Formal Modelling and Analysis of Dynamic Reconfiguration of Dependable Systems

Anirban Bhattacharyya

A thesis submitted in partial fulfilment of
the requirements for the degree of
Doctor of Philosophy

Newcastle University
School of Computing Science

January 2013

Abstract

The contribution of this thesis is a novel way of formally modelling and analyzing dynamic process reconfiguration in dependable systems.

Modern dependable systems are required to be flexible, reliable, available and highly predictable. One way of achieving flexibility, reliability and availability is through dynamic reconfiguration. That is, by changing at runtime the structure of a system – consisting of its components and their communication links – or the hardware location of its software components. However, predicting the system's behaviour during its dynamic reconfiguration is a challenge, and this motivates our research.

Formal methods can determine whether or not a system's design is correct, and design correctness is a key factor in ensuring the system will behave predictably and reliably at runtime. Therefore, our approach is formal. Existing research on software reconfiguration has focused on planned reconfiguration and link mobility. The focus of this thesis is on unplanned process reconfiguration. That is, the creation, deletion and replacement of processes that is not designed into a system when it is manufactured. We describe a process algebra (CCS^{dp}) which is *CCS* extended with a new type of process (termed a *fraction process*) in order to model process reconfiguration. We have deliberately not introduced a new operator in CCS^{dp} in order to model unplanned reconfiguration. Instead, we define a bisimulation ($\sim_{of}$) that is used to identify a process for reconfiguration by behavioural matching. The use of behavioural matching based on $\sim_{of}$ (rather than syntactic or structural congruence-based matching) helps to make models simple and terse. However, $\sim_{of}$ is too weak to be a congruence. Therefore, we strengthen the conditions defining $\sim_{of}$ to obtain another bisimulation ($\sim_{dp}$) which is a congruence, and (therefore) can be used for equational reasoning. Our notion of fraction process is recursive to enable fractions to be themselves reconfigured. We bound the depth of recursion of a fraction and its successors in order to ensure that $\sim_{of}$ and $\sim_{dp}$ are decidable. Furthermore, we restrict the set of states in a model of a system to be finite, which also supports decidability of the two bisimulations and helps model checking. We evaluate CCS^{dp} in two ways. First, with respect to requirements used to evaluate other formalisms. Second, through a simple case study, in which the reconfiguration of an office workflow is modelled using CCS^{dp}.

Declaration

The work presented in this thesis was done by myself, unless otherwise stated, and the thesis was written by me.

Dedication

To the maker of all things, without whose help I could not have done my research.

Long is the way
And hard, that out of Hell leads up to light.

– John Milton, Paradise Lost, Book 2

Acknowledgements

I would like to thank the following people for their help during my research. First, John Shaw Fitzgerald (my supervisor) for giving me the opportunity to study for my PhD at Newcastle University. John's encouragement, advice, guidance and overall support have been invaluable – I cannot thank him enough! I also want to thank Clifford Bryn Jones and Maciej Koutny for their useful criticism of my research and advice. Jon Warwick arranged the EPSRC funding for my research and Matthew Harris administered my MARI trivial commutation, and I gratefully acknowledge their help.

I have been very fortunate to have done my research in a friendly environment and in the company of supportive colleagues. Manuel Mazzara gave freely of his time and we had many hours of stimulating argument. Manuel is that rarity among creatures – an open-minded process algebraist! I am particularly indebted to Manuel for his advice on simplifying the semantics of CCS^{dp}. I must also acknowledge Mario Bravetti for clarifying my understanding of labelled transition system semantics, and Luke Ong for his observation that different bisimulations can be used for different purposes. Massimo Strano suggested the reconfiguration of an office workflow when I was looking for a case study, and Joey Coleman advised me to use presentations to clarify ideas – targetted on 5-year olds! I must also thank Peter Nicolls for educating me about chemical processes, and Olivier Commowick for making his LaTeX template available on the World Wide Web.

On a personal note, I would like to thank Anna Lewis and Caroline Clark for getting me back on my feet during writing up after a painful bout of sciatica and back pain. My friends Diane and Katy were constant in their encouragement and ceaselessly nagged me to get the job done! My parents gave unwavering support over the years, and I am glad their long wait is over. I am most grateful to my wife Gill for her love and patience – I could not have done my research without her.

This is a revised version of my thesis. I would like to thank my examiners, José Luiz Fiadeiro and Lance Jason Steggles, for their detailed comments on the original submitted version.

This work has been partly funded by the EPSRC under the terms of a graduate studentship.

Contents

Chapter 1

Introduction

Contents

1.1 Background

The dialectics of human competition, that is, the interaction between opposing human forces, drives technological change. The classic example is war. The conflict between opposing forces during the Second World War directly caused the construction of the first electronic digital computers (see [Hod87], [Flo83], [Coo83], [Cha83] and [MN82]). The dialectics of competition still drives computing technology [Hun12], and requires computing systems, including dependable systems, to be more flexible, reliable, available and predictable [BF08]. One way of achieving greater flexibility, reliability and availability is through the use of dynamic reconfiguration. That is, by changing at runtime the structure of a system – consisting of its components and their communication links – or the hardware location of its software components. However, predicting the behaviour of a system during its dynamic reconfiguration is a challenge.

We define several terms used in the discussion.

Definition 1.1.1 *Flexibility is the ability to change or to be changed in order to deliver service under different operating conditions or to deliver a different service.*

For example, a highly flexible distributed system can maintain its performance during heavy loading on a particular computer by load balancing. The system can deliver a new service by importing a new component that provides the service. In this thesis, a *flexible system* is taken to mean a highly flexible system. Our notions of system and service are the same as those defined in [ALRL04].

Definition 1.1.2 *Reliability is the ability to deliver correct service with no interruption due to a failure. A correct service delivery is a service delivery that meets the service requirements. A failure is an incorrect service delivery.*

A highly reliable system delivers correct service on demand [ALRL04], and any downtime **not** due to a failure (e.g. due to maintenance) is ignored in measuring the reliability of the system. The system can maintain its reliability if a fault is activated in a component by replacing the faulty component with a redundant component. We use the term *reliable system* to mean a highly reliable system.

Definition 1.1.3 *Availability is the ability to deliver service with no interruption.*

A highly available system can maintain its availability during maintenance by using multiple replicas of a component, which is upgraded by replacing each of its replicas in turn whilst the other replicas deliver service [TMMS01]. We use the term *available system* to mean a highly available system.

Thus, dynamic reconfiguration helps to increase the flexibility, reliability and availability of a computing system. These attributes in turn facilitate the evolution of computing systems, including dependable systems (defined below). Therefore, dynamic reconfiguration supports the evolution of dependable systems, which was recently identified as a key problem domain by several software engineering researchers and practitioners [CHNF10].

Definition 1.1.4 *Predictability is the ability to determine whether or not a service delivery will be correct.*

A service delivery is predictable if it is possible to determine whether or not the service delivery will be correct. A system is predictable if its service delivery is predictable. We use the term *predictable service delivery* to mean a highly predictable service delivery, and the term *predictable system* to mean a highly predictable system.

Definition 1.1.5 *Dependability is the ability to deliver service that can justifiably be trusted to meet its requirements.*

Our definition of dependability is based on [ALRL04], and makes explicit the relationship between dependability and requirements. The definition identifies the need for justification of trust. That is, the need for evidence that justifies the trust that a service delivery will meet its requirements. It must be possible to use the evidence to determine whether or not the service delivery will be correct. That

is, the service delivery must be predictable. Dependability is a generic term that encompasses several attributes of a system, including reliability and availability, and we add flexibility to this set of attributes. We use the term *dependable system* to mean a highly dependable system.

The importance of ensuring predictable service delivery during dynamic reconfiguration can be understood by considering the application domains of dependable systems, such as control of air-traffic, chemical plants, power stations and automated production lines. Thus, dependable systems are usually large, complex, geographically distributed, expensive and (consequently) long-lived. They must be highly predictable and reliable, otherwise they will not be deployed. They must be also highly available, because shutting down the system is impractical, or the cost of downtime is too high, or because the controlled environment could become dangerously unstable if the system became unavailable. Therefore, system designers need a high level of confidence that a dependable system will meet its requirements during normal operation. Furthermore, any reconfiguration of the system (whether this is due to evolving requirements, improved computing technology, or to mask an internal error) must be made whilst it is operational, and with no loss of confidence. Failure to achieve this can have catastrophic consequences, which raises the question of how best to ensure that the service delivered by a dependable system is predictable and correct, both during the normal operation of the system and its dynamic reconfiguration.

Definition 1.1.6 *A service design is correct if the service requirements are satisfied by the design.*

A system's design is the design of the service that the system has the ability to deliver. A system's design is correct if its service design is correct.

The correctness of a service design is a key factor in achieving a predictable and correct service delivery. Formal reasoning can determine whether or not a service design is correct, and the reasoning is supported by formal approaches. Therefore, we are interested in formal approaches to dynamic reconfiguration. Furthermore, we are interested in software reconfiguration, because software is much more mutable than hardware. Therefore, we restrict our attention to formal approaches to dynamic software reconfiguration.

Existing research in dynamic software reconfiguration can be grouped into three cases (see Figure 1.1).

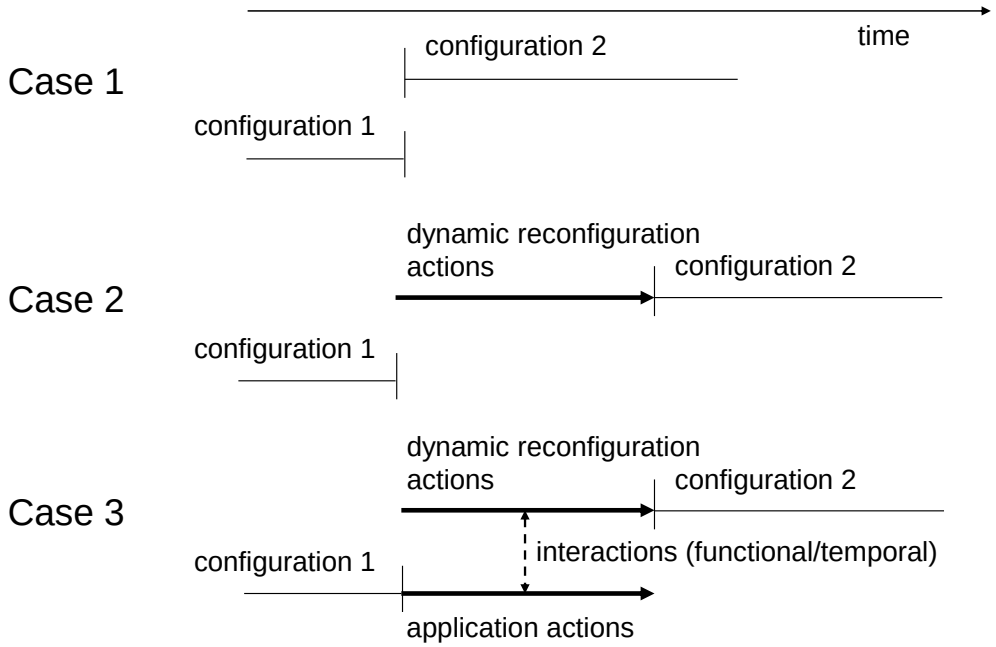


Figure 1.1: Dynamic Reconfiguration Cases.

Case 1 is the instantaneous reconfiguration of a system, in which the duration of the reconfiguration interval is negligible and any executing task in configuration 1 that is not in configuration 2 is aborted. This is the traditional method of software reconfiguration [Ves94], and is applicable to small, simple systems running on a uniprocessor.

Case 2 is the non-instantaneous reconfiguration of a system, in which the duration of the reconfiguration interval is significant and any executing task in configuration 1 is either aborted or suspended until the reconfiguration is complete. This is the most common method of software reconfiguration (see [SVK97], [AWvSN01], [BD93] and [KM90]), and is applicable to some large, complex, distributed systems. If the duration of the reconfiguration is bounded and the controlled environment can wait for the reconfiguration to complete, then the method can be used for hard real-time systems; otherwise, the environment can become irrecoverably unstable and suffer catastrophic failure.

Case 3 is the non-instantaneous reconfiguration of a system, in which the duration of the reconfiguration interval is significant and tasks in configuration 1 execute concurrently with reconfiguration tasks. This method avoids aborting tasks and reduces the delay on the application due to reconfiguration, but it introduces the possibility of functional and temporal interference between application and reconfiguration tasks. If the interference can be controlled, then this method is the most suitable for large, complex, distributed systems, including hard real-time systems. This is the least researched method of software reconfiguration. Existing research in Case 3 has focused on temporal interactions between application and reconfiguration tasks, and on achieving schedulability guarantees (for example, see [FW05] and [Mon04]). There is little research on functional interactions between application and reconfiguration tasks (see [MT00] and [BCDW04]).

Therefore, the focus of our research is on Case 3. Specifically, we aim to develop a process algebra that can model functional interactions between application and reconfiguration tasks, and can support the verification of system requirements defined over the reconfiguration interval. Furthermore, unlike existing formal approaches, our approach will enable unplanned dynamic process reconfiguration to be modelled abstractly, which is necessary for modelling dynamic evolution. Our objectives are to extend *CCS* with a special process for modelling reconfiguration (rather than a special operator), define the semantics of the resulting process algebra

(CCS^{dp}) and develop its equational theory. We will evaluate CCS^{dp} in two ways. First, with respect to requirements used to evaluate other formalisms. Second, through the reconfiguration of a simple office workflow used as a case study.

1.2 Approach

In modelling functional interactions between application and reconfiguration tasks, using a single formalism to model the two types of task is simpler than using different formalisms because the overhead of translation between formalisms is avoided. Therefore, we focus on approaches based on a single formalism.

We need to model the process through which an application performs its computations and the process through which the application is reconfigured, in order to model the functional interaction between the two processes. Process algebras are well-suited to modelling processes and process interaction. Furthermore, the structure of a process expression changes naturally as a result of its computation, which facilitates modelling the structural change of a process expression due to reconfiguration actions. Process algebras can model link reconfiguration [Mil99] and process relocation [Uny01], and also non-terminating and concurrently executing tasks, which are common in control systems. Therefore, we focus on process algebras as the basis for our formalism.

Existing research on process algebras for dynamic software reconfiguration has focused on *planned reconfiguration*. That is, reconfiguration which is incorporated in the original design of a system. There is very little research on *unplanned reconfiguration*, that is, reconfiguration which is **not** incorporated in the original design of a system, which is relevant for legacy systems and for the evolution of systems. Furthermore, much of the research on planned reconfiguration is on link reconfiguration and process relocation. There is little research on replacement of non-terminating processes, which is the most common type of process in control systems. Therefore, we focus our attention on process reconfiguration.

Our process algebra is CCS^{dp}; so called because it is based on *CCS* [Mil99] and is focused on **d**ynamic **p**rocess reconfiguration. *CCS* is used as the base for three reasons. First, *CCS* is one of the simplest process algebras that is capable of modelling computations. Therefore, it is easier to extend to suit our modelling requirements than a more complex process algebra; and (for the same reason) it is

an easier environment in which to experiment with new modelling constructs and 'tune' them to our requirements than a more complex process algebra. Second, *CCS* is the base for π-calculi [Mil99]. Therefore, it should be possible to extend CCS^{dp} to model link reconfiguration and process relocation. Third, *CCS* has no facility for link reconfiguration. Therefore, it should be possible to extend CCS^{dp} with real-time constructs to model process reconfiguration in control systems that do not require link reconfiguration. The key activity in process reconfiguration is process replacement, which we model using a special process (termed a *fraction process*). We do not use a special reconfiguration operator (such as the interrupt operator in *CSP* [Hoa85] or the workunit in $\texttt{web}\pi_\infty$[Maz06]) because an operator requires syntactic proximity between the operands, which implies that both the process to be replaced and the process replacing it must be in the model of the system, which (in turn) implies that both processes must represent quantities in the system, because the model is supposed to be an abstraction of the system and (therefore) should not contain any fictitious construct. Therefore, a reconfiguration operator allows only planned reconfiguration to be modelled, and it precludes the modelling of unplanned reconfiguration. In contrast, fraction processes enable both planned and unplanned reconfiguration to be modelled, and allow models to be changed incrementally in a modular fashion that corresponds to the reconfiguration of an application by a patch.

The absence of implementation mechanisms (such as messages, interrupts and timers) in the definition of a fraction process enables the fraction to model the effects of dynamic reconfiguration and omit the mechanisms that cause it. Consequently, CCS^{dp} can be used to specify the process of dynamic reconfiguration. Furthermore, the models can be analysed to determine reconfiguration paths and verify their properties without being encumbered with unnecessary implementation detail.

The semantics of CCS^{dp} is given in terms of a labelled transition system (LTS) only. This is done because the LTS semantics is sufficient for equational reasoning and verification based on model checking, which are the main forms of reasoning supported by process algebras. It is conventional to provide a reduction semantics for a new process algebra. This is done for two reasons. First, to enable people unfamiliar with process algebras to understand the meaning of process expressions more easily. Second, to conform to the tradition of writing a reduction semantics that dates back to the λ-calculus [Bar84]. Thus, given the LTS semantics, the reduction semantics is unnecessary for determining the meaning of process

expressions in CCS^{dp}, and (therefore) it is omitted.

The proof technique used to establish results in CCS^{dp} had to meet two requirements. First, the proofs had to be more rigorous than conventional proofs in process algebra. This is because CCS^{dp} is an unconventional process algebra. In a conventional process algebra, conventional proofs typically omit steps in the reasoning that can be filled by readers with sufficient background knowledge. However, in an unconventional process algebra, an omitted step in a conventional proof may not correspond to any background knowledge, and thereby cause doubt about the truth of the proposition to be proved. Second, the proofs had to be understandable and verifiable by a greater diversity of researchers in theoretical computing than only process algebra specialists. This is because the idea of using a special process (rather than a special operator) to provide functionality in a formalism may be useful beyond the domain of process algebra. Therefore, we used a proof technique inspired by [BFL+94], which is based on Gentzen's system of Natural Deduction [Pra65],[GTL89].

1.3 Thesis Structure

The rest of the thesis is organized as follows.

In Chapter 2, we identify the different approaches to dynamic software reconfiguration, and focus on dynamic architecture description languages (DADLs) as the most promising approach to managing dynamic reconfiguration of large systems. We identify dynamic reconfiguration issues and some dependability issues, and use these to examine a selection of DADLs in order to evaluate their suitability for the dynamic reconfiguration of dependable systems and to identify their essential features for modelling and analysis. Thus, we determine modelling and analysis requirements of system designers on formalisms, which cannot be done by a review of formalisms alone.

In Chapter 3, we identify the different formal approaches to the study of dynamic reconfiguration, and focus on π-calculi as the most promising approach to the modelling and analysis of dynamic reconfiguration of dependable systems. A selection of π-calculi is then evaluated using the modelling and analysis requirements identified in Chapter 2. Thus, limitations of the current research in this field are identified, which provides a focused motivation for our research. Specifically,

we show that existing π-calculi cannot model unplanned process reconfiguration because of their use of special operators to perform process reconfiguration (such as the workunit in $\texttt{web}\pi_\infty$), which justifies our use of a special process to model process reconfiguration.

In Chapter 4, we define the syntax and LTS semantics of CCS^{dp} and develop an equational theory. A strong bisimulation ($\sim_{of}$) is defined to enable a dynamic binding (termed *process matching*) to be made between a fraction process and the processes it can reconfigure. The use of behavioural process matching based on $\sim_{of}$ helps to maximize the terseness of process expressions for unplanned process reconfiguration. However, because the $\sim_{of}$ bisimulation is not a process congruence, another strong bisimulation ($\sim_{dp}$) is defined that is a process congruence and (therefore) can be used for equational reasoning. We prove a number of results for $\sim_{of}$ and $\sim_{dp}$, which facilitate the use of $\sim_{of}$ for process matching and the use of $\sim_{dp}$ for equational reasoning. We briefly discuss the consistency of CCS^{dp} and the restrictions that make $\sim_{of}$ and $\sim_{dp}$ decidable. We also discuss different forms of matching, based on syntactic equality, structural congruence and behaviour. Finally, CCS^{dp} is evaluated with respect to the requirements used to evaluate π-calculi in Chapter 3.

In Chapter 5, we evaluate CCS^{dp} using the dynamic reconfiguration of a simple office workflow. We describe the workflow and its reconfiguration, define the requirements on the reconfiguration, express the workflow and its reconfiguration in CCS^{dp}, then evaluate CCS^{dp} using the requirements.

In Chapter 6, we briefly explore modifications of CCS^{dp} in order to overcome some of its limitations identified in Chapter 4 and Chapter 5. Specifically, changes to the syntax, LTS semantics and other definitions to accomodate the restriction operator ν. Also, a process identification scheme to target a specific process instance for reconfiguration.

In Chapter 7, we summarize the findings of the research, discuss their significance, and identify future work.

Publications

Preliminary versions of the work presented in chapters 2, 4 and 5 of the thesis are in the following publications:

1. M. Mazzara and A. Bhattacharyya. On modelling and analysis of dynamic reconfiguration of dependable real-time systems. In *Proceedings of the Third International Conference on Dependability (DEPEND 2010)*, 2010.

2. A. Bhattacharyya and J. S. Fitzgerald. Development of a formalism for modelling and analysis of dynamic reconfiguration of dependable real-time systems: A technical diary. In *Proceedings of the 2008 RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems*, pages 67-72, 2008.

3. M. Mazzara, F. Abouzaid, N. Dragoni and A. Bhattacharyya. Toward design, modelling and analysis of dynamic workflow reconfiguration - A process algebra perspective. In *Proceedings of the 8th International Workshop on Web Services and Formal Method (WSFM 2011)*, Lecture Notes in Computer Science Vol. 7176, pages 64–78. Springer-Verlag, 2012.

Chapter 2

Review of Systems Supporting Dynamic Reconfiguration

Contents

In this chapter, the different approaches to dynamic software reconfiguration are briefly described in order to identify issues that occur in dynamically reconfiguring software in dependable systems. These issues are then used to evaluate examples of the most promising approach to dynamic software reconfiguration, namely, dynamic architecture description languages (DADLs), in order to identify their key features for modelling and analysis. These features are then used to define requirements on a formalism for the modelling and analysis of dynamic software reconfiguration.

Research into the dynamic reconfiguration of systems dates back at least to the 1960s [CV65]. Since then, the research has followed a variety of approaches, based on programming languages, object execution environments, operating systems, distributed systems, module interconnection languages, architecture description languages and architecture modification languages. Furthermore, systems have been constructed, such as Simplex [SRG96] and Chimera II [SVK97], which support the dynamic reconfiguration of hard real-time applications. We briefly describe typical examples of each approach and their limitations, and then discuss which approach is the most promising for the dynamic reconfiguration of dependable systems.

The term *system consistency* is used throughout this chapter, and we define it as follows.

Definition 2.0.1 *In a computing system, consistency is the restriction that two or more specified entities should have the same value in some well-defined sense. The entities are consistent if they all have the same value; otherwise, the entities are inconsistent. A system is consistent if its consistency restrictions are satisfied; otherwise, the system is inconsistent.*

For example, the states of different replicas of a component are required to be the same at specified times. If the replicas are all in the same state at a specified time, then the replicas are consistent at that time; otherwise, the replicas are inconsistent at that time, and the system is inconsistent at that time.

2.1 Programming Languages

Several programming languages, including Lisp, Smalltalk and C++ [HG98], allow executable code of software components of a system to be changed at runtime,

thereby enabling the system to evolve dynamically. However, there is no associated formalism to determine the system's behaviour during its dynamic reconfiguration.

Fabry describes an indirection mechanism to handle the dynamic replacement of abstract data types (ADTs) [Fab76]. ADTs are versioned, and each version is associated with an indirect segment. Clients access an ADT through its indirect segment, and the creation of a new version of the ADT results in the client reference to the indirect segment being updated (through a capability mechanism) to reference the indirect segment of the new version. It is assumed that all clients of an ADT will be able to interact with its newest version. No formal proof is given that use of the mechanism will preserve system consistency.

2.2 Object Execution Environments

The CORBA 2.0 ORB and Microsoft's COM provide facilities for the dynamic creation and deletion of objects; the dynamic registration and deregistration of object interfaces at an interface repository; dynamically discovering the interfaces supported by objects, the methods provided and their parameters; and for dynamically constructing and making invocations to the discovered methods [OHE96]. Clearly, these facilities support the dynamic reconfiguration of a system (including system evolution), but there is no associated formalism to determine the system's behaviour when these facilities are used.

2.3 Operating Systems

Multics uses dynamic linking, with hardware support in the form of indirect addressing and a linkage fault indicator, to reduce the linking time and memory utilisation of programs [MD87]. Shared libraries in SunOS 4.0 and dynamic link libraries in Microsoft Windows are used for the same purpose. If dynamic linking is used in combination with a modular programming language, it supports the dynamic evolution of a program. Indeed, the increasing speed of raw processing operations relative to I/O operations led Franz to propose the loadtime generation of executable code [Fra97]. However, it is left entirely to the programmer to determine the consequences of these dynamic changes on the program's behaviour, without the assistance of any recommended formalism.

Gupta and Jalote describe a scheme for the dynamic upgrade of a program written in a procedural language [GJ93]. The upgrade involves creating a process using the new version of the program, suspending the process of the old version of the program when none of its changed functions is executing (i.e. on the runtime stack), transferring the state of the old process to the new process using programmer-supplied code, terminating the old process and resuming execution of the new process. The program to be upgraded runs as a child process of a modification shell, so that its runtime stack can be monitored; and the program is linked to a special library, so that descriptors and offsets of the open files held by the old process can be transferred to the new process. The granularity of change is a function, and functions can be added, replaced or deleted. The notion of a valid upgrade of a program is defined formally as a change in which the program terminates satisfying the post-condition of either the old version or the new version, and conditions are defined that guarantee the valid upgrade of a program. The research has several limitations. First, continuously executing functions cannot be replaced. Second, a program is replaced instantaneously and on a 1-to-1 basis, which is problematic for large systems. Third, the issue of interference between functions is not addressed. Fourth, the scheme is not applicable to distributed systems (although the researchers claim otherwise) due to the absence of a mechanism for managing dynamic reconfiguration in distributed processes. Fifth, the process of dynamic reconfiguration is not formally modelled. In a later work [GJB96], Gupta et al. redefine the notion of valid upgrade as a change in a program that guarantees the reachability of a reachable state of the new version, and sufficiency conditions are given for a valid upgrade that maps a reachable state of the old version to its corresponding reachable state in the new version. The design correctness of both versions of the program is assumed. A major limitation of the formalism is that it is state-based, which makes it unsuitable for modelling distributed systems, since distributed systems do not have global state.

2.4 Distributed Systems

Distributed systems are the natural environment for dynamic reconfiguration, and (unsurprisingly) most of the research has concentrated on building mechanisms for the dynamic reconfiguration of distributed systems, with no associated formalism for determining a system's behaviour during its reconfiguration. For a review of mechanisms supporting dynamic reconfiguration of distributed systems, see [SF93]. Nevertheless, the issue of ensuring consistency in a system undergoing

dynamic reconfiguration has been investigated by a number of projects.

2.4.1 Argus

Argus is a statically typed object-based programming language and execution environment for fault-tolerant distributed applications that are required to maintain data online, for long time intervals, and in a well-defined state [LS83]. An application consists of subsystems, with each subsystem containing one or more multi-threaded guardians (i.e. fault-tolerant objects). A guardian is the granularity of dynamic reconfiguration. It is located on a single physical node (i.e. computer), communicates through remote procedure calls (RPCs), and it provides an interface consisting of a set of handlers (i.e. methods) through which its behaviour and state can be accessed. The interface and body of a guardian are implemented as separate objects, with the interface object containing the identifiers (i.e. ports) of the handlers in the guardian's body; an indirection which helps the dynamic change of guardians. The clients of a guardian can access its body only through its interface object, which helps to encapsulate the guardian. The Argus system has a catalogue service mapping string names and object types to objects, so that communication links between guardians can be made dynamically. Dynamic reconfiguration in Argus consists of the creation, deletion and replacement of guardians, and the relocation of guardians on physical nodes.

The focus of the work on dynamic reconfiguration is guardian replacement, and Bloom formally defines the conditions for replacing one subsystem by another so as to preserve application correctness. For guardian replacement on a 1-to-1 basis[1]: the pre-condition is that the old guardian must be quiescent (i.e. it must not be engaged in any action or communication). The post-condition is that the new guardian must preserve or extend the provided interface and the state and state transitions of the old guardian, such that any extension must be functionally invisible to the clients of the old guardian. The invariant condition is that the old guardian must remain quiescent during the replacement process. Guardian quiescence is user- initiated, and it involves either waiting until the old guardian is not engaged in any transaction, or aborting all the transactions in which the guardian is engaged [BD93]. Guardian quiescence is used to ensure that the old guardian is in a well-defined state during its state transfer to the new guardian, which results in a well-defined state for the new guardian; and the atomicity of

[1]The conditions for subsystem replacement are identical to those for guardian replacement on a 1-to-1 basis, with the subsystem being treated as a single unit (like a guardian).

Argus transactions ensures the consistency of the application. Bloom excludes guardian replacement involving interface extensions that interfere with preserved handlers, because this causes the change to become functionally visible to the clients of the old guardian.

The research has three limitations. First, it does not address change in the dependence between client and server objects. As a result, the conditions for guardian replacement are too strong. Second, Argus uses a flat type system (taken from CLU) that is unsuitable for guardian replacement, especially where the replacement relationship between guardians is not 1-to-1. Third, Argus has no formal semantics, and its process of dynamic reconfiguration is not modelled formally [Blo83]. Therefore, it is not possible to prove the functional and temporal correctness of an application.

2.4.2 Eternal

Eternal is an execution environment (running on CORBA) that supports transparent fault-tolerance and dynamic upgrade of CORBA applications [MMSN98]. An application consists of CORBA objects that communicate by RPCs. Eternal uses active replication of objects to provide fault-tolerance with high availability, with each replica located on a distinct physical node. A reliable multicast communication with total ordering of messages is used to ensure consistency between the replicas [MMSA+96]. All replicas are single-threaded. An object is the granularity of dynamic reconfiguration.

The focus of the work on dynamic reconfiguration is object upgrade, which is controlled by the Eternal Evolution Manager [TMMS01]. The Evolution Manager is used to create a composite object that contains both the old and new versions of the object to be upgraded, and has conversion code mapping old state to new state to keep the states of the two versions consistent. This enables the composite to behave like either version, depending on the state of the upgrade process, and it reduces the duration of the switchover between the two versions. For an interface preserving upgrade to an object, the upgrade process consists of replacing in turn each replica of the old version with a replica of the composite, whilst the other replicas continue to provide the old service to the object's clients. Following each replacement, the state of an old version is transferred to the new replica, so that the new replica can provide the service of the old version. During the state transfer, the object being upgraded is required to be quiescent to ensure consistency between its replicas. When all the old replicas have been replaced, an atomic switchover

is performed, after which the replicas of the composite object provide the service of the new version. During the atomic switchover, the object being upgraded is required to be quiescent to ensure consistency between its replicas. Replicas containing only the new version of the object then replace the replicas of the composite object, by a process similar to that described above. The Evolution Manager uses a CORBA method invocation graph to determine when the objects affected by the upgrade will be quiescent.

For an interface upgrade to an object, the upgrade process involves changing the object's clients, which must be included in the atomic switchover. The process is similar to that described above, with the clients being replaced last. However, the conditions for the switchover can be weakened through the use of wrapper functions in the server's composite object, which convert old method invocations into new method invocations. This enables the client objects to be changed after the switchover of the server object, and also independently of each other.

The research has two main limitations. First, it is not possible to upgrade an object that is never quiescent, such as an object executing an infinite polling loop. Second, Eternal has no formal model, and so it is not possible to prove the correctness of a dynamic reconfiguration.

2.4.3 Dynamic Reconfiguration Service

The Dynamic Reconfiguration Service (DRS) is an execution environment (running on CORBA) that supports transparent dynamic reconfiguration of CORBA applications [AWvSN01]. An application consists of multi-threaded CORBA objects that communicate by message passing, with each object located on a single physical node. An object is the granularity of dynamic reconfiguration; and the dynamic reconfiguration operations are the creation, deletion and replacement of objects, and the relocation of objects on physical nodes.

The focus of the work is object replacement, which is controlled by the Reconfiguration Manager. For object replacement on a 1-to-1 basis[2]: the pre-condition is that the old object must be reconfigurable (i.e. designed for reconfiguration) and quiescent. The post-condition is that the new object must preserve or extend the interface of the old object and the state and state transitions of the old object, such

[2]Object replacement on a set basis is identical to object replacement on a 1-to-1 basis, with the set of objects being treated as a single unit (like an object).

that any extension must be functionally invisible to the clients of the old object. The invariant condition is that the old object must remain quiescent during its reconfiguration. The process of object replacement is user-initiated, and it involves driving the old object into a quiescent state by filtering its clients' request messages: the Reconfiguration Manager notifies the old object it is to be reconfigured; and the facilities of the CORBA ORB and the information it holds on object invocations (e.g. a method invocation graph) are then used to block all request messages sent to the object that would prevent its quiescence; all other messages are allowed to proceed. If the object is active (i.e. time-triggered) it is changed to become reactive (i.e. event-triggered) by invoking a method of its reconfiguration interface. When the old object notifies the Reconfiguration Manager it is quiescent, its state is transferred to the new object. The state of either object can be inspected (and modified if necessary) using methods of its reconfiguration interface, to ensure that invariants on the application-state hold. Finally, the new object replaces the old object, and the blocked messages are redirected to the new object. The DRS Location Agent helps to rebind the client objects with the new server object.

The research has three limitations. First, replacement of a set of objects is done simultaneously, which does not scale well. Second, the issue of interference between new and preserved methods of an object is not addressed in object replacement. Third, DRS has no formal model, and so it is not possible to prove the correctness of a dynamic reconfiguration.

2.4.4 Related Work

There are distributed systems (in addition to those discussed above) that address the issue of consistency; see [ED97], [SM02] and [ALS03]. However, their mechanisms and processes for ensuring consistency are not significantly different from those that have been discussed already.

2.5 Module Interconnection Languages

Module interconnection languages (MILs) resulted from the idea that 'programming-in-the-large' involves solving a different set of problems in comparison to 'programming-in-the-small' [DK75]. Thus, MILs are complementary to ordinary programming languages, and they are used to define the composition and connectivity structure of large software systems.

A MIL defines a system recursively as a composition of modules. Each module is a container of resources, where a resource is a nameable programming construct, such as a variable, data type, or function, which can be made available for reference by other modules. A module definition identifies: the resources provided by the module to other modules (using the `provides` construct), the resources required by the module from other modules (using `requires`), the external modules to which access is needed (using `has_access_to`), and the decomposition of the module into functions and child modules (using `consists_of`). The designer of a module can impose restrictions on its child functions and modules by identifying resources they must provide (using `must_provide`). The different versions of a module implemented in different programming languages can also be identified (using `realisation`). Figure 2.1 gives an example of the use of these constructs in a module definition. For a review of MILs, see [PDN86].

MILs are useful during system construction, because they enable modules to be designed in parallel, and to be compiled and linked independently, and because they ensure a degree of inter-module compatibility by checking the required and provided interfaces of connected modules (based on matching names). Furthermore, an MIL can enforce a system structure, since a module's interface cannot be changed without explicitly changing the system's structure; which is also useful during evolution of the system.

MILs have limitations: they do not specify types, or module behaviour, or interactions between modules. Hence, although MILs can describe the structure of a large software system, they cannot be used to prove its correctness. Nevertheless, the issue of ensuring consistency in a system undergoing dynamic reconfiguration has been investigated by a number of MIL projects, including Polylith [Pur94], Durra [BWD+93] and Aster [BISZ98].

2.6 Architecture Description Languages

Architecture description languages (ADLs) are used to describe the structure of a system, the behaviour of its components and their interactions, and to express restrictions on these. Architectures and ADLs are active research areas.

```
module ABC
    author      'Joseph Green'
    date        '1st January 2004'

    provides    a, b, c
    requires    x, y
    consists_of function XA, module YBC

    function XA
        must_provide    a
        requires        x
        has_access_to   module Z
        real x, integer a
        realisation
            version FORTRAN resources file (<FORTRANXA>)
            end FORTRAN
            version Pascal resources file (<PascalXA>)
            end Pascal
            version ALGOL resources file (<ALGOLXA>)
            end ALGOL
    end XA

    module YBC
        must_provide    b, c
        requires        a, y
        real y, integer a, b, c
    end YBC

end ABC
```

Figure 2.1: A module definition in an MIL [PDN86].

The ISO/IEC/IEEE 42010 standard [ISO11] defines an architecture as:

'*fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution*'.

The ISO/IEC/IEEE 42010 standard defines an architecture description as a:

'*work product used to express an architecture*'.

The ISO/IEC/IEEE 42010 standard defines an ADL as:

'*any form of expression for use in architecture descriptions*'.

Typically, an ADL consists of components, connectors, configurations, and a language for defining restrictions on these. For a review of ADLs, see [MT00].

A component is used to model a computational unit that can have state, and it can be implemented in a variety of ways – as a procedure, an object, or an entire program. A component has an interface consisting of a set of interaction points (e.g. ports) through which it communicates with its environment. Interaction points that provide a service to the component's environment are distinguished from those that require a service. A component can have a type and a semantics, and restrictions can be defined on its behaviour. In some ADLs, such as C2, system evolution is supported through component subtyping [MRT99]. Real-time ADLs enable the non-functional properties of a component to be defined: MetaH enables the period of a periodic component to be defined [BEJV96]; and UniCon enables the priority of a component to be defined [SDK+95].

A connector is used to model communication between components, and it can be implemented in a great variety of ways – as a separately compiled entity, such as a message router; or as an embedded construct, such as a shared variable, buffer, instruction to a linker, method invocation, communication protocol, or a pipe [MT00]. The interface of a connector consists of interaction points that can be connected to the interaction points of components or other connectors. The type of a connector is an abstraction of interaction between components, such as an RPC or an asynchronous communication, and its semantics is a specification of communication protocols. The restrictions on a connector are usually defined on

its semantics and connectivity. Connectors achieve indirection in communication between components, and thereby help to model dynamic binding between components. Connectors also support system evolution through subtyping and incremental data filtering. Regarding the non-functional properties of connectors, UniCon enables connector attributes to be defined for schedulability analysis [SDK+95].

A configuration is used to model the composition and connectivity structure of a system, and it is a graph of components, connectors, and their inter-connections. It is the natural locus for defining global restrictions on a system, such as end-to-end deadlines, and the absence of deadlocks and interaction cycles. A configuration is also very useful for the modelling and analysis of the dynamic reconfiguration of a system.

A subset of ADLs, termed dynamic architecture description languages (DADLs), focus on the problems of managing dynamic reconfiguration of a system, and these are examined in detail in Section 2.10.

2.7 Architecture Modification Languages

The research on architecture modification languages (AMLs) is in its early stages. Its purpose is to create an interchange architecture or meta-architecture, so that an architectural description in one ADL can be transformed into an architectural description in another ADL.

In principle, an AML should be able to model and analyse fundamental changes in a system that are beyond the scope of a DADL. However, at present the main AML is ACME, which provides no support for modelling dynamic reconfiguration. More importantly, since the modelling and analysis of dynamic reconfiguration of a system is still a research issue, it is advisable to investigate these simpler forms of change more fully before attempting to study more difficult forms of change.

2.8 Discussion

It is clear from the above that most of the approaches to the dynamic reconfiguration of a system have concentrated on mechanisms to implement dynamic reconfiguration, rather than on formal frameworks to predict the system's behaviour during

the reconfiguration. The exception is the research on ADLs, which emphasises the need for an explicit abstract model of a system's components and their interactions and the system's configuration. If the ADL has a formal semantics, the model can be used both to design the system and to prove its requirements. A formal semantics also makes the ADL particularly amenable to tool support. Furthermore, dynamic reconfiguration of large systems is a problem domain of the DADLs. Therefore, we focus on DADLs, and evaluate a selection of them using issues relevant to dynamic reconfiguration and dependability in order to determine modelling and analysis requirements on formalisms.

2.9 Issues

Issues for the dynamic reconfiguration of dependable systems can be identified by inspection of the published literature, some of which has been reviewed above and in Section 2.10. The issues have been separated into dynamic reconfiguration issues and dependability issues for greater clarity. The dependability issues are relevant for dependable systems in general, whereas the dynamic reconfiguration issues are particularly relevant for systems with a reconfiguration requirement. Collectively, the issues help to evaluate the suitability of a DADL and to identify its essential features for solving both dynamic reconfiguration and dependability problems.

2.9.1 Dynamic Reconfiguration Issues

1. What can be dynamically reconfigured in a system?

   The granularity of dynamic reconfiguration is the smallest item of the system that can be changed independently, and it affects the conditions under which the change can be performed safely. It is sometimes necessary to change a set of items, rather than an individual item, and this has a significant effect on the change conditions and (thereby) on the complexity of the reconfiguration process.

2. How can an item be dynamically reconfigured in a system?

   The process of dynamic reconfiguration of the system typically involves performing multiple basic operations. Restricting these operations to a small canonical set helps to simplify the modelling and analysis of the process. The operations include creation, deletion and replacement of a software component, and relocation of the component to a different physical node.

3. What method is used to manage the dynamic reconfiguration of a system?

   The process of dynamic reconfiguration of a large system is usually complex, and typically involves making multiple changes, with restrictions on their order, duration and synchronisation. Therefore, it is necessary to use a method to manage the process.

4. When can an item be dynamically reconfigured in a system?

   In a dependable system, an item should be dynamically reconfigured only when it is safe to do so. Therefore, the reconfiguration process must satisfy conditions that will guarantee the system's requirements are met both during and after execution of the process. These conditions can be thought of as the pre-conditions, invariant conditions, and post-conditions of the process, and usually involve type information and the system's state. In a real-time system, the conditions can be time-dependent, so timely detection of the conditions becomes an issue. This leads to the requirement that both detection of occurrence of the change conditions and execution of the dynamic reconfiguration process must be fully automated and must satisfy timing restrictions.

5. What are the effects of the interaction between the application actions of a system and the dynamic reconfiguration actions?

   The environment of a dependable system typically contains processes that are controlled by actions within the system. Since the system has high availability requirements, it must continue to operate normally during its dynamic reconfiguration. This suggests that the application actions of the system and the dynamic reconfiguration actions must run concurrently, which implies that they interact. Hence, it is necessary to define a formal model of the application actions of the system, the dynamic reconfiguration actions, and their interaction, and to use these to prove that the system will meet its requirements during its dynamic reconfiguration.

2.9.2 Dependability Issues

Dependability requires predictability, and predictability is a higher-order property of a system, which is defined with respect to first-order properties (such as functionality) and second-order properties (such as timeliness), which (in turn) are defined with respect to a model. Therefore, we identify dependability issues in terms of models found in many dependable systems that are required to be dynamically reconfigurable.

1. What concurrency model is used?

   Concurrency is a feature of many dependable systems. Therefore, the system must have a model of concurrent events and actions. In many real-time systems the granularity of concurrency is a thread in a software component, which is either single-threaded or multi-threaded.

2. Is the system amenable to analysis for both functional and temporal correctness?

   Formal proof provides the strongest guarantee of the functional correctness of a system's design, and schedulability analysis provides the strongest guarantee of the temporal correctness of the system's implementation. Therefore, the design of the system must have a formal semantics, and its resource allocations must be amenable to schedulability analysis [Pri99]. This implies that the time of occurrence of all time-critical events, and the durations of all time-critical actions and state transitions of the system and associated communication between system components, must be known during implementation of the system; or the bounds on their variability must be known.

2.10 Dynamic Architecture Description Languages

DADLs focus on the problems of managing dynamic reconfiguration of large systems. Therefore, we examine a selection of DADLs, using the issues identified in the previous section, in order to evaluate their suitability for managing the dynamic reconfiguration of dependable systems and to identify their essential features for modelling and analysis. A summary of these features is given in Table 2.1.

2.10.1 Darwin

Darwin is a statically typed DADL for configuring parallel and distributed applications that run on the Regis execution environment [MDK93]. It is an extension of the Conic configuration language [MKS89]. However, unlike Conic, Darwin can express dynamic connections between components, and is largely independent of the languages used to implement leaf-level components. Darwin is also less restrictive than Conic in locating software components on physical nodes.

A Darwin application is structured as a hierarchy of interacting components, which can be parameterised with simple types. The hierarchical structure has no

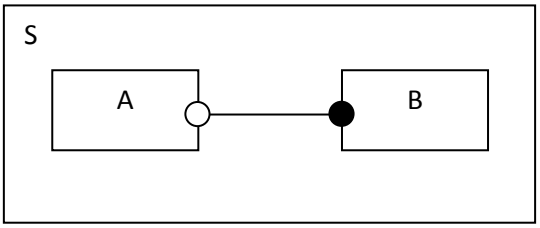
runtime representation, and the application is implemented using the leaf-level components (termed processes), each of which is located on a single physical node. The components interact through ports at their interface using synchronous and asynchronous message passing. A component's interface is partitioned into ports through which it provides services to other components (termed provided ports), and ports through which it accesses services required from other components (termed required ports). A port is a queue of messages with simple types, and a required port holds a reference to the provided port of another component (to which it has been bound). Ports are defined in the interface definition of their component. The interface definition of a composite component also defines the types and instances of its sub-components, the bindings between their ports, and the location of instances on physical nodes. A provided port can be bound to multiple required ports, and a required port can be bound to only one provided port. A port is a first class object, so that bindings between ports can be created dynamically.

Figure 2.2 shows the diagram and definition of a simple system (`S`) in Darwin, consisting of a client component (`A`) and a server component (`B`), taken from [MDEK95]. `require r` defines `r` as a required port of `Client`; `provide p` defines `p` as a provided port of `Server`; `inst` instantiates components `A` and `B` as sub-components of `System`; and `bind` defines the binding between `r` and `p`. Darwin also has a construct (`forall`) to declare an array of components or bindings; and a construct (`@`) to define the location of a component instance on an abstract processor. A direct communication path between any two abstract processors is assumed [MDK93].

2.10.1.1 Evaluation with respect to Dynamic Reconfiguration Issues

Dynamic reconfiguration in Darwin consists of the creation of components and port bindings. Deletion and modification of bindings is not allowed, in order to prevent interference between different dynamic reconfiguration processes. There are two forms of dynamic reconfiguration: lazy instantiation and dynamic instantiation, both declared using the `dyn` construct.

The original method of handling dynamic reconfiguration has the following conditions [KM90]: For binding creation, the pre-condition and invariant condition is that the component with the required port of the binding is quiescent. Component creation has no pre-condition, since on creation a component is quiescent and has no external binding. These conditions ensure system consistency during the dynamic reconfiguration process. Component quiescence is achieved using the




```
component Client {
    require r;
}
component Server {
    provide p;
}
component System {
inst
    A:Client;
    B:Server;
bind
    A.r -- B.p;
}
```

Figure 2.2: A simple client server configuration in Darwin [MDEK95].

`passivate` operation of a component, which makes it passive (i.e. the component is not engaged, and will not be engaged, in any transaction it initiated; although it can still engage in transactions it did not initiate). Invoking the `passivate` operation on a target component, and on any component capable of initiating a transaction involving the target, eventually results in quiescence of the target component [KM90]. However, this implies the environment must wait for a service if any of the passivated components are involved in the service provision; which is problematic for dependable systems because of their high availability requirement.

A less restrictive method for achieving component quiescence is proposed in [MGK96]. The target components are sent a blocking message to make them passive. However, a blocked component can become temporarily unblocked on being invoked by a component outside the target set, which in turn may be servicing a transaction caused by another component of the target set. Therefore, in order to prevent deadlock of the system, any component of the system outside the target set that is involved in servicing a transaction caused by a target component is also sent a blocking message. Hence, when all the target components are blocked, any further service request on the target set is queued, thereby ensuring quiescence of the target components; the blocked components outside the target set are unblocked, to minimise the disruption to system operation. This method makes a number of assumptions about the transaction structure of the system: it assumes that the system is free of deadlocks and livelocks in the absence of dynamic reconfiguration; that there is no cyclic dependency between transactions; and that a component does not interleave transactions. Both methods assume a transaction terminates and its initiating component is informed of its completion. Furthermore, a component has to be designed for reconfiguration in order to support its dynamic reconfiguration.

Darwin has a formal semantics (defined using the π-calculus) that enables the configuration between components, and the process of dynamic reconfiguration, to be formally expressed and analysed. For example, the client server configuration shown in Figure 2.2 can be modelled as follows [MDEK95]:

The `provide` construct, which declares a service to be provided through port `p`, is expressed as the agent $PROV(p,s)$, with $PROV(p,s) \stackrel{def}{=} !(p(x).\bar{x}s)$ where p is the access name, s is the service reference, and x is the location where service s is required.

PROV is replicated, because a provided port can be bound to arbitrarily many required ports.

Thus, the `Server` component is expressed as:

$$Server(p) \stackrel{def}{=} (vs)(PROV(p,s)|Server'(s)) \equiv (vs)(!(p(x).\overline{x}s)|Server'(s))$$

The `require` construct, which declares a service to be required through port `r`, is expressed as the agent $REQ(r,l)$, with $REQ(r,l) \stackrel{def}{=} r(y).\overline{y}l$ where r is the access name, y is the service provider, and l is the location where the service is required.

REQ is not replicated, because a required port can be bound to at most one provided port.

Thus, the `Client` component is expressed as:

$$Client(r) \stackrel{def}{=} (vl)(REQ(r,l)|Client'(l)) \equiv (vl)(r(y).\overline{y}l|Client'(l))$$

The `bind` construct, which declares a binding between two ports, is expressed as the agent $BIND(r,p)$, with $BIND(r,p) \stackrel{def}{=} \overline{r}p$

Clearly, the composition of $REQ(r,l)$ with $BIND(r,p)$ results in the *REQ* agent receiving the access name of the *PROV* agent, so that *REQ* and *PROV* can communicate. Thus, `System` is expressed as the composition:

$$System \stackrel{def}{=} (vr_A, p_B)(Client(r_A)|BIND(r_A,p_B)|Server(p_B))$$

Proving the correctness of this configuration consists of proving that client `A` will receive the service reference s provided by server `B`. Elaborating the definition of *System* gives:

$$System \equiv (vr_A,p_B)((vl)(REQ(r_A,l)|Client'(l))|\overline{r}_A p_B|(vs)(PROV(p_B,s)|Server'(s)))$$

$$\equiv (vr_A,p_B,l,s)(REQ(r_A,l)|\overline{r}_A p_B|PROV(p_B,s)|Client'(l)|Server'(s))$$

$$\equiv (vr_A,p_B,l,s)(r_A(y).\overline{y}l|\overline{r}_A p_B|!(p_B(x).\overline{x}s)|Client'(l)|Server'(s))$$

$$\rightarrow (vr_A,p_B,l,s)(\overline{p_B}l|!(p_B(x).\overline{x}s)|Client'(l)|Server'(s))$$

$\rightarrow (vr_A, p_B, l, s)(\bar{l}s|!(p_B(x).\bar{x}s)|Client'(l)|Server'(s))$

Clearly, the *Client'* agent needs to perform an input action in order to accept the service reference s, so that it can use the service; so that

$Client'(l) \stackrel{def}{=} l(z).Client''$

$\Longrightarrow System \rightarrow (vr_A, p_B, l, s)(!(p_B(x).\bar{x}s)|Client''(s/z)|Server'(s))$ Q.E.D.

The definition of `System` can be modified to demonstrate the dynamic instantiation of components (see Figure 2.3). `System` has a provided port `d`, which is used to access the service that dynamically instantiates `Client` components:

```
component System {
provide d <dyn>;
inst
    B:Server;
bind
    d -- dyn Client;
    Client.r -- B.p;
}
```

Figure 2.3: Dynamic instantiation of Client components in Darwin [MDEK95].

The modified `System` is modelled in [MDEK95] as:

$System \stackrel{def}{=} (vp_B, d, m)(Server(p_B)|PROV(d, m)|!(m().(vr)(Client(r)|BIND(r, p_B))))$

Thus, an agent requiring a new *Client* would access $PROV(d, m)$ using $\overline{d}$, and obtain the reference m to the service for instantiating *Client*s. The agent would then communicate with the guard $m()$ using $\overline{m}$, resulting in the instantiation of $Client(r)$ and its binding to $Server(p_B)$.

Darwin has three limitations with respect to dynamic reconfiguration. First, component replacement and deletion, and binding deletion are not supported. Second, because instantiation of components and bindings is performed without any ordering restriction, it is possible for a communication from a component to block without a known time bound. Third, the requirement for quiescence implies that the application actions of a system are suspended during the dynamic reconfigura-

tion process, thereby jeopardizing the availability of the system. Thus, the Darwin research does not analyse the interaction between the application actions and the dynamic reconfiguration actions of a system.

2.10.1.2 Evaluation with respect to Dependability Issues

Darwin does not fully address the dependability issues: its model of concurrency depends on the languages used to implement the processes. Processes communicate on a point-to-point basis with no time bound, and dynamic instantiation can result in a component being blocked with an unknown time bound. Hence, a Darwin design is not amenable to schedulability analysis.

A Darwin design can be analysed for safety using reachability analysis, and for liveness using labelled transition system analysis, although this has not been integrated with the analysis for correctness of a dynamic reconfiguration [MKG99].

2.10.2 Wright

Wright is a statically typed DADL for the formal modelling and analysis of software systems [AG94]. It was originally designed for systems with a static configuration [All97], and was later applied to dynamically reconfigurable systems.

A Wright system is structured as a graph of components and connectors, where a component represents a computational unit, and a connector specifies interactions between components. Each component is defined using its ports and a computation: the ports constitute the interface of the component, and are expressed as CSP processes defined in terms of their respective port events. The computation defines the behaviour of the component, and is also expressed as a CSP process defined in terms of the port events of the component. Each connector is defined using its roles and a glue: The roles are similar to ports, and specify the behaviour required of any component port attached to them, and are expressed as CSP processes defined in terms of their respective role events. The glue is similar to a computation, and defines the behaviour of the connector as a CSP process defined in terms of the role events of the connector. Thus, the behaviour of the system is expressed as the parallel composition of its computation and glue processes, and its configuration manager (termed configuror) process.

The definitions of components and connectors are type definitions, which are collected into styles for reuse. A style also contains constraints on instances of its component and connector types. A particular configuration of a system uses a specific style, a set of component and connector instances, and the bindings between the ports and roles of these instances. A port can be bound to exactly one role, and a role can be bound to exactly one port or another role, in a given configuration.

Figure 2.4 shows the diagram and definition of a simple client server system in Wright, consisting of a client component (C), a server component (S) and a connector (L), taken from [ADG98]. The *Client* − *Server* style defines: the *Client* component type with a single port (p) that can handle any number of iterations of the events: request output followed by reply input; the *Server* component type with a single port (p) that can handle any number of iterations of the events: request input followed by reply output; and the *Link* connector type with two roles (c and s), where c specifies the behaviour required of a *Client*'s port p, and s specifies the behaviour required of a *Server*'s port p. The *Link*'s *Glue* specifies that the sequence of corresponding input and output events of a *Link* should not be interrupted. The style also has the constraint that all *Client* instances must be connected to a unique *Server* instance. The *Simple* configuration uses the *Client* − *Server* style to declare the instances C, S and L; and defines the bindings between the ports of C, S and the roles of L.

2.10.2.1 Evaluation with respect to Dynamic Reconfiguration Issues

Dynamic reconfiguration in Wright consists of the creation and deletion of component and connector instances (using the *new* and *del* constructs), and the creation and deletion of bindings between ports and roles (using the *attach* and *detach* constructs), as shown in Figure 2.5.

The dynamic reconfiguration process transforms a system from one configuration to another, where each configuration meets the safety requirements of the system and can be maintained indefinitely. The process is controlled by the configurator, and involves synchronising all the component and connector instances that are directly affected by the reconfiguration to be at a rendezvous point using control events. When all the participating instances are at the rendezvous, the configuror performs the reconfiguration using a combination of the *new*, *del*, *attach* and *detach* operations, and the instances then continue their execution in the new configuration of the system. The pre-condition of the reconfiguration is that the

Style Client − Server

Component Client
Port p = $\overline{request} \rightarrow reply \rightarrow p \sqcap \S$
Computation = $internalCompute \rightarrow \overline{p.request} \rightarrow p.reply \rightarrow Computation \sqcap \S$

Component Server
Port p = $request \rightarrow \overline{reply} \rightarrow p \,[]\, \S$
Computation = $p.request \rightarrow internalCompute \rightarrow \overline{p.reply} \rightarrow Computation \,[]\, \S$

Connector Link
Role c = $\overline{request} \rightarrow reply \rightarrow c \sqcap \S$
Role s = $request \rightarrow \overline{reply} \rightarrow s \,[]\, \S$
Glue = $c.request \rightarrow \overline{s.request} \rightarrow Glue \,[]\, s.reply \rightarrow \overline{c.reply} \rightarrow Glue \,[]\, \S$

Constraints
$\exists$ a unique *Component s*,
such that $\forall$*Component c* : $TypeServer(s) \wedge TypeClient(c) \Longrightarrow connected(c, s)$

EndStyle

Configuration Simple
Style Client − Server
Instances
C : *Client*; *L* : *Link*; *S* : *Server*
Attachments
C.p as L.c; *S.p as L.s*

Figure 2.4: Specification of a simple client server system in Wright [ADG98].

Style Dynamic − Client − Server

Component Client
Port p = $\overline{request} \rightarrow reply \rightarrow p \sqcap \S$
Computation = $internalCompute \rightarrow \overline{p.request} \rightarrow p.reply \rightarrow Computation \sqcap \S$

Component PrimaryServer
Port p = $\S \,[]\, (request \rightarrow \overline{reply} \rightarrow p \sqcap control.down \rightarrow (\S \,[]\, control.up \rightarrow p))$
Computation = $\S \,[]\, (p.request \rightarrow internalCompute \rightarrow \overline{p.reply} \rightarrow$
$Computation \sqcap control.down \rightarrow (\S \,[]\, control.up \rightarrow Computation))$

Component SecondaryServer
Port p = $\S \,[]\, (control.on \rightarrow \mu Loop.(request \rightarrow \overline{reply} \rightarrow Loop \,[]\, control.off \rightarrow p \,[]\, \S))$
Computation = $\S \,[]\, control.on \rightarrow \mu Loop.$
$(p.request \rightarrow internalCompute \rightarrow \overline{p.reply} \rightarrow Loop \,[]\, control.off \rightarrow Computation [] \S)$

Connector DLink
Role c = $\overline{request} \rightarrow \overline{reply} \rightarrow c \sqcap \S$
Role s = $(request \rightarrow \overline{reply} \rightarrow s \sqcap control.changeOk \rightarrow s) \,[]\, \S$
Glue = $c.request \rightarrow (\overline{s.request} \rightarrow Glue \,[]\, control.changeOk \rightarrow RequestToSend) \,[]$
$s.reply \rightarrow \overline{c.reply} \rightarrow Glue \,[]\, \S \,[]\, control.changeOk \rightarrow Glue$

where RequestToSend = $\overline{s.request} \rightarrow Glue \,[]\, control.changeOk \rightarrow RequestToSend$

Constraints
$\exists$ a unique *Component s*,
such that $\forall Component\ c : TypeServer(s) \wedge TypeClient(c) \Longrightarrow connected(c, s)$

EndStyle

Configuror DynamicClient − Server

Style Dynamic − Client − Server
new.C : Client $\rightarrow$ *new.Primary : PrimaryServer* $\rightarrow$ *new.Secondary : SecondaryServer*
$\rightarrow$ *new.L : Dlink* $\rightarrow$ *attach.C.p.to.L.c* $\rightarrow$ *attach.Primary.p.to.L.s* $\rightarrow$ *WaitForDown*
where
WaitForDown =
(*Primary.control.down* $\rightarrow$ *Secondary.control.on* $\rightarrow$ *L.control.changeOk* $\rightarrow$
Style Dynamic − Client − Server
detach.Primary.p.from.L.s $\rightarrow$ *attach.Secondary.p.to.L.s* $\rightarrow$ *WaitForUp*) [] §

WaitForUp =
(*Primary.control.up* $\rightarrow$ *Secondary.control.off* $\rightarrow$ *L.control.changeOk* $\rightarrow$
Style Dynamic − Client − Server
detach.Secondary.p.from.L.s $\rightarrow$ *attach.Primary.p.to.L.s* $\rightarrow$ *WaitForDown*) [] §

Figure 2.5: Specification of dynamic reconfiguration of a simple client server system in Wright [ADG98].

system configuration is free of any deadlock, and all the component and connector instances are at the rendezvous with no message in transit. The invariant condition is that the instances remain at the rendezvous. The post-condition is that the new system configuration is free of deadlock. Clearly, an instance has to be designed for reconfiguration in order to participate in the reconfiguration process. However, the instance is not required to be quiescent or passive, which reduces the disruption to the system's operation.

The definition of the client server system can be modified to demonstrate the dynamic replacement of a server component due to its failure (see Figure 2.5), taken from [ADG98]. The reconfiguration is transparent to a *Client*, and so its definition is unchanged. The behaviour of a *PrimaryServer* is similar to a *Server* until it fails, when it generates the *control.down* event for the $Configuror$, and then it either terminates or waits for *control.up* (its restart event). The *SecondaryServer* is started by the *control.on* event from the $Configuror$ after the *PrimaryServer* fails, then behaves like a *Server* until it receives the $control.off$ event from the $Configuror$, when it either returns to its initial state or terminates. The behaviour of a *DLink* connector is similar to a *Link* connector until it receives the *control.changeOk* event (indicating reconfiguration of its server), when it attempts to resend the client's request to the new server. Thus, the original connector definition must be modified to handle reconfiguration events that occur during its normal transactions, and so it must buffer messages of incomplete transactions. The $Configuror$ creates the initial configuration of the system, and then controls the reconfiguration process by receiving and issuing the control events, and creating and deleting the bindings between the roles of the *DLink* and the ports of the *PrimaryServer* and *SecondaryServer*.

The main issue with Wright is due to its use of CSP, in which port and role names are not first class objects, resulting in a cumbersome event model: each event is labelled with the configuration in which it occurs, by 'sandwiching' the event between its associated port and role. Thus, $\overline{s.request}$ in *DLink* is expressed as $\overline{L.s.request.Primary.p}$; and reconfiguration implies a relabelling of the event if it is required to have an effect in the new configuration (e.g. $\overline{L.s.request.Secondary.p}$). Hence, the same event must have a distinct label for each configuration in which it is used.

A Wright design can be formally checked for deadlocks using model checking. Furthermore, because the design and the configurator are both modelled using CSP, it is possible to check for deadlocks in the dynamic reconfiguration process, and to analyse interactions between the application actions of a system and the dynamic reconfiguration process [ADG98].

2.10.2.2 Evaluation with respect to Dependability Issues

Wright does not fully address the dependability issues: its model of concurrency consists of a set of sequential processes that execute concurrently. However, processes communicate synchronously on a point-to-point basis with no time bound. Hence, a Wright design is not amenable to schedulability analysis.

2.10.3 Rapide

Rapide is a strongly typed object-oriented DADL for prototyping event-driven real-time distributed systems [LKA+95]. Thus, it resembles a very high-level programming language, although it can be used to design both software and hardware systems. It has no formal semantics, and so simulation is used to confirm the correctness of a system.

Rapide represents the structure of a system (termed `architecture`) as a graph of components linked by connections, and restricted by constraints. Each component has a type (termed `interface`) that defines a set of functions provided by the component to other components (using `provides`), a set of functions required by the component from other components (using `requires`), a set of input events accepted by the component (using `in action`), and a set of output events generated by the component (using `out action`). The interface also defines the behavioural states and state transitions of the component, and the constraints on this behaviour. Connections are defined 'in-line' within the architecture in terms of the interfaces, and connect corresponding required and provided functions, and corresponding output and input events. Components interact synchronously using their required/provided function connections, and asynchronously using their output/input event connections.

The behaviour of a system is represented using event patterns. An event is a uniquely identified tuple, consisting of the component generating the event, the activity associated with the event, and the event's data values, time of occurrence

and duration. An event pattern is an expression that evaluates to a partially ordered set (termed poset) of computationally dependent events (termed causal events) and their causal and timing relationships. The event pattern can be a basic pattern (e.g. a function call), or a composite pattern created using multiple constructs, including iteration (*), guard, conjunction (`and`) and disjunction (`or`). Event patterns are of fundamental importance in Rapide, and are used in different ways: to trigger component and connection behaviour on specific posets of events; to generate posets of events in response to a trigger; and to express constraints on posets of events. Thus, event patterns can be used to define connections that dynamically reconfigure the system, as shown in Figure 2.6.

2.10.3.1 Evaluation with respect to Dynamic Reconfiguration Issues

Dynamic reconfiguration in Rapide consists of the creation and deletion of components and connections, and occurs through the execution of event patterns in connections in response to the firing of triggers on events in the connections. There is no recommended method for performing dynamic reconfiguration. However, the dynamic reconfiguration events must satisfy the constraints defined in the architecture definition.

Figure 2.6 gives an example of a simple system with dynamic binding between client and server components, taken from [LV95]. The system (`Client_Server_Network`) consists of a `Trader` component (`NTT`), an array of `Client` components (`clients`), an array of `Server` components (`servers`), and three sets of connections.

`?S`, `?J`, `?C`, `?N` and `?param` are placeholder variables, which can be bound only to a component evaluated from an event pattern. Rapide also has an iterator variable (named using `!`), which is used as a universal quantifier over a type. The `&` construct indicates a name, which is used in Rapide to control the dereferencing of components. Thus, the architecture binds the `Register` event of a `Server` with the `Register_Server` event of a `Trader`, and the `Find_Server` function of a `Client` with the `Server_Lookup` function of a `Trader` (effectively aliasing them). The third set of connections binds the `Request_Job` function of a `Client` with the `Do_Job` function of a `Server`. However, it is clear from the interface definition that a `Client` cannot dereference a `Server`'s name, and the dereferencing is done by the architecture. This indirection helps to achieve dynamic binding between client and server components.

```
type Server is interface
provides
function Do_Job(J : Job; P : Parameters) return data;

out action Register(J : Job);
...
end;


type Client is interface
requires
function Request_Job(J : Job; P : Parameters; Pn : &Server) return data;

function Find_Server(J : Job) return &Server;
...
end;


type Trader is interface
provides
function Server_Lookup(J : Job) return &Server;

in action Register_Server(J : Job; S : &Server);
...
behaviour
Jobs : array [Job] of &Server;
?J : Job;
?N : &Server;

function Server_Lookup(J : Job) return &Server is
begin
return Jobs[J];
end;

Register_Server(?J, ?N) => Jobs[?J] := ?N;;
end;
```

```
with Client, Server, Trader;
architecture Client_Server_Network is
NTT : Trader;
clients : array [1..NUM_CLIENTS] of Client;
servers : array [1..NUM_SERVERS] of Server;
?S : Server;
?J : Job;
?C : Client;
?N : &Server;
?param : Parameters;

connect
?S.Register(?J) to NTT.Register_Server(?J, &?S);;

?C.Find_Server(?J) to NTT.Server_Lookup(?J);;

?C.Request_Job(?J, ?param, ?N) to *?N.Do_Job(?J, ?param);;

end Client_Server_Network;
```

Figure 2.6: Dynamic binding between clients and servers in Rapide [LV95].

The main limitation of Rapide is that it has no formal semantics. The execution of a system produces a poset of causal events that can be checked for conformance against the system's constraints. However, the poset is not exhaustive, and so the correctness of the system can be only confirmed, rather than proved. Nevertheless, the use of event patterns to express both state transitions of components and dynamic reconfiguration enables interactions between the application actions of the system and its dynamic reconfiguration to be studied.

2.10.3.2 Evaluation with respect to Dependability Issues

Rapide does not fully address the dependability issues: its model of concurrency consists of single-threaded components (expressed using the transition =>) and multi-threaded (expressed using the transition ||>). Components communicate by message passing, which can be either point-to-point or broadcast, and synchronous or asynchronous. However, the absence of a formal semantics implies that a Rapide design is not amenable to formal proof of functional correctness. Rapide does not support schedulability analysis (although it has a notion of deadline [Luc02]), but it could do so if the periods of actions and the bounds on their inter-release times could be represented.

ADL/System	Reconfiguration Items	Reconfiguration Operations	Support for State Tranfer?	Method for Dynamic Reconfiguration?	Pre-Conditions for Dynamic Reconfiguration	Post-Conditions for Dynamic Reconfiguration	Invariant Conditions for Dynamic Reconfiguration	Formal Semantics of Static System	Formal Semantics of Dynamic Reconfiguration Process	Modelling of Interactions between Application and Dynamic Reconfiguration Actions?
Gupta et al.	functions and procedures	function/procedure creation, replacement, deletion	yes	yes	none of the changed functions/procedures is executing, state is well-defined	program can reach a well-defined state	program not executing	no	no	no
Argus	guardians	guardian creation, replacement, deletion, relocation	yes	yes	guardian quiescence	for guardian replacement: the change must be transparent to the clients of the old guardian	guardian quiescence	no	no	no
Eternal	objects	object replacement	yes	yes	old and new versions of an object must be functionally related	replacement must be transparent to the unchanged objects of the system	object quiescence during state transfer and object switchover	no	no	no
DRS	objects	object creation, replacement, deletion, relocation	yes	yes	object must be reconfigurable and quiescent	for object replacement: the change must be transparent to the clients of the old object	object quiescence	no	no	no
Darwin	components and links	component and link creation	no	yes	component must be reconfigurable and quiescent	component quiescence	component quiescence	configuration defined in π-calculus	in π-calculus	no
Wright	components, connectors and links	component, connector and link creation, deletion	no	yes	system configuration must be free of any deadlock, component/connector must be reconfigurable and at a rendezvous	system configuration must be free of deadlock	component/connector must remain at the rendezvous	in CSP	in CSP	yes
Rapide	components and connections	component and connection creation, deletion	no	no	defined by event pattern of a connection's trigger	defined by event pattern of a connection's body	undefined	no	no	yes

Table 2.1: Summary of Dynamic Reconfiguration Features of Architectures and Systems.

2.10.4 Related Work

There are ADLs (in addition those discussed above) that address the problem of managing dynamic reconfiguration of large systems, such as Olan [BABR96] [BBRVD98] and Weaves [GR91]. However, their principles, methods and constructs for managing dynamic reconfiguration are not significantly different from those that have been discussed already. Furthermore, none of them has a formal semantics. In addition, there are web services composition languages (WSCLs), such as WS-BPEL [AAA+07] and WS-CDL [KBR+05], which are ADLs designed for the composition of services provided by the world wide web, which is a naturally dynamic computing system. However, their support for dynamic reconfiguration is limited to error handling. The formal semantics of the error handling mechanisms of WS-BPEL 2.0 has been defined using the asynchronous π-calculus $\texttt{web}\pi_\infty$[LM07].

2.10.5 Discussion

It is clear from the review that no DADL fully addresses the issues of managing dynamic reconfiguration of dependable systems. Considering the most relevant DADLs, each has a different mix of deficiencies (see Table 2.1): Darwin does not handle component replacement and deletion, or the deletion of bindings; and because affected actions are suspended, it does not analyse the interaction between application actions and dynamic reconfiguration actions of a system. However, Darwin uses a π-calculus to define the formal semantics of its dynamic reconfiguration process very simply. In contrast, Wright can fully describe dynamic reconfiguration and the interaction between application actions and dynamic reconfiguration actions; but its use of CSP leads to a cumbersome formal semantics. Rapide can fully describe dynamic reconfiguration and the interaction between application actions and dynamic reconfiguration actions; but it lacks a formal semantics, and (therefore) it can only confirm the correctness of a system, rather than prove the correctness. None of the DADLs fully addresses the dependability issues, especially with respect to schedulability analysis. Finally, none of the DADLs addresses the issue of how to manage unplanned dynamic reconfiguration, which is important for managing the evolution of dependable systems [MMR10].

2.11 Requirements on a Formalism

The strengths and weaknesses of the DADLs and systems reviewed above (summarized in Table 2.1) help to determine requirements on a formalism for the modelling

and analysis of dynamic reconfiguration of dependable systems. A preliminary version of these requirements is given in [MB10]. A formalism should meet the following requirements.

2.11.1 Dynamic Reconfiguration Requirements

1. It should be possible to model, and to identify instances of, software components and tasks, and their communication links.
2. It should be possible to model the creation, deletion and replacement of software components and tasks, and the creation and deletion of their communication links.
3. It should be possible to model the relocation of software components and tasks on physical nodes.
4. It should be possible to model state transfer between software components and between tasks.
5. It should be possible to model both planned and unplanned reconfiguration.
6. It should be possible to model the functional interactions between application tasks and reconfiguration tasks.
7. It should be possible to model the temporal interactions between application tasks and reconfiguration tasks.
8. It should be possible to express and to verify the functional correctness requirements of application tasks and reconfiguration tasks.
9. It should be possible to express and to verify the temporal correctness requirements of application tasks and reconfiguration tasks.

2.11.2 Dependability Requirements

1. It should be possible to model the concurrent execution of tasks.
2. It should be possible to model state transitions of software components and tasks.

2.11.3 General Requirements

1. The formalism should be as terse as possible, in order to facilitate its use.
2. The formalism should be supported by tools; otherwise, it will not be used.

Chapter 3

Review of Formalisms Supporting Dynamic Reconfiguration

Contents

The material reviewed in Chapter 2 shows that DADLs are the most promising approach to managing dynamic reconfiguration of dependable systems. However, the material also shows that no existing DADL is entirely suitable for the purpose. The most serious defect is the lack of a simple but powerful formalism that can model and analyze application actions and dynamic reconfiguration actions (including real-time actions) and their functional and temporal interactions for both planned and unplanned reconfiguration.

In developing a formalism suitable for DADLs for dependable systems, following the traditional approach involves representing application actions and reconfiguration actions by different formalisms (see [KK88] and [KGC89]). This is understandable, because the two sets of actions are logically different. However, the use of different formalisms creates the overhead of translation between the formalisms in order to study interactions between the two sets of actions. Therefore, we avoid the overhead by focusing on a single formalism to model both sets of actions.

In developing a single formalism suitable for DADLs for dependable systems, notice that it is more difficult to express dynamic reconfiguration features than to express real-time features necessary for the modelling and analysis of temporal interactions between application and reconfiguration tasks. Therefore, it is likely to be easier (and hence more productive) to modify a formalism that can represent dynamic reconfiguration with real-time constructs (for example, duration in order to support schedulability analysis), than to modify a real-time formalism with constructs for expressing dynamic reconfiguration, or to invent a completely new formalism 'from scratch'. Hence, it is advisable to follow the first approach.

Model-based formalisms, such as Z [Spi89] and VDM [Jon80], can express configurations [RS94] and the difference between configurations. However, the process of dynamic reconfiguration cannot be described easily, which makes it difficult to describe interactions between application actions and reconfiguration actions. Process algebras are more promising in this respect because they can represent processes and functional interaction between processes, and their description of a computational model in terms of actions can be analysed for schedulability if the actions contain sufficient timing information. However, standard process algebras, such as *CCS* [Mil89] and *CSP* [Hoa85], do not treat components and their bindings as first class objects, which leads to a cumbersome notation for expressing dynamic reconfiguration [ADG98]. Mobile process algebras, such as π-calculi, are interesting because of their treatment of component bindings as first class objects, which enables dynamic reconfiguration of communication links to be expressed simply. This suggests that a timed π-calculus may be a suitable formalism for modelling application actions and reconfiguration actions, and their interactions.

Other candidate formalisms include graph grammars, such as Garp [KK88] and the Δ-Grammar [KGC89], and the Chemical Abstract Machine [BB92]. These graph grammars represent the configuration of a system as a directed graph, with a node representing an agent and an arc representing a communication path. Nodes interact asynchronously by message passing through ports. System reconfiguration is expressed as graph rewrites by agents (termed Δ transitions), in which a node is replaced by a subgraph. Thus, the graph grammars are similar to a process algebra. However, they specify the effects of dynamic reconfiguration rather than model the reconfiguration process.

The Chemical Abstract Machine (CHAM) is based on the GAMMA formalism defined in [BM90]. GAMMA models a data value as a molecule, the system's state as a solution (i.e. a finite multiset) of molecules, and a computation as a sequence of reactions between molecules, defined by transformation rules between solutions and guarded by reaction conditions. Different reactions can run in parallel if their source multisets are disjoint; otherwise a non-deterministic choice is made as to which reaction will occur. GAMMA uses multisets in order to avoid unnecessary ordering restrictions in the specification of an algorithm caused by the use of list-based data structures. CHAM extends GAMMA by allowing the user to define the syntax of a molecule; a membrane construct is used to encapsulate a solution, so that it behaves like a single molecule, thereby enabling a large system to be structured as a hierarchy of solutions; and an airlock construct is used to control reactions between a given solution and its environment. System reconfiguration is expressed as rewrites of multisets of molecules [Met96]. CHAM has been used to specify software architectures [IW95], and to specify the dynamic reconfiguration of software architectures [Wer99]. However, like the graph grammars discussed above, CHAM does not model the reconfiguration process. Furthermore, the concepts underlying the CHAM constructs are different from those normally used by architects to design systems, so that ensuring a CHAM description is an abstraction of an architect's description becomes an issue. In contrast, the 'conceptual gap' between the architect's description and a process algebraic description is much less.

Therefore, we focus on π-calculi, and examine a selection of these using the requirements identified in Section 2.11 in order to evaluate their suitability for the modelling and analysis of dynamic reconfiguration of dependable systems.

3.1 Milner's, Parrow's and Walker's π-calculus

This is the original π-calculus, developed by Milner, Parrow and Walker as an extension of *CCS*, in which port names are treated as first class objects [MPW92]. It is based on *ECCS* [EN86], which extended behavioural equivalence in *CCS* to processes communicating port names.

In this π-calculus, a system is modelled as a collection of processes that communicate synchronously by message passing through ports on a point-to-point basis. The syntax of a process P is defined as follows [Mil99]:

$$P ::= \sum_{i \in I} \pi_i.P_i \mid P_1|P_2 \mid \nu a P \mid \; !P \mid 0 \tag{3.1}$$

$$\text{where } \pi_i ::= x_i(\vec{y}_i) \mid \overline{x}_i < \vec{y}_i > \mid \tau \;\text{ and } I \text{ is a finite set.}$$

Thus, π_i is an action prefix[1], which can be an input action (x_i) that receives a vector of values on port x_i and uses the vector to substitute the vector of names $\vec{y}_i$; or an output action ($\overline{x}_i$) that sends a vector of values $\vec{y}_i$ on port $\overline{x}_i$; or an unobservable action (τ) internal to the process. The identification of a communication action with its associated port (inherited from *CCS*) gives the formalism great simplicity. The summation enables P to behave non-deterministically, in a manner selected by its environment (using an input/output action) or by P itself (using τ). The composition operator (|) enables P to be decomposed into, and composed from, parallel processes (P_1 and P_2). The restriction operator (ν) restricts the scope of a port name (a) to P. The replication operator (!) produces an infinite replication and composition of its operand (i.e. $P|P|P|...$). The original syntax of a process did not contain the replication operator [MPW92], but it is necessary in order to make the formalism Turing complete. 0 is the *NIL* process, which performs no action.

Notice that this is the process syntax in the polyadic version of the π-calculus, where a message consists of a vector of names. The process syntax in the monadic version is similar, except that a message consists of a single name (e.g. y). Clearly, the monadic version is a simple case of the polyadic version; but it can be used to encode any expression in the polyadic version. We use the polyadic version for completeness with respect to definitions given below.

The operational semantics of this π-calculus are defined by the following seven transition rules, and alpha-conversion[2] [Mil99]:

$$SUM_C \;:\; \frac{\alpha A \xrightarrow{\alpha} A}{M + \alpha A + N \xrightarrow{\alpha} A} \quad \textit{where } \alpha ::= x(\vec{y}) \mid \overline{x} < \vec{y} > \mid \tau$$

[1]Some researchers include port name matching (i.e. [x=y]π_i) in the basic definition of π_i [SW01]; and other researchers treat the matching as an extension [Ber04]. We follow the latter approach, in order to keep the process syntax simple.

[2]Alpha-conversion is the renaming of one or more bound names of a process. It is usually done in order to avoid a name conflict in a substitution. A bound name of a process is a name whose scope is restricted to the process. In contrast, a free name of a process is a name in the process whose scope is not restricted to the process.

$$L\text{–}REACT_C \ : \ \frac{P \xrightarrow{x} (\vec{y}).P' \qquad Q \xrightarrow{\bar{x}} \nu\vec{z}{<}\vec{w}{>}.Q'}{P|Q \xrightarrow{\tau} \nu\vec{z}(\{\vec{w}/\vec{y}\}P'|Q')} \quad \textit{assuming } \vec{z} \textit{ not free in } (\vec{y}).P' \textit{ and } |\vec{w}| = |\vec{y}|$$

$$R\text{–}REACT_C \ : \ \frac{P \xrightarrow{\bar{x}} \nu\vec{z}{<}\vec{w}{>}.P' \qquad Q \xrightarrow{x} (\vec{y}).Q'}{P|Q \xrightarrow{\tau} \nu\vec{z}(\{\vec{w}/\vec{y}\}Q'|P')} \quad \textit{assuming } \vec{z} \textit{ not free in } (\vec{y}).Q' \textit{ and } |\vec{w}| = |\vec{y}|$$

$$L{-}PAR_C \ : \ \frac{P \xrightarrow{\alpha} A}{P|Q \xrightarrow{\alpha} A|Q} \qquad\qquad R{-}PAR_C \ : \ \frac{Q \xrightarrow{\alpha} A}{P|Q \xrightarrow{\alpha} P|A}$$

$$RES_C \ : \ \frac{P \xrightarrow{\alpha} A}{\nu xP \xrightarrow{\alpha} \nu xA} \ if \alpha \notin \{x, \bar{x}\}$$

$$REP_C \ : \ \frac{P|!P \xrightarrow{\alpha} A}{!P \xrightarrow{\alpha} A}$$

SUM_C states that summation preserves sequential transitions. The $REACT_C$ rules define reactions between two processes that contain an *abstraction* and a *concretion* respectively. An *abstraction* is an expression (e.g. $(\vec{y}).P'$) that enables the binding of a vector of port names ($\vec{y}$) to a process (P') to be expressed in a uniform manner. A *concretion* is an expression (e.g. $\nu\vec{z}{<}\vec{w}{>}.Q'$) that passes a vector of values ($\vec{w}$) to an abstraction along with an extension in scope of a vector of restricted port names ($\nu\vec{z}$). Thus, a concretion is a dual of an abstraction. The $REACT_C$ rules state that if a process containing a concretion ($\nu\vec{z} < \vec{w} > .Q'$) and a process containing an abstraction ($(\vec{y}).P'$) are composed, then the values of the concretion are passed to the abstraction ($\{\vec{w}/\vec{y}\}P'$) and the restriction on port names in the concretion is extended over the result ($\nu\vec{z}(\{\vec{w}/\vec{y}\}P'|Q')$), provided the restricted names are not free in the abstraction and the message vectors ($\vec{w}$ and $\vec{y}$) have the same length. The PAR_C rules state that composition preserves the transitions of its constituent processes. The RES_C rule states that restriction preserves a transition of its process if the transition name is not restricted. The REP_C rule implies that $P|!P$ and $!P$ have identical transitions.

Notice that (unlike basic *CCS*) this π-calculus is asymmetric, since receiving and sending processes are treated differently. For example, a receiving process P is equal to $x.F$ where $F = (\vec{y}).P'$ (an abstraction); and its sending process Q is equal to $\bar{x}.C$ where $C = \nu\vec{z}{<}\vec{w}{>}.Q'$ (a concretion). The composition $P|Q$ results in P being replaced with the residue $(\vec{y}).P'$, and Q being replaced with the residue $\nu\vec{z}{<}\vec{w}{>}.Q'$,

and we have the asymmetric transition $P|Q \xrightarrow{\tau} \nu\vec{z}(\{\vec{w}/\vec{y}\}P'|Q')$, assuming $\vec{z}$ not free in $(\vec{y}).P'$ and $|\vec{w}| = |\vec{y}|$. Furthermore, because the calculus is synchronous, the sending process waits until the receiving process is ready to receive its message. Hence, the end time of the sending action depends on the start time of the receiving action.

Abstractions and concretions are termed *agents*, and the length of the message vector is termed the *arity* of the agent. A process is considered to be an agent of arity 0, and it is considered to be both an abstraction and a concretion.

3.1.1 Evaluation using Requirements

Dynamic reconfiguration in this π-calculus consists of the creation of processes (achieved using !), and the creation and deletion of bindings between processes (achieved by passing port names in communications between processes):

$!P$ creates an infinity of processes composed together (i.e. $P|P|P|...$).

If $P \stackrel{def}{=} \overline{x}<y>.P'$ and $Q \stackrel{def}{=} x(u).u(v).Q'$ and $R \stackrel{def}{=} \overline{y}<w>.R'$, then in $P|Q|R$ P passes the port name y to Q (thereby substituting u by y) so that Q can communicate with R. The binding between Q and R can be deleted by a subsequent substitution of u.

The physical relocation of a process can be modelled as a change in its bindings to other processes.

State transfer between processes can be modelled as communication.

It is not possible to delete a process in an unplanned manner, although a process can terminate at the end of its execution (e.g. $x(a).y(b).0$). Hence, it is not possible to model the unplanned replacement of a process. Furthermore, it is not possible to identify a specific instance of a process for reconfiguration.

Functional interaction between application actions and reconfiguration actions is modelled as interleavings of actions. Temporal interaction between actions cannot be modelled, because the duration of an action cannot be expressed.

Functional correctness is expressed in terms of equivalence of process expressions, and is verified by equational reasoning. Thus, if a process P is correct and a process

expression $C[P]$ that contains P is correct, and if P is substituted by a process Q, and Q is equivalent to P (in some sense), then the substitution will preserve the correctness of the process expression $C[Q]$. The derivation of equivalence is as follows [Mil99].

Let $\mathcal{P}^\pi$ be the set of all π-calculus processes, and let $\mathcal{S}$ be a binary relation over $\mathcal{P}^\pi$. $\mathcal{S}$ is termed a *strong simulation* if whenever $P\mathcal{S}Q$,

$$if\ P \xrightarrow{\alpha} A\ then\ \exists B\ such\ that\ Q \xrightarrow{\alpha} B\ and\ A\mathcal{S}B$$

Thus, any behaviour of P can be simulated by Q, but not necessarily vice versa. If both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are strong simulations, then $\mathcal{S}$ is termed a *strong bisimulation*.

If $\mathcal{S}$ is a strong bisimulation such that $P\mathcal{S}Q$, then P and Q are termed *strongly equivalent* (written $P \sim Q$). Thus, any behaviour of P can be simulated by Q, and vice versa. Notice that if A and B are abstractions (e.g. $A = (\vec{x}).A'$ and $B = (\vec{x}).B'$), then for $A \sim B$ we must have $\forall \vec{y}\,(\{\vec{y}/\vec{x}\}A' \sim \{\vec{y}/\vec{x}\}B')$.

Strong equivalence between processes is important for process substitution in the context of a process expression (termed a *process context*), which has the following syntax:

$$C ::= [\,] \mid \pi.C + M \mid \nu a C \mid C|P \mid P|C \mid\ !C \tag{3.2}$$

Thus, a process context is a process expression with a hole (i.e. $[\,]$) that can be filled by a process (i.e. $C[Q]$). The elementary contexts are $\pi.[\,] + M$, $\nu a[\,]$, $[\,]|P$, $P|[\,]$ and $![\,]$; and $[\,]$ is the identity context, since $[Q] = Q$.

Clearly, filling the hole of a process context with different processes can produce different results. Therefore, a notion of *process congruence* needs to be defined between processes that produces equivalent results when congruent processes are substituted for each other in a process context.

An equivalence relation $\cong$ over $\mathcal{P}^\pi$ is termed a *process congruence* if it is preserved by all elementary process contexts. Thus, if $P \cong Q$ then the following must hold:

$$\pi.P + M \ \cong\ \pi.Q + M \quad where\ \pi ::= x(\vec{y}) \mid \overline{x}\langle\vec{y}\rangle \ \mid\ \tau$$

$$\nu\vec{z}P \ \cong\ \nu\vec{z}Q$$

$$P|R \ \cong\ Q|R$$

$$R|P \cong R|Q$$

$$!P \cong !Q$$

The conditions on agents for producing equivalent results in a process context (termed *agent congruence*) are slightly stronger than those for process congruence. *Agent congruence* is a process congruence with the following two conditions:

$$\nu\vec{x}<\vec{y}>.P \cong \nu\vec{x}<\vec{y}>.Q \quad for\ concretions$$

$$\forall\vec{y}(\{\vec{y}/\vec{x}\}P \cong \{\vec{y}/\vec{x}\}Q) \quad for\ abstractions$$

It can be proved that an equivalence relation is preserved by all elementary process contexts if and only if it is preserved by all process contexts. Furthermore, strong equivalence is a process congruence. Hence, strongly equivalent processes can be substituted in any given process context with equivalent results. Strong equivalence is also an agent congruence. Hence, strongly equivalent agents can be substituted in any given process context with equivalent results.

A stronger notion of congruence between processes is *structural congruence*, in which congruent processes can be transformed into one another. These processes are structurally similar, rather than being merely behaviourally similar:

Processes $P, Q \in \mathcal{P}^{\pi}$ are termed *structurally congruent* (written $P \equiv Q$) if and only if they can be transformed into one another using the following rules:

$$Alpha-conversion$$

$$Reordering\ of\ terms\ in\ a\ summation$$

$$P|0 \equiv P,\ P|Q \equiv Q|P,\ P|(Q|R) \equiv (P|Q)|R$$

$$\nu x(P|Q) \equiv P|\nu xQ \quad if\ x\ not\ free\ in\ P$$

$$\nu x0 \equiv 0,\ \nu xyP \equiv \nu yxP$$

$$!P \equiv P|!P$$

Structural congruence leads to a new transition rule:

$$STRUCT\ :\ \frac{Q \equiv P \quad P \rightarrow P' \quad P' \equiv Q'}{Q \rightarrow Q'}$$

Regarding the use of verification techniques other than equational reasoning, the application actions of a system and its dynamic reconfiguration actions are both modelled using labelled transitions. Hence, it is possible to model their

interleavings, and to analyse the effects using standard techniques (such as reachability analysis).

There is a weaker equivalence between processes than strong equivalence, termed *weak equivalence*, which ignores the internal behaviour of a process. Therefore, long process expressions can be weakly equivalent to much shorter process expressions, which helps to model and analyze large systems. The derivation of weak equivalence is as follows [Mil99].

Let $\mathcal{S}$ be a binary relation over $\mathcal{P}^{\pi}$. $\mathcal{S}$ is termed a *weak simulation* if whenever $P\mathcal{S}Q$,

$$if\ P \Rightarrow P'\ then\ \exists Q'\ such\ that\ Q \Rightarrow Q'\ and\ P'\mathcal{S}Q'$$

$$if\ P \overset{x<\vec{y}>}{\Rightarrow} P'\ then\ \exists Q'\ such\ that\ Q \overset{x<\vec{y}>}{\Rightarrow} Q'\ and\ P'\mathcal{S}Q'$$

$$if\ P \overset{\bar{x}}{\Rightarrow} C\ then\ \exists D\ such\ that\ Q \overset{\bar{x}}{\Rightarrow} D\ and\ C\mathcal{S}D$$

where $\overset{\alpha}{\Rightarrow}$ is an α transition preceded and/or followed by 0 or more τ transitions,
and $\alpha ::= x(\vec{y}) \mid \bar{x}<\vec{y}> \mid \tau$,
and C, D are concretions.

Thus, any behaviour of P can be weakly simulated by Q, but not necessarily vice versa. P can be thought of as a specification of Q. If both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are weak simulations, then $\mathcal{S}$ is termed a *weak bisimulation*.

If $\mathcal{S}$ is a weak bisimulation such that $P\mathcal{S}Q$, then P and Q are termed *weakly equivalent* (written $P \approx Q$). Thus, any behaviour of P can be weakly simulated by Q, and vice versa; so that P and Q have the same specification. As with strong equivalence, if A and B are abstractions (e.g. $A = (\vec{x}).A'$ and $B = (\vec{x}).B'$), then for $A \approx B$ we must have $\forall \vec{y}\,(\{\vec{y}/\vec{x}\}A' \approx \{\vec{y}/\vec{x}\}B')$.

Weak equivalence is an agent congruence. Hence, weakly equivalent agents can be substituted in any given process context with equivalent results.

In this π-calculus, the model of concurrency is that of a multiset of sequential processes that execute concurrently, perform state transitions, and communicate synchronously on a point-to-point basis with no time bound. Hence, an expression in this formalism is not amenable to schedulability analysis.

Support for functional correctness is provided through proofs of equivalence and congruence; and the operational semantics support reachability analysis. Model checking using temporal logic is problematic, due to the dynamic creation of processes that can be involved in a proposition [CKCB01]. A suitable timed temporal logic is required.

There is a type system based on sorts. A set of processes is associated with a partial function (termed a sorting) which maps the sort of an action (σ) to the list of sorts ($ob(\sigma)$) corresponding to the action's message vector:

$$ob : \Sigma \rightharpoonup \Sigma^* \quad where\ \Sigma\ is\ a\ set\ of\ sorts. \tag{3.3}$$

For type correctness, each subterm of the form $x(\vec{y}).P$ or $\overline{x} < \vec{y} > .P$ must satisfy the condition:
$if\ x : \sigma\ then\ \vec{y} : ob(\sigma)$; and x and $\overline{x}$ must have the same sort (σ). A sorting is a partial function because not all the elements of a message vector may be used as actions in a given set of processes. An action with no message vector is mapped to the empty sort list (ε).

Support for temporal correctness is highly problematic in this π-calculus. First, the ! operator produces an infinite composition of processes, which causes problems in modelling systems with finite resources. Second, the synchronous communication model results in a timing dependency between the sending and receiving processes, which complicates the schedulability analysis. Third, the process expressions are not amenable to schedulability analysis.

Inspite of the limitations of this π-calculus, it is a simple and powerful formalism, and (consequently) considerable research on process mobility is based on it. Tool support includes the Mobility Workbench [VM94], which checks for open bisimilarity between processes and for deadlocks; and TyPiCal [Kob06], which is a type-based static analyzer for checking deadlock freedom and termination.

3.2 Higher-Order π-calculi

Higher-order π-calculi are distinguished from first-order π-calculi (such as the original π-calculus) by their treatment of processes as first class objects in communications (rather than only port names).

In higher-order π-calculi, a system is modelled as a collection of processes that communicate by message passing through ports, where the messages can contain port names or processes [Tho90] [San93]. The syntax of a process P is defined as follows:

$$P ::= \sum_{i \in I} \pi_i.P_i \mid P_1|P_2 \mid \nu a P \mid \ !x(\vec{y}).P \mid X \mid 0 \tag{3.4}$$

$$\text{where } \pi_i ::= x_i(\vec{y}_i) \mid \overline{x}_i<\vec{y}_i> \ \mid \tau \ \text{ and } I \text{ is a finite set.}$$

Thus, the action prefix π_i can be an input action (x_i) that receives a vector of port names or processes, and uses the vector to substitute the vector of names $\vec{y}_i$; or an output action ($\overline{x}_i$) that sends a vector $\vec{y}_i$ of port names or processes; or an unobservable action (τ) internal to P. The process syntax is similar to that of the original π-calculus, with two exceptions. First, the lazy replication operator (!) uses a guard ($x(\vec{y})$) to control the replication of P. It can simulate the unguarded ! of the original π-calculus, but has better computational properties. Second, X indicates the execution of a received process.

The operational semantics of synchronous higher-order π-calculi consist of four rules of the original π-calculus (i.e. SUM_C, $L-PAR_C$, $R-PAR_C$ and RES_C) and the following three transition rules:

$$L{-}REACT \ : \ \frac{P \xrightarrow{x(\vec{y})} P' \qquad Q \xrightarrow{\overline{x}<\vec{w}>} Q'}{P|Q \xrightarrow{\tau} \{\vec{w}/\vec{y}\}P'|Q'} \quad assuming \ |\vec{w}| = |\vec{y}|$$

$$R{-}REACT \ : \ \frac{P \xrightarrow{\overline{x}<\vec{w}>} P' \qquad Q \xrightarrow{x(\vec{y})} Q'}{P|Q \xrightarrow{\tau} P'|\{\vec{w}/\vec{y}\}Q'} \quad assuming \ |\vec{w}| = |\vec{y}|$$

$$LAZY{-}REP \ : \ \overline{x}<\vec{P}>|!x(\vec{y}).Q \xrightarrow{\tau} \{\vec{P}/\vec{y}\}Q|!x(\vec{y}).Q$$

The $REACT$ rules determine the result of the interaction between two complementary actions. The result is asymmetric, since (in $L{-}REACT$) a receiving process P continues to $\{\vec{w}/\vec{y}\}P'$, whereas the sending process Q continues to Q'. Notice that $\vec{w}$ can be a vector of port names or processes. Obviously, $\vec{w}$ and $\vec{y}$ must have identical lengths and sorts for the two processes to interact. Notice also that the scope of any restriction on $<\vec{w}>$ is not extended over the result (unlike the original π-calculus). The $LAZY{-}REP$ rule states that a process (Q) can be replicated only through an interaction with its guard ($x(\vec{y})$), which enables a process to be created only when it is needed.

3.2.1 Evaluation using Requirements

As in the original π-calculus, processes can be created (achieved using lazy replication rather than unguarded replication), and bindings between processes can be created and deleted by passing port names in communications between processes.

The physical relocation of a process can be modelled by passing the process in communications between processes and changing its bindings to other processes.

State transfer between processes can be modelled as communication.

Unplanned process deletion and unplanned process replacement cannot be modelled; and it is not possible to identify a specific instance of a process for reconfiguration.

As in the original π-calculus, functional interaction between application and reconfiguration actions is modelled as action interleavings, and can be analyzed using standard techniques (such as reachability analysis). Temporal interaction cannot be modelled. Functional correctness is based on equivalence of process expressions and substitution of processes, and is verified by equational reasoning. However, the definition of equivalence in higher-order π-calculi is different from that in the original π-calculus, because of process passing [Tho90].

Let $\mathcal{P}$ be the set of all concurrent processes, and let $\mathcal{S}$ be a binary relation over $\mathcal{P}$. $\mathcal{S}$ is termed a *strong higher-order simulation* if whenever $P\mathcal{S}Q$,

$$if\ P \xrightarrow{\alpha} P'\ then\ \exists Q', \beta\ such\ that\ Q \xrightarrow{\beta} Q'\ and\ P'\mathcal{S}Q'\ and\ \alpha\widehat{\mathcal{S}}\beta$$

$$where\ \widehat{\mathcal{S}} = \{(\alpha,\beta) \mid (\alpha = x(P'') \ \wedge\ \beta = x(Q'') \ \wedge\ P''\mathcal{S}Q'') \vee$$
$$(\alpha = \overline{x}<P''> \ \wedge \beta = \overline{x}<Q''> \ \wedge P''\mathcal{S}Q'') \vee$$
$$(\alpha = \tau = \beta)\}$$

Thus, any behaviour of P can be simulated by Q, provided the process passed in the transition of Q simulates the process passed in the transition of P. If both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are strong higher-order simulations, and $\widehat{\mathcal{S}}$ is common to both $\mathcal{S}$ and $\mathcal{S}^{-1}$, then $\mathcal{S}$ is termed a *strong higher-order bisimulation*.

If $\mathcal{S}$ is a strong higher-order bisimulation such that $P\mathcal{S}Q$, then P and Q are termed *strongly higher-order equivalent* (written $P \sim Q$). Thus, any behaviour of P can be simulated by Q, and vice versa.

Strong higher-order equivalence is a process congruence [Tho90]. Hence, strongly higher-order equivalent processes can be substituted in any given process context with equivalent results.

The issue of whether higher-order π-calculi are more expressive than first-order π-calculi was addressed in [San93]. Sangiorgi developed a higher-order π-calculus (HOπ) by extending the sorting scheme of the original π-calculus to allow processes and parameterised processes of arbitrarily high order to be passed in communications (i.e. an ω-order extension of the original π-calculus). A notion of equivalence (based on *barbed bisimulation*) was defined that applies uniformly over different calculi, so that equivalence between process expressions can be preserved by their encodings in different calculi. The derivation of barbed bisimulation is as follows.

Let $\mathcal{P}$ be the set of all concurrent processes, and let $\mathcal{S}$ be a binary relation over $\mathcal{P}$. $\mathcal{S}$ is termed a *barbed simulation* if whenever $P\mathcal{S}Q$,

$$if\ P \rightarrow P'\ then\ Q \rightarrow Q'\ and\ P'\mathcal{S}Q' \quad where\ \rightarrow\ is\ a\ transition.$$

$$\forall a\ if\ P \downarrow_a\ then\ Q \downarrow_a \quad where\ a\ is\ a\ port$$

The first condition states that any transition of P can be simulated by Q. $\downarrow_a$ is an observation predicate on port a which detects the possibility of communication with the environment through a. Thus, the second condition states that if P can communicate with its environment through its port a, then Q can communicate with its environment through its port a. If both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are barbed simulations, then $\mathcal{S}$ is termed a *barbed bisimulation*.

If $\mathcal{S}$ is a barbed bisimulation such that $P\mathcal{S}Q$, then P and Q are termed *barbed-bisimilar* (written $P \dot{\sim} Q$). Thus, any transition of P can be simulated by Q, and vice versa; and the resulting states have the same observation set.

The definition of *weak barbed bisimulation* (written $\dot{\approx}$) is similar to the definition of barbed bisimulation, with the transition $Q \rightarrow Q'$ replaced by $Q \Rightarrow Q'$ and the predicate $Q \downarrow_a$ replaced by $Q \Downarrow_a$; where $\Rightarrow$ is the reflexive and transitive closure of

$\rightarrow$, and $Q \Downarrow_a$ means $Q \Rightarrow Q'' \downarrow_a$ for some Q''.

Weak barbed bisimulation is used to define *weak barbed congruence*: two processes P, Q are termed *weakly barbed-congruent* (written $P \approx^c Q$) if $C[P] \dot{\approx} C[Q]$ for all contexts $C[\ ]$. The importance of $\approx^c$ is that it is preserved by encodings in different calculi in a fully abstract way. That is, two source language terms are weakly barbed-congruent if and only if their translations are weakly barbed-congruent. Thus, Sangiorgi proved the semantic equivalence between the original π-calculus and HOπ [San93] [SW01].

Higher-order π-calculi are useful in expressing process communication in an abstract manner, especially if process passing is involved. However, there are a number of reasons for modelling processes in a first-order π-calculus [SW01]. First, name passing is more widely used in computing systems in comparison to process passing. Second, first-order π-calculi can express partial access to a process, and can enable process sharing; whereas in higher-order π-calculi a receiving process can only execute a received process or communicate it. Third, the theory of first-order π-calculi is simpler and more tractable than the theory of higher-order π-calculi.

The concurrency model is that of a multiset of concurrently executing sequential processes that perform state transitions and communicate with no time bound. Hence, process expressions are not amenable to schedulability analysis.

Functional correctness is supported through proofs of equivalence and congruence, and the operational semantics support reachability analysis (as in the original π-calculus).

The type system is based on sorts, and varies over higher-order π-calculi. In CHOCS, the sort of a process is defined as follows [Tho90]:

$$\begin{aligned} &p \text{ is a process with sort } L \text{ (written } p :: L) \text{ if and only if} \\ &\forall p' \textit{ reachable from } p \textit{ through } 0 \textit{ or more transitions } (p' \xrightarrow{a(\vec{y}) \mid \vec{a}<\vec{y}>} p'' \implies a \in L) \end{aligned} \tag{3.5}$$

Temporal correctness is supported through guarded replication, which avoids the instantaneous creation of an infinite composition of processes in a system with finite resources. However, the process expressions are not amenable to schedulability

analysis, because the communication delays are undefined.

3.3 Asynchronous π-calculus

The asynchronous π-calculus ($A\pi$) is a subcalculus of the original π-calculus, defined in [HT91] and [Bou92]. It was intended to be a minimal calculus for expressing mobility.

In $A\pi$, a system is modelled as a collection of processes that communicate asynchronously by message passing through ports, where the messages can contain port names. The syntax of a process P is defined as follows:

$$P ::= x(\vec{y}).P \mid \overline{x}<\vec{y}>.0 \mid P_1|P_2 \mid \nu aP \mid\ !x(\vec{y}).P \mid 0 \tag{3.6}$$

In computing systems, asynchronous communication can be thought of as communication that does not involve waiting by the communicating processes. Thus, a sending process does not wait for a receiving process to be ready to receive before sending its message over the communication medium; and a receiving process does not wait for a sending process to be ready to send – it either receives its message from the medium (if this has arrived) or executes another action. In $A\pi$, sending processes have non-waiting behaviour, and output messages are unordered; and so output actions ($\overline{x} < \vec{y} >$) have no continuation. Furthermore, an unguarded output action is considered 'already executed', rather than 'to be executed'. Receiving processes have waiting behaviour, and input messages are ordered; and so an action prefix is an input action ($x(\vec{y})$). The remainder of the syntax has been described previously.

The operational semantics of $A\pi$ are essentially defined by the following eight asynchronous transition rules [Ber04]:

Let $l ::= x(\vec{v}) \mid \overline{x}<(\nu\vec{z})\vec{y}> \mid \tau$

Let $fn(l)$ and $fn(Q)$ be the sets of free names of l and Q respectively, and let $bn(l)$ be the set of bound names of l.

$$OUT\ :\ \frac{}{\overline{x}<\vec{y}>.0 \xrightarrow{\overline{x}<\vec{y}>}_a 0} \qquad\qquad IN_a\ :\ \frac{}{0 \xrightarrow{x(\vec{y})}_a \overline{x}<\vec{y}>.0}$$

$$PAR \ : \ \frac{P \xrightarrow{l}_a P' \qquad bn(l) \cap fn(Q) = \emptyset}{P|Q \xrightarrow{l}_a P'|Q}$$

$$COM \ : \ \frac{}{\overline{x}<\vec{y}>.0|x(\vec{v}).Q \xrightarrow{\tau}_a \{\vec{y}/\vec{v}\}Q}$$

$$LAZY-REP \ : \ \frac{}{\overline{x}<\vec{y}>.0|!x(\vec{v}).Q \xrightarrow{\tau}_a \{\vec{y}/\vec{v}\}Q|!x(\vec{v}).Q}$$

$$RES \ : \ \frac{P \xrightarrow{l}_a Q \qquad x \notin fn(l) \cup bn(l)}{(\nu x)P \xrightarrow{l}_a (\nu x)Q}$$

$$OPEN \ : \ \frac{P \xrightarrow{\overline{x}<(\nu \vec{y})\vec{z}>}_a Q \qquad v \neq x, v \in \{\vec{z}\} - \{\vec{y}\}}{(\nu v)P \xrightarrow{\overline{x}<(\nu \vec{y},v)\vec{z}>}_a Q}$$

$$STRUCT \ : \ \frac{P \equiv P' \qquad P' \xrightarrow{l}_a Q' \qquad Q' \equiv Q}{P \xrightarrow{l}_a Q}$$

These rules define the transitions of a process that can be observed by an asynchronous observer. The *OUT* rule states that a sending process performs its output action and continues to 0. The IN_a rule can be understood as the observer sending a message to the 0 process. The message cannot be received by 0; and so it exists alongside the 0 process, waiting to be received (i.e. $0\,|\,\overline{x}<\vec{y}>$). The observer cannot detect that the message has not been consumed, since it is asynchronous. Thus, the 0 process behaves as if it performs an input action ($x(\vec{y})$). The *PAR* rule states that composition preserves the transitions of its constituent processes. The *COM* and $LAZY - REP$ rules show how the vector of output names ($\vec{y}$) substitute the vector of input names ($\vec{v}$) when two complementary actions interact. The *RES* rule states that restriction preserves a transition of its process if the transition is not restricted. The *OPEN* rule states that a restriction will preserve an output transition of a process if the transition name is not restricted; and the restriction will be exported with the output vector if the vector contains the restricted name. The *STRUCT* rule states that transitions are preserved by structural congruence.

There are a corresponding set of rules defining the transitions of a process that can be observed by a synchronous observer. These synchronous transitions are very similar to the asynchronous transitions defined above, with one exception:

$$IN \ : \ \frac{}{x(\vec{v}).P \xrightarrow{x(\vec{z})} \{\vec{z}/\vec{v}\}P}$$

3.3.1 Evaluation using Requirements

Dynamic reconfiguration in $A\pi$ consists of the creation of processes (achieved using lazy replication), and the creation and deletion of bindings between processes (achieved by passing port names in process communications).

Physical relocation of a process is modelled implicitly (as in the original π-calculus).

State transfer between processes can be modelled as communication.

Unplanned process deletion and unplanned process replacement cannot be modelled; and it is not possible to identify a specific instance of a process for reconfiguration (as in the previous π-calculi).

Functional interaction between application and reconfiguration actions is modelled as action interleavings, which can be analyzed using standard techniques; temporal interaction cannot be modelled; and functional correctness is based on equivalence of process expressions and substitution of processes, and is verified by equational reasoning (as in the previous π-calculi). *Asynchronous bisimulation* (written $\approx_a$) is a form of weak equivalence (discussed above) with asynchronous transitions; and *synchronous bisimulation* (written $\approx_s$) is a form of weak equivalence with synchronous transitions. Furthermore, both $\approx_a$ and $\approx_s$ are congruences. Hence, asynchronously/synchronously bisimilar processes can be substituted in any given process context with correspondingly equivalent results.

The concurrency model is that of a multiset of concurrently executing sequential processes that perform state transitions and communicate asynchronously with no time bound. Hence, process expressions are not amenable to schedulability analysis.

Support for functional correctness is provided through proofs of equivalence and congruence, and the operational semantics support reachability analysis (as in the previous π-calculi).

The type system is based on sorts and sortings, and is not significantly different from that used in the original π-calculus.

Support for temporal correctness is provided through guarded replication, which

is suitable for process creation in systems with finite resources; and asynchronous communication, which simplifies the timing dependency between sending and receiving processes. However, the process expressions are not amenable to schedulability analysis, because of the absence of timing information.

$A\pi$ is a simple and powerful formalism, despite its limitations, and it is a basis of research on process mobility [Ama00], fault-tolerance [Ber04] and composition of web services [Maz06]. Tool support includes Pict [PT00], which is a strongly-typed programming language.

3.4 Related Work

There are a number of π-calculi (in addition to those discussed above) that can be used to model and analyse dynamic reconfiguration of dependable systems.

The fusion calculus is an extension of the π-calculus [PV98]. It simplifies the π-calculus by treating input and output actions symmetrically, by having a single binding operator (called scope) and by having a single bisimulation congruence (called hyperequivalence). When complementary actions interact, the corresponding input and output names are identified using an equivalence relation (called a fusion), the visibility of which is controlled by the scope operator. Fusion causes symmetry in communications, but it has a fundamental weakness: the scope of a fusion includes any process composed with the processes related by the fusion, so that a fusion represents an implicit shared state. This is problematic, since shared state does not exist in distributed systems. Other weaknesses are the absence of real-time constructs and operators, and the lack of facilities for unplanned process creation, process deletion and process replacement.

The Φ-calculus is an extension of the π-calculus [RS03]. It was designed to model discrete systems that control continuously changing environments. An embedded system is modelled as a pair (E, P), where E is an environment and P is a hybrid process expression in the Φ-calculus. (E, P) changes according to: π-actions that change P; time actions that change E continuously; and e-actions that change both E and P discretely. Weak bisimulation is extended with e-transitions, and a notion of embedded bisimulation is defined such that if weakly bisimilar processes are substituted in the same environment, then the resulting embedded systems are embedded bisimilar. The Φ-calculus has a

number of deficiencies for our purpose: it is more complex than necessary, since it models both discrete and continuous change. It has a synchronous communication model, which complicates schedulability analysis. The unguarded replication operator is problematic when used to create processes in systems with finite resources. Unplanned process deletion and process replacement cannot be modelled.

A timed extension of $A\pi$ has been developed to model the reliability of communications in distributed systems, including message loss, site failure and timeouts for error detection [Ber04]. The process syntax of $A\pi$ is extended with a timer construct $timer^t(x(\vec{v}).P, Q)$, which executes $x(\vec{v})$ if a message is received before t time increments have passed, and otherwise executes Q. The work is interesting because of its integration of $A\pi$ with discrete time. However, the calculus has two main deficiencies for our purpose: unplanned process deletion and process replacement cannot be modelled, and specific instances of processes cannot be identified for reconfiguration.

$\texttt{web}\pi_\infty$ is an extension of $A\pi$ designed to model the composition of web services [Maz06]. The process syntax of $A\pi$ is extended with the construct $\langle\!|P\ ;\ Q|\!\rangle_x$ (termed a workunit) in order to model error handling. The workunit executes P until either P terminates (whereupon the workunit terminates) or an interrupt is received on channel x during the execution of P. The interrupt can be sent either by P or by a process in the context of the workunit, and causes the premature termination of P (without rollback) and the execution of Q. Thus, workunits can be used to model event-triggered planned reconfiguration. However, unplanned process deletion and process replacement cannot be modelled, and specific instances of processes cannot be identified for reconfiguration.

3.5 Discussion

It is clear from the review that most of the research on π-calculi has concentrated on link reconfiguration in non-real-time systems, with functional interaction between application and reconfiguration actions modelled as action interleavings, and functional correctness verified by equational reasoning using congruent process expressions and by model checking. Temporal interaction is not modelled and (therefore) temporal correctness cannot be verified. Unplanned dynamic reconfiguration is not modelled, and specific instances of processes cannot be identified for reconfiguration. The most serious of these defects, with respect to dynamic recon-

figuration, is the lack of constructs for modelling *unplanned dynamic reconfiguration*, which is defined as follows:

Definition 3.5.1 *Unplanned dynamic reconfiguration is the runtime change of a system's composition, structure or resource use that is not determined by the system's design.*

Unplanned dynamic reconfiguration is relevant for legacy systems that are not designed to be reconfigured, but have sufficient laxity to be reconfigured; and for systems that are required to evolve to meet new requirements.

The inability of the reviewed π-calculi to model unplanned dynamic reconfiguration is due to their use of special reconfiguration operators, as the following argument explains. A model of a system should be an abstraction of the system. An abstraction of a system is a representation of the system such that every construct of the abstraction represents one or more elements of the system, or is determined from one or more elements of the system, but not every element of the system has to be represented in the abstraction. Thus, an abstraction is a simplified representation of a system, and a property of the abstraction should be also a property of the system (if all the factors that determine the property in the system are represented in the abstraction). Therefore, if a model is an abstraction of a system, then if the model has a property then the system should have the same property. However, if a model has a fictitious construct, then making an inference about the system from the model can be problematic. The notion of *fictitious construct* is defined as follows:

Definition 3.5.2 *A fictitious construct of a model of a system is a construct that does not represent any element or collection of elements of the system, and is not determined from any element or collection of elements of the system.*

If a model of a system has a fictitious construct, then the model is not an abstraction of the system, and (therefore) if the model has a property based on the fictitious construct then the system may or may not have the same property. This uncertainty, caused by the use of a fictitious construct in a model, undermines confidence in the validity of the model. π-calculi use special reconfiguration operators to model dynamic process reconfiguration. For example, in $\texttt{web}\pi_\infty$, the workunit $\langle\!|P\ ;\ Q|\!\rangle_x$ executes P until an interrupt is received on channel x, whereupon Q executes instead of P. The use of a special operator to model process reconfiguration requires syntactic proximity of the operands, namely, the old process (P) and the new process (Q); and this requirement implies that the system model (containing

P) and the model of the reconfiguring system (containing *Q*) cannot be syntactically separated. However, syntactic separation of the system model and the model of the reconfiguring system is exactly what is required for modelling unplanned dynamic process reconfiguration as an abstraction. Therefore, the reviewed π-calculi cannot be used to model unplanned dynamic process reconfiguration as an abstraction. Instead, they are used to model unplanned reconfiguration as planned reconfiguration by introducing fictitious constructs (such as new processes) into the system model, which undermines confidence in the model's validity, because the use of fictitious constructs can result in fictitious properties.

Another defect of π-calculi with respect to dependability is their emphasis on link reconfiguration rather than process reconfiguration. Many dependable systems are control systems, which consist of non-terminating processes with a stable communications topology. For these systems, process reconfiguration due to a mode change is much more relevant than link reconfiguration.

Therefore, our research focuses on process reconfiguration. In a process algebra, the only alternative to using a special operator to model process reconfiguration is to use a special process. Therefore, a new type of process (termed a *fraction process*) is defined in the following chapter to model both planned and unplanned process reconfiguration. Our process algebra (CCS^{dp}) is based on *CCS*, which was chosen for the following reasons. First, *CCS* is one of the simplest process algebras that is capable of modelling computations. Therefore, it is easier to extend to suit our modelling requirements than a more complex process algebra; and (for the same reason) it is an easier environment in which to experiment with new modelling constructs and 'tune' them to our requirements than a more complex process algebra. Second, *CCS* is the base for π-calculi. Therefore, it should be possible to extend CCS^{dp} to model link reconfiguration and process relocation. Third, *CCS* has no facility for link reconfiguration. Therefore, it should be possible to extend CCS^{dp} with real-time constructs to model process reconfiguration in control systems that do not require link reconfiguration.

Chapter 4

Basic CCS^{dp}

Contents

In this chapter, for simplicity, *CCS*^{dp} is developed using basic *CCS* [Mil99] without the restriction operator (ν). We define the process syntax and explain the design decisions behind the formulation of the fraction process. We then define the labelled transition system (LTS) semantics of basic *CCS*^{dp} and strong of-bisimulation ($\sim_{of}$), and explain the decision to use behavioural matching between processes (based on strong of-bisimulation) in the semantics. We prove that strong of-bisimulation is an equivalence relation, which is useful for behavioural matching, but that it

is not a congruence, and so cannot be used for equational reasoning. Therefore, we define a stronger relation – strong dp-bisimulation ($\sim_{dp}$) – and prove this is a congruence. The decidability of strong of-bisimulation and strong dp-bisimulation is necessary for the automation of equational reasoning and model checking of process expressions in CCS^{dp}; and the automation is necessary for the usability of CCS^{dp}. Therefore, we impose restrictions on the structure of processes in order to facilitate decidability of both bisimulations. We discuss alternative ways of matching processes, using syntactic equality, structural congruence and strong observation equivalence, and identify tradeoffs. Finally, CCS^{dp} is evaluated with respect to the requirements used to evaluate π-calculi in the previous chapter.

4.1 Syntax

Let $\mathcal{N}$ be the countable set of names (e.g. a, b, c) that represent both input ports and input actions of the processes in basic CCS^{dp}; and let $\overline{\mathcal{N}}$ be the countable set of complementary names (e.g. $\overline{a}$, $\overline{b}$, $\overline{c}$) that represent both output ports and output actions of the processes in basic CCS^{dp}, where $\overline{\mathcal{N}} \triangleq \{\bar{l} \mid l \in \mathcal{N}\}$. Let $\mathcal{PN}$ be the countable set of names (e.g. A, B, C) of the processes in basic CCS^{dp}. The sets $\mathcal{N}$, $\overline{\mathcal{N}}$ and $\mathcal{PN}$ are assumed to be pairwise disjoint.

Thus, given $a \in \mathcal{N}$, a represents the input action on the input port a of a process; and $\overline{a}$ represents the complementary output action on the output port $\overline{a}$ of a process. The interaction between complementary actions (such as a and $\overline{a}$) is represented by the special action τ, which is internal to a process.

Let $\mathcal{L}$ be the set of names that represent both ports and actions of the processes in basic CCS^{dp}, where $\mathcal{L} \triangleq \mathcal{N} \cup \overline{\mathcal{N}}$.

We define the function $\bar{} : \mathcal{L} \longrightarrow \mathcal{L}$ such that $\overline{x} \triangleq \begin{cases} \bar{l} & \text{if } \exists l \in \mathcal{N}(x = l) \\ l & \text{elseif } \exists \bar{l} \in \overline{\mathcal{N}}(x = \bar{l}) \end{cases}$

so that $\forall l \in \mathcal{N}(\bar{\bar{l}} = l)$ (as required by convention).

Let $\mathcal{I}$ be the set of input and output ports/actions of the processes in basic CCS^{dp}, and their internal action (τ), where $\mathcal{I} \triangleq \mathcal{L} \cup \{\tau\}$.

Let $\mathcal{P}$ be the set of processes in basic CCS^{dp}.

The syntax of a process P in $\mathcal{P}$ is defined as follows (using the style in [SW01]):

$$\begin{array}{lll} P & ::= & PN{<}\widetilde{\beta}{>} \;\mid\; M \;\mid\; P|P \;\mid\; \frac{P}{P} \\ M & ::= & 0 \;\mid\; \alpha.P \;\mid\; M+M \end{array} \tag{4.1}$$

where $PN \in \mathcal{PN}$, $\widetilde{\beta}$ is a tuple of elements of $\mathcal{N} \cup \overline{\mathcal{N}}$, and $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$.

Thus, the syntax of basic CCS^{dp} is the syntax of basic CCS without ν extended with the $\frac{P'}{P}$ construct.

As in CCS, 0 is the *NIL* process, which has no behaviour. It is typically used at the end of a trace of a process to indicate termination of the process.

Prefix (e.g. $\alpha.P$) models sequential action. For example, $\alpha \in \mathcal{N}$ represents the input action on the input port α of a process, $\overline{\alpha} \in \overline{\mathcal{N}}$ represents the complementary output action on the output port $\overline{\alpha}$ of a process, and τ represents the internal action of a process. After performing α, the process $\alpha.P$ continues as P.

Summation (e.g. $M + M'$) models non-deterministic choice of actions by a process. Notice that 0 can be represented as the empty summation $\sum_{\emptyset}$ (by convention). Notice also that a non-0 term in a summation is guarded by a prefix action in order to prevent the creation of an infinite number of processes, which complicates reasoning.

$A{<}\widetilde{\beta}{>}$ models the invocation of a constant process named A, instantiated with a tuple of port/action names $\widetilde{\beta}$. $A(\widetilde{\beta})$ has a unique definition, which can be recursive.

Parallel composition (e.g. $P|P'$) models the execution of concurrent processes and their direct functional interaction, as well as process composition and decomposition. Interaction between processes is synchronous and point-to-point.

A *fraction* (e.g. $\frac{P'}{P}$) is a process that models process replacement and deletion. On creation, the fraction $\frac{P'}{P}$ identifies any instance of a process matching its denominator process P with which it is composed in parallel, and replaces that process atomically with the numerator process P'. If no such process instance exists, the fraction continues to exist until such a process is created (or the fraction is itself deleted or replaced). If there is more than one such process instance, a non-deterministic choice is made as to which process is replaced. Similarly, if more than one fraction can

replace a process instance, a non-deterministic choice is made as to which fraction replaces the process. Deletion of a process P is achieved by parallel composition with $\frac{0}{P}$. If P progresses to Q, then $\frac{P'}{P}$ will not replace Q by P' (unless Q matches P). Notice that a fraction has no communication behaviour; its only behaviour is to replace a process with which it is composed in parallel that matches its denominator. The matching is done by behaviour using a bisimulation, as explained in Section 4.2.

Operator Precedence

In *CCS*, the precedence of the operators (in decreasing order) is:
relabelling (highest); prefix; parallel composition; summation (lowest).

However, in CCS^{dp}, the syntax of summation implies that parallel composition must have a lower precedence than summation. For example, $M + M'|P$ should mean $M + (M'|P)$ (by the *CCS* precedence rules); but a parallel composition of processes cannot be a term in a summation (by the CCS^{dp} syntax rules for summation). Therefore, in CCS^{dp}, $M + M'|P$ is taken to mean $(M + M')|P$.

Therefore, in CCS^{dp}, the precedence of the operators (in decreasing order) is:
fraction formation (highest); relabelling; prefix; summation; parallel composition (lowest).

Free Names and Bound Names

In general, the name of a port/action of a process is termed a *bound* name of the process if its scope is restricted to the process; otherwise, it is termed a *free* name. In the absence of value-passing and the restriction operator, all names of ports/actions of processes in $\mathcal{P}$ are free names.

Given $p \in \mathcal{P}$, let $nm(p)$, $fn(p)$ and $bn(p)$ be the set of port/action names, free names and bound names of p respectively, and are defined as follows:

$nm : \mathcal{P} \longrightarrow \mathbb{P}\mathcal{L}$ such that $nm(p) \triangleq fn(p) \cup bn(p)$

$fn : \mathcal{P} \longrightarrow \mathbb{P}\mathcal{L}$ such that

$$fn(p) \triangleq \begin{cases} \emptyset & \text{if } p = 0 \\ \{\beta\} \cup fn(p_1) & \text{elseif } p = \beta.p_1 \ \wedge \ \beta \in \mathcal{N} \cup \overline{\mathcal{N}} \\ fn(M_1) \cup fn(M_2) & \text{elseif } p = M_1 + M_2 \\ fn(p_1) \cup fn(p_2) & \text{elseif } p = p_1 | p_2 \\ fn(p_1) \cup fn(p_2) & \text{elseif } p = \frac{p_1}{p_2} \\ Set(\widetilde{\beta}) & \text{elseif } p = A < \widetilde{\beta} > \end{cases}$$

$bn : \mathcal{P} \longrightarrow \mathbb{P}\mathcal{L}$ such that $bn(p) \triangleq \emptyset$

4.1.1 Rationale

Special reconfiguration operators cannot be used to model unplanned dynamic reconfiguration (see Section 3.5). In a process algebra, the only alternative is to use a special reconfiguration process, and operator overloading for composing a system model with a model of a reconfiguring system. The standard operator for composing process expressions is parallel composition (|), and its associativity and commutativity properties allow modularity of process expressions. Therefore, we decided to overload the parallel composition operator for dynamic reconfiguration. Operator overloading also simplifies the syntax of CCS^{dp}.

The syntactic separation of the process to be reconfigured from the reconfiguring process creates the need for a modelling mechanism to bind the two processes dynamically. We term this modelling mechanism *process matching*, or simply *matching* (if there is no possibility of confusion with the port/action matching used in π-calculi). Matching is expressed using similarity between processes, specifically, behavioural similarity based on a bisimulation, as explained in Section 4.2.

The basic process reconfiguration operations are creation, deletion and replacement. Process creation is readily described in *CCS*, but not unplanned process deletion or replacement. An intuitive solution is to define an inverse process (P^{-1}) to model process deletion [BF08], since process replacement is simply a combination of process creation and deletion. However, using P^{-1} in a concurrent model with non-deterministic transitions can result in deletion of the wrong process, and in creation of an infinite number of processes (see [BF08]). Therefore,

the fraction process ($\frac{P'}{P}$) was defined to model process replacement and deletion [BF08], which resolves the problem (provided there is only one matching process to be reconfigured). Furthermore, the syntactic recursion of fraction processes enables a fraction to be itself reconfigured, which cannot be achieved as easily with inverse processes. For example, the inverse of P^{-1} is P, but P can evolve to P' instead of deleting P^{-1}; whereas a fraction process can perform only reconfiguration actions.

The dichotomy between operator-based and process-based reconfiguration of processes, and their relationship to planned and unplanned reconfiguration of systems, can be summarized as follows. Operator-based modelling of reconfiguration requires syntactic proximity between the operands. This implies both the reconfigured and the reconfiguring processes must be within the system model, which implies both the reconfigured and the reconfiguring components must be within the system (because the model must be an abstraction of the system), which is planned reconfiguration. In contrast, process-based modelling of reconfiguration does not require syntactic proximity between the reconfigured and reconfiguring processes (because the processes are bound dynamically using a bisimulation). This enables the reconfiguring process to be located either within the system model or in the context of the system model, which implies the reconfiguring component can be located within the system or in the environment of the system, which are (respectively) planned reconfiguration and unplanned reconfiguration. Furthermore, the step through which a reconfiguring process is added to the context of the system model is performed outside the calculus, and thereby captures the fact that system evolution is unplanned. This is illustrated in Section 5.3.3. Notice that a fraction is a process (**not** an operator) because it has the syntax of a process (see Equation 4.1).

4.2 Labelled Transition System Semantics

A structural operational semantics (SOS) of a process algebra is a set of inference rules that define the possible actions (termed *transitions*) that any process in the algebra can perform. The conclusion of each SOS rule defines a transition of a process, and the hypothesis of the rule defines the weakest precondition of the transition [Plo04].

An SOS defines a directed graph (termed a *transition system*), in which each vertex (termed a *configuration*) represents the cartesian product of an SOS rule and a process or only a process, and each arc represents a transition of the source process.

If the arcs are labelled, the graph is termed a *labelled transition system* (LTS).

CCS^{dp} has a structural operational semantics. This was done for the following reasons. First, CCS^{dp} is based on *CCS*, and the semantics of *CCS* is defined using an SOS [Mil99], [Mil89]. Second, dependable systems are typically concurrent with non-deterministic behaviour, and it is easier to express the semantics of concurrent non-deterministic systems using SOS than using other semantics [Hut10]. Third, SOS integrates the representations of process behaviour and process structure, which facilitates modelling of the reconfiguration behaviour of a process, the computational behaviour of the process, their effect on the structure of the process, and thereby their interaction. The semantics of CCS^{dp} is expressed using a labelled transition system, because the labels facilitate proof of properties of processes and their transitions.

4.2.1 LTS Rules

Let $\mathcal{R}$ be the countable set of reconfiguration actions of the processes in $\mathcal{P}$ (e.g. τ_{r_X}, τ_{r_Y}, τ_{r_Z}) that create a process in $\mathcal{P}$; and let $\overline{\mathcal{R}}$ be the countable set of complementary reconfiguration actions of the processes in $\mathcal{P}$ (e.g. $\overline{\tau}_{r_X}$, $\overline{\tau}_{r_Y}$, $\overline{\tau}_{r_Z}$) that delete a process in $\mathcal{P}$, where $\overline{\mathcal{R}} \triangleq \{\overline{\tau}_{r_X} \mid \tau_{r_X} \in \mathcal{R}\}$ (see the *Creat* and *Delet* rules below). Each action in $\mathcal{R}$ is represented by τ_{r_X}, with $X \in \mathcal{P}$. The sets $\mathcal{N}$, $\overline{\mathcal{N}}$, $\{\tau\}$, $\mathcal{R}$, $\overline{\mathcal{R}}$ and $\mathcal{PN}$ are assumed to be pairwise disjoint.

Let C be the set of reconfiguration actions of the processes in $\mathcal{P}$, where $C \triangleq \mathcal{R} \cup \overline{\mathcal{R}}$. We extend the definition of the function $\bar{}$ as follows:

$$\bar{} : \mathcal{L} \cup C \longrightarrow \mathcal{L} \cup C \text{ such that } \overline{\lambda} \triangleq \begin{cases} \bar{l} & \text{if } \exists l \in \mathcal{N} (\lambda = l) \\ l & \text{elseif } \exists \bar{l} \in \overline{\mathcal{N}} (\lambda = \bar{l}) \\ \overline{\tau}_{r_X} & \text{elseif } \exists\, \tau_{r_X} \in \mathcal{R} \,(\lambda = \tau_{r_X}) \\ \tau_{r_X} & \text{elseif } \exists\, \overline{\tau}_{r_X} \in \overline{\mathcal{R}} \,(\lambda = \overline{\tau}_{r_X}) \end{cases}$$

so that $\forall \lambda \in \mathcal{L} \cup C \,(\overline{\overline{\lambda}} = \lambda)$ (as required by convention).

Let $\mathcal{A}$ be the set of actions of the processes in $\mathcal{P}$, where $\mathcal{A} \triangleq \mathcal{I} \cup C$.

The LTS rules for basic CCS^{dp} are a superset of the LTS rules for basic *CCS* without ν, consisting of an unchanged rule of basic *CCS* (i.e. *Sum*) plus basic *CCS* rules applicable to reconfiguration transitions (i.e. *React*, *L-Par*, *R-Par* and *Ident*) plus additional

rules to describe new reconfiguration behaviour (i.e. *Creat*, *Delet*, *CompDelet*, *L-React* and *R-React*). See Table 4.1. Notice that $\mathcal{P}^+$ is the set of positive processes of $\mathcal{P}$, which is defined in Section 4.2.2. The notion of strong of-bisimulation on $\mathcal{P}$ ($\sim_{of}$) is defined in Section 4.2.3.

Sum	$\dfrac{k \in I}{\sum_{i\in I}\alpha_i.P_i \xrightarrow{\alpha_k} P_k}$ where I is a finite indexing set	
React	$\dfrac{\lambda \in \mathcal{L}\cup\mathcal{C} \ \wedge\ P \xrightarrow{\lambda} P' \ \wedge\ Q \xrightarrow{\overline{\lambda}} Q'}{P\mid Q \xrightarrow{\tau} P'\mid Q'}$	
L-Par	$\dfrac{\mu \in \mathcal{A} \ \wedge\ P \xrightarrow{\mu} P'}{P\mid Q \xrightarrow{\mu} P'\mid Q}$	R-Par $\dfrac{\mu \in \mathcal{A} \ \wedge\ Q \xrightarrow{\mu} Q'}{P\mid Q \xrightarrow{\mu} P\mid Q'}$
Ident	$\dfrac{\lvert\widetilde{b}\rvert=\lvert\widetilde{a}\rvert \ \wedge\ \mu\in\mathcal{A} \ \wedge\ P[\frac{\widetilde{b}}{\widetilde{a}}] \xrightarrow{\mu} P'}{A\langle\widetilde{b}\rangle \xrightarrow{\mu} P'}$ where $A(\widetilde{a}) \triangleq P$	
Creat	$\dfrac{P \sim_{of} Q \ \wedge\ P \in \mathcal{P}^+}{\frac{P'}{P} \xrightarrow{\tau_{r_Q}} P'}$	Delet $\dfrac{P \sim_{of} Q \ \wedge\ P \in \mathcal{P}^+}{P \xrightarrow{\overline{\tau}_{r_Q}} 0}$
CompDelet	$\dfrac{R \sim_{of} R_1\mid R_2 \ \wedge\ P \xrightarrow{\overline{\tau}_{r_{R_1}}} P' \ \wedge\ P' \xrightarrow{\overline{\tau}_{r_{R_2}}} P''}{P \xrightarrow{\overline{\tau}_{r_R}} P''}$	
L-React	$\dfrac{R \sim_{of} R_1\mid R_2 \ \wedge\ P \xrightarrow{\overline{\tau}_{r_{R_1}}} P' \ \wedge\ P' \xrightarrow{\tau_{r_R}} P'' \ \wedge\ Q \xrightarrow{\overline{\tau}_{r_{R_2}}} Q'}{P\mid Q \xrightarrow{\tau} P''\mid Q'}$	
R-React	$\dfrac{R \sim_{of} R_1\mid R_2 \ \wedge\ P \xrightarrow{\overline{\tau}_{r_{R_1}}} P' \ \wedge\ Q \xrightarrow{\overline{\tau}_{r_{R_2}}} Q' \ \wedge\ Q' \xrightarrow{\tau_{r_R}} Q''}{P\mid Q \xrightarrow{\tau} P'\mid Q''}$	

Table 4.1: Labelled Transition System Semantics of Basic CCS^{dp}.

$$\text{Sum} \ : \ \frac{k \in I}{\sum_{i\in I}\alpha_i.P_i \xrightarrow{\alpha_k} P_k}$$

The *Sum* rule states that summation preserves the transitions of constituent processes as a non-deterministic choice of alternative transitions.

For example, *REC* is a process that can receive one of a finite number of alternative orders drawn from the set O (see below). *REC* receives an order k in O by performing the transition $Receipt_k$, after which it becomes the two processes *WORKFLOW* and $\overline{InventoryCheck}_k$ that execute concurrently. By convention, we

omit the 0 process following $\overline{InventoryCheck}_o$ and $\overline{InventoryCheck}_k$.

$REC \triangleq \sum_{o \in O} Receipt_o.(WORKFLOW \mid \overline{InventoryCheck}_o)$
$REC \xrightarrow{Receipt_k} WORKFLOW \mid \overline{InventoryCheck}_k$

$$\text{React} : \frac{\lambda \in \mathcal{L} \cup \mathcal{C} \quad \wedge \quad P \xrightarrow{\lambda} P' \quad \wedge \quad Q \xrightarrow{\overline{\lambda}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

The *React* rule states that if two processes can perform complementary transitions, then their parallel composition can result in a τ transition in which both processes undergo their respective complementary transitions atomically.

For example, *IC* is a process that can receive an inventory check request for order k in O by performing the transition $InventoryCheck_k$ (by the *Sum* rule, see below). $\overline{InventoryCheck}_k$ sends an inventory check request for order k, and $InventoryCheck_k$, $\overline{InventoryCheck}_k$ are complementary transitions. Therefore, the parallel composition of *IC* and $\overline{InventoryCheck}_k$ can react by performing a τ transition.

$IC \triangleq \sum_{o \in O} InventoryCheck_o.\tau.(\overline{InventoryCheckNotOK}_o + \overline{InventoryCheckOK}_o)$
$IC \mid \overline{InventoryCheck}_k \xrightarrow{\tau} \tau.(\overline{InventoryCheckNotOK}_k + \overline{InventoryCheckOK}_k) \mid 0$

$$\text{L-Par} : \frac{\mu \in \mathcal{A} \quad \wedge \quad P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q} \qquad \text{R-Par} : \frac{\mu \in \mathcal{A} \quad \wedge \quad Q \xrightarrow{\mu} Q'}{P|Q \xrightarrow{\mu} P|Q'}$$

The *L-Par* and *R-Par* rules state that parallel composition preserves the transitions of constituent processes.

For example, the parallel composition of *REC* and *IC* can perform both the transitions of *REC* and the transitions of *IC*.

$REC \xrightarrow{Receipt_k} WORKFLOW \mid \overline{InventoryCheck}_k$
(by the *Sum* rule, for any k in O) and
$REC \mid IC \xrightarrow{Receipt_k} (WORKFLOW \mid \overline{InventoryCheck}_k) \mid IC$
(by the *L-Par* rule) and
$IC \xrightarrow{InventoryCheck_l} \tau.(\overline{InventoryCheckNotOK}_l + \overline{InventoryCheckOK}_l)$
(by the *Sum* rule, for any l in O) and
$REC \mid IC \xrightarrow{InventoryCheck_l} REC \mid \tau.(\overline{InventoryCheckNotOK}_l + \overline{InventoryCheckOK}_l)$
(by the *R-Par* rule).

$$\text{Ident} : \frac{|\widetilde{b}| = |\widetilde{a}| \quad \wedge \quad \mu \in \mathcal{A} \quad \wedge \quad P[\frac{\widetilde{b}}{\widetilde{a}}] \xrightarrow{\mu} P'}{A < \widetilde{b} > \xrightarrow{\mu} P'} \text{ where } A(\widetilde{a}) \triangleq P$$

The *Ident* rule states that if a relabelled constant process can perform a given transition, then the constant process instantiated with the new labelling can also undergo the same transition.

For example, the process named $A(a_1, a_2)$ is defined as follows: $A(a_1, a_2) \triangleq a_1.\tau.\overline{a}_2$
Relabelling $A(a_1, a_2)$ with (b_1, b_2) results in the process expression $b_1.\tau.\overline{b}_2$,
which is the same process expression named by $A(b_1, b_2)$.
Therefore, $A(a_1, a_2)$ relabelled with (b_1, b_2) and $A(b_1, b_2)$ have identical transitions.

$$\text{Creat} : \frac{P \sim_{of} Q \quad \wedge \quad P \in \mathcal{P}^+}{\frac{P'}{P} \xrightarrow{\tau_{r_Q}} P'} \qquad \text{Delet} : \frac{P \sim_{of} Q \quad \wedge \quad P \in \mathcal{P}^+}{P \xrightarrow{\overline{\tau}_{r_Q}} 0}$$

Intuitively, it should be possible to reconfigure only processes that exist, and we identify these processes as the processes with behaviour (termed *positive processes*). Processes without behaviour (i.e. 0 and 0-like processes) are terminated processes, and (therefore) no longer exist, and (therefore) cannot be reconfigured. Therefore, the hypotheses of *Creat* and *Delet* restrict reconfiguration transitions to positive processes. A more detailed explanation of the need for positive processes is given in Section 4.2.2.

The *Creat* rule states that if P is a positive process ($P \in \mathcal{P}^+$) that matches Q using *strong of-bisimulation* ($P \sim_{of} Q$), then the fraction process $\frac{P'}{P}$ can perform the reconfiguration transition τ_{r_Q} that results in the creation of P'. The *Delet* rule is complementary to the *Creat* rule. It states that if P is a positive process that matches Q using strong of-bisimulation, then P can be deleted by performing the reconfiguration transition $\overline{\tau}_{r_Q}$ that is complementary to the reconfiguration transition τ_{r_Q} performed by some fraction that creates a process. Thus, P' replaces P as a result of the reaction between $\frac{P'}{P}$ performing τ_{r_Q} and P performing $\overline{\tau}_{r_Q}$ (defined by the *React* rule).

For example, REC is a positive process (because it has behaviour). Therefore, $\frac{REC'}{REC}$ can perform the reconfiguration transition $\tau_{r_{REC}}$ that results in the creation of the process REC' (because $\sim_{of}$ is reflexive). REC can perform the reconfiguration transition $\overline{\tau}_{r_{REC}}$, which results in its deletion (because REC is a positive process). The transitions $\tau_{r_{REC}}$ and $\overline{\tau}_{r_{REC}}$ are complementary. Therefore, the parallel composition

of $\frac{REC'}{REC}$ and REC can react by performing a τ transition that results in the creation of REC' and the deletion of REC, which is the replacement of REC by REC'.

$REC \triangleq \sum_{o \in O} Receipt_o.(WORKFLOW \mid \overline{InventoryCheck}_o)$ (as above).
$\frac{REC'}{REC} \xrightarrow{\tau_{r_{REC}}} REC'$ (by the *Creat* rule) and
$REC \xrightarrow{\overline{\tau}_{r_{REC}}} 0$ (by the *Delet* rule).
Therefore, $\frac{REC'}{REC} \mid REC \xrightarrow{\tau} REC' \mid 0$ (by the *React* rule).

Strong of-bisimulation is used for matching for two reasons. First, strong of-bisimulation is a relation between processes. The use of a relation avoids modelling reconfiguration mechanisms and operators, which (respectively) simplifies models and facilitates the modelling of unplanned reconfiguration. The use of reconfiguration transitions in the LTS semantics, and not in the syntax, also helps to achieve these objectives. Thus, the relation is a pre-condition that allows a process to be reconfigured only when it is in a specified state, which is an important requirement for reconfigurable systems. Furthermore, the separation of fractions and mechanisms enables a fraction process to be combined with general purpose triggering mechanisms (such as prefixes, interrupts and timeouts) without changing their semantics. Second, strong of-bisimulation helps to maximize the terseness of expressions modelling reconfiguration, which simplifies modelling.

The notions of positive process and strong of-bisimilarity are defined below. Notice that reconfiguration transitions do not involve any communication. Therefore, the interaction between complementary reconfiguration transitions does not require a port or a communication channel.

$$\text{CompDelet} : \frac{R \sim_{of} R_1 | R_2 \quad \wedge \quad P \xrightarrow{\overline{\tau}_{r_{R_1}}} P' \quad \wedge \quad P' \xrightarrow{\overline{\tau}_{r_{R_2}}} P''}{P \xrightarrow{\overline{\tau}_{r_R}} P''}$$

The *CompDelet* rule states that consecutive delete transitions of a process can be composed into a single delete transition of the process. The rule is applicable only if it is used in combination with *L-Par* or *R-Par*.

For example, REC and IC are both positive processes (because both REC and IC have behaviour). Therefore, REC can perform the reconfiguration transition $\overline{\tau}_{r_{REC}}$, which results in its deletion; and IC can perform the reconfiguration transition $\overline{\tau}_{r_{IC}}$, which results in its deletion (because $\sim_{of}$ is reflexive). Therefore, the parallel

composition of *REC* and *IC* can perform the transitions $\overline{\tau}_{r_{REC}}$ and $\overline{\tau}_{r_{IC}}$ consecutively (using the *L-Par* and *R-Par* rules), which results in 0|0. The same result can be produced by performing the reconfiguration transition $\overline{\tau}_{r_{REC|IC}}$ (because $\sim_{of}$ is reflexive).

$REC \mid IC \xrightarrow{\overline{\tau}_{r_{REC}}} 0 \mid IC$ (by the *Delet* and *L-Par* rules) and
$0 \mid IC \xrightarrow{\overline{\tau}_{r_{IC}}} 0 \mid 0$ (by the *Delet* and *R-Par* rules).
Therefore, $REC \mid IC \xrightarrow{\overline{\tau}_{r_{REC|IC}}} 0 \mid 0$ (by the *CompDelet* rule, because $\sim_{of}$ is reflexive).

$$\text{L-React} : \frac{R \sim_{of} R_1|R_2 \quad \wedge \quad P \xrightarrow{\overline{\tau}_{r_{R_1}}} P' \quad \wedge \quad P' \xrightarrow{\tau_{r_R}} P'' \quad \wedge \quad Q \xrightarrow{\overline{\tau}_{r_{R_2}}} Q'}{P|Q \xrightarrow{\tau} P''|Q'}$$

$$\text{R-React} : \frac{R \sim_{of} R_1|R_2 \quad \wedge \quad P \xrightarrow{\overline{\tau}_{r_{R_1}}} P' \quad \wedge \quad Q \xrightarrow{\overline{\tau}_{r_{R_2}}} Q' \quad \wedge \quad Q' \xrightarrow{\tau_{r_R}} Q''}{P|Q \xrightarrow{\tau} P'|Q''}$$

In a process expression, the denominator of a fraction process can match the parallel composition of processes that are located on different sides of the fraction. The *L-React* and *R-React* rules state that a reconfiguration reaction can occur in this case, with all the processes participating in the reaction undergoing their respective transitions atomically.

For example, if *REC*, *REC'*, *IC*, *IC'* are processes (and *REC*, *IC* defined as above),
then $REC \xrightarrow{\overline{\tau}_{r_{REC}}} 0$ and $IC \xrightarrow{\overline{\tau}_{r_{IC}}} 0$
(by the *Delet* rule, because *REC*, *IC* are positive processes and $\sim_{of}$ is reflexive),
and $\frac{REC' \mid IC'}{REC \mid IC} \xrightarrow{\tau_{r_{REC \mid IC}}} REC' \mid IC'$
(by the *Creat* rule, because the parallel composition of positive processes is a positive process and $\sim_{of}$ is reflexive).
Therefore, $REC \mid \frac{REC' \mid IC'}{REC \mid IC} \xrightarrow{\overline{\tau}_{r_{REC}}} 0 \mid \frac{REC' \mid IC'}{REC \mid IC}$ (by the *L-Par* rule), and
$0 \mid \frac{REC' \mid IC'}{REC \mid IC} \xrightarrow{\tau_{r_{REC \mid IC}}} 0 \mid (REC' \mid IC')$ (by the *R-Par* rule), and
$\left(REC \mid \frac{REC' \mid IC'}{REC \mid IC}\right) \mid IC \xrightarrow{\tau} (0 \mid (REC' \mid IC')) \mid 0$ (by the *L-React* rule).
Thus, the fraction $\frac{REC' \mid IC'}{REC \mid IC}$ atomically replaces the two processes *REC* and *IC* that are located on different sides of the fraction.
Similarly, $\frac{REC' \mid IC'}{REC \mid IC} \mid IC \xrightarrow{\overline{\tau}_{r_{IC}}} \frac{REC' \mid IC'}{REC \mid IC} \mid 0$ (by the *R-Par* rule), and
$\frac{REC' \mid IC'}{REC \mid IC} \mid 0 \xrightarrow{\tau_{r_{REC \mid IC}}} (REC' \mid IC') \mid 0$ (by the *L-Par* rule), and

$REC \mid \left(\frac{REC' \mid IC'}{REC \mid IC} \mid IC\right) \xrightarrow{\tau} 0 \mid ((REC' \mid IC') \mid 0)$ (by the *R-React* rule).

In *CCS*, the parallel composition operator is associative and commutative with respect to strong bisimulation, and it is desirable to retain these properties of parallel composition with respect to strong of-bisimulation in CCS^{dp}, because associativity and commutativity support equational reasoning. However, the denominator of a fraction can match the parallel composition of two or more processes, which enables the fraction to replace multiple processes atomically. The replacement of these processes must be possible even if the processes are parenthesized differently or are reordered, in order to preserve the associativity and commutativity of parallel composition with respect to strong of-bisimulation. The *CompDelet* rule helps to ensure associativity, and the *L-React* and *R-React* rules help to ensure commutativity, of parallel composition with respect to strong of-bisimulation.

For example, the *CompDelet* rule is necessary in order to prove:

$$((a.0 \mid b.0) \mid c.0) \mid \frac{d.0}{a.0 \mid b.0} \;\sim_{of}\; (a.0 \mid (b.0 \mid c.0)) \mid \frac{d.0}{a.0 \mid b.0} \tag{4.2}$$

And the *L-React* and *R-React* rules are necessary in order to prove:

$$(b.0 \mid \frac{d.0}{a.0 \mid b.0}) \mid a.0 \;\sim_{of}\; a.0 \mid (b.0 \mid \frac{d.0}{a.0 \mid b.0}) \tag{4.3}$$

The LTS transitions are defined to be the smallest relation on $\mathcal{P}$ that satisfies the LTS rules. Therefore, a process $p \in \mathcal{P}$ performs a transition $p \xrightarrow{\mu} p'$ with $\mu \in \mathcal{A}$ and $p' \in \mathcal{P}$ iff the hypothesis of some LTS rule that determines the $p \xrightarrow{\mu} p'$ transition is satisfied.

4.2.2 Positive Processes and Zero Processes

0 is the identity of the summation and parallel composition operators in the equivalences and congruences of *CCS*, and it is desirable to retain this property of 0 in CCS^{dp} because it helps to manipulate process expressions during reasoning. However, the identity property of 0 in combination with fraction processes with a 0-valued denominator is problematic. For example, we would like the following

bisimilarity to hold, by the identity properties of 0:

$$\frac{a.0}{0}|0 \;\sim_{of}\; \frac{a.0}{0} \tag{4.4}$$

If the restriction on the denominators of fractions to be positive processes in the *Creat* and *Delet* rules is elided, we have:

$$\frac{a.0}{0}|0 \xrightarrow{\tau} a.0 \;\text{ and }\; \frac{a.0}{0} \xrightarrow{\tau}\!\!\!\!\!\!\rightarrow \tag{4.5}$$

This is a contradiction with respect to equation 4.4, since the transitions of $\frac{a.0}{0}|0$ and $\frac{a.0}{0}$ should be the same (by equation 4.4).

One solution to the contradiction is to define any fraction with denominator matching 0 as undefined (as done with the number 0 in arithmetic). However, this causes problems in formulating well-defined processes, and also if the denominator of a fraction is a complex process expression that matches 0.

A simpler solution is to define any fraction process with denominator matching 0 as a *zero process*, and to exclude all zero processes from reconfiguration transitions defined by the LTS rules, so that a reconfiguration reaction cannot occur. This is the approach we have adopted. Therefore, we distinguish processes that can be reconfigured (positive processes) from processes that cannot be reconfigured (zero processes).

Let $\mathcal{P}^+$ be the set of *positive processes* of $\mathcal{P}$, where $\mathcal{P}^+$ is defined to be the smallest subset of $\mathcal{P}$ that satisties the following conditions:

1. $\forall \alpha \in \mathcal{I}\ \forall p \in \mathcal{P}\ (\alpha.p \in \mathcal{P}^+)$
2. $\forall p, q \in \mathcal{P}\ (p + q \in \mathcal{P} \wedge (p \in \mathcal{P}^+ \vee q \in \mathcal{P}^+) \Longrightarrow p + q \in \mathcal{P}^+)$
3. $\forall p, q \in \mathcal{P}\ (p \in \mathcal{P}^+ \vee q \in \mathcal{P}^+ \Longrightarrow p|q \in \mathcal{P}^+)$
4. $\forall p \in \mathcal{P}\ \forall q \in \mathcal{P}^+ \left(\frac{p}{q} \in \mathcal{P}^+\right)$
5. $\forall \beta \in \mathcal{I}\ \forall X \in \mathcal{PN}\ (\beta.X \in \mathcal{P}^+)$

Thus, p is a positive process if it can perform an input or an output or a τ prefix action, or a reconfiguration action that creates a process. Notice that if p is a positive process, then so is $p+0$, $p|0$, $\frac{0}{p+0}$ and $\frac{0}{p|0}$. However, the restriction that $\mathcal{P}^+$ must be the smallest set satisfying the above properties excludes processes such as 0, 0 + 0, 0|0, $\frac{p}{0}$, $\frac{p}{(\frac{p}{0})}$ and so on. These excluded processes are collected into the set $\mathcal{P}^0$, termed the

set of *zero processes* of $\mathcal{P}$, which is defined as the smallest subset of $\mathcal{P}$ that satisties the following conditions:

1. $0 \in \mathcal{P}^0$
2. $\forall p, q \in \mathcal{P}^0 \ (p + q \in \mathcal{P} \Longrightarrow p + q \in \mathcal{P}^0)$
3. $\forall p, q \in \mathcal{P}^0 \ (p|q \in \mathcal{P}^0)$
4. $\forall p \in \mathcal{P} \ \forall q \in \mathcal{P}^0 \left(\frac{p}{q} \in \mathcal{P}^0\right)$

A process in $\mathcal{P}$ should be either in $\mathcal{P}^+$ or in $\mathcal{P}^0$. However, the syntax of $\mathcal{P}$ allows processes to be defined for which this is not true. For example:

$$A \triangleq A \mid A \tag{4.6}$$

The definition of A is syntactically correct, but it is not a member of $\mathcal{P}^+$ or of $\mathcal{P}^0$. Therefore, we restrict $\mathcal{P}$ as folllows:

$$\mathcal{P} \triangleq \mathcal{P}^+ \cup \mathcal{P}^0 \tag{4.7}$$

Clearly, $\mathcal{P}^+$ and $\mathcal{P}^0$ should be disjoint, which suggests $\{\mathcal{P}^+, \mathcal{P}^0\}$ is a partition of $\mathcal{P}$. However, before this can be proved, the notion of strong of-bisimulation needs to be defined and a number of preliminary results need to be proved, for which $\mathcal{P}$ is restricted further.

Following convention, notice that:

1. Every $p \in \mathcal{P}$ is the result of one or more applications of the production rules of $\mathcal{P}^+$ or $\mathcal{P}^0$ with finite depth of inference.
2. Every transition of every $p \in \mathcal{P}$ is a result of one or more applications of the LTS semantic rules with finite depth of inference.

The first condition ensures every process in $\mathcal{P}$ can be represented in 'closed form', which facilitates modelling. The second condition ensures every transition of every process in $\mathcal{P}$ can be inferred after a finite number of steps, which facilitates modelling and analysis. Both conditions are used to prove properties of processes in $\mathcal{P}$ by finite induction.

4.2.3 Strong of-Bisimulation

Behavioural matching is a feature of CCS^{dp}, and its purpose is to maximize the terseness of models that can be reconfigured. Behavioural matching determines

a reconfiguration reaction between positive processes, and it is expressed using strong of-bisimulation ($\sim_{of}$), which is defined as follows.

For all p in $\mathcal{P}$, $\mathcal{T}_p$ is defined to be the set of actions in $\mathcal{T}$ that p can perform, where $\mathcal{T} \in \{\mathcal{N}, \overline{\mathcal{N}}, \mathcal{L}, \mathcal{I}, \mathcal{R}, \overline{\mathcal{R}}, \mathcal{C}, \mathcal{A}\}$.

S is defined to be a *strong observational and fractional simulation* (or equivalently, *strong of-simulation*) on $\mathcal{P}$ iff $S \subseteq \mathcal{P}$ x $\mathcal{P}$ and the following two conditions hold $\forall (p, q) \in S$:

$$\textit{Observation}: \quad \forall \alpha \in \mathcal{I}_p \; \forall p' \in \mathcal{P} \; (p \xrightarrow{\alpha} p' \implies \alpha \in \mathcal{I}_q \wedge \exists q' \in \mathcal{P} \; (q \xrightarrow{\alpha} q' \wedge (p', q') \in S))$$

$$\textit{Fraction}: \quad \forall \tau_{r_X} \in \mathcal{R}_p \; \forall p'' \in \mathcal{P} \; (p \xrightarrow{\tau_{r_X}} p'' \implies \tau_{r_X} \in \mathcal{R}_q \wedge \exists q'' \in \mathcal{P} \; (q \xrightarrow{\tau_{r_X}} q'' \wedge (p'', q'') \in S))$$

A process p is defined to be *strongly of-simulated by* process q (or equivalently, q *strongly of-simulates* p), written $p \rightsquigarrow_{of} q$, iff there exists a strong of-simulation S on $\mathcal{P}$ with $(p, q) \in S$.

The *Observation* condition of strong of-simulation is intended for processes that can behave like processes in *CCS*, and it is the same as the condition for strong simulation in *CCS*. It states that in order for q to simulate p, any input or output or τ action that p can perform to become p', must be also performable by q to become q', and q' must simulate p'.

The *Fraction* condition of strong of-simulation is intended for processes that can behave like fraction processes. It states that in order for q to simulate p, any reconfiguration action that p can perform to create p'', must be also performable by q to create q'', and q'' must simulate p''.

The two conditions of strong of-simulation are very similar, and (therefore) can be readily combined into a single condition. However, we prefer to keep them separate in order to show the difference between strong bisimulation in *CCS* and strong of-bisimulation in CCS^{dp} more clearly.

A strong of-simulation S on $\mathcal{P}$ is defined to be a *strong observational and fractional bisimulation* (or equivalently, *strong of-bisimulation*) on $\mathcal{P}$ iff both S and S^{-1} are strong of-simulations on $\mathcal{P}$.

Process p is defined to be *strongly observationally and fractionally bisimilar* to process q (or equivalently, p is *strongly of-bisimilar* to q), written $p \sim_{of} q$, iff there exists a strong of-bisimulation S on $\mathcal{P}$ with $(p, q) \in S$.

Following convention, we represent the largest strong of-bisimulation on $\mathcal{P}$ by $\sim_{of}$, where $\sim_{of} \triangleq \bigcup\{S \mid S \text{ is a strong of-bisimulation on } \mathcal{P}\}$. Notice that $\sim_{of}$ is non-empty ($\because (0,0) \in \sim_{of}$, by Lemma 4.3.1).

4.2.4 Structure of the LTS Semantics

Following convention [Mil99], the LTS semantics is represented as the mathematical structure $(\mathcal{P}, \mathcal{T})$, where $\mathcal{P}$ is the set of processes (or equivalently, states) in CCS^{dp}, and $\mathcal{T}$ is the set of transitions of the processes in CCS^{dp}.

The *CCS*-type transitions in CCS^{dp}, such as the transitions defined by the *Sum* rule, have the structure $\mathcal{P} \times \mathcal{I} \times \mathcal{P}$.

The reconfiguration transitions in CCS^{dp}, such as the transitions by the *Creat* and *Delet* rules, have the structure $(\mathcal{P} \times \mathcal{P}) \times \mathcal{P} \times \mathcal{C} \times \mathcal{P}$, because $\sim_{of}$ is necessary in order to determine the transitions in $\mathcal{C}$ that a given process p in $\mathcal{P}$ can perform.

Combining the structures of the *CCS*-type transitions and the reconfiguration transitions gives the structure of the transitions in CCS^{dp}.

$\therefore \mathcal{T} = (\mathcal{P} \cup (\mathcal{P} \times \mathcal{P}) \times \mathcal{P}) \times \mathcal{A} \times \mathcal{P}$
$\implies (\mathcal{P}, \mathcal{T}) = (\mathcal{P}, (\mathcal{P} \cup (\mathcal{P} \times \mathcal{P}) \times \mathcal{P}) \times \mathcal{A} \times \mathcal{P})$

Creat is the basic rule that determines the reconfiguration transitions in $\mathcal{R}$; the hypothesis of *Creat* depends on $\sim_{of}$; and $\sim_{of}$ is defined in terms of the transitions in $\mathcal{I} \cup \mathcal{R}$. This suggests there is an inductive relationship between the transitions in $\mathcal{I} \cup \mathcal{R}$ and $\sim_{of}$. In fact, it is clear from the *Creat* rule that this inductive relationship is based on the depth of fractional recursion of the denominator of a process. Induction is a powerful technique for proving properties of recursive processes and relationships between recursive processes, such as bisimilarity. However, proof of bisimilarity typically requires the inductive hypothesis to apply to all the processes to which a given process can evolve. Therefore, in order to facilitate the use of induction to prove strong of-bisimilarity between processes, such as $p|q \sim_{of} q|p$, we bound the depth of fractional recursion of the denominators of

fractions and their successors, using the following definitions:

$succ : \mathcal{P} \times \mathbb{N} \longrightarrow \mathbb{P}\,\mathcal{P}$ such that

$$succ(p,i) \triangleq \begin{cases} \{p\} & \text{if } i = 0 \\ \{q' \in \mathcal{P} \mid \exists q \in succ(p,i-1)\ (\exists \mu \in \mathcal{I}_q \cup \mathcal{R}_q\ (q \xrightarrow{\mu} q'))\} & \text{else} \end{cases}$$

$succ(p,i)$ is the set of i^{th} *successor processes* (or equivalently, *successors*) of p. That is, the set of processes reached after i consecutive transitions in $\mathcal{I} \cup \mathcal{R}$ starting from p, with $succ(p,0) = \{p\}$.

For example, let $p \triangleq a.0 \mid \frac{\overline{a}.0}{a.0}$
then $succ(p,0) = \{p\} = \{a.0 \mid \frac{\overline{a}.0}{a.0}\}$
$\Longrightarrow succ(p,1) = \{0 \mid \frac{\overline{a}.0}{a.0},\ a.0 \mid \overline{a}.0,\ \overline{a}.0\}$
$\Longrightarrow succ(p,2) = \{0 \mid \overline{a}.0,\ a.0 \mid 0,\ 0 \mid 0,\ 0\}$
$\Longrightarrow succ(p,3) = \{0 \mid 0\}$
$\Longrightarrow succ(p,4) = \emptyset$

$successors : \mathcal{P} \longrightarrow \mathbb{P}\,\mathcal{P}$ such that $successors(p) \triangleq \bigcup_{i \in \mathbb{N}} succ(p,i)$

$successors(p)$ is the set of all the successors of p, including p. That is, the set of all the processes reached after zero, one or more consecutive transitions in $\mathcal{I} \cup \mathcal{R}$ starting from p.

For example, $p \triangleq a.0 \mid \frac{\overline{a}.0}{a.0}$
$\Longrightarrow successors(p) = succ(p,0) \cup succ(p,1) \cup succ(p,2) \cup succ(p,3)$
$\Longrightarrow successors(p) = \{a.0 \mid \frac{\overline{a}.0}{a.0},\ 0 \mid \frac{\overline{a}.0}{a.0},\ a.0 \mid \overline{a}.0,\ \overline{a}.0,\ 0 \mid \overline{a}.0,\ a.0 \mid 0,\ 0 \mid 0,\ 0\}$

$factors_m : \mathcal{P} \longrightarrow \mathbb{P}_m\,\mathcal{P}$ such that

$$factors_m(p) \triangleq \begin{cases} \{p_1\}_m \uplus \{p_2\}_m \uplus factors_m(p_1) \uplus factors_m(p_2) & \text{if } p = p_1 | p_2 \\ \emptyset_m & \text{else} \end{cases}$$

where $\mathbb{P}_m$ is the power multiset, which is the set of all the multisets of $\mathcal{P}$, $\{p_1\}_m$ and $\{p_2\}_m$ are the multisets consisting of the processes p_1 and p_2 (respectively), $\uplus$ is the multiset union operation, and $\emptyset_m$ is the empty multiset.

The elements of $factors_m(p)$ are termed the *factors* of p.
$\therefore$ The factors of $a.0 \mid \frac{\overline{a}.0}{a.0}$ are $a.0$ and $\frac{\overline{a}.0}{a.0}$.
p is termed a *singleton process* (or equivalently, a *singleton*) iff $factors_m(p) = \emptyset_m$.
$\therefore a.0 + b.\frac{\overline{a}.0}{a.0}$ is a singleton.

$sfdrdepth : \mathcal{P} \longrightarrow \mathbb{N}$ such that
$sfdrdepth(p) \triangleq max\{fdrdepth(s) \mid s \in successors(p)\}$ where

$fdrdepth : \mathcal{P} \longrightarrow \mathbb{N}$ such that

$$fdrdepth(s) \triangleq \begin{cases} 0 & \text{if } \mathcal{R}_s = \emptyset \\ 1 + max\{sfdrdepth(X) \mid \tau_{r_X} \in \mathcal{R}_s\} & \text{else} \end{cases}$$

$fdrdepth(p)$ is the fractional denominator recursion depth of p. If p is a fraction process, $fdrdepth(p)$ is determined from the maximum of the depths of fractional recursion of the denominator of p and of the successors of the denominator. If p has one or more fraction factors, $fdrdepth(p)$ is determined from the maximum of the depths of fractional recursion of the denominators of the fraction factors of p and of the successors of the denominators. If p is not a fraction process and does not have a fraction factor, then $fdrdepth(p) = 0$.

$sfdrdepth(p)$ is the successors' fractional denominator recursion depth of p. That is, the maximum value of the $fdrdepth$ function taken over all the successors of p, including p. If p and its successors are *CCS* processes, then $sfdrdepth(p) = 0$.

For example, $p \triangleq a.0|\frac{\bar{a}.0}{a.0}$
$\Longrightarrow fdrdepth(p) = 1 \ \wedge \ sfdrdepth(p) = max\{1,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0\} = 1$

We restrict $\mathcal{P}$ to the domain of $sfdrdepth$. Thus, for each process p in $\mathcal{P}$, all the successors of p have a uniform upper bound (determined by p) on their depths of fractional recursion. This restriction facilitates the use of induction to prove strong of-bisimilarities between process expressions with a fraction factor.

There is a close relationship between $\sim_{of}$ and $sfdrdepth$, which is proved in Theorem 4.2.1 using the following lemma and its corollary: Lemma 4.2.1 states that if process p is strongly of-simulated by process q, then each i^{th} successor of p is strongly of-simulated by some i^{th} successor of q. Corollary 4.2.1 states that if process p is strongly of-simulated by process q, then $sfdrdepth(p) \leq sfdrdepth(q))$. Theorem 4.2.1 states that if process p is strongly of-bisimilar to process q, then the $sfdrdepth$ values of p and q are the same.

Lemma 4.2.1 $\forall p, q \in \mathcal{P}$
$(p \rightsquigarrow_{of} q \Longrightarrow \forall i \in \mathbb{N}\ \forall p' \in succ(p, i)\ (\exists q' \in succ(q, i)\ (p' \rightsquigarrow_{of} q')))$.
Proof: See Section A.1 in Appendix A.

Corollary 4.2.1 $\forall p,q\in\mathcal{P}\ (p\rightsquigarrow_{of} q \implies sfdrdepth(p) \leq sfdrdepth(q))$.
Proof: See Section A.2 in Appendix A.

Theorem 4.2.1 $\forall p,q\in\mathcal{P}\ (p\sim_{of} q \implies sfdrdepth(p) = sfdrdepth(q))$.

Proof: Suppose $p,q\in\mathcal{P}\ (p\sim_{of} q)$
then $\exists$ strong of-bisimulation S on $\mathcal{P}$ with $(p,q)\in S$ (by definition of $p\sim_{of} q$)
$\implies S$ is a strong of-simulation on $\mathcal{P}$ with $(p,q)\in S\ \ \wedge$
S^{-1} is a strong of-simulation on $\mathcal{P}$ with $(q,p)\in S^{-1}$
(by definition of strong of-bisimulation on $\mathcal{P}$)
$\implies p\rightsquigarrow_{of} q$ (by definition of $p\rightsquigarrow_{of} q$) $\ \wedge\ q\rightsquigarrow_{of} p$ (by definition of $q\rightsquigarrow_{of} p$)
$\implies sfdrdepth(p) \leq sfdrdepth(q)\ \wedge\ sfdrdepth(q) \leq sfdrdepth(p)$
(by Corollary 4.2.1)
$\implies sfdrdepth(p) = sfdrdepth(q)$ (by arithmetic).
$\therefore \forall p,q\in\mathcal{P}\ (p\sim_{of} q \implies sfdrdepth(p) = sfdrdepth(q))$ ($\because p,q\in\mathcal{P}$ are arbitrary). Q.E.D.
■

Theorem 4.2.1 enables $\mathcal{P}$ to be partially ordered into layers using the $sfdrdepth$ value of processes. This layering of $\mathcal{P}$ results in the following inductive relationships (expressed as proof obligations) between the transitions in $\mathcal{I}\cup\mathcal{R}$ and $\sim_{of}$:

1. CCS LTS rules $\ \vdash\ \longrightarrow_0$
2. $\forall n\in\mathbb{N}\ (\longrightarrow_{\leq n}\ \vdash\ \sim_{of_{\leq n}})$
3. $\forall n\in\mathbb{N}\ (\text{LTS rules}\wedge\sim_{of_{\leq n}}\ \vdash\ \longrightarrow_{\leq n+1})$

where $\longrightarrow_0$ is the smallest relation on $\mathcal{P}$ between processes with $sfdrdepth$ value 0 that satisfies the LTS rules, $\longrightarrow_{\leq n}$ is the smallest relation on $\mathcal{P}$ between processes with $sfdrdepth$ value $\leq n$ that satisfies the LTS rules, $\sim_{of_{\leq n}}$ is the largest strong of-bisimulation on $\mathcal{P}$ that contains only processes with $sfdrdepth$ value $\leq n$, and $\longrightarrow_{\leq n+1}$ is the smallest relation on $\mathcal{P}$ between processes with $sfdrdepth$ value $\leq n+1$ that satisfies the LTS rules.

Therefore, $\sim_{of} = \lim_{n\to\infty}\sim_{of_{\leq n}}$, which is the largest strong of-bisimulation on $\mathcal{P}$ under the restriction $\mathcal{P} = dom(sfdrdepth)$.

4.3 Equational Reasoning

Equational reasoning is the conventional form of reasoning in a process algebra, and is usually done using a bisimulation that is a *process congruence*. Process congruence in basic CCS^{dp} is defined as follows.

Definition 4.3.1 *A process context K of $\mathcal{P}$ is a process expression that contains a process variable, denoted by $[\cdot]$, and has the following syntax:*

$$K[\cdot] ::= [\cdot] \mid \alpha.K[\cdot] + M \mid K[\cdot]|P \mid P|K[\cdot] \mid \frac{K[\cdot]}{P} \mid \frac{P}{K[\cdot]}$$

where $\alpha \in \mathcal{I}$, M is any summation in $\mathcal{P}$, and P is any process in $\mathcal{P}$.

The process variable $[\cdot]$ is usually referred to as a 'hole'. The result of replacing the hole (literally) with a process p in $\mathcal{P}$ is denoted by $K[p]$. The contexts $\alpha.[\cdot] + M$, $[\cdot]|P$, $P|[\cdot]$, $\frac{[\cdot]}{P}$ and $\frac{P}{[\cdot]}$ are termed the *elementary process contexts* of $\mathcal{P}$, and are used to define process congruence.

Definition 4.3.2 *$\cong$ is a process congruence on $\mathcal{P}$ iff the following conditions hold:*

1. *$\cong$ is an equivalence relation on $\mathcal{P}$*
2. *$\forall p, q \in \mathcal{P}$ if $p \cong q$ then the following conditions hold:*
   (a) $\forall \alpha \in \mathcal{I}\ (\alpha.p + M \cong \alpha.q + M)$ *where M is any summation in $\mathcal{P}$*
   (b) $\forall r \in \mathcal{P}\ (p|r \cong q|r)$
   (c) $\forall r \in \mathcal{P}\ (r|p \cong r|q)$
   (d) $\forall r \in \mathcal{P}\ (\frac{p}{r} \cong \frac{q}{r})$
   (e) $\forall r \in \mathcal{P}\ (\frac{r}{p} \cong \frac{r}{q})$

Thus, $\cong$ is a process congruence on $\mathcal{P}$ iff $\cong$ is an equivalence relation on $\mathcal{P}$ that is preserved by the elementary process contexts of $\mathcal{P}$.

Following convention, the terms *process context*, *elementary process context* and *process congruence* are equivalent to the terms *context*, *elementary context* and *congruence* (respectively), unless otherwise stated.

4.3.1 Strong of-Bisimulation is an Equivalence Relation

Strong of-bisimulation has a number of properties which are useful for equational reasoning, and which we now prove.

Lemma 4.3.1 *$\sim_{of}$ is reflexive on $\mathcal{P}$.*

Proof: $\sim_{of}$ is reflexive on $\mathcal{P}$ iff $\forall p \in \mathcal{P}\ (p \sim_{of} p)$ (by definition of reflexivity).
If $\exists$ strong of-bisimulation S on $\mathcal{P}$ with $\forall p \in \mathcal{P}\ ((p, p) \in S)$
then $\forall p \in \mathcal{P}\ (p \sim_{of} p)$ (by definition of $p \sim_{of} p$).
Therefore, we find such an S.

Let $S \triangleq \{(p,p) \mid p \in \mathcal{P}\}$. Thus, S is the identity function on $\mathcal{P}$.

$\forall p \in \mathcal{P}\,((p,p) \in S)$ (by definition of S).

S is a strong of-bisimulation on $\mathcal{P}$
$\iff S, S^{-1}$ are strong of-simulations on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$)
$\iff S, S^{-1}$ are binary relations on $\mathcal{P}$ $\wedge$
for all elements of S, S^{-1} the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied
(by definition of strong of-simulation on $\mathcal{P}$).
We show S is a strong of-simulation on $\mathcal{P}$ by proving S is a binary relation on $\mathcal{P}$ and for all elements of S the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied, then prove S^{-1} is a strong of-simulation on $\mathcal{P}$.

$S \subseteq \mathcal{P}$ x $\mathcal{P}$ (by definition of S).

Verifying the *Observation* condition of strong of-simulation on $\mathcal{P}$ for $(p,p) \in S$:
For $\alpha \in \mathcal{I}_p$ and $p' \in \mathcal{P}$,
if $p \xrightarrow{\alpha} p'$
(by the hypothesis of the *Observation* condition of strong of-simulation on $\mathcal{P}$)
then $\alpha \in \mathcal{I}_p$ (by definition of α) $\wedge$ $p' \in \mathcal{P}$ (by definition of p') $\wedge$ $p \xrightarrow{\alpha} p'$ $\wedge$
$(p',p') \in S$ (by definition of S)
$\implies \forall \alpha \in \mathcal{I}_p\ \forall p' \in \mathcal{P}\ (p \xrightarrow{\alpha} p' \implies \alpha \in \mathcal{I}_p\ \wedge\ \exists p' \in \mathcal{P}\ (p \xrightarrow{\alpha} p'\ \wedge\ (p',p') \in S))$
($\because \alpha \in \mathcal{I}_p$ and $p' \in \mathcal{P}$ are arbitrary)
$\implies$ the *Observation* condition of strong of-simulation on $\mathcal{P}$ holds for $(p,p) \in S$
(by definition of the *Observation* condition of strong of-simulation on $\mathcal{P}$).

Verifying the *Fraction* condition of strong of-simulation on $\mathcal{P}$ for $(p,p) \in S$:
For $\tau_{r_X} \in \mathcal{R}_p$ and $p'' \in \mathcal{P}$,
if $p \xrightarrow{\tau_{r_X}} p''$
(by the hypothesis of the *Fraction* condition of strong of-simulation on $\mathcal{P}$)
then $\tau_{r_X} \in \mathcal{R}_p$ (by definition of τ_{r_X}) $\wedge$ $p'' \in \mathcal{P}$ (by definition of p'') $\wedge$ $p \xrightarrow{\tau_{r_X}} p''$ $\wedge$
$(p'',p'') \in S$ (by definition of S)
$\implies \forall \tau_{r_X} \in \mathcal{R}_p\ \forall p'' \in \mathcal{P}\ (p \xrightarrow{\tau_{r_X}} p'' \implies \tau_{r_X} \in \mathcal{R}_p\ \wedge\ \exists p'' \in \mathcal{P}\ (p \xrightarrow{\tau_{r_X}} p''\ \wedge\ (p'',p'') \in S))$
($\because \tau_{r_X} \in \mathcal{R}_p$ and $p'' \in \mathcal{P}$ are arbitrary)
$\implies$ the *Fraction* condition of strong of-simulation on $\mathcal{P}$ holds for $(p,p) \in S$
(by definition of the *Fraction* condition of strong of-simulation on $\mathcal{P}$).

$\therefore S$ is a strong of-simulation on $\mathcal{P}$
(by definition of strong of-simulation on $\mathcal{P}$, $\because (p,p) \in S$ is arbitrary)
$\implies S$ and S^{-1} are strong of-simulations on $\mathcal{P}$ ($\because S^{-1} = S$)
$\implies S$ is a strong of-bisimulation on $\mathcal{P}$

(by definition of strong of-bisimulation on $\mathcal{P}$)
$\Longrightarrow \forall p \in \mathcal{P}\ (p \sim_{of} p)$ (by definitions of $p \sim_{of} p$ and S)
$\Longrightarrow \sim_{of}$ is reflexive on $\mathcal{P}$ (by definition of reflexivity). Q.E.D. ■

Lemma 4.3.2 *$\sim_{of}$ is symmetric on $\mathcal{P}$.*

Proof: $\sim_{of}$ is symmetric on $\mathcal{P}$ iff $\forall (p,q) \in \mathcal{P} \times \mathcal{P}\ (p \sim_{of} q \Longrightarrow q \sim_{of} p)$
(by definition of symmetry).

$p \sim_{of} q \implies \exists$ strong of-bisimulation S on $\mathcal{P} \wedge (p,q) \in S$
(by definition of $p \sim_{of} q$)
$\implies \exists$ strong of-simulations S, S^{-1} on $\mathcal{P} \wedge (p,q) \in S \wedge (q,p) \in S^{-1}$
(by definitions of strong of-bisimulation on $\mathcal{P}$ and inverse relations)
$\implies \exists$ strong of-bisimulation S^{-1} on $\mathcal{P} \wedge (q,p) \in S^{-1}$
($\because (S^{-1})^{-1} = S$ and by definition of strong of-bisimulation on $\mathcal{P}$)
$\implies q \sim_{of} p$
(by definition of $q \sim_{of} p$).

$\therefore \forall (p,q) \in \mathcal{P} \times \mathcal{P}\ (p \sim_{of} q \Longrightarrow q \sim_{of} p)$ ($\because p, q \in \mathcal{P}$ are arbitrary)
$\Longrightarrow \sim_{of}$ is symmetric on $\mathcal{P}$ (by definition of symmetry). Q.E.D. ■

Transitivity is an important property of simulations as well as bisimulations. For example, in order to prove $p|q \sim_{of} q|p$ (in Theorem 4.3.11), we need the transitivity of strong dp-simulation on $\mathcal{P}$ (defined in Section 4.3.3), which we prove using the transitivity of strong of-simulation on $\mathcal{P}$ (see Corollary 4.3.1 below). Lemma 4.3.3 is the basic proposition, which states that the composition of two strong of-simulations on $\mathcal{P}$ is a strong of-simulation on $\mathcal{P}$. Corollary 4.3.1 states that $\rightsquigarrow_{of}$ is transitive, Corollary 4.3.2 states that the composition of two strong of-bisimulations on $\mathcal{P}$ is a strong of-bisimulation on $\mathcal{P}$, and the third corollary (Lemma 4.3.4) states that $\sim_{of}$ is transitive.

Lemma 4.3.3 *$\forall$strong of-simulations U, V on $\mathcal{P}$ (UV is a strong of-simulation on $\mathcal{P}$).*
Proof: See Section A.3 in Appendix A.

Corollary 4.3.1 $\forall p, q, r \in \mathcal{P}\ (p \rightsquigarrow_{of} q \wedge q \rightsquigarrow_{of} r \Longrightarrow p \rightsquigarrow_{of} r)$.

Proof: Suppose $p, q, r \in \mathcal{P}\ (p \rightsquigarrow_{of} q \wedge q \rightsquigarrow_{of} r)$
then $\exists$ strong of-simulation U on $\mathcal{P}$ $((p,q) \in U)$ (by definition of $p \rightsquigarrow_{of} q$) $\wedge$
$\exists$ strong of-simulation V on $\mathcal{P}$ $((q,r) \in V)$ (by definition of $q \rightsquigarrow_{of} r$)
$\Longrightarrow UV$ is a strong of-simulation on $\mathcal{P}$ (by Lemma 4.3.3) $\wedge$
$(p,r) \in UV$ (by composition of binary relations)
$\Longrightarrow p \rightsquigarrow_{of} r$ (by definition of $p \rightsquigarrow_{of} r$).

$\therefore \forall p,q,r \in \mathcal{P}\ (p \rightsquigarrow_{of} q\ \wedge\ q \rightsquigarrow_{of} r \implies p \rightsquigarrow_{of} r)$
(because $\implies$ is transitive and $p,q,r \in \mathcal{P}$ with $p \rightsquigarrow_{of} q\ \wedge\ q \rightsquigarrow_{of} r$ are arbitrary). Q.E.D. ■

Corollary 4.3.2 *$\forall$strong of-bisimulations U, V on $\mathcal{P}$ (UV is a strong of-bisimulation on $\mathcal{P}$).*

Proof: Suppose U, V are strong of-bisimulations on $\mathcal{P}$
then U, U^{-1}, V, V^{-1} are strong of-simulations on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$)
$\implies UV, V^{-1}U^{-1}$ are strong of-simulations on $\mathcal{P}$ (by Lemma 4.3.3)
$\implies UV, (UV)^{-1}$ are strong of-simulations on $\mathcal{P}$
($\because (UV)^{-1} = V^{-1}U^{-1}$, by algebra of binary relations)
$\implies UV$ is a strong of-bisimulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$).
$\therefore \forall$strong of-bisimulations U, V on $\mathcal{P}$ (UV is a strong of-bisimulation on $\mathcal{P}$)
($\because U, V$ are arbitrary strong of-bisimulations on $\mathcal{P}$). Q.E.D. ■

Lemma 4.3.4 *$\sim_{of}$ is transitive on $\mathcal{P}$.*

Proof: $\sim_{of}$ is transitive on $\mathcal{P}$ iff $\forall p,q,r \in \mathcal{P}\ (p \sim_{of} q\ \wedge\ q \sim_{of} r \implies p \sim_{of} r)$
(by definition of transitivity).

Suppose $p,q,r \in \mathcal{P}\ (p \sim_{of} q\ \wedge\ q \sim_{of} r)$
then $\exists$ strong of-bisimulation U on $\mathcal{P}$ $((p,q) \in U)$ (by definition of $p \sim_{of} q$) $\wedge$
$\exists$ strong of-bisimulation V on $\mathcal{P}$ $((q,r) \in V)$ (by definition of $q \sim_{of} r$)
$\implies UV$ is a strong of-bisimulation on $\mathcal{P}$ (by Corollary 4.3.2) $\wedge$
$(p,r) \in UV$ (by composition of binary relations)
$\implies p \sim_{of} r$ (by definition of $p \sim_{of} r$).
$\therefore \forall p,q,r \in \mathcal{P}\ (p \sim_{of} q\ \wedge\ q \sim_{of} r \implies p \sim_{of} r)$
(because $\implies$ is transitive and $p,q,r \in \mathcal{P}$ with $p \sim_{of} q\ \wedge\ q \sim_{of} r$ are arbitrary)
$\implies \sim_{of}$ is transitive on $\mathcal{P}$ (by definition of transitivity). Q.E.D. ■

Theorem 4.3.1 *$\sim_{of}$ is an equivalence relation on $\mathcal{P}$.*

Proof: $\sim_{of}$ is an equivalence relation on $\mathcal{P}$ iff $\sim_{of}$ is reflexive, symmetric and transitive on $\mathcal{P}$
(by definition of equivalence relation).

$\sim_{of}$ is reflexive on $\mathcal{P}$ (by Lemma 4.3.1) $\wedge$
$\sim_{of}$ is symmetric on $\mathcal{P}$ (by Lemma 4.3.2) $\wedge$
$\sim_{of}$ is transitive on $\mathcal{P}$ (by Lemma 4.3.4)
$\implies \sim_{of}$ is an equivalence relation on $\mathcal{P}$ (by definition of equivalence relation).
Q.E.D. ■

Theorem 4.3.1 is important for a number of reasons. First, equivalence is necessary for process congruence, which enables equational reasoning about processes. Second, equivalence helps to group matching processes into an equivalence class, and thereby helps to identify the reconfiguration transitions of a process. Third, equivalence helps to prove Theorem 4.3.2, which states that $\{\mathcal{P}^+, \mathcal{P}^0\}$ is a partition of $\mathcal{P}$. Theorem 4.3.2 is proved using the following two lemmas: Lemma 4.3.5 states that every positive process has a transition in $\mathcal{I} \cup \mathcal{R}$, and Lemma 4.3.7 states that no zero process has a transition in $\mathcal{I} \cup \mathcal{R}$. Thus, the two lemmas help to prove that $\mathcal{P}^+$ and $\mathcal{P}^0$ are disjoint. The proof of Lemma 4.3.7 requires Lemma 4.3.6, which states that a process is positive if and only if the process can perform a delete transition (i.e. a transition in $\overline{\mathcal{R}}$).

Lemma 4.3.5 $\forall p \in \mathcal{P}^+ (\mathcal{I}_p \cup \mathcal{R}_p \neq \emptyset)$
Proof: See Section A.4 in Appendix A.

The proof of Lemma 4.3.7 requires the notion of a *factor tree* of a process, which is a context of the factors of the process, and is defined as follows:

Definition 4.3.3 *The factor tree of a process $p \in \mathcal{P}$ is the binary tree of processes rooted on p such that the immediate subnodes of a node are the processes used to produce the node process using a parallel composition production rule 3 of $\mathcal{P}$.*

Thus, in a factor tree, if a node process p has subnodes, then $p \in \mathcal{P}^+$ is produced using the $\mathcal{P}^+$ production rule 3, and $p \in \mathcal{P}^0$ is produced using the $\mathcal{P}^0$ production rule 3. Every process $p \in \mathcal{P}$ has a factor tree, because p is the root of its factor tree.

For example, let $p \triangleq (a.A \mid \frac{b.0}{a.A}) \mid (\overline{a}.0 + 0 \mid 0)$. The factor tree of p is shown in Figure 4.1.

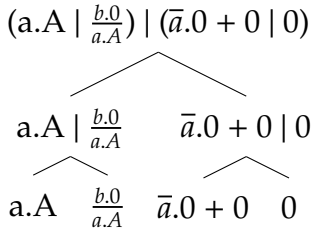


Figure 4.1: Factor Tree of a Process.

Definition 4.3.4 *The depth of a singly-rooted tree is the number of nodes in the longest path with distinct nodes that connects the root node to a leaf node of the tree.*

The depth of the factor tree of p is finite, because every process $p \in \mathcal{P}$ is the result of applications of the $\mathcal{P}$ production rules with finite depth of inference, and the factor tree of p is obtained by pruning this inference tree of processes that produces p. The pruning of the inference tree of p is done by starting from the root node p, working down each branch (if any) until a subnode is reached that is **neither** produced by production rule 3 of $\mathcal{P}^+$ **nor** by production rule 3 of $\mathcal{P}^0$, and cutting the branches (if any) below the subnode. These subnodes (if any) are the leaf nodes of the factor tree of p; otherwise, p has no subnode, and the factor tree of p is p. Therefore, the nodes of the factor tree of p are the processes in $\{p\}_m \uplus factors_m(p)$ (by definition of $factors_m(p)$). Thus, the factor tree of p describes the syntactic relationship between a factor of p (if any) and its context p.

The inference tree that produces $p \in \mathcal{P}$ is also the parse tree of p, because the production rules of $\mathcal{P}^+$ and $\mathcal{P}^0$ that produce p are inference rules and are based on the process syntax of basic CCS^{dp}. Therefore, the factor tree of p is the parse tree of p with either p or the singleton factors of p as its leaf node(s).

Lemma 4.3.6 $\forall p \in \mathcal{P}(p \in \mathcal{P}^+ \iff \overline{\mathcal{R}}_p \neq \emptyset)$
Proof: See Section A.5 in Appendix A.

Lemma 4.3.7 $\forall p \in \mathcal{P}^0(\mathcal{I}_p \cup \mathcal{R}_p = \emptyset)$
Proof: See Section A.6 in Appendix A.

Theorem 4.3.2 $\{\mathcal{P}^+, \mathcal{P}^0\}$ *is a partition of* $\mathcal{P}$

Proof: $\{\mathcal{P}^+, \mathcal{P}^0\}$ is a partition of $\mathcal{P}$ iff the following conditions hold
(by definition of set partition):

1. $\mathcal{P}^+, \mathcal{P}^0$ are non-empty subsets of $\mathcal{P}$
2. $\mathcal{P}^+ \cap \mathcal{P}^0 = \emptyset$
3. $\mathcal{P}^+ \cup \mathcal{P}^0 = \mathcal{P}$

Therefore, we prove $\mathcal{P}^+, \mathcal{P}^0$ are non-empty subsets of $\mathcal{P}$, $\mathcal{P}^+ \cap \mathcal{P}^0 = \emptyset$ and $\mathcal{P}^+ \cup \mathcal{P}^0 = \mathcal{P}$.

Case 1. $\mathcal{P}^+, \mathcal{P}^0$ are non-empty subsets of $\mathcal{P}$

Proof: $0 \in \mathcal{P}^0$ (by production rule 1 of $\mathcal{P}^0$)
$\Longrightarrow 0 \in \mathcal{P}^+ \cup \mathcal{P}^0$ (by set theory)
$\Longrightarrow 0 \in \mathcal{P}$ (by definition of $\mathcal{P}$)
$\Longrightarrow \tau.0 \in \mathcal{P}^+$ (by production rule 1 of $\mathcal{P}^+$, $\because \tau \in \mathcal{I}$)
$\Longrightarrow \mathcal{P}^+$ is non-empty (by set theory).
And $\mathcal{P}^+ \subseteq \mathcal{P}^+ \cup \mathcal{P}^0$ (by set theory)

$\Longrightarrow \mathcal{P}^+ \subseteq \mathcal{P}$ (by definition of $\mathcal{P}$)
$\Longrightarrow \mathcal{P}^+$ is a non-empty subset of $\mathcal{P}$ ($\because \mathcal{P}^+$ is non-empty).

$0 \in \mathcal{P}^0$ (by production rule 1 of $\mathcal{P}^0$)
$\Longrightarrow \mathcal{P}^0$ is non-empty (by set theory).
And $\mathcal{P}^0 \subseteq \mathcal{P}^+ \cup \mathcal{P}^0$ (by set theory)
$\Longrightarrow \mathcal{P}^0 \subseteq \mathcal{P}$ (by definition of $\mathcal{P}$)
$\Longrightarrow \mathcal{P}^0$ is a non-empty subset of $\mathcal{P}$ ($\because \mathcal{P}^0$ is non-empty). Q.E.D.

Case 2. $\mathcal{P}^+ \cap \mathcal{P}^0 = \emptyset$

Proof: If $\mathcal{P}^+ \cap \mathcal{P}^0 \neq \emptyset$
then $\exists p \in \mathcal{P}^+ \cap \mathcal{P}^0$ (by set theory)
$\Longrightarrow p \in \mathcal{P}^+ \wedge p \in \mathcal{P}^0$ (by set theory)
$\Longrightarrow \mathcal{I}_p \cup \mathcal{R}_p \neq \emptyset$ (by Lemma 4.3.5) $\wedge$ $\mathcal{I}_p \cup \mathcal{R}_p = \emptyset$ (by Lemma 4.3.7)
$\Longrightarrow$ *false*.
$\therefore \mathcal{P}^+ \cap \mathcal{P}^0 = \emptyset$ (by contradiction). Q.E.D.

Case 3. $\mathcal{P}^+ \cup \mathcal{P}^0 = \mathcal{P}$

Proof: $\mathcal{P}^+ \cup \mathcal{P}^0 = \mathcal{P}$ (by definition of $\mathcal{P}$). Q.E.D.

$\therefore \{\mathcal{P}^+, \mathcal{P}^0\}$ is a partition of $\mathcal{P}$ (by definition of set partition). Q.E.D. ■

The distinction between positive processes and zero processes was made in order to retain in CCS^{dp} the identity property of 0 with respect to parallel composition in the equivalences and congruences of *CCS*. Theorem 4.3.3 states that all zero processes have the required identity property with respect to parallel composition and strong of-bisimulation. The theorem is proved using the following two lemmas: Lemma 4.3.7 states that no zero process has a transition in $\mathcal{I} \cup \mathcal{R}$, and Lemma 4.3.8 states that no zero process has a transition in $\overline{\mathcal{R}}$. Together, the two lemmas imply that no zero process has a transition.

Lemma 4.3.8 $\forall p \in \mathcal{P}^0 (\overline{\mathcal{R}}_p = \emptyset)$

Proof: If this lemma is *false*
then $\exists p \in \mathcal{P}^0 (\overline{\mathcal{R}}_p \neq \emptyset)$ (by definitions of $\neg$ and $\forall$)
$\Longrightarrow p \in \mathcal{P}$ (by set theory and definition of $\mathcal{P}$) $\wedge$ $\overline{\mathcal{R}}_p \neq \emptyset$
$\Longrightarrow p \in \mathcal{P}^+$ (by Lemma 4.3.6)
$\Longrightarrow p \notin \mathcal{P}^0$ (by Theorem 4.3.2; which is a contradiction).
$\therefore$ This lemma is *true*
($\because$ the LTS semantics of basic CCS^{dp} is consistent and decidable (see Section 4.4)).
Q.E.D. ■

Theorem 4.3.3 $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{of} p \ \wedge \ p \sim_{of} z|p)$

Proof: See Section A.7 in Appendix A. The proof technique is to prove the conjuncts separately by discharging the following two proof obligations:
$\vdash \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{of} p)$ and $\vdash \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p \sim_{of} z|p)$. Each proof obligation is discharged by producing a witness value, that is, a binary relation on $\mathcal{P}$ which contains the pair of processes that are required to be strongly of-bisimilar, showing the relation is a strong of-simulation on $\mathcal{P}$, then showing the inverse relation is a strong of-simulation on $\mathcal{P}$. ■

4.3.2 Strong of-Bisimulation is not a Process Congruence

Theorem 4.3.1 states that strong of-bisimulation is an equivalence relation on $\mathcal{P}$, and Theorem 4.3.4 states that strong of-bisimulation preserves the elementary contexts $\alpha.[\cdot] + M$, $\frac{[\cdot]}{P}$ and $\frac{P}{[\cdot]}$ that correspond to the congruence conditions 2(a), 2(d) and 2(e) (respectively). The proof of Theorem 4.3.4 is simplified by Lemma 4.3.9, which states that two strongly of-bisimilar processes are either both positive or both zero processes. However, strong of-bisimulation does **not** satisfy the congruence conditions 2(b) and 2(c), as shown by the two examples below.

Lemma 4.3.9 $\forall p, q \in \mathcal{P}\ (p \sim_{of} q \implies p \in \mathcal{P}^+ \wedge q \in \mathcal{P}^+ \ \vee \ p \in \mathcal{P}^0 \wedge q \in \mathcal{P}^0)$

Proof: Suppose $p, q \in \mathcal{P}\ (p \sim_{of} q)$
then $p \in \mathcal{P}^+ \ \vee \ p \notin \mathcal{P}^+$ (by definition of $\mathcal{P}$ and set theory).
If $p \in \mathcal{P}^+$
then $\mathcal{I}_p \cup \mathcal{R}_p \neq \emptyset$ (by Lemma 4.3.5)
$\implies \exists \alpha \in \mathcal{I}_p \cup \mathcal{R}_p\ \exists p' \in \mathcal{P}\ (p \xrightarrow{\alpha} p')$ (by definitions of $\mathcal{I}_p$ and $\mathcal{R}_p$)
$\implies \alpha \in \mathcal{I}_q \cup \mathcal{R}_q \wedge \ \exists q' \in \mathcal{P}\ (q \xrightarrow{\alpha} q')$ ($\because p \sim_{of} q$)
$\implies \mathcal{I}_q \cup \mathcal{R}_q \neq \emptyset$ (by set theory).
If $q \in \mathcal{P}^0$ then $\mathcal{I}_q \cup \mathcal{R}_q = \emptyset$ (by Lemma 4.3.7; which is a contradiction).
$\therefore q \notin \mathcal{P}^0$
$\implies q \in \mathcal{P}^+$ (by Theorem 4.3.2)
$\implies p \in \mathcal{P}^+ \wedge q \in \mathcal{P}^+$ ($\because p \in \mathcal{P}^+$, by assumption).

If $p \notin \mathcal{P}^+$
then $p \in \mathcal{P}^0$ (by Theorem 4.3.2)
$\implies \mathcal{I}_p \cup \mathcal{R}_p = \emptyset$ (by Lemma 4.3.7).
If $q \in \mathcal{P}^+$
then $\mathcal{I}_q \cup \mathcal{R}_q \neq \emptyset$ (by Lemma 4.3.5)
$\implies \exists \beta \in \mathcal{I}_q \cup \mathcal{R}_q\ \exists q'' \in \mathcal{P}\ (q \xrightarrow{\beta} q'')$ (by definitions of $\mathcal{I}_q$ and $\mathcal{R}_q$)
$\implies \beta \in \mathcal{I}_p \cup \mathcal{R}_p \wedge \ \exists p'' \in \mathcal{P}\ (p \xrightarrow{\beta} p'')$ ($\because p \sim_{of} q$)

$\Longrightarrow \mathcal{I}_p \cup \mathcal{R}_p \neq \emptyset$ (by set theory; which is a contradiction).
$\therefore q \notin \mathcal{P}^+$
$\Longrightarrow q \in \mathcal{P}^0$ (by Theorem 4.3.2)
$\Longrightarrow p \in \mathcal{P}^0 \wedge q \in \mathcal{P}^0$ ($\because p \in \mathcal{P}^0$).

$\therefore p \in \mathcal{P}^+ \vee p \notin \mathcal{P}^+ \Longrightarrow p \in \mathcal{P}^+ \wedge q \in \mathcal{P}^+ \ \vee \ p \in \mathcal{P}^0 \wedge q \in \mathcal{P}^0$
($\because \Longrightarrow$ is transitive and
$(predicate_1 \Longrightarrow predicate_2) \wedge (predicate_3 \Longrightarrow predicate_4) \Longrightarrow$
$(predicate_1 \vee predicate_3 \Longrightarrow predicate_2 \vee predicate_4)$).
$\therefore$ For $p, q \in \mathcal{P}$, $p \sim_{of} q \Longrightarrow p \in \mathcal{P}^+ \wedge q \in \mathcal{P}^+ \ \vee \ p \in \mathcal{P}^0 \wedge q \in \mathcal{P}^0$ ($\because \Longrightarrow$ is transitive)
$\Longrightarrow \forall p, q \in \mathcal{P}\ (p \sim_{of} q \Longrightarrow p \in \mathcal{P}^+ \wedge q \in \mathcal{P}^+ \ \vee \ p \in \mathcal{P}^0 \wedge q \in \mathcal{P}^0)$ ($\because p, q \in \mathcal{P}$ are arbitrary).
Q.E.D. ■

Theorem 4.3.4 *$\sim_{of}$ Preserves the Elementary Contexts $\alpha.[\cdot] + M$, $\frac{[\cdot]}{P}$ and $\frac{P}{[\cdot]}$*

Proof: See Section A.8 in Appendix A. The proof technique is to prove $\sim_{of}$ preserves the three elementary contexts separately by discharging the following three proof obligations:
$\vdash \forall p, q \in \mathcal{P}\ (p \sim_{of} q \Longrightarrow \forall \alpha \in \mathcal{I}\ (\alpha.p + M \sim_{of} \alpha.q + M)$
where M is any summation in $\mathcal{P}$),
$\vdash \forall p, q \in \mathcal{P}\ (p \sim_{of} q \Longrightarrow \forall r \in \mathcal{P}\ (\frac{p}{r} \sim_{of} \frac{q}{r}))$ and
$\vdash \forall p, q \in \mathcal{P}\ (p \sim_{of} q \Longrightarrow \forall r \in \mathcal{P}\ (\frac{r}{p} \sim_{of} \frac{r}{q}))$.
Each proof obligation is discharged by producing a witness value, which is a binary relation on $\mathcal{P}$ that contains the pair of processes that are required to be strongly of-bisimilar, showing the relation is a strong of-simulation on $\mathcal{P}$, then showing the inverse relation is a strong of-simulation on $\mathcal{P}$. The first proof obligation is discharged using the witness value
$\{(\alpha.p + M, \alpha.q + M), (p, q), (r, r) \mid$
$\alpha \in \mathcal{I} \ \wedge \ p, q \in \mathcal{P}\ (p \sim_{of} q) \ \wedge \ M$ is any summation in $\mathcal{P} \ \wedge \ r \in \mathcal{P}\}$.
The second proof obligation is discharged using the witness value
$\{(\frac{p}{r}, \frac{q}{r}), (p, q) \mid p, q, r \in \mathcal{P} \ \wedge \ p \sim_{of} q\}$.
The third proof obligation is discharged using the witness value
$\{(\frac{r}{p}, \frac{r}{q}), (r, r) \mid p, q, r \in \mathcal{P} \ \wedge \ p \sim_{of} q\}$. ■

$\sim_{of}$ does **not** satisfy the congruence conditions 2(b) and 2(c), because:

1. $\exists p, q, r \in \mathcal{P}\ (p \sim_{of} q \ \wedge \ \neg(p|r \sim_{of} q|r))$
2. $\exists p, q, r \in \mathcal{P}\ (p \sim_{of} q \ \wedge \ \neg(r|p \sim_{of} r|q))$

Example of 1: $\exists \mathbf{p}, \mathbf{q}, \mathbf{r} \in \mathcal{P}\ (\mathbf{p} \sim_{\mathbf{of}} \mathbf{q} \ \wedge \ \neg(\mathbf{p}|\mathbf{r} \sim_{\mathbf{of}} \mathbf{q}|\mathbf{r}))$

Let $p \triangleq a.0|b.0$ and $q \triangleq a.b.0 + b.a.0$ and $r \triangleq \frac{0}{b.0}$.

We demonstrate $p \sim_{of} q$ by comparing the transitions of p and q, and the transitions of their corresponding successors, side by side; and then demonstrate $\neg(p|r \sim_{of} q|r)$

by identifying a transition of $p|r$ that cannot be performed by $q|r$.

$p \xrightarrow{a} 0\|b.0$	(by the *Sum* and *L-Par* rules)	$q \xrightarrow{a} b.0$	(by the *Sum* rule)
$0\|b.0 \xrightarrow{b} 0\|0$	(by the *Sum* and *R-Par* rules)	$b.0 \xrightarrow{b} 0$	(by the *Sum* rule)
$0\|0$	has no transition	0	has no transition.
$p \xrightarrow{b} a.0\|0$	(by the *Sum* and *R-Par* rules)	$q \xrightarrow{b} a.0$	(by the *Sum* rule)
$a.0\|0 \xrightarrow{a} 0\|0$	(by the *Sum* and *L-Par* rules)	$a.0 \xrightarrow{a} 0$	(by the *Sum* rule)
$0\|0$	has no transition	0	has no transition.

Thus, all the transitions of p, q and their successors in $\mathcal{I}$ have been identified; and there is no transition in $\mathcal{R}$, because p, q and their successors are not fractions and have no fraction factor. Therefore, all the transitions of p, q and their successors in $\mathcal{I} \cup \mathcal{R}$ have been identified; and it should be clear that $p \sim_{of} q$.

Now $p \xrightarrow{\overline{\tau}_{r_{b.0}}} a.0|0$ (by the *Delet* and *R-Par* rules) and $r \xrightarrow{\tau_{r_{b.0}}} 0$ (by the *Creat* rule)
$\Longrightarrow p|r \xrightarrow{\tau} (a.0|0)|0$ (by the *React* rule).
But $q \overset{\overline{\tau}_{r_{b.0}}}{\nrightarrow}$ ($\because$ q is not strongly of-bisimilar to $b.0$, and q is a singleton)
$\Longrightarrow q|r \overset{\tau}{\nrightarrow}$ ($\because$ the hypothesis of *React* does not hold $\wedge\ \tau \notin \mathcal{I}_q\ \wedge\ \tau \notin \mathcal{I}_r$).
$\therefore \neg(p|r \sim_{of} q|r)$ (by definition of strong of-bisimilarity)
$\Longrightarrow \sim_{of}$ is not a congruence (by definition of process congruence).

Example of 2: $\exists \mathbf{p, q, r} \in \mathcal{P}\ (\mathbf{p} \sim_{\mathbf{of}} \mathbf{q}\ \wedge\ \neg(\mathbf{r}|\mathbf{p} \sim_{\mathbf{of}} \mathbf{r}|\mathbf{q}))$

Let $p \triangleq \frac{a.0}{b.0}|\frac{a.0}{b.0}$ and $q \triangleq \frac{\frac{a.0}{b.0}|a.0}{b.0}$ and $r \triangleq \frac{0}{\frac{a.0}{b.0}}$.

We demonstrate $p \sim_{of} q$ by comparing the transitions of p and q, and the transitions of their corresponding successors, side by side; and then demonstrate $\neg(r|p \sim_{of} r|q)$ by identifying a transition of $r|p$ that cannot be performed by $r|q$.

$p \xrightarrow{\tau_{r_{b.0}}} a.0\|\frac{a.0}{b.0}$	(by *Creat* and *L-Par*)	$q \xrightarrow{\tau_{r_{b.0}}} \frac{a.0}{b.0}\|a.0$	(by *Creat*).
$p \xrightarrow{\tau_{r_{b.0}}} \frac{a.0}{b.0}\|a.0$	(by *Creat* and *R-Par*)	$q \xrightarrow{\tau_{r_{b.0}}} \frac{a.0}{b.0}\|a.0$	(by *Creat*).

$a.0\|\frac{a.0}{b.0} \xrightarrow{a} 0\|\frac{a.0}{b.0}$	(by *Sum* and *L-Par*)	$\frac{a.0}{b.0}\|a.0 \xrightarrow{a} \frac{a.0}{b.0}\|0$	(by *Sum* and *R-Par*).
$a.0\|\frac{a.0}{b.0} \xrightarrow{\tau_{r_{b.0}}} a.0\|a.0$	(by *Creat* and *R-Par*)	$\frac{a.0}{b.0}\|a.0 \xrightarrow{\tau_{r_{b.0}}} a.0\|a.0$	(by *Creat* and *L-Par*).
$\frac{a.0}{b.0}\|a.0 \xrightarrow{\tau_{r_{b.0}}} a.0\|a.0$	(by *Creat* and *L-Par*)	$\frac{a.0}{b.0}\|a.0 \xrightarrow{\tau_{r_{b.0}}} a.0\|a.0$	(by *Creat* and *L-Par*).
$\frac{a.0}{b.0}\|a.0 \xrightarrow{a} \frac{a.0}{b.0}\|0$	(by *Sum* and *R-Par*)	$\frac{a.0}{b.0}\|a.0 \xrightarrow{a} \frac{a.0}{b.0}\|0$	(by *Sum* and *R-Par*).

$0|\frac{a.0}{b.0} \xrightarrow{\tau_{r_{b.0}}} 0|a.0$ (by *Creat* and *R-Par*) $\quad \frac{a.0}{b.0}|0 \xrightarrow{\tau_{r_{b.0}}} a.0|0$ (by *Creat* and *L-Par*).

$a.0|a.0 \xrightarrow{a} 0|a.0$ (by *Sum* and *L-Par*) $\quad a.0|a.0 \xrightarrow{a} 0|a.0$ (by *Sum* and *L-Par*).

$a.0|a.0 \xrightarrow{a} a.0|0$ (by *Sum* and *R-Par*) $\quad a.0|a.0 \xrightarrow{a} a.0|0$ (by *Sum* and *R-Par*).

$\frac{a.0}{b.0}|0 \xrightarrow{\tau_{r_{b.0}}} a.0|0$ (by *Creat* and *L-Par*) $\quad \frac{a.0}{b.0}|0 \xrightarrow{\tau_{r_{b.0}}} a.0|0$ (by *Creat* and *L-Par*).

$0|a.0 \xrightarrow{a} 0|0$ (by *Sum* and *R-Par*) $\quad 0|a.0 \xrightarrow{a} 0|0$ (by *Sum* and *R-Par*).

$a.0|0 \xrightarrow{a} 0|0$ (by *Sum* and *L-Par*) $\quad a.0|0 \xrightarrow{a} 0|0$ (by *Sum* and *L-Par*).

$0|0$ has no transition $\quad 0|0$ has no transition.

Thus, all the transitions of p, q and their successors in $\mathcal{I} \cup \mathcal{R}$ have been identified; and it should be clear that $p \sim_{of} q$.

Now $p \xrightarrow{\overline{\tau}_{r_{\frac{a.0}{b.0}}}} 0|\frac{a.0}{b.0}$ (by the *Delet* and *L-Par* rules) and $r \xrightarrow{\tau_{r_{\frac{a.0}{b.0}}}} 0$ (by the *Creat* rule)
$\Longrightarrow r|p \xrightarrow{\tau} 0|(0|\frac{a.0}{b.0})$ (by the *React* rule).

If $r|q$ performs a τ transition then the transition must be a reaction
($\because$ neither r nor q can perform a τ action).
A reaction performed by $r|q$ must be a reconfiguration reaction
($\because$ neither r nor q can perform an action in $\mathcal{I}$).
A reconfiguration reaction performed by $r|q$ must delete r or q
($\because$ r, q are singleton processes).
$\therefore$ If $r|q$ performs a τ transition then r or q is deleted
(by the transitivity of implication).
If r is deleted then $\frac{0}{\frac{a.0}{b.0}} \sim_{of} b.0$
(by the hypotheses of the *Delet* and *Creat* rules, and Theorem 4.3.1)
$\Longrightarrow sfdrdepth(\frac{0}{\frac{a.0}{b.0}}) = sfdrdepth(b.0)$ (by Theorem 4.2.1)
$\Longrightarrow 2 = 0$ (by definition of $sfdrdepth$; which is a contradiction).
$\therefore$ r is not deleted by a reaction transition of $r|q$.
If q is deleted then $\frac{\frac{a.0}{b.0}|a.0}{b.0} \sim_{of} \frac{a.0}{b.0}$
(by the hypotheses of the *Delet* and *Creat* rules, and Theorem 4.3.1)
$\Longrightarrow \frac{a.0}{b.0}|a.0 \sim_{of} a.0$ (by definition of strong of-bisimilarity)
$\Longrightarrow sfdrdepth(\frac{a.0}{b.0}|a.0) = sfdrdepth(a.0)$ (by Theorem 4.2.1)
$\Longrightarrow 1 = 0$ (by definition of $sfdrdepth$; which is a contradiction).
$\therefore$ q is not deleted by a reaction transition of $r|q$
$\therefore$ $r|q$ does not perform a τ transition (by contradiction).
$\therefore \neg(r|p \sim_{of} r|q)$ (by definition of strong of-bisimilarity)
$\Longrightarrow \sim_{of}$ is not a congruence (by definition of process congruence).

The above two examples suggest that strong of-bisimulation is too weak to be a congruence. Therefore, it is necessary to strengthen the conditions that define strong of-bisimulation in order to obtain a stronger bisimulation ($\sim_{dp}$) that is a congruence, and (therefore) can be used for equational reasoning. However, a stronger bisimulation reduces the set of processes matched by a given process, which can reduce the terseness of process expressions. Therefore, in order to retain the terseness of process expressions, we use strong of-bisimulation for matching, and use the stronger bisimulation for congruence-based equational reasoning.

4.3.3 Strong dp-Bisimulation

Strong of-bisimulation is not a congruence because it is based on equality of behavioural state, whereas congruence is based on equality of process structure. Therefore, strong of-bisimulation equates processes with identical behavioural state but with different numbers of factors (as shown in the above examples), whereas congruence distinguishes between processes with different numbers of factors. This suggests that adding a condition to strong of-bisimulation that achieves a bijection between the factors of strongly of-bisimilar processes should be sufficient to produce a bisimulation that is a congruence. Therefore, we define *strong dp-bisimulation* ($\sim_{dp}$) as follows.

S is defined to be a *strong dynamic process reconfigurational simulation* (or equivalently, *strong dp-simulation*) on $\mathcal{P}$ iff $S \subseteq \mathcal{P}$ x $\mathcal{P}$ and the following three conditions hold $\forall (p,q) \in S$:

$$\textit{Observation}: \quad \forall \alpha \in \mathcal{I}_p \; \forall p' \in \mathcal{P} \; (p \xrightarrow{\alpha} p' \implies \alpha \in \mathcal{I}_q \wedge \exists q' \in \mathcal{P} \; (q \xrightarrow{\alpha} q' \wedge (p',q') \in S))$$

$$\textit{Fraction}: \quad \forall \tau_{r_X} \in \mathcal{R}_p \; \forall p'' \in \mathcal{P} \; (p \xrightarrow{\tau_{r_X}} p'' \implies \tau_{r_X} \in \mathcal{R}_q \wedge \exists q'' \in \mathcal{P} \; (q \xrightarrow{\tau_{r_X}} q'' \wedge (p'',q'') \in S))$$

$$\textit{Deletion}: \quad \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \; \forall p''' \in \mathcal{P} \; (p \xrightarrow{\overline{\tau}_{r_Y}} p''' \implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_q \wedge \exists q''' \in \mathcal{P} \; (q \xrightarrow{\overline{\tau}_{r_Y}} q''' \wedge (p''',q''') \in S))$$

A process p is defined to be *strongly dp-simulated by* process q (or equivalently, q *strongly dp-simulates* p), written $p \rightsquigarrow_{dp} q$, iff there exists a strong dp-simulation S on $\mathcal{P}$ with $(p,q) \in S$.

Thus, strong dp-simulation is strong of-simulation extended with the *Deletion* condition, which states that in order for q to simulate p, any reconfiguration delete action that p can perform to become p''', must be also performable by q to become q''', and q''' must simulate p'''.

The three conditions of strong dp-simulation are very similar, and (therefore) can be readily combined into a single condition. However, we prefer to keep them separate in order to show the differences between strong bisimulation in *CCS* and strong of-bisimulation and strong dp-bisimulation in CCS^{dp} more clearly.

A strong dp-simulation S on $\mathcal{P}$ is defined to be a *strong dynamic process reconfigurational bisimulation* (or equivalently, *strong dp-bisimulation*) on $\mathcal{P}$ iff both S and S^{-1} are strong dp-simulations on $\mathcal{P}$.

Process p is defined to be *strongly dynamic process reconfigurationally bisimilar* to process q (or equivalently, p is *strongly dp-bisimilar* to q), written $p \sim_{dp} q$, iff there exists a strong dp-bisimulation S on $\mathcal{P}$ with $(p, q) \in S$.

Following convention, we represent the largest strong dp-bisimulation on $\mathcal{P}$ by $\sim_{dp}$, where $\sim_{dp} \triangleq \bigcup\{S \mid S \text{ is a strong dp-bisimulation on } \mathcal{P}\}$. Notice that $\sim_{dp}$ is non-empty ($\because (0,0) \in \sim_{dp}$, by Lemma 4.3.11).

The similarity between the definitions of strong dp-simulation on $\mathcal{P}$ and strong of-simulation on $\mathcal{P}$ suggests several intuitive logical relationships between dp-based relations and of-based relations that facilitate proof of congruence of $\sim_{dp}$, and which we now prove. Lemma 4.3.10 states that every strong dp-simulation is also a strong of-simulation. Corollary 4.3.3 states that if process p is strongly dp-simulated by process q, then p is strongly of-simulated by q. Corollary 4.3.4 states that every strong dp-bisimulation is also a strong of-bisimulation. Corollary 4.3.5 states that if process p is strongly dp-bisimilar to process q, then p is strongly of-bisimilar to q. Theorem 4.3.5 states that the largest strong dp-bisimulation ($\sim_{dp}$) is a proper subset of the largest strong of-bisimulation ($\sim_{of}$).

Lemma 4.3.10 *Every strong dp-simulation on $\mathcal{P}$ is a strong of-simulation on $\mathcal{P}$.*

Proof: If S is a strong dp-simulation on $\mathcal{P}$
then $S \subseteq \mathcal{P} \times \mathcal{P} \ \wedge$
$\forall (p, q) \in S$ (*Observation* $\wedge$ *Fraction* $\wedge$ *Deletion* conditions of strong dp-simulation on $\mathcal{P}$) hold
(by definition of strong dp-simulation on $\mathcal{P}$)
$\Longrightarrow S \subseteq \mathcal{P} \times \mathcal{P} \ \wedge$
$\forall (p, q) \in S$ (*Observation* $\wedge$ *Fraction* conditions of strong dp-simulation on $\mathcal{P}$) hold
($\because$ $predicate_1 \wedge predicate_2 \wedge predicate_3 \Longrightarrow predicate_1 \wedge predicate_2$)
$\Longrightarrow S \subseteq \mathcal{P} \times \mathcal{P} \ \wedge$
$\forall (p, q) \in S$ (*Observation* $\wedge$ *Fraction* conditions of strong of-simulation on $\mathcal{P}$) hold

($\because$ the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are the same as the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$, respectively)
$\Longrightarrow$ S is a strong of-simulation on $\mathcal{P}$ (by definition of strong of-simulation on $\mathcal{P}$).
$\therefore$ Every strong dp-simulation on $\mathcal{P}$ is a strong of-simulation on $\mathcal{P}$
($\because$ S is an arbitrary strong dp-simulation on $\mathcal{P}$). Q.E.D. ■

Corollary 4.3.3 $\forall p, q \in \mathcal{P}\ (p \rightsquigarrow_{dp} q \Longrightarrow p \rightsquigarrow_{of} q)$

Proof: Suppose $p, q \in \mathcal{P}\ (p \rightsquigarrow_{dp} q)$
then $\exists$ strong dp-simulation S on $\mathcal{P}$ with $(p, q) \in S$ (by definition of $p \rightsquigarrow_{dp} q$)
$\Longrightarrow$ S is a strong of-simulation on $\mathcal{P}$ with $(p, q) \in S$ (by Lemma 4.3.10)
$\Longrightarrow$ $p \rightsquigarrow_{of} q$ (by definition of $p \rightsquigarrow_{of} q$).
$\therefore$ $\forall p, q \in \mathcal{P}\ (p \rightsquigarrow_{dp} q \Longrightarrow p \rightsquigarrow_{of} q)$ ($\because$ $p, q \in \mathcal{P}$ with $p \rightsquigarrow_{dp} q$ are arbitrary). Q.E.D. ■

Corollary 4.3.4 *Every strong dp-bisimulation on $\mathcal{P}$ is a strong of-bisimulation on $\mathcal{P}$.*

Proof: If S is a strong dp-bisimulation on $\mathcal{P}$
then S, S^{-1} are strong dp-simulations on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$)
$\Longrightarrow$ S, S^{-1} are strong of-simulations on $\mathcal{P}$ (by Lemma 4.3.10)
$\Longrightarrow$ S is a strong of-bisimulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$).
$\therefore$ Every strong dp-bisimulation on $\mathcal{P}$ is a strong of-bisimulation on $\mathcal{P}$
($\because$ S is an arbitrary strong dp-bisimulation on $\mathcal{P}$). Q.E.D. ■

Corollary 4.3.5 $\forall p, q \in \mathcal{P}\ (p \sim_{dp} q \Longrightarrow p \sim_{of} q)$

Proof: Suppose $p, q \in \mathcal{P}\ (p \sim_{dp} q)$
then $\exists$ strong dp-bisimulation S on $\mathcal{P}$ with $(p, q) \in S$ (by definition of $p \sim_{dp} q$)
$\Longrightarrow$ S is a strong of-bisimulation on $\mathcal{P}$ with $(p, q) \in S$ (by Corollary 4.3.4)
$\Longrightarrow$ $p \sim_{of} q$ (by definition of $p \sim_{of} q$).
$\therefore$ $\forall p, q \in \mathcal{P}\ (p \sim_{dp} q \Longrightarrow p \sim_{of} q)$ ($\because$ $p, q \in \mathcal{P}$ with $p \sim_{dp} q$ are arbitrary). Q.E.D. ■

Theorem 4.3.5 $\sim_{dp} \subset \sim_{of}$

Proof: Suppose $(p, q) \in \sim_{dp}$ ($\because \sim_{dp} \subseteq \mathcal{P} \times \mathcal{P}$, by definition of $\sim_{dp}$)
then $\exists$ strong dp-bisimulation S on $\mathcal{P}$ with $(p, q) \in S$ (by definition of $\sim_{dp}$)
$\Longrightarrow$ $p \sim_{dp} q$ (by definition of $p \sim_{dp} q$)
$\Longrightarrow$ $p \sim_{of} q$ (by Corollary 4.3.5)
$\Longrightarrow$ $\exists$ strong of-bisimulation T on $\mathcal{P}$ with $(p, q) \in T$ (by definition of $p \sim_{of} q$)
$\Longrightarrow$ $(p, q) \in \sim_{of}$ (by definition of $\sim_{of}$).
$\therefore$ $\forall (p, q) \in \sim_{dp}\ ((p, q) \in \sim_{of})$ ($\because$ $(p, q) \in \sim_{dp}$ is arbitrary)
$\Longrightarrow$ $\sim_{dp} \subseteq \sim_{of}$ (by set theory).
Now $(a.0|b.0,\ a.b.0 + b.a.0) \in \sim_{of} \wedge (a.0|b.0,\ a.b.0 + b.a.0) \notin \sim_{dp}$
(by an example in Section 4.3.2)

$\Longrightarrow \sim_{dp} \subset \sim_{of}$ ($\because \sim_{dp} \subseteq \sim_{of}$, and by set theory). Q.E.D. ■

Strong dp-bisimulation has a number of properties which are useful for equational reasoning, and which we now prove. Lemma 4.3.11 states that $\sim_{dp}$ is reflexive, Lemma 4.3.12 states that $\sim_{dp}$ is symmetric, and Lemma 4.3.14 states that $\sim_{dp}$ is transitive. The three lemmas help to prove Theorem 4.3.6, which states that $\sim_{dp}$ is an equivalence relation on $\mathcal{P}$, and (therefore) satisfies condition 1 of congruence. The transitivity of strong dp-simulation and strong dp-bisimulation is proved using Lemma 4.3.13, whose importance with respect to strong dp-based relations is similar to the importance of Lemma 4.3.3 with respect to strong of-based relations. Lemma 4.3.13 states that the composition of two strong dp-simulations on $\mathcal{P}$ is a strong dp-simulation on $\mathcal{P}$. Corollary 4.3.6 states that $\rightsquigarrow_{dp}$ is transitive, Corollary 4.3.7 states that the composition of two strong dp-bisimulations on $\mathcal{P}$ is a strong dp-bisimulation on $\mathcal{P}$, and Lemma 4.3.14 states that $\sim_{dp}$ is transitive.

Lemma 4.3.11 *$\sim_{dp}$ is reflexive on $\mathcal{P}$.*

Proof: $\sim_{dp}$ is reflexive on $\mathcal{P}$ iff $\forall p \in \mathcal{P}\,(p \sim_{dp} p)$ (by definition of reflexivity).
If $\exists$ strong dp-bisimulation S on $\mathcal{P}$ with $\forall p \in \mathcal{P}\,((p,p) \in S)$
then $\forall p \in \mathcal{P}\,(p \sim_{dp} p)$ (by definition of $p \sim_{dp} p$).
Therefore, we find such an S.

Let $S \triangleq \{(p,p) \mid p \in \mathcal{P}\}$.
$\forall p \in \mathcal{P}\,((p,p) \in S)$ (by definition of S).

S is a strong of-bisimulation on $\mathcal{P}$ (by the proof of Lemma 4.3.1)
$\Longrightarrow$ S is a strong of-simulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$)
$\Longrightarrow$ S is a binary relation on $\mathcal{P}$ $\wedge$
for all elements of S the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied
(by definition of strong of-simulation on $\mathcal{P}$)
$\Longrightarrow$ for all elements of S the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
($\because$ the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are the same as the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$, respectively).
We prove for all elements of S the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied.

Verifying the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ for $(p,p) \in S$:
For $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p$ and $p''' \in \mathcal{P}$,

if $p \xrightarrow{\overline{\tau}_{r_Y}} p'''$
(by the hypothesis of the *Deletion* condition of strong dp-simulation on $\mathcal{P}$)
then $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\tau}_{r_Y}$) $\wedge$ $p''' \in \mathcal{P}$ (by definition of p''') $\wedge$ $p \xrightarrow{\overline{\tau}_{r_Y}} p'''$ $\wedge$ $(p''', p''') \in S$ (by definition of S).
$\therefore \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \; \forall p''' \in \mathcal{P} \; (p \xrightarrow{\overline{\tau}_{r_Y}} p''' \implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \; \wedge \; \exists p''' \in \mathcal{P} \; (p \xrightarrow{\overline{\tau}_{r_Y}} p''' \; \wedge \; (p''', p''') \in S))$
($\because \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p$ and $p''' \in \mathcal{P}$ are arbitrary)
$\implies$ the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ holds for $(p, p) \in S$
(by definition of the *Deletion* condition of strong dp-simulation on $\mathcal{P}$).

$\therefore S$ is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$, $\because (p, p) \in S$ is arbitrary)
$\implies S$ and S^{-1} are strong dp-simulations on $\mathcal{P}$ ($\because S^{-1} = S$)
$\implies S$ is a strong dp-bisimulation on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$)
$\implies \forall p \in \mathcal{P} \; (p \sim_{dp} p)$ (by definitions of $p \sim_{dp} p$ and S)
$\implies \sim_{dp}$ is reflexive on $\mathcal{P}$ (by definition of reflexivity). Q.E.D. ■

Lemma 4.3.12 *$\sim_{dp}$ is symmetric on $\mathcal{P}$.*

Proof: $\sim_{dp}$ is symmetric on $\mathcal{P}$ iff $\forall (p, q) \in \mathcal{P} \times \mathcal{P} \; (p \sim_{dp} q \implies q \sim_{dp} p)$
(by definition of symmetry).

$p \sim_{dp} q \implies \exists$ strong dp-bisimulation S on $\mathcal{P} \; \wedge \; (p, q) \in S$
(by definition of $p \sim_{dp} q$)
$\implies \exists$ strong dp-simulations S, S^{-1} on $\mathcal{P} \; \wedge \; (p, q) \in S \; \wedge \; (q, p) \in S^{-1}$
(by definitions of strong dp-bisimulation on $\mathcal{P}$ and inverse relations)
$\implies \exists$ strong dp-bisimulation S^{-1} on $\mathcal{P} \; \wedge \; (q, p) \in S^{-1}$
($\because (S^{-1})^{-1} = S$ and by definition of strong dp-bisimulation on $\mathcal{P}$)
$\implies q \sim_{dp} p$
(by definition of $q \sim_{dp} p$).
$\therefore \; \forall (p, q) \in \mathcal{P} \times \mathcal{P} \; (p \sim_{dp} q \implies q \sim_{dp} p)$ ($\because p, q \in \mathcal{P}$ are arbitrary)
$\implies \sim_{dp}$ is symmetric on $\mathcal{P}$ (by definition of symmetry). Q.E.D. ■

Lemma 4.3.13 *$\forall$strong dp-simulations U, V on $\mathcal{P}$ (UV is a strong dp-simulation on $\mathcal{P}$).*
Proof: See Section A.9 in Appendix A.

Corollary 4.3.6 $\forall p, q, r \in \mathcal{P} \; (p \rightsquigarrow_{dp} q \; \wedge \; q \rightsquigarrow_{dp} r \implies p \rightsquigarrow_{dp} r)$.

Proof: Suppose $p, q, r \in \mathcal{P} \; (p \rightsquigarrow_{dp} q \; \wedge \; q \rightsquigarrow_{dp} r)$
then $\exists$ strong dp-simulation U on $\mathcal{P}$ $((p, q) \in U)$ (by definition of $p \rightsquigarrow_{dp} q$) $\wedge$
$\exists$ strong dp-simulation V on $\mathcal{P}$ $((q, r) \in V)$ (by definition of $q \rightsquigarrow_{dp} r$)

$\Longrightarrow UV$ is a strong dp-simulation on $\mathcal{P}$ (by Lemma 4.3.13) $\wedge$
$(p,r)\in UV$ (by composition of binary relations)
$\Longrightarrow p\rightsquigarrow_{dp} r$ (by definition of $p\rightsquigarrow_{dp} r$).
$\therefore \forall p,q,r\in\mathcal{P}\ (p\rightsquigarrow_{dp} q \ \wedge\ q\rightsquigarrow_{dp} r \Longrightarrow p\rightsquigarrow_{dp} r)$
(because $\Longrightarrow$ is transitive and $p,q,r\in\mathcal{P}$ with $p\rightsquigarrow_{dp} q \ \wedge\ q\rightsquigarrow_{dp} r$ are arbitrary). Q.E.D.
■

Corollary 4.3.7 *$\forall$strong dp-bisimulations U, V on $\mathcal{P}$*
(UV is a strong dp-bisimulation on $\mathcal{P}$).

Proof: Suppose U, V are strong dp-bisimulations on $\mathcal{P}$
then U, U^{-1}, V, V^{-1} are strong dp-simulations on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$)
$\Longrightarrow UV, V^{-1}U^{-1}$ are strong dp-simulations on $\mathcal{P}$ (by Lemma 4.3.13)
$\Longrightarrow UV, (UV)^{-1}$ are strong dp-simulations on $\mathcal{P}$
($\because (UV)^{-1} = V^{-1}U^{-1}$, by algebra of binary relations)
$\Longrightarrow UV$ is a strong dp-bisimulation on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$).
$\therefore \forall$ strong dp-bisimulations U, V on $\mathcal{P}$ (UV is a strong dp-bisimulation on $\mathcal{P}$)
($\because U, V$ are arbitrary strong dp-bisimulations on $\mathcal{P}$). Q.E.D. ■

Lemma 4.3.14 *$\sim_{dp}$ is transitive on $\mathcal{P}$.*

Proof: $\sim_{dp}$ is transitive on $\mathcal{P}$ iff $\forall p,q,r\in\mathcal{P}\ (p\sim_{dp} q \ \wedge\ q\sim_{dp} r \Longrightarrow p\sim_{dp} r)$
(by definition of transitivity).

Suppose $p,q,r\in\mathcal{P}\ (p\sim_{dp} q \ \wedge\ q\sim_{dp} r)$
then $\exists$ strong dp-bisimulation U on $\mathcal{P}$ $((p,q)\in U)$ (by definition of $p\sim_{dp} q$) $\wedge$
$\exists$ strong dp-bisimulation V on $\mathcal{P}$ $((q,r)\in V)$ (by definition of $q\sim_{dp} r$)
$\Longrightarrow UV$ is a strong dp-bisimulation on $\mathcal{P}$ (by Corollary 4.3.7) $\wedge$
$(p,r)\in UV$ (by composition of binary relations)
$\Longrightarrow p\sim_{dp} r$ (by definition of $p\sim_{dp} r$).
$\therefore \forall p,q,r\in\mathcal{P}\ (p\sim_{dp} q \ \wedge\ q\sim_{dp} r \Longrightarrow p\sim_{dp} r)$
(because $\Longrightarrow$ is transitive and $p,q,r\in\mathcal{P}$ with $p\sim_{dp} q \ \wedge\ q\sim_{dp} r$ are arbitrary)
$\Longrightarrow \sim_{dp}$ is transitive on $\mathcal{P}$ (by definition of transitivity). Q.E.D. ■

Theorem 4.3.6 *$\sim_{dp}$ is an equivalence relation on $\mathcal{P}$.*

Proof: $\sim_{dp}$ is an equivalence relation on $\mathcal{P}$ iff $\sim_{dp}$ is reflexive, symmetric and transitive on $\mathcal{P}$
(by definition of equivalence relation).

$\sim_{dp}$ is reflexive on $\mathcal{P}$ (by Lemma 4.3.11) $\wedge$
$\sim_{dp}$ is symmetric on $\mathcal{P}$ (by Lemma 4.3.12) $\wedge$
$\sim_{dp}$ is transitive on $\mathcal{P}$ (by Lemma 4.3.14)

$\Longrightarrow \sim_{dp}$ is an equivalence relation on $\mathcal{P}$ (by definition of equivalence relation). Q.E.D. ■

The identity property of zero processes in parallel compositions with respect to $\sim_{of}$ and $\sim_{dp}$ is established by Theorem 4.3.3 and Theorem 4.3.7 (respectively). Therefore, a zero process can be elided from a parallel composition with no impact on the behaviour of the process expression, up to $\sim_{of}$ and $\sim_{dp}$. The proof of Theorem 4.3.7 depends on several lemmas and on the notion of *positive singleton factor* of a process, which is defined as follows:

Definition 4.3.5 *A positive singleton factor of a process p is a positive process that is a singleton and a factor of p.*

There are two related definitions. A *positive factor* of a process p is a positive process that is a factor of p. A *zero factor* of a process p is a zero process that is a factor of p.

The multiset of positive singleton factors of a process is given by the function $factors^{+}_{m}$, which is defined as follows:

$factors^{+}_{m} : \mathcal{P} \longrightarrow \mathbb{P}_m\, \mathcal{P}$ such that
$factors^{+}_{m}(p) \triangleq \{f \in factors_m(p) \mid factors_m(f) = \emptyset \ \wedge \ f \in \mathcal{P}^{+}\}_m$

If p is a singleton, then $factors_m(p)$ is the empty multiset, and (therefore) $factors^{+}_{m}(p)$ is the empty multiset.

Theorem 4.3.3 and Theorem 4.3.7 show that the behaviour of a process is determined only by the behaviour of its positive singleton factors and their interaction, up to $\sim_{of}$ and $\sim_{dp}$. Furthermore, parallel composition is commutative and associative with respect to $\sim_{of}$ and $\sim_{dp}$ (see Theorem 4.3.11, Theorem 4.3.13, Theorem 4.3.12 and Theorem 4.3.14). Therefore, the behaviour of p can be derived from $factors^{+}_{m}(p)$, even though the multiset is devoid of any ordering or structural relationship between the factors of p. Thus, $factors^{+}_{m}(p)$ is a canonical representation of p. The properties of $factors^{+}_{m}(p)$, such as its cardinality, are sufficient to prove Theorem 4.3.7 and most of the new lemmas on which the proof of the theorem depends.

Lemma 4.3.15 states that if a parallel composition is a positive process, then the expression has a positive singleton factor. Lemma 4.3.16 states that a zero process has no positive singleton factor. Lemma 4.3.17 states that if a process can perform a delete transition, then either the result is 0 or the result has a factor. Lemma

4.3.18 states that if a process can perform a delete transition, then either the result is 0 or the transition deletes one or more positive singleton factors of the process. Lemma 4.3.19 and Lemma 4.3.20 are 'technical' lemmas that determine the process structure of the result of a deletion transition performed by the parallel composition of a zero process with any other process, given the result is a positive process. Together, the two lemmas imply that the zero process does **not** participate in the deletion transition. Theorem 4.3.7 states that a zero process is an identity of parallel composition with respect to $\sim_{dp}$.

Lemma 4.3.15 $\forall p, q \in \mathcal{P}\ (p|q \in \mathcal{P}^+ \implies factors^+_m(p|q) \neq \emptyset_m)$
Proof: See Section A.10 in Appendix A.

Lemma 4.3.16 $\forall p \in \mathcal{P}\ (p \in \mathcal{P}^0 \implies factors^+_m(p) = \emptyset_m)$
Proof: See Section A.11 in Appendix A.

Lemma 4.3.17 $\forall p, p' \in \mathcal{P}\ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p\ (p \xrightarrow{\overline{\tau}_{r_X}} p' \implies p' = 0 \ \vee\ factors_m(p') \neq \emptyset_m)$
Proof: See Section A.12 in Appendix A.

Lemma 4.3.18 $\forall p, p' \in \mathcal{P}\ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p\ (p \xrightarrow{\overline{\tau}_{r_X}} p' \implies p' = 0 \ \vee\ factors^+_m(p') \subset factors^+_m(p))$
Proof: See Section A.13 in Appendix A.

Lemma 4.3.19 $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{p|z}\ \forall (p|z)' \in \mathcal{P}^+$
$(p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)' \implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \wedge \exists p' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (p|z)' = p'|z))$
Proof: See Section A.14 in Appendix A.

Lemma 4.3.20 $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{z|p}\ \forall (z|p)' \in \mathcal{P}^+$
$(z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)' \implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \wedge \exists p' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (z|p)' = z|p'))$
Proof: See Section A.15 in Appendix A.

Theorem 4.3.7 $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{dp} p \ \wedge\ p \sim_{dp} z|p)$

Proof: See Section A.16 in Appendix A. The proof technique is to prove the conjuncts separately by discharging the following two proof obligations:
$\vdash \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{dp} p)$ and
$\vdash \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p \sim_{dp} z|p)$.
Each proof obligation is discharged by producing a witness value, which is a binary relation on $\mathcal{P}$ that contains the pair of processes that are required to be strongly dp-bisimilar, then showing the relation is a strong dp-bisimulation on $\mathcal{P}$.
The first proof obligation is discharged using the witness value
$\{(p|z, p), (z_1, z_2) \mid p \in \mathcal{P} \ \wedge\ z, z_1, z_2 \in \mathcal{P}^0\}$.
The second proof obligation is discharged using the witness value

$\{(p, z|p), (z_1, z_2) \mid p \in \mathcal{P} \ \wedge \ z, z_1, z_2 \in \mathcal{P}^0\}$.
Both relations are strong of-bisimulations on $\mathcal{P}$ (by the proof of Theorem 4.3.3 and because zero processes have no transition in $\mathcal{I} \cup \mathcal{R}$). In order to prove the relations are strong dp-bisimulations on $\mathcal{P}$, the transitions of the processes in $\overline{\mathcal{R}}$ must be considered. We use complete induction on the number of positive singleton factors of a process to prove the relations are strong dp-bisimulations on $\mathcal{P}$, because every transition in $\overline{\mathcal{R}}$ by a process either deletes the process or deletes one or more positive singleton factors of the process (by Lemma 4.3.18). ■

The multiset of positive singleton factors of a process is also useful for demonstrating the logical relationship between behavioural similarity and structural similarity of processes. Specifically, Theorem 4.3.8 states that strongly dp-bisimilar processes (in parallel composition with 0) have the same number of positive singleton factors. Notice that the parallel composition with 0 is necessary in order to preserve the numerical equality if one process is a singleton and the other is not. Therefore, strengthening strong of-simulation on $\mathcal{P}$ with the *Deletion* condition produces structural similarity between strongly dp-bisimilar processes (in parallel composition with 0) in terms of the number of positive singleton factors. The proof of Theorem 4.3.8 assumes the theorem is false, identifies a pair of strongly dp-bisimilar processes – one of which has the least number of positive singleton factors, performs a deletion transition on the other process that deletes exactly one positive singleton factor of the process, and shows a contradiction. The proof needs Lemma 4.3.21, which states that it is possible for a process to perform a deletion transition that deletes exactly one of its positive singleton factors (if any).

Lemma 4.3.21 $\forall p \in \mathcal{P} \ \forall f \in factors_m^+(p)$
$(\exists \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p \ \exists p' \in \mathcal{P} \ (p \xrightarrow{\overline{\tau}_{r_f}} p' \ \wedge \ factors_m^+(p) = factors_m^+(p') \uplus \{f\}_m))$
Proof: See Section A.17 in Appendix A.

Theorem 4.3.8 $\forall p, q \in \mathcal{P} \ (p \sim_{dp} q \implies |factors_m^+(p|0)| = |factors_m^+(q|0)|)$
Proof: See Section A.18 in Appendix A.

In Section 4.3.2, we demonstrated that $\sim_{of}$ does not satisfy the congruence conditions 2(b) and 2(c). However, if $p \sim_{of} q$ is strengthened to $p \sim_{dp} q$, then the congruence conditions 2(b) and 2(c) are satisfied for $\sim_{of}$ (see Theorem 4.3.9). Furthermore, $\sim_{dp}$ satisfies condition 2 of congruence (by Theorem 4.3.10), and satisfies condition 1 of congruence (by Theorem 4.3.6). Therefore, $\sim_{dp}$ is a congruence on $\mathcal{P}$ (by definition of congruence), and (therefore) can be used for equational reasoning.

Theorem 4.3.9 $\forall p,q\in\mathcal{P}\,(p\sim_{dp}q \implies \forall r\in\mathcal{P}\,(p|r\sim_{of}q|r)\ \wedge\ \forall r\in\mathcal{P}\,(r|p\sim_{of}r|q))$

Proof: See Section A.19 in Appendix A. The proof technique is to prove the conjuncts separately by discharging the following two proof obligations:
$\vdash \forall p,q\in\mathcal{P}\,(p\sim_{dp}q \implies \forall r\in\mathcal{P}\,(p|r\sim_{of}q|r))$ and
$\vdash \forall p,q\in\mathcal{P}\,(p\sim_{dp}q \implies \forall r\in\mathcal{P}\,(r|p\sim_{of}r|q))$.
Each proof obligation is discharged by producing a witness value, which is a binary relation on $\mathcal{P}$ that contains the pair of processes that are required to be strongly of-bisimilar, showing the relation is a strong of-simulation on $\mathcal{P}$, then showing the inverse relation is a strong of-simulation on $\mathcal{P}$. The first proof obligation is discharged using the witness value
$\{(p|r,q|r)\mid p,q,r\in\mathcal{P}\ \wedge\ p\sim_{dp}q\}$.
The second proof obligation is discharged using the witness value
$\{(r|p,r|q)\mid p,q,r\in\mathcal{P}\ \wedge\ p\sim_{dp}q\}$. ■

Theorem 4.3.10 $\sim_{dp}$ *Preserves all Elementary Contexts*

Proof: See Section A.20 in Appendix A. The proof technique is to prove $\sim_{dp}$ preserves the elementary contexts separately by discharging the following five proof obligations:
$\vdash \forall p,q\in\mathcal{P}\,(p\sim_{dp}q \implies \forall\alpha\in\mathcal{I}\,(\alpha.p+M\sim_{dp}\alpha.q+M)$
where M is any summation in $\mathcal{P}$),
$\vdash \forall p,q\in\mathcal{P}\,(p\sim_{dp}q \implies \forall r\in\mathcal{P}\,(p|r\sim_{dp}q|r))$,
$\vdash \forall p,q\in\mathcal{P}\,(p\sim_{dp}q \implies \forall r\in\mathcal{P}\,(r|p\sim_{dp}r|q))$,
$\vdash \forall p,q\in\mathcal{P}\,(p\sim_{dp}q \implies \forall r\in\mathcal{P}\,(\frac{p}{r}\sim_{dp}\frac{q}{r}))$ and
$\vdash \forall p,q\in\mathcal{P}\,(p\sim_{dp}q \implies \forall r\in\mathcal{P}\,(\frac{r}{p}\sim_{dp}\frac{r}{q}))$.
Each proof obligation is discharged by producing a witness value, which is a binary relation on $\mathcal{P}$ that contains the pair of processes that are required to be strongly dp-bisimilar, then showing the relation is a strong dp-bisimulation on $\mathcal{P}$.
The first proof obligation is discharged using the witness value
$\{(\alpha.p+M,\alpha.q+M),(p,q),(r,r)\mid$
$\alpha\in\mathcal{I}\ \wedge\ p,q\in\mathcal{P}\,(p\sim_{dp}q)\ \wedge\ M$ is any summation in $\mathcal{P}\ \wedge\ r\in\mathcal{P}\}$.
The second proof obligation is discharged using the witness value
$\{(p|r,q|r),(0,0)\mid p,q,r\in\mathcal{P}\ \wedge\ p\sim_{dp}q\}$
and the third proof obligation is discharged using the witness value
$\{(r|p,r|q),(0,0)\mid p,q,r\in\mathcal{P}\ \wedge\ p\sim_{dp}q\}$. Both this relation and the previous relation are strong of-bisimulations on $\mathcal{P}$ (by the proof of Theorem 4.3.9 and because the 0 process has no transition in $\mathcal{I}\cup\mathcal{R}$). We show the relations are strong dp-bisimulations on $\mathcal{P}$ by using complete induction on the depth of inference of

the applications of the LTS rules that determine the transitions of the processes in $\overline{\mathcal{R}}$.
The fourth proof obligation is discharged using the witness value
$\{(\frac{p}{r}, \frac{q}{r}), (p,q) \mid p,q,r \in \mathcal{P} \ \wedge \ p \sim_{dp} q\}$.
The fifth proof obligation is discharged using the witness value
$\{(\frac{r}{p}, \frac{r}{q}), (r,r) \mid p,q,r \in \mathcal{P} \ \wedge \ p \sim_{dp} q\}$. ■

Commutativity and associativity are useful for both matching and equational reasoning. Theorem 4.3.12 states that the parallel composition of processes is commutative with respect to $\sim_{dp}$, which facilitates equational reasoning. The proof of Theorem 4.3.12 uses the commutativity of parallel composition with respect to $\sim_{of}$, which facilitates matching and is stated in Theorem 4.3.11. The proof of Theorem 4.3.11 uses complete induction on the successors' fractional denominator recursion depth of a parallel composition (i.e. $sfdrdepth(p|q)$), and depends on several new lemmas: Lemma 4.3.25 states that the $sfdrdepth$ of a parallel composition is the maximum of the $sfdrdepth$ values of the two composed processes. Lemma 4.3.24 states that a process performing a deletion transition strongly dp-simulates the result. That is, a deletion transition does not add behaviour to a process. Lemma 4.3.23 states that strong dp-simulation on $\mathcal{P}$ is preserved by parallel composition. Lemma 4.3.22 states that a zero process is strongly dp-simulated by any process.

Lemma 4.3.22 $\forall z \in \mathcal{P}^0 \ \forall p \in \mathcal{P} \ (z \rightsquigarrow_{dp} p)$
Proof: See Section A.21 in Appendix A.

Lemma 4.3.23 $\forall p,q,r,s \in \mathcal{P} \ (p \rightsquigarrow_{dp} q \ \wedge \ r \rightsquigarrow_{dp} s \implies p|r \rightsquigarrow_{dp} q|s)$
Proof: See Section A.22 in Appendix A.

Lemma 4.3.24 $\forall p,p' \in \mathcal{P} \ \forall \overline{\tau}_{r_X} \in \overline{R}_p \ (p \xrightarrow{\overline{\tau}_{r_X}} p' \implies p' \rightsquigarrow_{dp} p)$
Proof: See Section A.23 in Appendix A.

Lemma 4.3.25 $\forall p,q \in \mathcal{P} \ (sfdrdepth(p|q) = max\{sfdrdepth(p), sfdrdepth(q)\})$
Proof: See Section A.24 in Appendix A.

Theorem 4.3.11 $\forall p,q \in \mathcal{P} \ (p|q \sim_{of} q|p)$
Proof: See Section A.25 in Appendix A.

Theorem 4.3.12 $\forall p,q \in \mathcal{P} \ (p|q \sim_{dp} q|p)$
Proof: See Section A.26 in Appendix A.

The associativity of parallel composition with respect to $\sim_{of}$ and $\sim_{dp}$ is established by Theorem 4.3.13 and Theorem 4.3.14 (respectively). The proof of Theorem 4.3.14

uses Theorem 4.3.13 and a new lemma. Lemma 4.3.26 is a 'technical' lemma that shows possible decompositions of a deletion transition performed by a parallel composition of processes into deletion transitions performed by the two composed processes, and the structure of the resulting process.

Theorem 4.3.13 $\forall p,q,r\in\mathcal{P}\ ((p|q)|r\sim_{of} p|(q|r))$
Proof: See Section A.27 in Appendix A.

Lemma 4.3.26 $\forall p,q\in\mathcal{P}\ \forall\overline{\tau}_{r_X}\in\overline{\mathcal{R}}_{p|q}\ \forall(p|q)'\in\mathcal{P}$
$(p|q \xrightarrow{\overline{\tau}_{r_X}} (p|q)' \Longrightarrow$
$(p|q)' = 0 \ \vee\ \exists p'\in\mathcal{P}\,(p \xrightarrow{\overline{\tau}_{r_X}} p' \wedge (p|q)' = p'|q) \ \vee\ \exists q'\in\mathcal{P}\,(q \xrightarrow{\overline{\tau}_{r_X}} q' \wedge (p|q)' = p|q') \ \vee$
$\exists\overline{\tau}_{r_{X_1}},\overline{\tau}_{r_{X_2}}\in\overline{\mathcal{R}}\ \exists p',q'\in\mathcal{P}\,(X\sim_{of} X_1|X_2 \wedge p \xrightarrow{\overline{\tau}_{r_{X_1}}} p' \wedge q \xrightarrow{\overline{\tau}_{r_{X_2}}} q' \wedge (p|q)' = p'|q'))$
Proof: See Section A.28 in Appendix A.

Theorem 4.3.14 $\forall p,q,r\in\mathcal{P}\ ((p|q)|r\sim_{dp} p|(q|r))$
Proof: See Section A.29 in Appendix A.

4.4 Consistency and Decidability

We define the terms *consistency* of a formalism (also known as *logical consistency*) and *decidability*, and briefly discuss how these two requirements are met by basic CCS^{dp}.

4.4.1 Consistency

In order to discuss *consistency*, the following definitions are needed. For a more detailed explanation of the definitions, see [BA01].

Definition 4.4.1 *A formula is an expression with a Boolean value.*

Examples of formulae in CCS^{dp} are $p \xrightarrow{\alpha} p'$, $p\sim_{of} q$ and $p\sim_{dp} q$.

Definition 4.4.2 *A theory is a set of formulae that is closed under logical consequence.*

For a formula A and a theory $\mathcal{T}$, $A\in\mathcal{T}$ iff $\mathcal{T}\models A$ (by Definition 4.4.2).

Definition 4.4.3 *A theory $\mathcal{T}$ is inconsistent iff for some formula A, $\mathcal{T}\models A$ and $\mathcal{T}\models\neg A$. A theory is consistent iff the theory is **not** inconsistent.*

Consistency of a formalism is required in order to predict a system's behaviour, as well as absence of behaviour, using a system model expressed using the formalism. Therefore, our theory of basic CCS^{dp} must be consistent.

In basic CCS^{dp}, the formulae are the process transitions and the simulations and bisimulations derived from the process transitions. Therefore, our theory of CCS^{dp} is inconsistent iff $\exists\, p, p' \in \mathcal{P}\ \exists\, \alpha \in \mathcal{A}_p\ (p \xrightarrow{\alpha} p' \ \wedge\ p \overset{\alpha}{\nrightarrow} p')$ by the application of one or more LTS rules with finite depth of inference (by Definition 4.4.3).

However, the absence of negative premises in the LTS rules implies the LTS rules define **only** transitions [Gro93]. Therefore, our theory of basic CCS^{dp} is not inconsistent, and (therefore) our theory of basic CCS^{dp} is consistent (by Definition 4.4.3).

4.4.2 Decidability

In order to discuss *decidability*, the following definitions are needed [BA01]:

Definition 4.4.4 *A closed formula is a formula with no free variable.*

Definition 4.4.5 *A theory $\mathcal{T}$ is decidable iff there exists a terminating procedure that decides for any closed formula A, true if $A \in \mathcal{T}$ and false if $A \notin \mathcal{T}$.*

Decidability is required in order to provide tool support for a formalism, without which the formalism will not be used by engineers. Therefore, formulae in basic CCS^{dp}, such as $p \sim_{of} q$, must be decidable.

The decidability of strong of-bisimilarity between processes helps to automate matching, and thereby the execution of reconfiguration transitions, which (in turn) facilitates the automation of equational reasoning and model checking. The decidability of strong dp-bisimilarity between processes also helps to automate equational reasoning and model checking. Therefore, ensuring decidability of formulae in our theory of basic CCS^{dp} is important. However, decidability it is not the focus of our research. Therefore, we identify several factors that help to achieve decidability, but leave a comprehensive answer for future work.

First, in full *CCS*, both observation equivalence and strong bisimulation are undecidable [Mil89]. However, if restriction and relabelling are removed from *CCS*, then strong bisimulation is decidable [CHM94]; which justifies our decision

to omit the restriction operator (ν) from basic CCS^{dp}, and to use only strong bisimulations.

Second, the evolution of a fraction process could result in successor fractions with strictly increasing depth of fractional recursion, which would complicate proofs and could make matching undecidable. Therefore, we bounded the depth of fractional recursion of the denominators of fractions and their successors.

Third, the LTS rules define a countably infinite number of reconfiguration transitions by a process. This problem is resolved by grouping the reconfiguration transitions into a finite number of equivalence classes using the denominators of the fractions in the model (which are finite in number) and the equivalence property of strong of-bisimulation on $\mathcal{P}$.

Fourth, it is possible for a fraction process to evolve into multiple processes due to its numerator, which could result in a system with an infinite number of states. Therefore, modellers are advised to construct models that consist of a finite number of processes and process definitions. This decision avoids imposing unnecessarily strong restrictions on basic CCS^{dp} before the issue of decidability of bisimulations between fraction processes has been thoroughly investigated, which requires a proof theory, and (therefore) is beyond the scope of this thesis.

4.5 Forms of Matching

Matching in basic CCS^{dp} is based on similarity of behaviour between processes, specifically, strong of-bisimulation on $\mathcal{P}$. However, there are alternative forms of matching, using syntactic equality (=), structural congruence (≡) or strong observation equivalence (∼). In this section, we briefly describe these alternatives and identify tradeoffs.

In order to model unplanned reconfiguration abstractly, matching affects only the semantics of CCS^{dp} and its derivatives. Therefore, the different forms of matching do not have any impact on the process syntax of CCS^{dp} nor on the definitions of positive and zero processes and process context. The impact on the LTS rules is described below.

4.5.1 Syntactic Equality-based Matching

The new LTS rules are $Creat_{=}$, $Delet_{=}$, $CompDelet_{=}$, $L\text{-}React_{=}$ and $R\text{-}React_{=}$, which are identical to their similarly named counterparts in basic CCS^{dp} with the matching relation $\sim_{of}$ replaced by $=$.

Syntactic equality is an equivalence relation on $\mathcal{P}$ (by definition of $=$); which facilitates matching. Strong dp-bisimilarity on $\mathcal{P}$ is interpreted with the transitions in $\mathcal{R}$ and $\overline{\mathcal{R}}$ defined by the new LTS rules, and is a congruence; which supports equational reasoning. Parallel composition is associative and commutative up to $\sim_{dp}$ (due to $CompDelet_{=}$, $L\text{-}React_{=}$ and $R\text{-}React_{=}$); which also support equational reasoning.

4.5.2 Structural Congruence-based Matching

Structural congruence is an equivalence relation on $\mathcal{P}$ defined by the set of axioms given in Table 4.2.

	$\alpha\text{-conversion}(P) \equiv P$	
$P\|0 \equiv P$	$P\|Q \equiv Q\|P$	$P\|(Q\|R) \equiv (P\|Q)\|R$
$M + 0 \equiv M$	$M_1 + M_2 \equiv M_2 + M_1$	$M_1 + (M_2 + M_3) \equiv (M_1 + M_2) + M_3$
	$P \equiv Q \wedge R \equiv S \Longleftrightarrow \frac{P}{R} \equiv \frac{Q}{S}$	
	$A<\widetilde{b}>\equiv P[\widetilde{b}/\widetilde{a}]$ if $A(\widetilde{a}) \triangleq P \wedge \lvert\widetilde{b}\rvert = \lvert\widetilde{a}\rvert$	

Table 4.2: Structural Congruence of Basic CCS^{dp}.

The structural congruence axioms for basic CCS^{dp} are a superset of the structural congruence axioms for basic CCS without ν. The additional axiom is for fraction processes, and states that two fractions are structurally congruent iff their respective numerators and denominators are structurally congruent.

The new LTS rules are $Creat_{\equiv}$, $Delet_{\equiv}$, $CompDelet_{\equiv}$, $L\text{-}React_{\equiv}$ and $R\text{-}React_{\equiv}$, which are identical to their similarly named counterparts in basic CCS^{dp} with the matching relation $\sim_{of}$ replaced by $\equiv$. However, because of the associativity and commutativity of parallel composition up to $\equiv$, $CompDelet_{\equiv}$, $L\text{-}React_{\equiv}$ and $R\text{-}React_{\equiv}$ can be replaced the $Struct_{\equiv}$ rule defined below:

$$\text{Struct}_{\equiv} \ : \ \frac{\mu \in \mathcal{A} \ \wedge \ Q \equiv P \ \wedge \ P \xrightarrow{\mu} P' \ \wedge \ P' \equiv Q'}{Q \xrightarrow{\mu} Q'}$$

The $Struct_{\equiv}$ rule states that structurally congruent processes (P, Q) have identical transitions in $\mathcal{A}$, and their respective results (P', Q') are structurally congruent.

Structural congruence is an equivalence relation on $\mathcal{P}$ (by definition of $\equiv$); which helps matching. Strong dp-bisimilarity on $\mathcal{P}$ is interpreted using the new LTS rules, and is a congruence; which helps equational reasoning. Parallel composition is associative and commutative up to $\sim_{dp}$ (due to $CompDelet_{=}$, $L\text{-}React_{=}$ and $R\text{-}React_{=}$ or $Struct_{\equiv}$); which also help equational reasoning.

4.5.3 Strong Observation Equivalence-based Matching

This form of matching is identical to that of basic CCS^{dp} with the restriction $\forall p \in \mathcal{P}\,(sfdrdepth(p) \leq 1)$. Therefore, only CCS processes can be reconfigured.

4.5.4 Comparison

The different forms of matching can be compared using the notions of *coverage*, *decidable processes* and *complexity*. The *coverage* of a process is the set of processes that can be matched using the given process. The *decidable processes* of a matching is the set of processes for which the matching relation is decidable. The *complexity* of matching with a process is the minimum computational complexity of the matching using the given process.

Regarding coverage, syntactic equality is the most restrictive form of matching.
P matches Q syntactically iff $P = Q$.
Therefore, $P|Q$ cannot match $Q|P$ syntactically ($\because P|Q \neq Q|P$).
However, $P|Q$ matches $Q|P$ structurally
($\because P|Q \equiv Q|P$, by the structural congruence axioms)
and $P = Q \Longrightarrow P$ matches Q structurally ($\because \equiv$ is an equivalence relation on $\mathcal{P}$).
However, $a.b.0 + b.a.0$ cannot match $a.0 \mid b.0$ structurally
(by definition of structural congruence).
But $a.b.0 + b.a.0$ can match $a.0 \mid b.0$ by strong of-bisimulation
(see example in Section 4.3.2),
and $P|Q$ matches $Q|P$ by strong of-bisimulation (by Theorem 4.3.11),
and $P = Q \Longrightarrow P$ matches Q by strong of-bisimulation ($\because \sim_{of}$ is reflexive on $\mathcal{P}$).

If we take coverage as the function $coverage : \mathcal{P} \times (\mathcal{P} \times \mathcal{P}) \longrightarrow \mathbb{P}\,\mathcal{P}$,
we have $coverage(p, =) \subset coverage(p, \equiv) \subset coverage(p, \sim_{of})$.

Regarding the decidable processes of a matching relation, syntactic equality is the most decidable relation, since it involves a syntactic comparison of two process expressions in closed form. Strong of-bisimulation is the least decidable relation, since it requires the removal of restriction and relabelling from the process syntax.

If we take the decidable processes of a matching relation as given by the function $DecProcs : \mathcal{P} \times \mathcal{P} \longrightarrow \mathbb{P}\,\mathcal{P}$,
it is conjectured that we have $DecProcs(\sim_{of}) \subset DecProcs(\equiv) \subset DecProcs(=)$.

Regarding complexity, it is conjectured that the complexity of matching with a process is least with syntactic equality-based matching; intermediate with structural congruence-based matching; and greatest with strong of-bisimulation-based matching.

Regarding the effect of matching on strong dp-bisimilarity, Theorem 4.3.8 states that strongly dp-bisimilar processes (in parallel composition with 0) have the same number of positive singleton factors. This theorem also holds for syntactic equality-based and structural congruence-based matching. There is a 1–to–1 correspondence between the positive singleton factors of strongly dp-bisimilar processes (in parallel composition with 0) in all three forms of matching. However, in syntactic equality-based matching, corresponding factors are syntactically equal (e.g. $a.0 = a.0$); in structural congruence-based matching, corresponding factors are structurally congruent (e.g. $a.0+b.0 \equiv b.0+a.0$); and in strong of-bisimulation-based matching, corresponding factors are strongly of-bisimilar (e.g. $a.0 + a.0 \sim_{of} a.0$).

The results and conjectures for strong observation equivalence-based matching are similar to those of strong of-bisimulation-based matching, if the restriction $\forall p \in \mathcal{P}\,(sfdrdepth(p) \leq 1)$ is used.

The proofs of the above conjectures are left for future work, because they are beyond the scope of this thesis.

In conclusion, regarding the different forms of matching discussed above, matching based on syntactic equality is the most amenable to tool support, because of its decidable processes and complexity. However, the coverage is minimal, and (therefore) process expressions modelling reconfiguration are the most ver-

bose. Furthermore, the syntactic structure of a process to be reconfigured must be known. Matching based on structural congruence is also amenable to tool support, but less so than matching based on syntactic equality, although reconfiguration models are more terse. The structure of a process to be reconfigured must be known. Matching based on strong of-bisimulation is the least amenable to tool support. However, it has the greatest coverage, and (therefore) process expressions modelling reconfiguration are the most terse. It also supports information hiding, because only behaviour is used for matching. Matching based on strong observation equivalence has the advantages of matching using strong of-bisimulation, but tool support is simpler, because fraction processes cannot be reconfigured.

4.6 Evaluation using Requirements

We evaluate basic CCS^{dp} with respect to the requirements used to evaluate π-calculi in Chapter 3.

In basic CCS^{dp}, software components and tasks are both modelled as processes, which can be identified by using unique names (as in *CCS*). Communication links between processes are modelled as pairs of complementary port names that belong to different processes. Communication links can be identified by using unique port names.

Planned process creation and deletion can be modelled as in *CCS* or using fraction processes. Planned process replacement is modelled using fraction processes. Planned link creation or deletion cannot be modelled directly, but can be modelled indirectly using process replacement. Unplanned process and link reconfiguration is modelled using fraction processes. However, it is not possible to delete or replace instances of processes selectively (e.g. by using a process name).

There is no notion of physical node in basic CCS^{dp}. Therefore, relocation of processes on physical nodes cannot be modelled.

Transfer of values in communication can be modelled using parameterless transitions. Therefore, state transfer can be modelled using parameterless transitions.

Functional and temporal interactions between application and reconfiguration tasks are modelled as interleavings of process transitions. However, there is no

notion of physical time in basic CCS^{dp}, and (therefore) the duration of actions and delays cannot be modelled.

Functional correctness can be expressed in terms of strong dp-bisimilarity between processes, which can be verified by equational reasoning. Model checking can also be used. However, temporal correctness cannot be expressed or verified, because there is no notion of physical time in basic CCS^{dp}.

Concurrent execution of tasks is modelled as concurrent execution of processes with an interleaving semantics.

State transitions of software components and tasks is modelled as process transitions.

Support for functional verification is limited to equational reasoning and model checking. The absence of a type system precludes type checking. Support for temporal correctness is problematic. First, the absence of a time model precludes schedulability analysis of processes. Second, if a time model were added to the formalism, the schedulability analysis of processes would be still complicated by the synchronous communication model, which results in a timing dependency between the sending and receiving processes.

Inspite of its limitations, basic CCS^{dp} is a useful and potentially powerful formalism for modelling unplanned reconfiguration abstractly, unlike other formalisms. The simplicity of basic CCS^{dp} should allow it to be extended (like basic *CCS*) to address its limitations, but avoid unnecessary complexity. Tool support is contingent on increasing the set of decidable processes of matching and reducing the complexity of matching. It may be possible to encode matching in Isabelle [NP02], in which case it will be possible to prove correctness of expressions in basic CCS^{dp}. However, this conjecture has to be confirmed.

Chapter 5

Evaluation of Basic CCS^{dp} using a Simple Office Workflow

Contents

In this chapter, basic *CCS*^{dp} is evaluated using as a case study the dynamic reconfiguration of a simple office workflow. A preliminary version of the workflow is given in [MADB12]. We describe the workflow and its reconfiguration, and define the requirements on the reconfiguration. Different designs for the workflow and its reconfiguration are then identified, and one of the designs is formulated in basic *CCS*^{dp}. The reconfiguration requirements are then used to evaluate the formulation, and thereby identify the strengths and weaknesses of basic *CCS*^{dp}.

5.1 Office Workflow for Order Processing

Our office workflow for order processing is a simplified version of real workflows commonly found in large and medium-sized organisations [EKR95]. The workflow initially contains the following activities:

1. `Order Receipt`: an order for a product is received from an existing customer. The order includes the customer's identifier and the product's identifier. An

evaluation of the order is initiated to determine whether or not the order is viable.

2. `Evaluation`: in evaluating an order, the product identity is used to perform an inventory check on the availability of the product. The customer identity is used to perform an internal credit check on the customer. If both the checks are positive, the order is accepted; otherwise the order is rejected.

3. `Rejection`: if the order is rejected, a notification of rejection is sent to the customer and the workflow terminates.

4. If the order is accepted, the following activities are initiated:

   (a) `Billing`: the customer is billed for the cost of the product ordered plus shipping costs.

   (b) `Shipping`: the product is shipped to the customer.

   (c) `Archiving`: the order is archived for future reference.

   (d) `Confirmation`: a notification of successful completion of the order is sent to the customer.

The workflow is structured using Configuration 1, which must meet the following requirements (see Figure 5.1).

Requirements on Configuration 1 of the Workflow

For each order:

1. `Order Receipt` must be performed first. That is, it must begin before any other activity.

2. `Evaluation` must be performed second.

3. If the output of `Evaluation` is negative, `Rejection` must be the third and final activity of the workflow.

4. If the output of `Evaluation` is positive, the following conditions must be satisfied:

   (a) `Billing` and `Shipping` must be the third set of activities to be performed.

   (b) `Billing` and `Shipping` must be performed concurrently.

(c) After the completion of both `Billing` and `Shipping`, `Archiving` must be performed.

(d) After the completion of `Archiving`, `Confirmation` must be performed.

5. Each activity must be performed at most once.

6. The workflow must terminate.

After some time, the management of an organization using the workflow can decide to change it in order to increase opportunities for sales, improve the synchronisation between `Billing` and `Shipping`, and to simplify the workflow. The new version of the workflow can be structured using Configuration 2, to meet the following requirements (see Figure 5.2).

Requirements on Configuration 2 of the Workflow

For each order:

1. `Order Receipt` must be performed first.

2. `Evaluation`: in evaluating an order, the product identity is used to perform an inventory check on the availability of the product. The customer identity is used to perform an internal credit check on the customer. If the internal credit check fails, an external credit check is performed on the customer. If the inventory check is positive, and either credit check is positive, the order is accepted; otherwise the order is rejected.

3. `Evaluation` must be performed second.

4. If the output of `Evaluation` is negative, `Rejection` must be the third and final activity of the workflow.

5. If the output of `Evaluation` is positive, the following conditions must be satisfied:

   (a) `Shipping` must be the third activity to be performed.

   (b) `Billing` must be the fourth activity to be performed.

   (c) `Archiving` must be the fifth and final activity to be performed.

6. Each activity must be performed at most once.

7. The workflow must terminate.

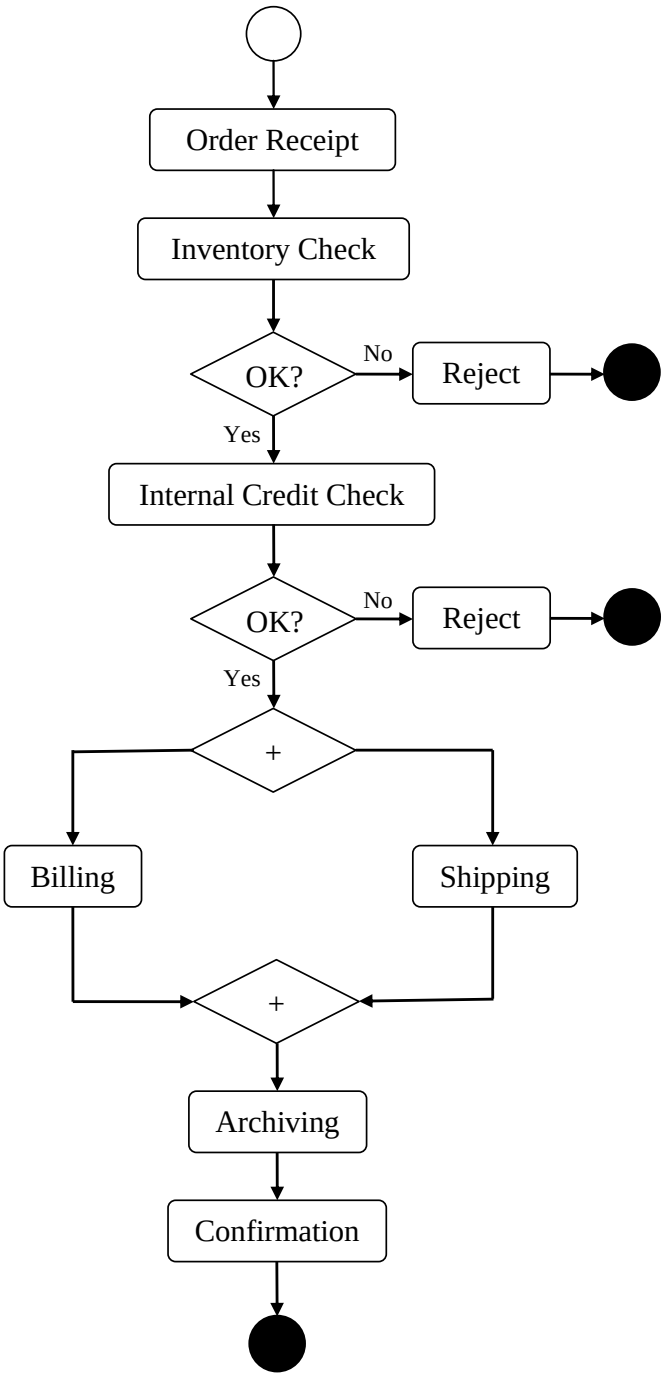


Figure 5.1: Flow chart of the requirements on Configuration 1.

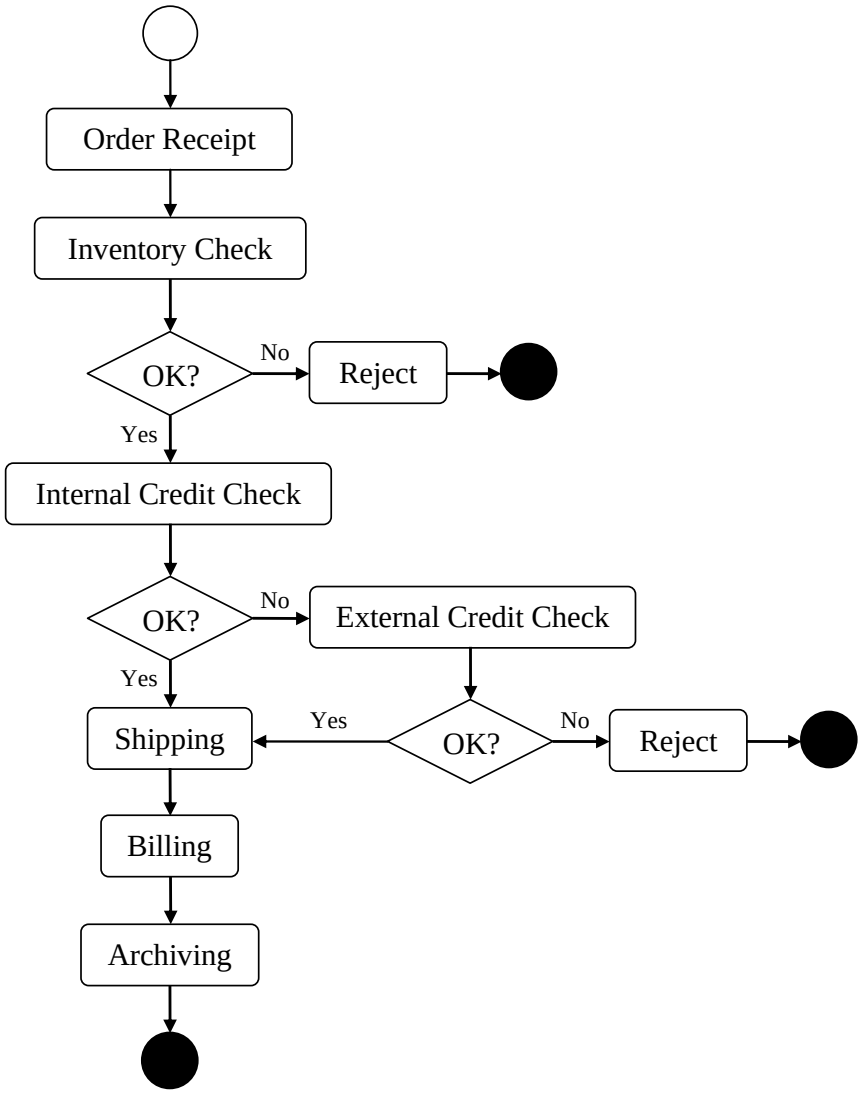


Figure 5.2: Flow chart of the requirements on Configuration 2.

In order to achieve a smooth transition from Configuration 1 to Configuration 2 of the workflow, the process of reconfiguration must meet the following requirements.

Requirements on Reconfiguration of the Workflow

1. Reconfiguration of a workflow should not necessarily result in the rejection of an order.

   In some systems, executing activities of configuration 1 are aborted during its reconfiguration to configuration 2 (see Case 2 in Figure 1.1). The purpose of this requirement is to avoid the occurrence of Case 2 and ensure the occurrence of Case 3.

2. No order should be significantly delayed by the reconfiguration.

3. All orders being processed that were accepted **before** the start of the reconfiguration must satisfy the requirements on Configuration 1 or the requirements on Configuration 2.

4. All orders accepted **after** the start of the reconfiguration must satisfy the requirements on Configuration 2.

5. The reconfiguration process must terminate.

5.2 Reconfiguration of a Design

The reconfiguration of the workflow depends on the workflow's design and implementation. There are at least four possible designs for the workflow:

1. There is at most one workflow, and the workflow handles a single order at a time.

   The workflow is mainly sequential: after an order is received, the thread performs a sequence of actions, with a choice at `Evaluation`, and a possible interleaved execution of `Billing` and `Shipping`. After the order has been processed, the thread is ready to receive a new order. This design corresponds to a cyclic executive.

2. There is at most one workflow, and the workflow can handle multiple orders at a time.

   The workflow is mainly concurrent: after an order is received, the thread forks internally into concurrent threads, such that different threads perform the different activities of the workflow – although the requirements on the configurations severely restrict the degree of internal concurrency of the workflow, and each thread performs the same activity for different orders.

3. There can be multiple workflows, and each workflow handles a single order at a time.

   After an order is received, the thread forks into two threads: one thread processes the order sequentially – as in Design 1 – but terminates after the order has been processed; the other thread waits to receive a new order.

4. There can be multiple workflows, and each workflow can handle multiple orders at a time.

   This design is a complex version of Design 2 with multiple workflows.

We choose to model Design 3 because it has a mixture of sequential and concurrent processing, and is simple and realistic.

There are three possible designs for the reconfiguration of the workflow:

1. The reconfiguration consists of a single action.
2. The reconfiguration consists of multiple actions performed sequentially.
3. The reconfiguration consists of multiple actions performed concurrently.

We choose to model Design 3 because it provides the maximum scope for analyzing interactions between application and reconfiguration actions.

5.3 Modelling the Workflow

We now formulate Design 3 of both configurations of the workflow and Design 3 of the reconfiguration in basic CCS^{dp}. The sets of possible customer identifiers, product identifiers and order identifiers is assumed to be finite.

Let C be the set of possible customer identifiers,
let I be the set of possible product identifiers,
let O be the set of possible order identifiers,
such that $|C|, |I|, |O| \in \mathbb{N}^+$

5.3.1 Modelling Configuration 1

Configuration 1 consists of a collection of activities. Each activity consists of a collection of actions, which are modelled as actions of processes in $\mathcal{P}$. For example, the actions of the `Order Receipt` activity are modelled as the actions of the *REC*

process. Actions of the `Evaluation` activity are modelled as actions of the processes *IC*, *ICH*, *CC* and *CCH*. The `Rejection` activity is modelled as the action $\overline{Reject}_{c,i,o}$ in the processes *ICH* and *CCH*. The `Confirmation` activity is modelled as the action $\overline{Confirm}_{c,i,o}$ in the *ARCH* process.

Configuration 1 of the workflow is denoted by the process *WORKFLOW*, and
$WORKFLOW \triangleq REC \mid IC \mid ICH \mid CC \mid CCH \mid BILL \mid SHIP \mid BSH \mid ARC \mid ARCH$

$REC \triangleq \sum_{c\in C,i\in I,o\in O} Receipt_{c,i,o}.(WORKFLOW \mid \overline{InventoryCheck}_{c,i,o})$
and denotes the `Order Receipt` activity.

Notice that the subscripted actions (such as $Receipt_{c,i,o}$) are distinct.
Thus, $Receipt_{c,i,o} = Receipt_{c',i',o'} \iff c = c' \wedge i = i' \wedge o = o'$
By convention, we omit the 0 process at the end of a trace of actions by a process.

$IC \triangleq \sum_{c\in C,i\in I,o\in O} InventoryCheck_{c,i,o}.\tau.(\overline{InventoryCheckNotOK}_{c,i,o} + \overline{InventoryCheckOK}_{c,i,o})$
and denotes the `Inventory Check` action in `Evaluation`.

$ICH \triangleq \sum_{c\in C,i\in I,o\in O} InventoryCheckNotOK_{c,i,o}.\overline{Reject}_{c,i,o} + InventoryCheckOK_{c,i,o}.\overline{CreditCheck}_{c,i,o}$
and denotes actions in `Evaluation` and `Rejection`.

$CC \triangleq \sum_{c\in C,i\in I,o\in O} CreditCheck_{c,i,o}.\tau.(\overline{CreditCheckNotOK}_{c,i,o} + \overline{CreditCheckOK}_{c,i,o})$
and denotes the `Internal Credit Check` action in `Evaluation`.

$CCH \triangleq \sum_{c\in C,i\in I,o\in O} CreditCheckNotOK_{c,i,o}.\overline{Reject}_{c,i,o} + CreditCheckOK_{c,i,o}.(\overline{Bill}_{c,i,o} \mid \overline{Ship}_{c,i,o})$
and denotes actions in `Evaluation` and `Rejection`.

$BILL \triangleq \sum_{c\in C,i\in I,o\in O} Bill_{c,i,o}.\tau.\overline{BillOK}_{c,i,o}$
and denotes the `Billing` activity.

$SHIP \triangleq \sum_{c\in C,i\in I,o\in O} Ship_{c,i,o}.\tau.\overline{ShipOK}_{c,i,o}$
and denotes the `Shipping` activity.

$BSH \triangleq \sum_{c\in C,i\in I,o\in O} BillOK_{c,i,o}.ShipOK_{c,i,o}.\overline{Archive}_{c,i,o} + ShipOK_{c,i,o}.BillOK_{c,i,o}.\overline{Archive}_{c,i,o}$
and denotes an action in `Archiving`.

$ARC \triangleq \sum_{c\in C,i\in I,o\in O} Archive_{c,i,o}.\tau.\overline{ArchiveOK}_{c,i,o}$
and denotes the main `Archiving` activity.

$ARCH \triangleq \sum_{c\in C,i\in I,o\in O} ArchiveOK_{c,i,o}.\overline{Confirm}_{c,i,o}$
and denotes the `Confirmation` activity.

$\therefore$ *WORKFLOW* =
$\sum_{c\in C,i\in I,o\in O} Receipt_{c,i,o}.(WORKFLOW \mid \overline{InventoryCheck}_{c,i,o})$
$\mid \sum_{c\in C,i\in I,o\in O} InventoryCheck_{c,i,o}.\tau.(\overline{InventoryCheckNotOK}_{c,i,o} + \overline{InventoryCheckOK}_{c,i,o})$
$\mid \sum_{c\in C,i\in I,o\in O} InventoryCheckNotOK_{c,i,o}.\overline{Reject}_{c,i,o} + InventoryCheckOK_{c,i,o}.\overline{CreditCheck}_{c,i,o}$
$\mid \sum_{c\in C,i\in I,o\in O} CreditCheck_{c,i,o}.\tau.(\overline{CreditCheckNotOK}_{c,i,o} + \overline{CreditCheckOK}_{c,i,o})$
$\mid \sum_{c\in C,i\in I,o\in O} CreditCheckNotOK_{c,i,o}.\overline{Reject}_{c,i,o} + CreditCheckOK_{c,i,o}.(\overline{Bill}_{c,i,o} \mid \overline{Ship}_{c,i,o})$
$\mid \sum_{c\in C,i\in I,o\in O} Bill_{c,i,o}.\tau.\overline{BillOK}_{c,i,o}$
$\mid \sum_{c\in C,i\in I,o\in O} Ship_{c,i,o}.\tau.\overline{ShipOK}_{c,i,o}$
$\mid \sum_{c\in C,i\in I,o\in O} BillOK_{c,i,o}.ShipOK_{c,i,o}.\overline{Archive}_{c,i,o} + ShipOK_{c,i,o}.BillOK_{c,i,o}.\overline{Archive}_{c,i,o}$
$\mid \sum_{c\in C,i\in I,o\in O} Archive_{c,i,o}.\tau.\overline{ArchiveOK}_{c,i,o}$
$\mid \sum_{c\in C,i\in I,o\in O} ArchiveOK_{c,i,o}.\overline{Confirm}_{c,i,o}$

The execution of Configuration 1 of the workflow is modelled as transitions of the *WORKFLOW* process. The following example shows a sequence of transitions of *WORKFLOW* that models the successful completion of an order in Configuration 1.

WORKFLOW
$\xrightarrow{Receipt_{c,i,o}}$ $WORKFLOW \mid \overline{InventoryCheck}_{c,i,o} \mid IC \mid$
ICH | CC | CCH | BILL | SHIP | BSH | ARC | ARCH
$\xrightarrow{\tau}$ $WORKFLOW \mid \tau.(\overline{InventoryCheckNotOK}_{c,i,o} + \overline{InventoryCheckOK}_{c,i,o}) \mid$
ICH | CC | CCH | BILL | SHIP | BSH | ARC | ARCH
$\xrightarrow{\tau}$ $WORKFLOW \mid (\overline{InventoryCheckNotOK}_{c,i,o} + \overline{InventoryCheckOK}_{c,i,o}) \mid ICH \mid$
CC | CCH | BILL | SHIP | BSH | ARC | ARCH
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{CreditCheck}_{c,i,o} \mid CC \mid$
CCH | BILL | SHIP | BSH | ARC | ARCH
$\xrightarrow{\tau}$ $WORKFLOW \mid \tau.(\overline{CreditCheckNotOK}_{c,i,o} + \overline{CreditCheckOK}_{c,i,o}) \mid$
CCH | BILL | SHIP | BSH | ARC | ARCH
$\xrightarrow{\tau}$ $WORKFLOW \mid (\overline{CreditCheckNotOK}_{c,i,o} + \overline{CreditCheckOK}_{c,i,o}) \mid CCH \mid$
BILL | SHIP | BSH | ARC | ARCH
$\xrightarrow{\tau}$ $WORKFLOW \mid (\overline{Bill}_{c,i,o} \mid \overline{Ship}_{c,i,o}) \mid BILL \mid SHIP \mid$
BSH | ARC | ARCH
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{Ship}_{c,i,o} \mid \tau.\overline{BillOK}_{c,i,o} \mid SHIP \mid$
BSH | ARC | ARCH
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{Ship}_{c,i,o} \mid \overline{BillOK}_{c,i,o} \mid SHIP \mid$
BSH | ARC | ARCH
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{BillOK}_{c,i,o} \mid \tau.\overline{ShipOK}_{c,i,o} \mid$
BSH | ARC | ARCH
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{BillOK}_{c,i,o} \mid \overline{ShipOK}_{c,i,o} \mid BSH \mid$
ARC | ARCH

$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{BillOK}_{c,i,o} \mid BillOK_{c,i,o}.\overline{Archive}_{c,i,o} \mid$
ARC | ARCH
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{Archive}_{c,i,o} \mid ARC \mid$
ARCH
$\xrightarrow{\tau}$ $WORKFLOW \mid \tau.\overline{ArchiveOK}_{c,i,o} \mid$
ARCH
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{ArchiveOK}_{c,i,o} \mid ARCH$
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{Confirm}_{c,i,o}$
$\xrightarrow{\overline{Confirm}_{c,i,o}}$ $WORKFLOW \mid 0$

5.3.2 Modelling Configuration 2

Configuration 2 is different in structure from Configuration 1, although some of the activities are unchanged (such as `Billing` and `Shipping`), and this difference is reflected in the processes used to model Configuration 2. For example, the *REC* process must be different in order to spawn a workflow with Configuration 2. The *CCH* process must be different in order to ensure that `Shipping` and `Billing` are performed in sequence. A new process *ECC* is needed in order to model the new action (`External Credit Check`) in the `Evaluation` activity. The *BSH* process must be different since the serialization of `Shipping` and `Billing` results in only one output for `Archiving`. Removal of the `Confirmation` activity implies the `Archiving` activity no longer produces an output, and (therefore) the *ARC* process must be different and the *ARCH* process is no longer needed.

Configuration 2 of the workflow is denoted by the process $WORKFLOW'$, and
$WORKFLOW' \triangleq REC' \mid IC \mid ICH \mid CC \mid CCH' \mid BILL \mid SHIP \mid BSH' \mid ARC'$

$REC' \triangleq \sum_{c\in C,i\in I,o\in O} Receipt_{c,i,o}.(WORKFLOW' \mid \overline{InventoryCheck}_{c,i,o})$
and denotes the `Order Receipt` activity.

$CCH' \triangleq \sum_{c\in C,i\in I,o\in O} CreditCheckNotOK_{c,i,o}.(\overline{ExtCreditCheck}_{c,i,o} \mid ECC) +$
$\text{CreditCheckOK}_{c,i,o}.\overline{Ship}_{c,i,o}.ShipOK_{c,i,o}.\overline{Bill}_{c,i,o}$
and denotes changes in `Evaluation` that initiate an `External Credit Check` and initiate `Shipping` and `Billing` sequentially.

$ECC \triangleq$
$\sum_{c\in C,i\in I,o\in O} ExtCreditCheckNotOK_{c,i,o}.\overline{Reject}_{c,i,o} + ExtCreditCheckOK_{c,i,o}.\overline{Ship}_{c,i,o}.ShipOK_{c,i,o}.\overline{Bill}_{c,i,o}$
and denotes the new `External Credit Check` handling action in `Evaluation` and the sequential initiation of `Shipping` and `Billing`.

$BSH' \triangleq \sum_{c\in C,i\in I,o\in O} BillOK_{c,i,o}.\overline{Archive}_{c,i,o}$
and denotes a changed action in `Archiving` that handles the serialization of `Shipping` and `Billing`.

$ARC' \triangleq \sum_{c\in C,i\in I,o\in O} Archive_{c,i,o}.\tau$
and denotes the main `Archiving` activity that now terminates the workflow.

$\therefore WORKFLOW' =$
$\sum_{c\in C,i\in I,o\in O} Receipt_{c,i,o}.(WORKFLOW' \mid \overline{InventoryCheck}_{c,i,o})$
$\mid \sum_{c\in C,i\in I,o\in O} InventoryCheck_{c,i,o}.\tau.(\overline{InventoryCheckNotOK}_{c,i,o} + \overline{InventoryCheckOK}_{c,i,o})$
$\mid \sum_{c\in C,i\in I,o\in O} InventoryCheckNotOK_{c,i,o}.\overline{Reject}_{c,i,o} + InventoryCheckOK_{c,i,o}.\overline{CreditCheck}_{c,i,o}$
$\mid \sum_{c\in C,i\in I,o\in O} CreditCheck_{c,i,o}.\tau.(\overline{CreditCheckNotOK}_{c,i,o} + \overline{CreditCheckOK}_{c,i,o})$
$\mid \sum_{c\in C,i\in I,o\in O} CreditCheckNotOK_{c,i,o}.(\overline{ExtCreditCheck}_{c,i,o} \mid ECC) +$
$\quad \text{CreditCheckOK}_{c,i,o}.\overline{Ship}_{c,i,o}.ShipOK_{c,i,o}.\overline{Bill}_{c,i,o}$
$\mid \sum_{c\in C,i\in I,o\in O} Bill_{c,i,o}.\tau.\overline{BillOK}_{c,i,o}$
$\mid \sum_{c\in C,i\in I,o\in O} Ship_{c,i,o}.\tau.\overline{ShipOK}_{c,i,o}$
$\mid \sum_{c\in C,i\in I,o\in O} BillOK_{c,i,o}.\overline{Archive}_{c,i,o}$
$\mid \sum_{c\in C,i\in I,o\in O} Archive_{c,i,o}.\tau$

The execution of Configuration 2 of the workflow is modelled as transitions of the *WORKFLOW′* process. The following example shows a sequence of transitions of *WORKFLOW′* that models the successful completion of an order in Configuration 2.

WORKFLOW′
$\xrightarrow{Receipt_{c,i,o}}$ $WORKFLOW' \mid \overline{InventoryCheck}_{c,i,o} \mid IC \mid$
ICH | CC | CCH′ | BILL | SHIP | BSH′ | ARC′
$\xrightarrow{\tau}$ $WORKFLOW' \mid \tau.(\overline{InventoryCheckNotOK}_{c,i,o} + \overline{InventoryCheckOK}_{c,i,o}) \mid$
ICH | CC | CCH′ | BILL | SHIP | BSH′ | ARC′
$\xrightarrow{\tau}$ $WORKFLOW' \mid (\overline{InventoryCheckNotOK}_{c,i,o} + \overline{InventoryCheckOK}_{c,i,o}) \mid ICH \mid$
CC | CCH′ | BILL | SHIP | BSH′ | ARC′
$\xrightarrow{\tau}$ $WORKFLOW' \mid \overline{CreditCheck}_{c,i,o} \mid CC \mid$
CCH′ | BILL | SHIP | BSH′ | ARC′
$\xrightarrow{\tau}$ $WORKFLOW' \mid \tau.(\overline{CreditCheckNotOK}_{c,i,o} + \overline{CreditCheckOK}_{c,i,o}) \mid$
CCH′ | BILL | SHIP | BSH′ | ARC′
$\xrightarrow{\tau}$ $WORKFLOW' \mid (\overline{CreditCheckNotOK}_{c,i,o} + \overline{CreditCheckOK}_{c,i,o}) \mid CCH' \mid$
BILL | SHIP | BSH′ | ARC′
$\xrightarrow{\tau}$ $WORKFLOW' \mid \overline{ExtCreditCheck}_{c,i,o} \mid ECC \mid$
BILL | SHIP | BSH′ | ARC′
$\xrightarrow{\overline{ExtCreditCheck}_{c,i,o}}$ $WORKFLOW' \mid ECC \mid$
BILL | SHIP | BSH′ | ARC′
$\xrightarrow{\tau}$ $WORKFLOW' \mid \overline{Ship}_{c,i,o}.ShipOK_{c,i,o}.\overline{Bill}_{c,i,o} \mid SHIP \mid$
BILL | BSH′ | ARC′ (assuming the environment performs a $\overline{ExtCreditCheckOK}_{c,i,o}$)
$\xrightarrow{\tau}$ $WORKFLOW' \mid ShipOK_{c,i,o}.\overline{Bill}_{c,i,o} \mid \tau.\overline{ShipOK}_{c,i,o} \mid$
BILL | BSH′ | ARC′

$\xrightarrow{\tau}$ $WORKFLOW' \mid ShipOK_{c,i,o}.\overline{Bill}_{c,i,o} \mid \overline{ShipOK}_{c,i,o} \mid$
BILL | BSH′ | ARC′
$\xrightarrow{\tau}$ $WORKFLOW' \mid \overline{Bill}_{c,i,o} \mid BILL \mid$
BSH′ | ARC′
$\xrightarrow{\tau}$ $WORKFLOW' \mid \tau.\overline{BillOK}_{c,i,o} \mid$
BSH′ | ARC′
$\xrightarrow{\tau}$ $WORKFLOW' \mid \overline{BillOK}_{c,i,o} \mid BSH' \mid$
ARC′
$\xrightarrow{\tau}$ $WORKFLOW' \mid \overline{Archive}_{c,i,o} \mid ARC'$
$\xrightarrow{\tau}$ $WORKFLOW' \mid \tau$
$\xrightarrow{\tau}$ $WORKFLOW' \mid 0$

5.3.3 Modelling the Reconfiguration

The workflow is reconfigured by a reconfiguration manager (modelled by the process *RM*) that is activated after receiving a triggering message and reconfigures the workflow from Configuration 1 to Configuration 2. There are two different ways of reconfiguring the workflow (depending on its state of execution), and they are triggered by different messages. The *trigger*1 guard models receipt of the message that is used to trigger reconfiguration of the workflow if it has **not** yet started to execute. After the release of *trigger*1, *RM* replaces the process *WORKFLOW* with the process *WORKFLOW*′, and replicates itself. The *trigger*2 guard models receipt of the message that is used to trigger reconfiguration of the workflow if it has started to execute. After the release of *trigger*2, *RM* deletes the process *ARCH*, replaces the processes *CCH*, *BSH* and *ARC* with the processes *CCH*′, *BSH*′ and *ARC*′ (respectively), and replicates itself.

The reconfiguration manager is denoted by the process *RM*, and
$RM \triangleq trigger1.\left(\frac{WORKFLOW'}{WORKFLOW} \mid RM\right) + trigger2.\left(\frac{CCH'}{CCH} \mid \frac{BSH'}{BSH} \mid \frac{ARC'}{ARC} \mid \frac{0}{ARCH} \mid RM\right)$

Thus, *RM* performs two operations of unplanned process reconfiguration, namely, the deletion and replacement of processes that are not designed to be reconfigured.

The process of reconfiguration of the workflow is expressed as reactions between *WORKFLOW* and *RM* in the expression $WORKFLOW \mid RM$. The step through which the reconfiguring process *RM* is added to the context of the process *WORKFLOW*, that is, the step through which *WORKFLOW* becomes $WORKFLOW \mid RM$, is performed outside basic CCS^{dp}, and thereby captures the fact that the reconfiguration is unplanned (as mentioned previously in Section 4.1.1).

5.4 Evaluation using the Reconfiguration Requirements

One of the purposes of modelling is analysis. Therefore, one way to evaluate basic CCS^{dp} is to examine how well the expression $WORKFLOW \mid RM$ supports the verification of the reconfiguration requirements.

5.4.1 Verification of Requirement 2

Reconfiguration requirement 2 states: 'No order should be significantly delayed by the reconfiguration'.

This requirement has a notion of duration. However, duration is not expressed in $WORKFLOW \mid RM$ (because duration cannot be expressed in basic CCS^{dp}). Therefore, reconfiguration requirement 2 cannot be verified.

5.4.2 Verification of Requirements 1, 3, 4 and 5

There are two standard approaches to the analysis of process expressions: equational reasoning and model checking. Equational reasoning works by equating process expressions that are congruent in structure or in behaviour, which requires an invariant property of a model that can be represented as a process expression. However, in the workflow, the invariant property is not clear. Furthermore, the simplicity of the workflow made it difficult to find simple expressions that could be equated for the verification of the requirements. Therefore, using equational reasoning based on $\sim_{dp}$ to verify reconfiguration requirements 1, 3, 4 and 5 is problematic.

In contrast, reconfiguration requirements 1, 3, 4 and 5 can all be expressed using a temporal logic and verified by model checking (see [MADB12]). Furthermore, model checking is a more widely applicable verification technique than equational reasoning. Therefore, it is desirable that expressions in CCS^{dp} are amenable to model checking, and we outline the argument that CCS^{dp} processes are amenable to model checking.

5.4.2.1 Model Checking CCS^{dp} Processes

Model checking is a technique designed to verify the correctness of concurrent systems with a finite number number of states and transitions [CGP02]. It has a number of advantages: it is highly amenable to automation [BBF+01], it can identify an error state of a system (if any) where the system has failed to meet the specification, and it can check partial or incomplete representations of a system. Its principal disadvantage is the 'state explosion' problem, although research has made progress on this issue with the use of binary decision diagrams.

In order to describe model checking, the following definitions are needed. For a more detailed explanation of the definitions, see [Rot00] and [CGP02].

Definition 5.4.1 *A model is a non-empty mathematical structure that satisfies a set of formulae.*

Definition 5.4.2 *An atomic proposition is a formula whose Boolean value can be calculated from a state alone.*

Definition 5.4.3 *Given a set of atomic propositions $\mathscr{A}$, a Kripke structure over $\mathscr{A}$ is a mathematical structure of the form (S, S_0, R, L) such that:*

1. *S is a finite set of states.*
2. *S_0 is the set of initial states, and $S_0 \subseteq S$.*
3. *R is a total transition relation on S, and $R \subseteq S \; x \; S$.*
4. *$L : S \longrightarrow \mathbb{P}\mathscr{A}$ is a total function on S that labels each state in S with the set of atomic propositions in $\mathscr{A}$ that are true in that state.*

For some systems, the set of initial states is not important, and their Kripke structures are simplified to the form (S, R, L). A path π in a Kripke structure from a state s is an infinite sequence of states, that is, $\pi = s_0 s_1 s_2 ...$ such that $s_0 = s \; \wedge \; \forall i \in \mathbb{N} \, ((s_i, s_{i+1}) \in R)$.

Definition 5.4.4 *Model checking is an activity that decides whether or not a Kripke structure is a model of a set of formulae expressed in a temporal logic.*

If a system is represented by a Kripke structure (S, R, L), and a system failure is expressed as a formula f in a temporal logic, then model checking can identify the states in S (if any) that satisfy f, and thereby identify the states of the system (if

any) where the system fails as defined.

The first step in model checking is the production of a Kripke structure designed for the verification of a requirement. Now, processes are essentially states with transitions. Therefore, it is conceptually feasible to produce a state transition system from a CCS^{dp} expression. Our restriction (in Section 4.4.2) that models consist of a finite number of processes and process definitions ensures a finite number of states. The restriction that every process in $\mathcal{P}$ is the result of one or more applications of the production rules of $\mathcal{P}^+$ or $\mathcal{P}^0$ with finite depth of inference, and the equivalence of $\sim_{of}$, ensure a finite number of transitions for each state. Therefore, a process expression in basic CCS^{dp} corresponds to a finite state transition system. Each state can be associated with a set of atomic propositions. Therefore, it is conceptually feasible to produce a Kripke structure from a basic CCS^{dp} expression. More specifically, the state space of a Kripke structure can be partitioned into sets of dimensions, and a process p can be mapped to coordinates in dimensions determined by $fdrdepth(p)$.

For example, the process *WORKFLOW* can be mapped to the first two coordinates of the state *(order_id, workflow_state, reconfiguration_state)* in a Kripke structure, because $fdrdepth(WORKFLOW) = 0$; and the process *RM* (after the release of *trigger*1 or *trigger*2) can be mapped to the third coordinate, because
$fdrdepth\left(\frac{WORKFLOW'}{WORKFLOW} \mid RM\right) = 1$ and
$fdrdepth\left(\frac{CCH'}{CCH} \mid \frac{BSH'}{BSH} \mid \frac{ARC'}{ARC} \mid \frac{0}{ARCH} \mid RM\right) = 1.$

Representing multiple *WORKFLOW* processes executing in parallel is not problematic (in principle), because multiple Kripke structures can be composed to produce a single Kripke structure (although the state space increases considerably). Furthermore, the requirements can be verified by model checking a single *WORKFLOW* in which interactions with the other *WORKFLOW* processes are represented by transitions that change the *reconfiguration_state* coordinate of the given *WORKFLOW*.

The verification of a requirement can be performed incrementally: the $WORKFLOW \mid RM$ process is mapped to its state in the Kripke structure, and the value of the temporal logic formula is calculated. Then, for each transition, the successor process is mapped to its state in the Kripke structure, and the value of the temporal logic formula is calculated. The second step is repeated until the

formula's Boolean value is calculated, which either verifies or refutes the requirement. Notice that the commutativity, associativity and the congruence property of $\sim_{dp}$ (see Theorem 4.3.12, Theorem 4.3.14 and Theorem 4.3.10) allow states in the Kripke structure to be identified, which simplifies the Kripke structure.

5.4.2.2 Problems identified by Model Checking WORKFLOW | RM

Our initial experiments in model checking were informal and done manually, because of the lack of a model checker that performs process matching (to our knowledge). We focused on the termination requirement on workflows and on reconfiguration, and traced the transitions of the process *WORKFLOW* | *RM* directly in basic CCS^{dp} for convenience. The termination requirement on workflows can be taken as the requirement that every sequence of transitions of every instance of the *WORKFLOW* and *WORKFLOW'* processes must reach 0 (for a successfully completed order, and ignoring the spawned *WORKFLOW* or *WORKFLOW'* process). The termination requirement on reconfiguration can be taken as the requirement that every sequence of transitions of every instance of the *RM* process that has its trigger released must reach 0 (ignoring the spawned *RM* process). The termination requirements on workflows and on reconfiguration can be combined.

The following example shows a sequence of transitions of *WORKFLOW* | *RM* that terminates as required. The workflow starts its execution in Configuration 1 and changes its configuration during execution to Configuration 2.

$$
\begin{aligned}
&WORKFLOW \mid RM \\
&\xrightarrow{Receipt_{c,i,o}} WORKFLOW \mid \overline{InventoryCheck}_{c,i,o} \mid IC \mid \\
&\text{ICH} \mid \text{CC} \mid \text{CCH} \mid \text{BILL} \mid \text{SHIP} \mid \text{BSH} \mid \text{ARC} \mid \text{ARCH} \mid \text{RM} \\
&\xrightarrow{\tau} WORKFLOW \mid \tau.(\overline{InventoryCheckNotOK}_{c,i,o} + \overline{InventoryCheckOK}_{c,i,o}) \mid \\
&\text{ICH} \mid \text{CC} \mid \text{CCH} \mid \text{BILL} \mid \text{SHIP} \mid \text{BSH} \mid \text{ARC} \mid \text{ARCH} \mid \text{RM} \\
&\xrightarrow{\tau} WORKFLOW \mid (\overline{InventoryCheckNotOK}_{c,i,o} + \overline{InventoryCheckOK}_{c,i,o}) \mid ICH \mid \\
&\text{CC} \mid \text{CCH} \mid \text{BILL} \mid \text{SHIP} \mid \text{BSH} \mid \text{ARC} \mid \text{ARCH} \mid \text{RM} \\
&\xrightarrow{\tau} WORKFLOW \mid \overline{CreditCheck}_{c,i,o} \mid CC \mid \\
&\text{CCH} \mid \text{BILL} \mid \text{SHIP} \mid \text{BSH} \mid \text{ARC} \mid \text{ARCH} \mid \text{RM} \\
&\xrightarrow{trigger2} WORKFLOW \mid \overline{CreditCheck}_{c,i,o} \mid CC \mid \\
&\text{CCH} \mid \text{BILL} \mid \text{SHIP} \mid \text{BSH} \mid \text{ARC} \mid \text{ARCH} \mid \left(\tfrac{CCH'}{CCH} \mid \tfrac{BSH'}{BSH} \mid \tfrac{ARC'}{ARC} \mid \tfrac{0}{ARCH} \mid RM\right) \\
&\xrightarrow{\tau} WORKFLOW \mid \overline{CreditCheck}_{c,i,o} \mid CC \mid \\
&\text{CCH'} \mid \text{BILL} \mid \text{SHIP} \mid \text{BSH} \mid \text{ARC} \mid \text{ARCH} \mid \left(\tfrac{BSH'}{BSH} \mid \tfrac{ARC'}{ARC} \mid \tfrac{0}{ARCH} \mid RM\right) \\
&\xrightarrow{\tau} WORKFLOW \mid \overline{CreditCheck}_{c,i,o} \mid CC \mid \\
&\text{CCH'} \mid \text{BILL} \mid \text{SHIP} \mid \text{BSH'} \mid \text{ARC} \mid \text{ARCH} \mid \left(\tfrac{ARC'}{ARC} \mid \tfrac{0}{ARCH} \mid RM\right)
\end{aligned}
$$

$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{CreditCheck}_{c,i,o} \mid CC \mid$
CCH′ | BILL | SHIP | BSH′ | ARC′ | ARCH | $\left(\frac{0}{ARCH} \mid RM\right)$
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{CreditCheck}_{c,i,o} \mid CC \mid$
CCH′ | BILL | SHIP | BSH′ | ARC′ | 0 | RM
$\xrightarrow{\tau}$ $WORKFLOW \mid \tau.(\overline{CreditCheckNotOK}_{c,i,o} + \overline{CreditCheckOK}_{c,i,o}) \mid$
CCH′ | BILL | SHIP | BSH′ | ARC′ | 0 | RM
$\xrightarrow{\tau}$ $WORKFLOW \mid (\overline{CreditCheckNotOK}_{c,i,o} + \overline{CreditCheckOK}_{c,i,o}) \mid CCH' \mid$
BILL | SHIP | BSH′ | ARC′ | 0 | RM
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{Ship}_{c,i,o}.ShipOK_{c,i,o}.\overline{Bill}_{c,i,o} \mid SHIP \mid$
BILL | BSH′ | ARC′ | 0 | RM
$\xrightarrow{\tau}$ $WORKFLOW \mid ShipOK_{c,i,o}.\overline{Bill}_{c,i,o} \mid \tau.\overline{ShipOK}_{c,i,o} \mid$
BILL | BSH′ | ARC′ | 0 | RM
$\xrightarrow{\tau}$ $WORKFLOW \mid ShipOK_{c,i,o}.\overline{Bill}_{c,i,o} \mid \overline{ShipOK}_{c,i,o} \mid$
BILL | BSH′ | ARC′ | 0 | RM
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{Bill}_{c,i,o} \mid BILL \mid$
BSH′ | ARC′ | 0 | RM
$\xrightarrow{\tau}$ $WORKFLOW \mid \tau.\overline{BillOK}_{c,i,o} \mid$
BSH′ | ARC′ | 0 | RM
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{BillOK}_{c,i,o} \mid BSH' \mid$
ARC′ | 0 | RM
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{Archive}_{c,i,o} \mid ARC' \mid$
0 | RM
$\xrightarrow{\tau}$ $WORKFLOW \mid \tau \mid 0 \mid RM$
$\xrightarrow{\tau}$ $WORKFLOW \mid 0 \mid 0 \mid RM$

The following example shows another sequence of transitions of $WORKFLOW \mid RM$ where the workflow fails to terminate (i.e. deadlocks) due to non-determinism of the transitions.

$WORKFLOW \mid RM$
$\xrightarrow{Receipt_{c,i,o}}$ $WORKFLOW \mid \overline{InventoryCheck}_{c,i,o} \mid IC \mid$
ICH | CC | CCH | BILL | SHIP | BSH | ARC | ARCH | RM
$\xrightarrow{\tau}$ $WORKFLOW \mid \tau.(\overline{InventoryCheckNotOK}_{c,i,o} + \overline{InventoryCheckOK}_{c,i,o}) \mid$
ICH | CC | CCH | BILL | SHIP | BSH | ARC | ARCH | RM
$\xrightarrow{\tau}$ $WORKFLOW \mid (\overline{InventoryCheckNotOK}_{c,i,o} + \overline{InventoryCheckOK}_{c,i,o}) \mid ICH \mid$
CC | CCH | BILL | SHIP | BSH | ARC | ARCH | RM
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{CreditCheck}_{c,i,o} \mid CC \mid$
CCH | BILL | SHIP | BSH | ARC | ARCH | RM
$\xrightarrow{trigger2}$ $WORKFLOW \mid \overline{CreditCheck}_{c,i,o} \mid CC \mid$
CCH | BILL | SHIP | BSH | ARC | ARCH | $\left(\frac{CCH'}{CCH} \mid \frac{BSH'}{BSH} \mid \frac{ARC'}{ARC} \mid \frac{0}{ARCH} \mid RM\right)$

$\xrightarrow{\tau}$ $WORKFLOW \mid \tau.(\overline{CreditCheckNotOK}_{c,i,o} + \overline{CreditCheckOK}_{c,i,o}) \mid$
CCH $\mid$ BILL $\mid$ SHIP $\mid$ BSH $\mid$ ARC $\mid$ ARCH $\mid$ $\left(\frac{CCH'}{CCH} \mid \frac{BSH'}{BSH} \mid \frac{ARC'}{ARC} \mid \frac{0}{ARCH} \mid RM\right)$
$\xrightarrow{\tau}$ $WORKFLOW \mid (\overline{CreditCheckNotOK}_{c,i,o} + \overline{CreditCheckOK}_{c,i,o}) \mid CCH \mid$
BILL $\mid$ SHIP $\mid$ BSH $\mid$ ARC $\mid$ ARCH $\mid$ $\left(\frac{CCH'}{CCH} \mid \frac{BSH'}{BSH} \mid \frac{ARC'}{ARC} \mid \frac{0}{ARCH} \mid RM\right)$
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{Bill}_{c,i,o} \mid \overline{Ship}_{c,i,o} \mid BILL \mid SHIP \mid$
BSH $\mid$ ARC $\mid$ ARCH $\mid$ $\left(\frac{CCH'}{CCH} \mid \frac{BSH'}{BSH} \mid \frac{ARC'}{ARC} \mid \frac{0}{ARCH} \mid RM\right)$
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{Bill}_{c,i,o} \mid \overline{Ship}_{c,i,o} \mid BILL \mid SHIP \mid$
BSH′ $\mid$ ARC $\mid$ ARCH $\mid$ $\left(\frac{CCH'}{CCH} \mid \frac{ARC'}{ARC} \mid \frac{0}{ARCH} \mid RM\right)$
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{Bill}_{c,i,o} \mid \overline{Ship}_{c,i,o} \mid BILL \mid SHIP \mid$
BSH′ $\mid$ ARC′ $\mid$ ARCH $\mid$ $\left(\frac{CCH'}{CCH} \mid \frac{0}{ARCH} \mid RM\right)$
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{Bill}_{c,i,o} \mid \overline{Ship}_{c,i,o} \mid BILL \mid SHIP \mid$
BSH′ $\mid$ ARC′ $\mid$ $\left(\frac{CCH'}{CCH} \mid RM\right)$
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{Ship}_{c,i,o} \mid \tau.\overline{BillOK}_{c,i,o} \mid SHIP \mid$
BSH′ $\mid$ ARC′ $\mid$ $\left(\frac{CCH'}{CCH} \mid RM\right)$
$\xrightarrow{\tau}$ $WORKFLOW \mid \tau.\overline{BillOK}_{c,i,o} \mid \tau.\overline{ShipOK}_{c,i,o} \mid$
BSH′ $\mid$ ARC′ $\mid$ $\left(\frac{CCH'}{CCH} \mid RM\right)$
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{BillOK}_{c,i,o} \mid \tau.\overline{ShipOK}_{c,i,o} \mid$
BSH′ $\mid$ ARC′ $\mid$ $\left(\frac{CCH'}{CCH} \mid RM\right)$
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{BillOK}_{c,i,o} \mid \overline{ShipOK}_{c,i,o} \mid BSH' \mid$
ARC′ $\mid$ $\left(\frac{CCH'}{CCH} \mid RM\right)$
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{ShipOK}_{c,i,o} \mid \overline{Archive}_{c,i,o} \mid ARC' \mid \left(\frac{CCH'}{CCH} \mid RM\right)$
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{ShipOK}_{c,i,o} \mid \tau \mid \left(\frac{CCH'}{CCH} \mid RM\right)$
$\xrightarrow{\tau}$ $WORKFLOW \mid \overline{ShipOK}_{c,i,o} \mid \left(\frac{CCH'}{CCH} \mid RM\right)$

Notice that the original *WORKFLOW* fails to terminate because there is no process to synchronise with its action $\overline{ShipOK}_{c,i,o}$. The original *RM* failed to replace *CCH* in the original *WORKFLOW*, but it can still terminate by replacing *CCH* in the new (i.e. wrong) *WORKFLOW*.

The problems caused by non-deterministic transitions can be minimized by using Design 1 for the reconfiguration. That is, by removing the term in *RM* guarded by *trigger*2, so that $RM \triangleq trigger.\left(\frac{WORKFLOW'}{WORKFLOW} \mid RM\right)$. However, atomic reconfiguration is a restrictive form of reconfiguration. An alternative way of handling non-deterministic transitions is to use a priority scheme for processes or transitions.

A second problem that arose during our informal model checking was the replacement of the wrong process by a fraction. For example, suppose we have two *WORKFLOW* processes and two *RM* processes in existence at the same time. For clarity, the four processes are distinguished by parentheses, which have no seman-

tic significance because parallel composition is associative in basic CCS^{dp}. Also suppose reconfiguration transitions have a higher priority than other transitions. Then the following sequence of transitions is possible.

$$
\begin{aligned}
&WORKFLOW \mid RM \\
&\xrightarrow{Receipt_{c,i,o}} WORKFLOW \mid \\
&(\overline{InventoryCheck}_{c,i,o} \mid IC \mid ICH \mid CC \mid CCH \mid BILL \mid SHIP \mid BSH \mid ARC \mid ARCH) \mid RM \\
&\xrightarrow{\tau} WORKFLOW \mid \\
&(\tau.(\overline{InventoryCheckNotOK}_{c,i,o} + \overline{InventoryCheckOK}_{c,i,o}) \mid \\
&\text{ICH} \mid \text{CC} \mid \text{CCH} \mid \text{BILL} \mid \text{SHIP} \mid \text{BSH} \mid \text{ARC} \mid \text{ARCH}) \mid \text{RM} \\
&\xrightarrow{\tau} WORKFLOW \mid \\
&((\overline{InventoryCheckNotOK}_{c,i,o} + \overline{InventoryCheckOK}_{c,i,o}) \mid \\
&\text{ICH} \mid \text{CC} \mid \text{CCH} \mid \text{BILL} \mid \text{SHIP} \mid \text{BSH} \mid \text{ARC} \mid \text{ARCH}) \mid \text{RM} \\
&\xrightarrow{\tau} WORKFLOW \mid \\
&(\overline{CreditCheck}_{c,i,o} \mid CC \mid CCH \mid BILL \mid SHIP \mid BSH \mid ARC \mid ARCH) \mid RM \\
&\xrightarrow{trigger2} WORKFLOW \mid \\
&(\overline{CreditCheck}_{c,i,o} \mid CC \mid CCH \mid BILL \mid SHIP \mid BSH \mid ARC \mid ARCH) \mid \\
&\left(\tfrac{CCH'}{CCH} \mid \tfrac{BSH'}{BSH} \mid \tfrac{ARC'}{ARC} \mid \tfrac{0}{ARCH} \mid RM\right) \\
&\xrightarrow{trigger1} WORKFLOW \mid \\
&(\overline{CreditCheck}_{c,i,o} \mid CC \mid CCH \mid BILL \mid SHIP \mid BSH \mid ARC \mid ARCH) \mid \\
&\left(\tfrac{CCH'}{CCH} \mid \tfrac{BSH'}{BSH} \mid \tfrac{ARC'}{ARC} \mid \tfrac{0}{ARCH}\right) \mid \\
&\left(\tfrac{WORKFLOW'}{WORKFLOW} \mid RM\right) \\
&\xrightarrow{\tau} (REC \mid IC \mid ICH \mid CC \mid CCH' \mid BILL \mid SHIP \mid BSH \mid ARC \mid ARCH) \mid \\
&(\overline{CreditCheck}_{c,i,o} \mid CC \mid CCH \mid BILL \mid SHIP \mid BSH \mid ARC \mid ARCH) \mid \\
&\left(\tfrac{BSH'}{BSH} \mid \tfrac{ARC'}{ARC} \mid \tfrac{0}{ARCH}\right) \mid \\
&\left(\tfrac{WORKFLOW'}{WORKFLOW} \mid RM\right) \\
&\xrightarrow{\tau} (REC \mid IC \mid ICH \mid CC \mid CCH' \mid BILL \mid SHIP \mid BSH' \mid ARC \mid ARCH) \mid \\
&(\overline{CreditCheck}_{c,i,o} \mid CC \mid CCH \mid BILL \mid SHIP \mid BSH \mid ARC \mid ARCH) \mid \\
&\left(\tfrac{ARC'}{ARC} \mid \tfrac{0}{ARCH}\right) \mid \\
&\left(\tfrac{WORKFLOW'}{WORKFLOW} \mid RM\right) \\
&\xrightarrow{\tau} (REC \mid IC \mid ICH \mid CC \mid CCH' \mid BILL \mid SHIP \mid BSH' \mid ARC' \mid ARCH) \mid \\
&(\overline{CreditCheck}_{c,i,o} \mid CC \mid CCH \mid BILL \mid SHIP \mid BSH \mid ARC \mid ARCH) \mid \\
&\left(\tfrac{0}{ARCH}\right) \mid \\
&\left(\tfrac{WORKFLOW'}{WORKFLOW} \mid RM\right) \\
&\xrightarrow{\tau} (REC \mid IC \mid ICH \mid CC \mid CCH' \mid BILL \mid SHIP \mid BSH' \mid ARC') \mid \\
&(\overline{CreditCheck}_{c,i,o} \mid CC \mid CCH \mid BILL \mid SHIP \mid BSH \mid ARC \mid ARCH) \mid \\
&0 \mid \\
&\left(\tfrac{WORKFLOW'}{WORKFLOW} \mid RM\right)
\end{aligned}
$$

Thus, the fractions intended to reconfigure the executing workflow have reconfigured the other workflow instead, and the fraction intended to replace the workflow that had not yet started to execute cannot progress. The intention was to reconfigure both workflows, but only one workflow (the wrong one) was reconfigured. One way to solve this problem is to use a process identifier to reconfigure specific processes selectively.

The experience of modelling and informal model checking revealed three other problems. In a process expression with $n + 1$ positive singleton factors, one of which is a fraction, the matching of the denominator of the fraction will require $2^n - 1$ comparisons between the denominator and other factors in the process expression. Therefore, the computational complexity of matching is exponential at each transition. The use of a process identifier can reduce this complexity.

The office workflow is a simple case study. However, the workflow is modelled using up to 10 concurrent processes. We had to be careful to avoid port/action name clashes when defining the processes. More complex case studies are expected to contain many more processes. In such cases, unless there is a way to scope port/action names, unintended reactions between processes will be unavoidable.

Writing processes without parameter passing is clumsy and increases the verbosity of process expressions, which makes them inconvenient to write and to read.

5.5 Strengths and Weaknesses of Basic CCS^{dp}

The case study reveals the following strengths and weaknesses of basic *CCS*^{dp}.

Strengths

1. Unplanned process reconfiguration can be modelled simply and tersely.
2. Process expressions can be verified against requirements by model checking.

Weaknesses

1. No value-passing in process transitions, which increases the verbosity of process expressions.
2. Duration of an action cannot be modelled.

3. No facility for controlling the non-determinism of process transitions.
4. No facility for the selective reconfiguration of a specific process instance.
5. Computational complexity of matching is exponential at each transition.
6. No facility for scoping port/action names.

Chapter 6

Towards Full CCS^{dp}

Contents

In this chapter, we briefly explore modifications of basic CCS^{dp} in order to overcome some of its limitations identified in Chapter 4 and Chapter 5. Specifically, the addition of the restriction operator (ν), which scopes port names and (thereby) supports the modelling of large systems; and a process identification scheme, which supports the selective reconfiguration of specific process instances and (thereby) also reduces the computational complexity of matching.

6.1 Basic CCS^{dp} +ν

We define the syntax, LTS semantics, and the sets of positive and zero processes of basic CCS^{dp} +ν, and make observations.

6.1.1 Syntax

Let $\mathcal{P}_\nu$ be the set of processes in basic CCS^{dp} +ν.

As in basic CCS^{dp}, $\mathcal{N}$ is the countable set of names (e.g. a, b, c) that represent both input ports and input actions of the processes in $\mathcal{P}_\nu$; and $\overline{\mathcal{N}}$ is the countable set of complementary names (e.g. $\overline{a}$, $\overline{b}$, $\overline{c}$) that represent both output ports and output actions of the processes in $\mathcal{P}_\nu$. Let $\mathcal{PN}_\nu$ be the countable set of names (e.g. A, B,

C) of the processes in $\mathcal{P}_\nu$. The sets $\mathcal{N}$, $\overline{\mathcal{N}}$ and $\mathcal{PN}_\nu$ are assumed to be pairwise disjoint.

As in basic CCS^{dp}, the interaction between complementary actions (such as a and $\overline{a}$) is represented by the special action τ, which is internal to a process. By convention, $\forall l \in \mathcal{N}(\bar{\bar{l}} \triangleq l)$. $\mathcal{L}$ is the set of names that represent both ports and actions of the processes in $\mathcal{P}_\nu$, where $\mathcal{L} \triangleq \mathcal{N} \cup \overline{\mathcal{N}}$. $\mathcal{I}$ is the set of input and output ports/actions of the processes in $\mathcal{P}_\nu$, and their internal action (τ), where $\mathcal{I} \triangleq \mathcal{L} \cup \{\tau\}$.

The syntax of a process P in $\mathcal{P}_\nu$ is defined as follows:

$$\begin{array}{lll} P & ::= & PN<\widetilde{\beta}> \mid M \mid P|P \mid (\nu\widetilde{\beta})P \mid \frac{P}{P} \\ M & ::= & 0 \mid \alpha.P \mid M+M \end{array}$$

where $PN \in \mathcal{PN}_\nu$, $\widetilde{\beta}$ is a tuple of elements of $\mathcal{N} \cup \overline{\mathcal{N}}$, and $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$.

As in CCS, $(\nu\widetilde{\beta})P$ models restriction of the scope of a tuple of port/action names $\widetilde{\beta}$ to a process P. The meaning of the other syntactic constructs is as in basic CCS^{dp} (suitably reinterpreted for processes in $\mathcal{P}_\nu$). Thus, $\mathcal{P}_\nu$ is a superset of $\mathcal{P}$.

Operator Precedence

In basic $CCS^{dp}+\nu$, the precedence of the operators (in decreasing order) is:
fraction formation (highest); restriction and relabelling (right associative); prefix; summation; parallel composition (lowest).

Free Names

Given $p \in \mathcal{P}_\nu$, let $fn(p)$ be the set of port/action free names of p, and is defined as follows:

$fn : \mathcal{P}_\nu \longrightarrow \mathbb{P}\mathcal{L}$ such that

$$fn(p) \triangleq \begin{cases} \emptyset & \text{if } p = 0 \\ \{\beta\} \cup fn(p_1) & \text{elseif } p = \beta.p_1 \ \wedge \ \beta \in \mathcal{N} \cup \overline{\mathcal{N}} \\ fn(M_1) \cup fn(M_2) & \text{elseif } p = M_1 + M_2 \\ fn(p_1) \cup fn(p_2) & \text{elseif } p = p_1|p_2 \\ fn(p_1) - (Set(\widetilde{\beta}) \cup Set(\bar{\bar{\widetilde{\beta}}})) & \text{elseif } p = (\nu\widetilde{\beta})p_1 \\ fn(p_1) \cup fn(p_2) & \text{elseif } p = \frac{p_1}{p_2} \\ Set(\widetilde{\beta}) & \text{elseif } p = A<\widetilde{\beta}> \end{cases}$$

6.1.2 Labelled Transition System Semantics

Let $\mathcal{R}_\nu$ be the countable set of reconfiguration actions of the processes in $\mathcal{P}_\nu$ (e.g. τ_{r_X}, τ_{r_Y}, τ_{r_Z}) that create a process in $\mathcal{P}_\nu$, and let $\overline{\mathcal{R}}_\nu$ be the countable set of complementary reconfiguration actions of the processes in $\mathcal{P}_\nu$ (e.g. $\overline{\tau}_{r_X}$, $\overline{\tau}_{r_Y}$, $\overline{\tau}_{r_Z}$) that delete a process in $\mathcal{P}_\nu$ (see the *Creat* and *Delet* rules below). Each action in $\mathcal{R}_\nu$ is represented by τ_{r_X}, with $X \in \mathcal{P}_\nu$, and $\forall \tau_{r_X} \in \mathcal{R}_\nu$ ($\overline{\overline{\tau}}_{r_X} \triangleq \tau_{r_X}$).

The sets $\mathcal{N}$, $\overline{\mathcal{N}}$, $\{\tau\}$, $\mathcal{R}_\nu$, $\overline{\mathcal{R}}_\nu$ and $\mathcal{PN}_\nu$ are assumed to be pairwise disjoint.

Let $\mathcal{C}_\nu$ be the set of reconfiguration actions of the processes in $\mathcal{P}_\nu$, where $\mathcal{C}_\nu \triangleq \mathcal{R}_\nu \cup \overline{\mathcal{R}}_\nu$.

Let $\mathcal{A}_\nu$ be the set of actions of the processes in $\mathcal{P}_\nu$, where $\mathcal{A}_\nu \triangleq \mathcal{I} \cup \mathcal{C}_\nu$.

The LTS rules for basic $CCS^{dp}+\nu$ are a superset of the LTS rules for basic CCS^{dp} (suitably reinterpreted for processes in $\mathcal{P}_\nu$) plus additional rules to describe new behaviour due to the restriction operator (i.e. *Res*, *ResFract* and *ResRecon*). See Table 6.1.

$$\text{Res} : \frac{\alpha \in \mathcal{I} \quad \wedge \quad P \xrightarrow{\alpha} P' \quad \wedge \quad \alpha \notin Set(\widetilde{a}) \cup Set(\overline{\widetilde{a}})}{(\nu \widetilde{a})P \xrightarrow{\alpha} (\nu \widetilde{a})P'}$$

The *Res* rule states that restriction preserves a transition in $\mathcal{I}$ of a process, provided the transition is not in the set of restricted actions nor in the set of complementary actions of the restricted actions.

$$\text{ResFract} : \frac{\frac{P'}{P} \xrightarrow{\tau_{r_Q}} P' \quad \wedge \quad (\nu \widetilde{a})P \in \mathcal{P}_\nu^+}{(\nu \widetilde{a})\frac{P'}{P} \xrightarrow{\tau_{r_{(\nu \widetilde{a})Q}}} (\nu \widetilde{a})P'}$$

The *ResFract* rule for restricted fraction processes corresponds to the *Creat* rule for fraction processes. It states that if a fraction process $\frac{P'}{P}$ can perform a reconfiguration transition τ_{r_Q} to create P', then the restricted fraction process $(\nu \widetilde{a})\frac{P'}{P}$ can perform the reconfiguration transition $\tau_{r_{(\nu \widetilde{a})Q}}$ to create $(\nu \widetilde{a})P'$, provided $(\nu \widetilde{a})P$ is a positive process. Thus, the restricted fraction $(\nu \widetilde{a})\frac{P'}{P}$ behaves like the fraction of the restrictions $\frac{(\nu \widetilde{a})P'}{(\nu \widetilde{a})P}$.

$$\text{ResRecon} : \frac{\rho_Q \in \{\tau_{r_Q}, \overline{\tau}_{r_Q}\} \quad \wedge \quad P \xrightarrow{\rho_Q} P' \quad \wedge \quad fn(Q) \cap (Set(\widetilde{a}) \cup Set(\overline{\widetilde{a}})) = \emptyset}{(\nu \widetilde{a})P \xrightarrow{\rho_Q} (\nu \widetilde{a})P'}$$

The *ResRecon* rule is the *Res* rule modified for reconfiguration transitions. It states that a restriction $(\nu \widetilde{a})P$ preserves the reconfiguration transitions τ_{r_Q}, $\overline{\tau}_{r_Q}$ of the process

Sum $\dfrac{k\in I}{\sum_{i\in I}\alpha_i.P_i \xrightarrow{\alpha_k} P_k}$

React $\dfrac{\lambda\in\mathcal{L}\cup\mathcal{C}_\nu \ \wedge\ P\xrightarrow{\lambda}P' \ \wedge\ Q\xrightarrow{\overline{\lambda}}Q'}{P|Q\xrightarrow{\tau}P'|Q'}$

L-Par $\dfrac{\mu\in\mathcal{A}_\nu \ \wedge\ P\xrightarrow{\mu}P'}{P|Q\xrightarrow{\mu}P'|Q}$

R-Par $\dfrac{\mu\in\mathcal{A}_\nu \ \wedge\ Q\xrightarrow{\mu}Q'}{P|Q\xrightarrow{\mu}P|Q'}$

Ident $\dfrac{|\widetilde{b}|=|\widetilde{a}| \ \wedge\ \mu\in\mathcal{A}_\nu \ \wedge\ P[\frac{\widetilde{b}}{\widetilde{a}}]\xrightarrow{\mu}P'}{A<\widetilde{b}>\xrightarrow{\mu}P'}$ where $A(\widetilde{a})\triangleq P$

Creat $\dfrac{P\sim_{of}Q \ \wedge\ P\in\mathcal{P}_\nu^+}{\frac{P'}{P}\xrightarrow{\tau_{r_Q}}P'}$

Delet $\dfrac{P\sim_{of}Q \ \wedge\ P\in\mathcal{P}_\nu^+}{P\xrightarrow{\overline{\tau}_{r_Q}}0}$

CompDelet $\dfrac{R\sim_{of}R_1|R_2 \ \wedge\ P\xrightarrow{\overline{\tau}_{r_{R_1}}}P' \ \wedge\ P'\xrightarrow{\overline{\tau}_{r_{R_2}}}P''}{P\xrightarrow{\overline{\tau}_{r_R}}P''}$

L-React $\dfrac{R\sim_{of}R_1|R_2 \ \wedge\ P\xrightarrow{\overline{\tau}_{r_{R_1}}}P' \ \wedge\ P'\xrightarrow{\tau_{r_R}}P'' \ \wedge\ Q\xrightarrow{\overline{\tau}_{r_{R_2}}}Q'}{P|Q\xrightarrow{\tau}P''|Q'}$

R-React $\dfrac{R\sim_{of}R_1|R_2 \ \wedge\ P\xrightarrow{\overline{\tau}_{r_{R_1}}}P' \ \wedge\ Q\xrightarrow{\overline{\tau}_{r_{R_2}}}Q' \ \wedge\ Q'\xrightarrow{\tau_{r_R}}Q''}{P|Q\xrightarrow{\tau}P'|Q''}$

Res $\dfrac{\alpha\in\mathcal{I} \ \wedge\ P\xrightarrow{\alpha}P' \ \wedge\ \alpha\notin Set(\widetilde{a})\cup Set(\overline{\widetilde{a}})}{(\nu\widetilde{a})P\xrightarrow{\alpha}(\nu\widetilde{a})P'}$

ResFract $\dfrac{\frac{P'}{P}\xrightarrow{\tau_{r_Q}}P' \ \wedge\ (\nu\widetilde{a})P\in\mathcal{P}_\nu^+}{(\nu\widetilde{a})\frac{P'}{P}\xrightarrow{\tau_{r_{(\nu\widetilde{a})Q}}}(\nu\widetilde{a})P'}$

ResRecon $\dfrac{\rho_Q\in\{\tau_{r_Q},\overline{\tau}_{r_Q}\} \ \wedge\ P\xrightarrow{\rho_Q}P' \ \wedge\ fn(Q)\cap(Set(\widetilde{a})\cup Set(\overline{\widetilde{a}}))=\emptyset}{(\nu\widetilde{a})P\xrightarrow{\rho_Q}(\nu\widetilde{a})P'}$

Table 6.1: Labelled Transition System Semantics of Basic $CCS^{dp}+\nu$.

P, provided the free names of Q are not restricted by $(\nu\widetilde{a})$. This condition (on Q) is stronger than the corresponding condition (on α) in *Res*, because τ_{r_Q} and $\overline{\tau}_{r_Q}$ depend on the behaviour of the entire process Q for matching.

6.1.3 Positive Processes and Zero Processes

The sets of positive processes and zero processes of $\mathcal{P}_\nu$ are defined as follows.

Let $\mathcal{P}_\nu^+$ be the set of *positive processes* of $\mathcal{P}_\nu$, where $\mathcal{P}_\nu^+$ is defined to be the smallest

subset of $\mathcal{P}_\nu$ that satisties the following conditions:

1. $\forall \alpha \in \mathcal{I}\ \forall p \in \mathcal{P}_\nu\ (\alpha.p \in \mathcal{P}_\nu^+)$
2. $\forall p, q \in \mathcal{P}_\nu\ (p + q \in \mathcal{P}_\nu \wedge (p \in \mathcal{P}_\nu^+ \vee q \in \mathcal{P}_\nu^+) \Longrightarrow p + q \in \mathcal{P}_\nu^+)$
3. $\forall p, q \in \mathcal{P}_\nu\ (p \in \mathcal{P}_\nu^+ \vee q \in \mathcal{P}_\nu^+ \Longrightarrow p|q \in \mathcal{P}_\nu^+)$
4. $\forall p \in \mathcal{P}_\nu\ \forall q \in \mathcal{P}_\nu^+\ \left(\frac{p}{q} \in \mathcal{P}_\nu^+\right)$
5. $\forall \beta \in \mathcal{I}\ \forall X \in \mathcal{PN}_\nu\ (\beta.X \in \mathcal{P}_\nu^+)$
6. $\forall \widetilde{\gamma} \in \mathcal{L}^{|\widetilde{\gamma}|}\ \forall p \in \mathcal{P}_\nu\ (\mathcal{I}_{(\nu\widetilde{\gamma})p} \cup (\mathcal{R}_\nu)_{(\nu\widetilde{\gamma})p} \neq \emptyset \Longrightarrow (\nu\widetilde{\gamma})p \in \mathcal{P}_\nu^+)$

Thus, $\mathcal{P}_\nu^+$ is a superset of $\mathcal{P}^+$. It should be clear from Lemma 4.3.5 and condition 6 that every positive process in $\mathcal{P}_\nu$ can perform a transition in $\mathcal{I} \cup \overline{\mathcal{R}}_\nu$.

Let $\mathcal{P}_\nu^0$ be the set of *zero processes* of $\mathcal{P}_\nu$, where $\mathcal{P}_\nu^0$ is defined to be the smallest subset of $\mathcal{P}_\nu$ that satisties the following conditions:

1. $0 \in \mathcal{P}_\nu^0$
2. $\forall p, q \in \mathcal{P}_\nu^0\ (p + q \in \mathcal{P}_\nu \Longrightarrow p + q \in \mathcal{P}_\nu^0)$
3. $\forall p, q \in \mathcal{P}_\nu^0\ (p|q \in \mathcal{P}_\nu^0)$
4. $\forall p \in \mathcal{P}_\nu\ \forall q \in \mathcal{P}_\nu^0\ \left(\frac{p}{q} \in \mathcal{P}_\nu^0\right)$
5. $\forall \widetilde{\gamma} \in \mathcal{L}^{|\widetilde{\gamma}|}\ \forall p \in \mathcal{P}_\nu\ (\mathcal{I}_{(\nu\widetilde{\gamma})p} \cup (\mathcal{R}_\nu)_{(\nu\widetilde{\gamma})p} = \emptyset \Longrightarrow (\nu\widetilde{\gamma})p \in \mathcal{P}_\nu^0)$

Thus, $\mathcal{P}_\nu^0$ is a superset of $\mathcal{P}^0$. It should be clear from Lemma 4.3.7 and condition 5 that no zero process in $\mathcal{P}_\nu$ can perform a transition in $\mathcal{I} \cup \overline{\mathcal{R}}_\nu$.
Therefore, $\mathcal{P}_\nu^+$ and $\mathcal{P}_\nu^0$ are disjoint.

As in basic CCS^{dp}, we restrict $\mathcal{P}_\nu$ as follows:

$$\mathcal{P}_\nu \triangleq \mathcal{P}_\nu^+ \cup \mathcal{P}_\nu^0 \tag{6.1}$$

Furthermore, as in basic CCS^{dp}, every $p \in \mathcal{P}_\nu$ must be the result of one or more applications of the production rules of $\mathcal{P}_\nu^+$ or $\mathcal{P}_\nu^0$ with finite depth of inference; and every transition of every $p \in \mathcal{P}_\nu$ must be a result of one or more applications of the LTS semantic rules with finite depth of inference.

It is straightforward to prove that $\{\mathcal{P}_\nu^+, \mathcal{P}_\nu^0\}$ is a partition of $\mathcal{P}_\nu$.

The definitions of strong of-bisimulation and strong dp-bisimulation on $\mathcal{P}_\nu$ are identical to their respective definitions on $\mathcal{P}$ (suitably reinterpreted for processes in

$\mathcal{P}_v$); and we conjecture that the propositions proved for $\mathcal{P}$ also hold for $\mathcal{P}_v$ (if suitably reinterpreted). One important difference is the decidability of $\sim_{of}$. However, if restriction is only applied to constant processes (rather than to recursively defined processes) then the decidability of $\sim_{of}$ is not an issue. Furthermore, if models in CCS^{dp} are restricted to consist of a finite number of processes and process definitions, then decidability is not an issue.

6.2 On Process Identification

A process identifier is a simple construct for identifying a specific process instance. Therefore, the identifier can be used by a fraction to select a specific process instance for reconfiguration; and if the identifier is passed as a parameter to the fraction, it enables the fraction to reconfigure different process instances in a flexible and controlled manner. Furthermore, the identification of a specific process for reconfiguration precludes the matching of other processes, and thereby significantly reduces the computational complexity of matching.

6.2.1 A Process Identification Scheme

An identifier is allocated to each process that is to be considered for reconfiguration, such that the identifier is unique to the expression containing the process. Processes not to be considered for reconfiguration do not have an identifier.

For example, in the expression $p_1 \mid q_2 \mid r \mid x(\widetilde{m}).\frac{p'}{p}(\widetilde{m})$, only p_1, q_2 and their successors can be tested for matching, because only p_1 and q_2 have been identified. Process r and its successors will not be tested for matching.

If a process does not spawn a process after a transition, then the successor retains the identity of its parent. The identity is retained after a non-deterministic transition. Thus, a process identifier is used to identify a thread of a task or a system component, rather than its state.

For example, we can have $p_1 \xrightarrow{\bar{a}} s_1 \xrightarrow{b} t_1$. So, $\frac{p'}{p}(1)$ will test p_1, s_1 and t_1 for a match.

If the parent's transition results in the creation of two or more processes, then the child processes append to the identifier of the parent their own identifier that is locally unique among the set of possible child processes resulting from all the

possible transitions of the parent.

For example, we can have $q_2 \xrightarrow{c} u_{2,1}|v_{2,2}$ and $q_2 \xrightarrow{d} w_{2,3}|w_{2,4}$. So, $\frac{p'}{p}(2)$ will test q_2, and $u_{2,1}|v_{2,2}$, and $w_{2,3}|w_{2,4}$ for a match.

Thus, we use a hierarchical identification scheme for processes, with a unique identifier for each thread that is to be considered for reconfiguration. The scheme is hierarchical because processes in CCS^{dp} have a flat compositional structure. That is, there in no indication of super-processes containing sub-processes in the process syntax. Therefore, we use the identifier hierarchy to indicate which processes belong to a given task or component, and to retain this information during equational reasoning and model checking.

An identification issue arises when a child process is common to different transitions of a parent.

For example, we can have $x \xrightarrow{a} u|v$ and $x \xrightarrow{b} u|w$.

Normally, the identifier of the child process u would be the same after either transition, because u in $u|v$ and in $u|w$ can be considered to be the same instance of the same process, but resulting from different transitions. However, the environment of u can be different in the two transitions, and the modeller may want to distinguish the two instances of u for matching because of the different environments. Therefore, we leave the identifier allocation of common child processes to the modeller.

The identifier of a process replacing a matched process should satisfy the same restrictions as those on the identifiers of the child processes of the matched process. However, the flexibility of the identification scheme implies that other alternatives are possible, such as identifying the replacing process as a new top-level thread.

It is important to notice that CCS^{dp} without process identifiers is a class-based process algebra. That is, like numbers in arithmetic, the processes in CCS^{dp} are classes, and different instances of a process can be used interchangeably in any context with identical results. However, the use of process identifiers makes the modification of CCS^{dp} an instance-based process algebra, so that different instances of a process with different identifiers in identical contexts can produce different

results.

6.2.1.1 Matching using identifiers instead of $\sim_{\mathbf{of}}$

If each positive singleton process in a process expression has a globally unique identifier, then it is possible to identify uniquely the expression, all its sub-expressions, and all the expressions resulting from their transitions (and their sub-expressions). If the process expressions are all behaviourally distinct, then such an identification scheme would be a very efficient matching mechanism. However, this is not the case. Syntactically and structurally different processes can behave identically in any context. In order to match processes with identical behaviour but different identifiers, we would need much more complex matching criteria than $\sim_{of}$; which would complicate modelling. Alternatively, it would be necessary to identify syntactically or structurally different processes with identical behaviour and label them with a common identifier. However, the computational complexity of checking identical behaviour would be similar to that of behavioural matching; but the job would fall on the modeller rather than on the verification tool.

Therefore, we prefer to use a process identifier to identify a task/system component, and use $\sim_{of}$ to match a behavioural state of the task/system component.

6.3 Discussion

This thesis has introduced a novel construct – the fraction process – in order to model unplanned process reconfiguration abstractly. The construct is a special process rather than a special operator so that the fraction can be located outside the system model, that is, in the context of the system model, which is necessary for constructing an abstract model of unplanned process reconfiguration. Furthermore, the fraction can be added to the context of the system model in the same way that a patch is added to the environment of the system. The syntactic separation of the fraction from the process it replaces necessitates a dynamic binding between the two processes, that is, process matching. A strong bisimulation ($\sim_{of}$) is used for process matching because it produces terse process expressions. Other forms of matching based on syntactic equality and structural congruence have also been briefly discussed. A fraction can also replace another fraction. Therefore, our notion of reconfiguration is recursive.

CCS was used as the host for the fraction process, resulting in the new process algebra CCS^{dp}. This was done partly because of the simplicity of *CCS*, and partly because *CCS* is the common base for both π-calculi and some timed process algebras, which should help to insert the fraction process and its theory into other process algebras. An equational theory has been developed for CCS^{dp} using a stronger bisimulation than $\sim_{of}$ (i.e. $\sim_{dp}$). We have also argued that expressions in CCS^{dp} are amenable to model checking.

CCS^{dp} has been evaluated in two ways. First, with respect to criteria used to evaluate other formalisms. Second, by modelling a simple office workflow and informal model checking. The exercise revealed several limitations of basic CCS^{dp}. This chapter has briefly outlined two extensions of basic CCS^{dp} that address some of these limitations. One extension uses the ν operator, which helps to construct larger models but affects the decidability of $\sim_{of}$. Another extension uses a process identification scheme that allows processes to be selectively reconfigured and also reduces the computational complexity of matching.

Chapter 7

Concluding Remarks

Contents

In this chapter, we summarize the findings of the research, discuss their significance, and identify future work.

7.1 Conclusions

The thesis of this thesis is that the fraction process is a suitable construct for the modelling and analysis of unplanned dynamic reconfiguration.

The next generation of dependable systems will be required to evolve [CHNF10], and some of the systems (such as control systems) will be required to evolve dynamically. Furthermore, it will be impossible to foresee at design time all the future configurations of an evolving dependable system [MMR10]. This justifies the modelling and analysis of unplanned dynamic reconfiguration.

We reviewed different approaches to the implementation of dynamic software reconfiguration, and from these, identified requirements on a formalism for the modelling and analysis of dynamic reconfiguration in dependable systems. The requirements were then used to evaluate a number of formalisms. The review of systems, dynamic architecture description languages and formalisms revealed several issues in the formal modelling and analysis of dynamic reconfiguration. One issue is the modelling of unplanned process reconfiguration. Mobile process algebras are simple and widely used for modelling dynamic reconfiguration, and their ability to model application actions and reconfiguration actions in the same notation makes them suitable for modelling computational interactions between the two sets of actions. However, mobile process algebras cannot model unplanned reconfiguration abstractly, because they use special operators to model specialised

behaviour (such as reconfiguration), which requires syntactic proximity between the operands in a model, which (in turn) requires both the reconfigured and the reconfiguring components to be in the system. Therefore, existing process algebras can model **only** planned reconfiguration.

Therefore, we defined a special kind of process (the fraction process) and modelled reconfiguration as a reaction between a fraction and the process that it reconfigures. Being a process, the fraction can be located outside the boundary of the system model, that is, in the context of the system model, and thereby can model reconfiguration of the system that is not designed into the system, that is, unplanned reconfiguration. Furthermore, just as a patch can be introduced into the environment of the system by an execution platform, so the fraction process can be added to the context of the system model by a modelling environment. Moreover, if a software component can be passed into the system as a value, then the system model should be able to accept a process as a value. If this process is a fraction, then the evolution of the system can be modelled elegantly (although not in basic CCS^{dp}). The notion of fraction process is recursive in order to allow a reconfiguration subsystem or process to be itself reconfigured.

The use of *CCS* as a host formalism for the fraction process enables functional interactions between application actions and reconfiguration actions to be expressed simply – as interleavings of transitions. We conjecture that it is possible to extend in a similar manner other process algebras that have a parallel composition operator, and thereby enable the algebras to model unplanned dynamic reconfiguration simply.

The syntactic separation of the fraction from the process that it reconfigures necessitates a dynamic binding between the two processes, that is, process matching. A strong bisimulation ($\sim_{of}$) is used for matching in order to achieve terseness of process expressions. We made a distinction between processes that can be reconfigured (positive processes) and processes that cannot be reconfigured (zero processes) in order to preserve the identity property of 0. The combination of syntactic separation and dynamic binding between the processes is useful in other ways. First, it supports the flexible calculation of different reconfiguration paths between configurations, since only fraction processes need to be changed in order to determine new reconfiguration transitions. Second, it is not necessary to have access to the 'source code' of a system model in order to reconfigure it, since the

model can be changed by its interactions with fraction processes in its context.

We proved that $\sim_{of}$ is an equivalence relation, but not a process congruence. Therefore, the bisimulation conditions were strengthened to produce a congruence ($\sim_{dp}$) for equational reasoning. The equational theory was developed using a more rigorous proof technique than is conventional in order to establish that the theory of fraction processes is mathematically sound.

Basic CCS^{dp} was evaluated using the requirements used to evaluate the other formalisms and the reconfiguration of a simple office workflow used as a case study. The evaluation showed the ability of a fraction to model both planned and unplanned process reconfiguration simply and tersely. Furthermore, it was argued that process expressions in basic CCS^{dp} are amenable to model checking. However, the flexibility of matching causes problems.

First, the matching has to be decidable for tool support, but decidability involves removing the restriction operator from basic CCS^{dp}, bounding the depth of fractional recursion of fraction processes, and restricting models to consist of a finite number of processes and process definitions. However, the third restriction is realistic for many control systems. An alternative is to achieve decidability by sacrificing behavioural matching for structural congruence-based matching or even syntactic matching. Thus, the loss of information hiding in matching is balanced by the gain of decidability and tool support.

Second, the computational complexity of matching is exponential at each transition.

Third, matching cannot be controlled to reconfigure a specific process instance. This issue can be addressed using a process identification scheme, which can also reduce the computational complexity of matching significantly. However, equational reasoning becomes vacuous with the use of a process identification scheme, and another verification technique must be used (such as model checking).

The problem of controlling non-deterministic transitions is well known in process algebra, and it occurs in basic CCS^{dp}. A priority scheme for processes or transitions is required.

The case study also suggests that model checking is more useful in verifying workflows than equational reasoning.

We assert that CCS^{dp} is relevant for dependable systems, using the following argument. Dependable systems are required to deliver predictable and correct service, correct service design is a key factor in achieving predictable and correct service delivery, formal reasoning can determine whether or not a service design is correct, and CCS^{dp} supports formal reasoning.

7.2 Future Work

This research has explored the novel idea of using a special process to introduce dynamic reconfiguration functionality into a process algebra (rather than a special operator). As a result, there is considerable opportunity for future work.

To support reasoning, a systematic way of mapping a process expression in basic CCS^{dp} to a Kripke structure is needed (such as an algorithm). The mapping would be tested on the different designs of the office workflow and its reconfiguration. The mapping would also provide definitive evidence that basic CCS^{dp} processes can be model checked. A portfolio of case studies of dynamic reconfiguration of dependable systems should also be collected for modelling and verification.

To investigate the decidability of strong of-bisimulation, structural congruence and strong dp-bisimulation on $\mathcal{P}$, a proof theory is needed. It should also be possible to encode the proof theory in a theorem prover. The automated verification of the proofs given in this thesis should also be attempted.

The conjectures in Section 4.5.4 about the computational complexities of the different forms of matching need to be verified.

CCS^{dp} is more of a specification language than a programming language. Therefore, in order to produce dynamically reconfigurable implementations, a refinement calculus is needed supported by tools.

In order to facilitate modelling, a value-passing form of CCS^{dp} is needed. Alternatively, a value-passing process algebra can be extended with the fraction process. Port names should be first class in order to model link reconfiguration.

Dynamic reconfiguration is a common requirement in large real-time control systems. Therefore, CCS^{dp} needs to be extended with real-time constructs, such as time, duration of actions, preemption of actions, priority of an action, and process identity. Alternatively, a real-time process algebra can be extended with the fraction process. The consequences of using a non-interleaving semantics should also be investigated.

Finally, on a speculative note, since reactions in Nature occur between entities with opposite characteristics, it may be useful to formalise the notion of anti-similarity (i.e. the opposite of similarity) and to investigate matching based on anti-similarity. This may be useful for modelling physical reactions.

Bibliography

[AAA+07] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, and F. Curbera et al. Web services business process execution language version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf, 2007. [Online; accessed 03-Feb-2012]. 41

[ADG98] R. Allen, R. Douence, and D. Garlan. Specifying and analyzing dynamic software architectures. In *Proceedings of the 1st International Conference on Fundamental Approaches to Software Engineering*, pages 21–37, 1998. 32, 33, 34, 35, 36, 45

[AG94] R. Allen and D. Garlan. Formalizing architectural connection. In *Proceedings of the 16th International Conference on Software Engineering*, pages 71–80, 1994. 31

[All97] R. J. Allen. *A Formal Approach to Software Architecture*. PhD thesis, Carnegie Mellon University School of Computer Science, 1997. 31

[ALRL04] A. Avižienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004. 1, 2

[ALS03] S. Ajmani, B. Liskov, and L. Shrira. Scheduling and simulation: How to upgrade distributed systems. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems*, pages 43–48, 2003. 18

[Ama00] R. M. Amadio. An asynchronous model of locality, failure, and process mobility. *Theoretical Computer Science*, 240(1):147–176, 2000. 61

[AWvSN01] J. P. A. Almeida, M. Wegdam, M. van Sinderen, and L. Nieuwenhuis. Transparent dynamic reconfiguration for corba. In *Proceedings of the 3rd International Symposium on Distributed Objects and Applications*, pages 197–207, 2001. 5, 17

[BA01] M. Ben-Ari. *Mathematical Logic for Computer Science, 2nd edition*. Springer-Verlag, 2001. 107, 108

[BABR96] L. Bellissard, S. B. Atallah, F. Boyer, and M. Riveill. Distributed application configuration. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, pages 579–585, 1996. 41

[Bar84] H. P. Barendregt. *The Lambda Calculus: its syntax and semantics, 2nd edition*. Elsevier Science Publishers B. V., 1984. 7

[BB92] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248, 1992. 45

[BBF+01] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. McKenzie. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer, 2001. 128

[BBRVD98] L. Bellissard, F. Boyer, M. Riveill, and J. Y. Vion-Dury. System services for distributed application configuration. In *Proceedings of the 4th International Conference on Configurable Distributed Systems*, pages 53–60, 1998. 41

[BCDW04] J. S. Bradbury, J. R. Cordy, J. Dingel, and M. Wermelinger. A survey of self-management in dynamic software architecture specifications. In *Proceedings of the 1st ACM SIGSOFT Workshop on Self-Managed Systems*, pages 28–33, 2004. 5

[BD93] T. Bloom and M. Day. Reconfiguration and module replacement in argus: theory and practice. *Software Engineering Journal (Special Issue)*, 8(2):102–108, 1993. 5, 15

[BEJV96] P. Binns, M. Englehart, M. Jackson, and S. Vestal. Domain-specific software architectures for guidance, navigation and control. *International Journal of Software Engineering and Knowledge Engineering*, 6(2):201–227, 1996. 21

[Ber04] M. Berger. *Towards Abstractions for Distributed Systems*. PhD thesis, University of London Imperial College of Science, Technology and Medicine Department of Computing, 2004. 47, 58, 61, 62

[BF08] A. Bhattacharyya and J. S. Fitzgerald. Development of a formalism for modelling and analysis of dynamic reconfiguration of dependable real-time systems: A technical diary. In *Proceedings of the 2008 RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems*, pages 67–72, 2008. 1, 69, 70

[BFL+94] J. C. Bicarregui, J. S. Fitzgerald, P. A. Lindsay, R. Moore, and B. Ritchie. *Proof in VDM: A Practitioner's Guide*. Springer-Verlag London Limited, 1994. 8

[BISZ98] C. Bidan, V. Issarny, T. Saridakis, and A. Zarras. A dynamic reconfiguration service for corba. In *Proceedings of the 4th International Conference on Configurable Distributed Systems*, pages 35–42, 1998. 19

[Blo83] T. Bloom. *Dynamic Module Replacement in a Distributed Programming System*. PhD thesis, Massachusetts Institute of Technology Laboratory for Computer Science, 1983. 16

[BM90] J. P. Banâtre and D. Le Métayer. The gamma model and its discipline of programming. *Science of Computer Programming*, 15(1):55–77, 1990. 46

[Bou92] G. Boudol. Asynchrony and the π-calculus. Technical Report 1702, Institut National de Recherche en Informatique et en Automatique, May 1992. 58

[BWD+93] M. R. Barbacci, C. B. Weinstock, D. L. Doubleday, M. J. Gardner, and R. W. Lichota. Durra: a structure description language for developing distributed applications. *Software Engineering Journal (Special Issue)*, 8(2):83–94, 1993. 19

[CGP02] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2002. 128

[Cha83] W. W. Chandler. The installation and maintenance of colossus. *Annals of the History of Computing*, 5(3):260–262, 1983. 1

[CHM94] S. Christensen, Y. Hirshfeld, and F. Moller. Decidable subsets of ccs. *The Computer Journal*, 37(4):233–242, 1994. 108

[CHNF10] L. Coyle, M. Hinchey, B. Nuseibeh, and J. L. Fiadeiro. Guest editors' introduction: Evolving critical systems. *IEEE Computer*, 43(5):28–33, 2010. 2, 145

[CKCB01] S. M. Cho, H. H. Kim, S. D. Cha, and D. H. Bae. Specification and validation of dynamic systems using temporal logic. *IEE Proceedings Software*, 148(4):135–140, 2001. 53

[Coo83] A. W. M. Coombs. The making of colossus. *Annals of the History of Computing*, 5(3):253–259, 1983. 1

[CV65] F. J. Corbató and V. A. Vyssotsky. Introduction and overview of the multics system. In *Proceedings of the AFIPS Fall Joint Computer Conference*, pages 185–196, 1965. 12

[DK75] F. DeRemer and H. Kron. Programming-in-the-large versus programming-in-the-small. In *Proceedings of Conference on Reliable Software*, pages 114–121, 1975. 18

[ED97] H. Evans and P. Dickman. Drastic: A run-time architecture for evolving, distributed, persistent systems. In *Proceedings of the 11th European Conference on Object-Oriented Programming*, pages 243–275, 1997. 18

[EKR95] C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *Proceedings of the Conference on Organizational Computing Systems*, pages 10–21. ACM, 1995. 115

[EN86] U. Engberg and M. Nielsen. A calculus of communicating systems with label passing. Technical Report DAIMI PB-208, Aarhus University Department of Computer Science, May 1986. 46

[Fab76] R. S. Fabry. How to design a system in which modules can be changed on the fly. In *Proceedings of the 2nd International Conference on Software Engineering*, pages 470–476, 1976. 13

[Flo83] T. H. Flowers. The design of colossus. *Annals of the History of Computing*, 5(3):239–252, 1983. 1

[Fra97] M. Franz. Dynamic linking of software components. *IEEE Computer*, 30(3):74–81, 1997. 13

[FW05] S. Fischmeister and K. Winkler. Non-blocking deterministic replacement of functionality, timing, and data-flow for hard real-time systems at runtime. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 106–114. IEEE Computer Society, 2005. 5

[GJ93] D. Gupta and P. Jalote. On-line software version change using state transfer between processes. *Software - Practice and Experience*, 23(9):949–964, 1993. 14

[GJB96] D. Gupta, P. Jalote, and G. Barua. A formal framework for on-line software version change. *IEEE Transactions on Software Engineering*, 22(2):120–131, 1996. 14

[GR91] M. M. Gorlick and R. R. Razouk. Using weaves for software construction and analysis. In *Proceedings of the 13th International Conference on Software Engineering*, pages 23–34, 1991. 41

[Gro93] J. F. Groote. Negative system specifications with negative premises. *Theoretical Computer Science*, 118(2):263–299, 1993. 108

[GTL89] J-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1989. 8

[HG98] G. Hjálmtÿsson and R. Gray. Dynamic c++ classes: A lightweight mechanism to update code in a running program. In *Proceedings of the USENIX Annual Technical Conference (NO 98)*, pages 65–76, 1998. 12

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International (UK) Limited, 1985. 7, 45

[Hod87] A. Hodges. *Alan Turing: The Enigma of Intelligence*. Unwin Hyman Limited, 1987. 1

[HT91] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proceedings of the 5th European Conference on Object-Oriented Programming*, pages 133–147, 1991. 58

[Hun12] E. Hunt. Us government computer penetration programs and the implications for cyberwar. *Annals of the History of Computing*, 34(3):4–21, 2012. 1

[Hut10] H. Huttel. *Transitions and Trees: An Introduction to Structural Operational Semantics*. Cambridge University Press, 2010. 71

[ISO11] ISO/IEC/IEEE. *Systems and software engineering – Architecture description*. International standard; ISO 42010. ISO/IEC/IEEE, first edition, 2011. 21

[IW95] P. Inverardi and A. L. Wolf. Formal specification and analysis of software architectures using the chemical abstract machine model. *IEEE Transactions on Software Engineering*, 21(4):373–386, 1995. 46

[Jon80] C. B. Jones. *Software Development: A Rigorous Approach*. Prentice Hall International (UK) Limited, 1980. 45

[KBR+05] N. Kavantzas, D. Burdett, G. Ritzinger, A. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0. http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109, 2005. [Online; accessed 03-Feb-2012]. 41

[KGC89] S. M. Kaplan, S. K. Goering, and R. H. Campbell. Specifying concurrent systems with Δ-grammars. In *Proceedings of the 5th International Workshop on Software Specification and Design*, pages 20–27, 1989. 44, 45

[KK88] S. M. Kaplan and G. E. Kaiser. Garp: Graph abstractions for concurrent programming. In *Proceedings of the 2nd European Symposium on Programming*, pages 191–205, 1988. 44, 45

[KM90] J. Kramer and J. Magee. The evolving philosophers problem: Dynamic change management. *IEEE Transactions on Software Engineering*, 16(11):1293–1306, 1990. 5, 26, 28

[Kob06] N. Kobayashi. A new type system for deadlock-free processes. In *Proceedings of the 17th International Conference on Concurrency Theory (CONCUR 2006)*, pages 233–247. Springer-Verlag, 2006. 53

[LKA+95] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann. Specification and analysis of system architecture using rapide. *IEEE Transactions on Software Engineering*, 21(4):336–355, 1995. 36

[LM07] R. Lucchi and M. Mazzara. A pi-calculus based semantics for ws-bpel. *Journal of Logic and Algebraic Programming*, 70(1):96–118, 2007. 41

[LS83] B. Liskov and R. Scheifler. Guardians and actions: Linguistic support for robust, distributed programs. *ACM Transactions on Programming Languages and Systems*, 5(3):381–404, 1983. 15

[Luc02] D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2002. 39

[LV95] D. C. Luckham and J. Vera. An event-based architecture definition language. *IEEE Transactions on Software Engineering*, 21(9):717–734, 1995. 37, 39

[MADB12] M. Mazzara, F. Abouzaid, N. Dragoni, and A. Bhattacharyya. Toward design, modelling and analysis of dynamic workflow reconfiguration

- a process algebra perspective. In *Proceedings of the 8th International Workshop on Web Services and Formal Method (WSFM 2011)*, volume 7176 of *Lecture Notes in Computer Science*, pages 64–78. Springer-Verlag, 2012. 115, 127

[Maz06] M. Mazzara. *Towards Abstractions for Web Services Composition*. PhD thesis, University of Bologna Department of Computer Science, 2006. 7, 61, 62

[MB10] M. Mazzara and A. Bhattacharyya. On modelling and analysis of dynamic reconfiguration of dependable real-time systems. In *Proceedings of the 3rd International Conference on Dependability (DEPEND 2010)*, 2010. 42

[MD87] S. E. Madnick and J. J. Donovan. *Operating Systems*. McGraw-Hill, Inc., 1987. 13

[MDEK95] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. In *Proceedings of the 5th European Software Engineering Conference*, pages 137–153, 1995. 26, 27, 28, 30

[MDK93] J. Magee, N. Dulay, and J. Kramer. Structuring parallel and distributed programs. *Software Engineering Journal (Special Issue)*, 8(2):73–82, 1993. 25, 26

[Met96] D. Le Metayer. Software architecture styles as graph grammars. In *Proceedings of the 4th Symposium on the Foundations of Software Engineering*, pages 15–23, 1996. 46

[MGK96] K. Moazami-Goudarzi and J. Kramer. Maintaining node consistency in the face of dynamic change. In *Proceedings of the 3rd International Conference on Configurable Distributed Systems*, pages 62–69, 1996. 28

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall International (U.K.) Limited, 1989. 45, 71, 108

[Mil99] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999. 6, 7, 46, 47, 50, 52, 65, 71, 81

[MKG99] J. Magee, J. Kramer, and D. Giannakopoulou. *Software Architecture*, chapter Behaviour Analysis of Software Architectures, pages 35–49. Kluwer Academic Publishers, 1999. 31

[MKS89] J. Magee, J. Kramer, and M. Sloman. Constructing distributed systems in conic. *IEEE Transactions on Software Engineering*, 15(6):663–675, 1989. 25

[MMR10] T. Mens, J. Magee, and B. Rumpe. Evolving software architecture descriptions of critical systems. *IEEE Computer*, 43(5):42–48, 2010. 41, 145

[MMSA+96] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4):54–63, 1996. 16

[MMSN98] L. E. Moser, P. M. Melliar-Smith, and P. Narasimhan. Consistent object replication in the eternal system. *Theory and Practice of Object Systems*, 4(2):81–92, 1998. 16

[MN82] N. Metropolis and E. C. Nelson. Early computing at los alamos. *Annals of the History of Computing*, 4(4):348–357, 1982. 1

[Mon04] J. Montgomery. A model for updating real-time applications. *Real-Time Systems*, 27(2):169–189, 2004. 5

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts i and ii. *Information and Computation*, 100(1):1–77, 1992. 46, 47

[MRT99] N. Medvidovic, D. S. Rosenblum, and R. N. Taylor. A language and environment for architecture-based software development and evolution. In *Proceedings of the 21st International Conference on Software Engineering*, pages 44–53, 1999. 21

[MT00] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000. 5, 21

[NP02] T. Nipkow and L. C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, 2002. 114

[OHE96] R. Orfali, D. Harkey, and J. Edwards. *The Essential Distributed Objects Survival Guide*. John Wiley and Sons, Inc., 1996. 13

[PDN86] R. Prieto-Diaz and J. M. Neighbors. Module interconnection languages. *Journal of Systems and Software*, 6(4):307–334, 1986. 19, 20

[Plo04] G. D. Plotkin. A structural approach to operational semantics. *The Journal of Logic and Algebraic Programming*, 60–61:17–139, 2004. 70

[Pra65] D. Prawitz. *Natural Deduction: A Proof-Theoretical Study*. Almqvist Wiksell, 1965. 8

[Pri99] D. G. Priddin. *Method Integration for Real-Time System Design and Verification*. PhD thesis, University of York Department of Computer Science, 1999. 25

[PT00] B. C. Pierce and D. N. Turner. *Proof, Language and Interaction: Essays in Honour of Robin Milner*, chapter Pict: A Programming Language Based on the Pi-Calculus, pages 455–494. MIT Press, 2000. 61

[Pur94] J. M. Purtilo. The polylith software bus. *ACM Transactions on Programming Languages and Systems*, 16(1):151–174, 1994. 19

[PV98] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, pages 176–185, 1998. 61

[Rot00] P. Rothmaler. *Introduction to Model Theory*. Gordon and Breach Science Publishers, 2000. 128

[RS94] M. D. Rice and S. B. Seidman. A formal model for module interconnection languages. *IEEE Transactions on Software Engineering*, 20(1):88–101, 1994. 45

[RS03] W. C. Rounds and H. Song. The ϕ-calculus: A language for distributed control of reconfigurable embedded systems. In *Proceedings of the 6th International Workshop on Hybrid Systems: Computation and Control*, pages 435–449, 2003. 61

[San93] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh Department of Computer Science, 1993. 54, 56, 57

[SDK+95] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik. Abstractions for software architecture and tools to support them. *IEEE Transactions on Software Engineering*, 21(4):314–335, 1995. 21, 22

[SF93] M. E. Segal and O. Frieder. On-the-fly program modification: Systems for dynamic updating. *IEEE Software*, 10(2):53–65, 1993. 14

[SM02] M. Solarski and H. Meling. Towards upgrading actively replicated servers on-the-fly. In *Proceedings of the 26th Annual International Computer Software and Applications Conference*, pages 1038–1043, 2002. 18

[Spi89] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International (U.K.) Limited, 1989. 45

[SRG96] L. Sha, R. Rajkumar, and M. Gagliardi. Evolving dependable real-time systems. In *Proceedings of the IEEE Aerospace Applications Conference*, pages 335–346, 1996. 12

[SVK97] D. B. Stewart, R. A. Volpe, and P. K. Khosla. Design of dynamically reconfigurable real-time software using port-based objects. *IEEE Transactions on Software Engineering*, 23(12):759–776, 1997. 5, 12

[SW01] D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001. 47, 57, 67

[Tho90] B. Thomsen. *Calculi for Higher Order Communicating Systems*. PhD thesis, University of London Imperial College of Science, Technology and Medicine Department of Computing, 1990. 54, 55, 56, 57

[TMMS01] L. A. Tewksbury, L. E. Moser, and P. M. Melliar-Smith. Live upgrades of corba applications using object replication. In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 488–497, 2001. 2, 16

[Uny01] A. Unyapoth. *Nomadic π-Calculi: Expressing and Verifying Infrastructure for Mobile Computation*. PhD thesis, University of Cambridge Computer Laboratory, 2001. 6

[Ves94] S. Vestal. Mode changes in a real-time architecture description language. In *Proceedings of the 2nd International Workshop on Configurable Distributed Systems*, pages 136–146. IEEE Computer Society, 1994. 5

[VM94] B. Victor and F. Moller. The mobility workbench – a tool for the π-calculus. In *Proceedings of the 6th International Conference on Computer Aided Verification*, pages 428–440. Springer-Verlag, 1994. 53

[Wer99] M. A. Wermelinger. *Specification of Software Architecture Reconfiguration*. PhD thesis, University of Lisbon Department of Informatics, 1999. 46

Appendix A

Proofs of basic CCS^{dp}

A.1 Lemma 4.2.1 $\forall p,q\in\mathcal{P}\,(p\rightsquigarrow_{of} q \implies \forall i\in\mathbb{N}\;\forall p'\in succ(p,i)(\exists q'\in succ(q,i)\;(p'\rightsquigarrow_{of} q')))$

Proof: uses induction on i (the number of consecutive transitions in $\mathcal{I}\cup\mathcal{R}$ from p).

Suppose $p,q\in\mathcal{P}\;(p\rightsquigarrow_{of} q)$.
For $i\in\mathbb{N}$, let $Prop(i)$ be this lemma for i.

The proof by induction involves discharging the following two proof obligations:

1. $\vdash Prop(0)$
2. $\vdash \forall i\in\mathbb{N}\;(Prop(i)\implies Prop(i+1))$

Base Case: Proof of Prop(0)

$i=0$ (by definition of $Prop(0)$)
$\implies succ(p,0)=\{p\}$ (by definition of $succ(p,i)$) $\wedge$
$succ(q,0)=\{q\}$ (by definition of $succ(q,i)$).
If $p'\in succ(p,0)$
then $p'=p$ (by definition of $succ(p,0)$)
$\implies p'\rightsquigarrow_{of} q$ ($\because p\rightsquigarrow_{of} q$, by definition of p and q).
Let $q'\triangleq q$.
$p'\rightsquigarrow_{of} q'$ ($\because p'\rightsquigarrow_{of} q$, and by definition of q') $\wedge$
$q'\in succ(q,0)$ (by definitions of $succ(q,0)$ and q')
$\implies Prop(0)$ holds (by definition of $Prop(0)$). Q.E.D.

Induction Step: Proof of $\forall i\in\mathbb{N}\;(Prop(i)\implies Prop(i+1))$

For $i\in\mathbb{N}$, assume $Prop(i)$ holds (inductive hypothesis).
If $p'\in succ(p,i+1)$
then $\exists p_1\in succ(p,i)\;(\exists\mu\in\mathcal{I}_{p_1}\cup\mathcal{R}_{p_1}\;(p_1\xrightarrow{\mu}p'))$ (by definition of $succ(p,i+1)$)

$\implies \exists q_1\in succ(q,i)\ (p_1\rightsquigarrow_{of} q_1)$ (by the inductive hypothesis)
$\implies \exists q_1\in succ(q,i)$
$(\forall \mu_{p_1}\in\mathcal{I}_{p_1}\cup\mathcal{R}_{p_1}\ \forall p_1'\in\mathcal{P}\ (p_1\xrightarrow{\mu_{p_1}}p_1' \implies \mu_{p_1}\in\mathcal{I}_{q_1}\cup\mathcal{R}_{q_1}\ \wedge\ \exists q'\in\mathcal{P}\ (q_1\xrightarrow{\mu_{p_1}}q'\ \wedge\ (p_1',q')\in S)))$
where S is a strong of-simulation on $\mathcal{P}$ which contains (p_1,q_1)
(by definition of $p_1\rightsquigarrow_{of} q_1$)
$\implies \exists q_1\in succ(q,i)\ (p_1\xrightarrow{\mu}p' \implies \mu\in\mathcal{I}_{q_1}\cup\mathcal{R}_{q_1}\ \wedge\ \exists q'\in\mathcal{P}\ (q_1\xrightarrow{\mu}q'\ \wedge\ (p',q')\in S))$
($\because$ if a predicate holds $\forall\mu_{p_1}\in\mathcal{I}_{p_1}\cup\mathcal{R}_{p_1}$, and $\mu\in\mathcal{I}_{p_1}\cup\mathcal{R}_{p_1}$, then the predicate holds for $\mu\in\mathcal{I}_{p_1}\cup\mathcal{R}_{p_1}$; and if a predicate holds $\forall p_1'\in\mathcal{P}$, and $p'\in\mathcal{P}$, then the predicate holds for $p'\in\mathcal{P}$)
$\implies \exists q_1\in succ(q,i)\ (\mu\in\mathcal{I}_{q_1}\cup\mathcal{R}_{q_1}\ \wedge\ \exists q'\in\mathcal{P}\ (q_1\xrightarrow{\mu}q'\ \wedge\ (p',q')\in S))$
($\because p_1\xrightarrow{\mu}p'$, and by modus ponens)
$\implies \exists q'\in succ(q,i+1)\ ((p',q')\in S)$ (by definition of $succ(p,i+1)$)
$\implies \exists q'\in succ(q,i+1)\ (p'\rightsquigarrow_{of} q')$
(by definition of $p'\rightsquigarrow_{of} q'$, $\because S$ is a strong of-simulation on $\mathcal{P}$).
$\therefore Prop(i+1)$ holds ($\because p'\in succ(p,i+1)$ is arbitrary).
$\therefore \forall i\in\mathbb{N}\ (Prop(i)\implies Prop(i+1))$ ($\because i\in\mathbb{N}$ is arbitrary). Q.E.D.

$\therefore \forall i\in\mathbb{N}\ Prop(i)$ holds (by induction).
$\therefore \forall p,q\in\mathcal{P}\ (p\rightsquigarrow_{of} q \implies \forall i\in\mathbb{N}\ \forall p'\in succ(p,i)\ (\exists q'\in succ(q,i)\ (p'\rightsquigarrow_{of} q')))$
($\because p,q\in\mathcal{P}$ are arbitrary). Q.E.D.

A.2 Corollary 4.2.1

$$\forall p,q\in\mathcal{P}\ (p\rightsquigarrow_{of} q \implies sfdrdepth(p)\leq sfdrdepth(q))$$

Proof: Suppose $p,q\in\mathcal{P}\ (p\rightsquigarrow_{of} q)$
then $p\in\mathcal{P}$
$\implies sfdrdepth(p)\in\mathbb{N}$ (by definition of $sfdrdepth$)
$\implies max\{fdrdepth(s)\mid s\in successors(p)\}\in\mathbb{N}$ (by definition of $sfdrdepth(p)$)
$\implies \exists p'\in successors(p)\ (fdrdepth(p')=sfdrdepth(p))$
(by arithmetic and definition of $sfdrdepth(p)$)
$\implies \exists i\in\mathbb{N}\ \exists p'\in succ(p,i)\ (fdrdepth(p')=sfdrdepth(p))$
(by definition of $successors(p)$ and set theory)
$\implies \exists q'\in succ(q,i)\ (p'\rightsquigarrow_{of} q')$ (by Lemma 4.2.1)
$\implies \exists q'\in successors(q)\ (p'\rightsquigarrow_{of} q')$ (by set theory and definition of $successors(q)$)
$\implies \exists q'\in successors(q)\ \forall\mu_{p'}\in\mathcal{I}_{p'}\cup\mathcal{R}_{p'}\ \forall p''\in\mathcal{P}$
$(p'\xrightarrow{\mu_{p'}}p'' \implies \mu_{p'}\in\mathcal{I}_{q'}\cup\mathcal{R}_{q'}\ \wedge\ \exists q''\in\mathcal{P}\ (q'\xrightarrow{\mu_{p'}}q''\ \wedge\ (p'',q'')\in S))$ where S is a strong of-simulation on $\mathcal{P}$ which contains (p',q')

(by definition of $p' \rightsquigarrow_{of} q'$)
$\Longrightarrow \exists q' \in successors(q)$ $(\mathcal{I}_{p'} \cup \mathcal{R}_{p'} \subseteq \mathcal{I}_{q'} \cup \mathcal{R}_{q'})$ (by set theory)
$\Longrightarrow \exists q' \in successors(q)$ $(\mathcal{R}_{p'} \subseteq \mathcal{R}_{q'})$
($\because \mathcal{I}$ and $\mathcal{R}$ are disjoint, by definitions of $\mathcal{I}$ and $\mathcal{R}$)
$\Longrightarrow \exists q' \in successors(q)$ $(fdrdepth(p') \leq fdrdepth(q'))$
(by definitions of $fdrdepth(p')$ and $fdrdepth(q')$)
$\Longrightarrow \exists q' \in successors(q)$ $(sfdrdepth(p) \leq fdrdepth(q'))$
($\because fdrdepth(p') = sfdrdepth(p)$, by definition of p').
Now $q' \in successors(q)$
$\Longrightarrow fdrdepth(q') \leq max\{fdrdepth(r) \mid r \in successors(q)\}$ (by arithmetic)
$\Longrightarrow fdrdepth(q') \leq sfdrdepth(q)$ (by definition of $sfdrdepth(q)$).
$\Longrightarrow sfdrdepth(p) \leq fdrdepth(q')$
($\because sfdrdepth(p) = fdrdepth(p') \wedge fdrdepth(p') \leq fdrdepth(q'))$ $\wedge$
$fdrdepth(q') \leq sfdrdepth(q)$
$\Longrightarrow sfdrdepth(p) \leq sfdrdepth(q)$ (by arithmetic)
$\Longrightarrow \forall p, q \in \mathcal{P}$ $(p \rightsquigarrow_{of} q \implies sfdrdepth(p) \leq sfdrdepth(q))$ ($\because p, q \in \mathcal{P}$ are arbitrary).
Q.E.D.

A.3 Lemma 4.3.3 ∀strong of-simulations U, V on $\mathcal{P}$ (UV is a strong of-simulation on $\mathcal{P}$)

Proof: For strong of-simulations U, V on $\mathcal{P}$, let $W \triangleq UV$.
W is a strong of-simulation on $\mathcal{P}$ iff $W \subseteq \mathcal{P}$ x $\mathcal{P}$ and $\forall (w1, w3) \in W$ the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied
(by definition of strong of-simulation on $\mathcal{P}$).
Therefore, we prove $W \subseteq \mathcal{P}$ x $\mathcal{P}$, then prove $\forall (w1, w3) \in W$ the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied.

U, V are strong of-simulations on $\mathcal{P}$ (by definitions of U and V)
$\Longrightarrow U \subseteq \mathcal{P}$ x $\mathcal{P} \wedge V \subseteq \mathcal{P}$ x $\mathcal{P}$ (by definition of strong of-simulation on $\mathcal{P}$)
$\Longrightarrow UV \subseteq \mathcal{P}$ x $\mathcal{P}$ (by composition of binary relations)
$\Longrightarrow W \subseteq \mathcal{P}$ x $\mathcal{P}$ (by definition of W).

Now $W = \emptyset \vee W \neq \emptyset$ (by set theory).
If $W = \emptyset$
then $\forall (w1, w3) \in W$ the *Observation* and *Fraction* conditions of strong of-simulation

on $\mathcal{P}$ are satisfied
($\because \emptyset$ satisfies all conditions).

If $W \neq \emptyset$
then the proof that $\forall (w1, w3) \in W$ the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied is as follows.

Proof the *Observation* condition of strong of-simulation on $\mathcal{P}$ is satisfied for $(w1, w3) \in W \ \wedge \ W \neq \emptyset$:

$(w1, w3) \in W$ ($\because W \neq \emptyset$)

$\implies \exists w2 \in \mathcal{P}((w1, w2) \in U \wedge (w2, w3) \in V)$
(by definition of W and composition of binary relations)
$\implies \forall \alpha_{w1} \in \mathcal{I}_{w1}$
$\forall w1' \in \mathcal{P}$
$(w1 \xrightarrow{\alpha_{w1}} w1' \implies$
$\alpha_{w1} \in \mathcal{I}_{w2} \ \wedge \ \exists w2' \in \mathcal{P}(w2 \xrightarrow{\alpha_{w1}} w2' \ \wedge \ (w1', w2') \in U)$
$)$
$\wedge$
$\forall \alpha_{w2} \in \mathcal{I}_{w2}$
$\forall w2' \in \mathcal{P}$
$(w2 \xrightarrow{\alpha_{w2}} w2' \implies$
$\alpha_{w2} \in \mathcal{I}_{w3} \ \wedge \ \exists w3' \in \mathcal{P}(w3 \xrightarrow{\alpha_{w2}} w3' \ \wedge \ (w2', w3') \in V)$
$)$
($\because U, V$ are strong of-simulations on $\mathcal{P}$
and by the *Observation* condition of strong of-simulation on $\mathcal{P}$)

$$
\begin{aligned}
\Longrightarrow\ & \forall \alpha_{w1} \in \mathcal{I}_{w1} \\
& \quad \forall w1' \in \mathcal{P} \\
& \quad\quad [(w1 \xrightarrow{\alpha_{w1}} w1' \Longrightarrow \\
& \quad\quad\quad \alpha_{w1} \in \mathcal{I}_{w2} \ \wedge\ \exists w2' \in \mathcal{P}(w2 \xrightarrow{\alpha_{w1}} w2' \ \wedge\ (w1', w2') \in U) \\
& \quad\quad) \\
& \quad\quad \wedge \\
& \quad\quad (\forall \alpha_{w2} \in \mathcal{I}_{w2} \\
& \quad\quad\quad \forall w2' \in \mathcal{P} \\
& \quad\quad\quad\quad (w2 \xrightarrow{\alpha_{w2}} w2' \Longrightarrow \\
& \quad\quad\quad\quad\quad \alpha_{w2} \in \mathcal{I}_{w3} \ \wedge\ \exists w3' \in \mathcal{P}(w3 \xrightarrow{\alpha_{w2}} w3' \ \wedge\ (w2', w3') \in V) \\
& \quad\quad\quad\quad) \\
& \quad\quad)]
\end{aligned}
$$

($\because$ α_{w1}, $\mathcal{I}_{w1}$ and $w1'$ do not occur free in the 2^{nd} conjunct of the outer conjunction; and (therefore) do not affect the truth value of the 2^{nd} conjunct)

$$
\begin{aligned}
\Longrightarrow\ & \forall \alpha_{w1} \in \mathcal{I}_{w1} \\
& \quad \forall w1' \in \mathcal{P} \\
& \quad\quad (w1 \xrightarrow{\alpha_{w1}} w1' \Longrightarrow \\
& \quad\quad\quad [\alpha_{w1} \in \mathcal{I}_{w2} \\
& \quad\quad\quad \wedge \\
& \quad\quad\quad \exists w2' \in \mathcal{P}(w2 \xrightarrow{\alpha_{w1}} w2' \ \wedge\ (w1', w2') \in U) \\
& \quad\quad\quad \wedge \\
& \quad\quad\quad (\forall w2' \in \mathcal{P} \\
& \quad\quad\quad\quad (w2 \xrightarrow{\alpha_{w1}} w2' \Longrightarrow \\
& \quad\quad\quad\quad\quad \alpha_{w1} \in \mathcal{I}_{w3} \ \wedge\ \exists w3' \in \mathcal{P}(w3 \xrightarrow{\alpha_{w1}} w3' \ \wedge\ (w2', w3') \in V) \\
& \quad\quad\quad\quad) \\
& \quad\quad\quad)] \\
& \quad\quad)
\end{aligned}
$$

($\because$ if a predicate is satisfied $\forall \alpha_{w2} \in \mathcal{I}_{w2}$, and $\alpha_{w1} \in \mathcal{I}_{w2}$, then the predicate is satisfied for $\alpha_{w1} \in \mathcal{I}_{w2}$)

$$
\begin{aligned}
\Longrightarrow\ & \forall \alpha_{w1} \in \mathcal{I}_{w1} \\
& \quad \forall w1' \in \mathcal{P} \\
& \quad\quad (w1 \xrightarrow{\alpha_{w1}} w1' \Longrightarrow \\
& \quad\quad\quad \alpha_{w1} \in \mathcal{I}_{w2} \\
& \quad\quad\quad \wedge \\
& \quad\quad\quad \exists w2' \in \mathcal{P} \\
& \quad\quad\quad\quad [(w2 \xrightarrow{\alpha_{w1}} w2' \ \wedge\ (w1', w2') \in U) \\
& \quad\quad\quad\quad \wedge \\
& \quad\quad\quad\quad (w2 \xrightarrow{\alpha_{w1}} w2' \Longrightarrow \\
& \quad\quad\quad\quad\quad \alpha_{w1} \in \mathcal{I}_{w3} \ \wedge\ \exists w3' \in \mathcal{P}(w3 \xrightarrow{\alpha_{w1}} w3' \ \wedge\ (w2', w3') \in V) \\
& \quad\quad\quad\quad)] \\
& \quad\quad)
\end{aligned}
$$

(∵ if a predicate is satisfied $\forall w2' \in \mathcal{P}$,
then the predicate is satisfied for a particular $w2' \in \mathcal{P}$)

$$
\begin{aligned}
\Longrightarrow\ & \forall \alpha_{w1} \in \mathcal{I}_{w1} \\
& \quad \forall w1' \in \mathcal{P} \\
& \quad\quad (w1 \xrightarrow{\alpha_{w1}} w1' \Longrightarrow \\
& \quad\quad\quad \alpha_{w1} \in \mathcal{I}_{w2} \\
& \quad\quad\quad \wedge \\
& \quad\quad\quad \exists w2' \in \mathcal{P} \\
& \quad\quad\quad\quad ((w1', w2') \in U \ \wedge\ \alpha_{w1} \in \mathcal{I}_{w3} \ \wedge\ \exists w3' \in \mathcal{P}(w3 \xrightarrow{\alpha_{w1}} w3' \ \wedge\ (w2', w3') \in V)) \\
& \quad\quad)
\end{aligned}
$$

(by modus ponens, using
$w2 \xrightarrow{\alpha_{w1}} w2' \ \wedge$
$(w2 \xrightarrow{\alpha_{w1}} w2' \Longrightarrow \alpha_{w1} \in \mathcal{I}_{w3} \ \wedge\ \exists w3' \in \mathcal{P}(w3 \xrightarrow{\alpha_{w1}} w3' \ \wedge\ (w2', w3') \in V))$
$\Longrightarrow \alpha_{w1} \in \mathcal{I}_{w3} \ \wedge\ \exists w3' \in \mathcal{P}(w3 \xrightarrow{\alpha_{w1}} w3' \ \wedge\ (w2', w3') \in V))$

$$
\begin{aligned}
\Longrightarrow\ & \forall \alpha_{w1} \in \mathcal{I}_{w1} \\
& \quad \forall w1' \in \mathcal{P} \\
& \quad\quad (w1 \xrightarrow{\alpha_{w1}} w1' \Longrightarrow \\
& \quad\quad\quad \exists w2' \in \mathcal{P} \\
& \quad\quad\quad\quad ((w1', w2') \in U \ \wedge\ \alpha_{w1} \in \mathcal{I}_{w3} \ \wedge\ \exists w3' \in \mathcal{P}(w3 \xrightarrow{\alpha_{w1}} w3' \ \wedge\ (w2', w3') \in V)) \\
& \quad\quad)
\end{aligned}
$$

(∵ $predicate_1 \wedge predicate_2 \Longrightarrow predicate_2$
and $\Longrightarrow$ is transitive)

$\implies \forall \alpha_{w1} \in \mathcal{I}_{w1}$
$\forall w1' \in \mathcal{P}$
$(w1 \xrightarrow{\alpha_{w1}} w1' \implies$
$\alpha_{w1} \in \mathcal{I}_{w3}$
$\wedge$
$\exists w2' \in \mathcal{P}$
$((w1', w2') \in U \ \wedge \ \exists w3' \in \mathcal{P}(w3 \xrightarrow{\alpha_{w1}} w3' \ \wedge \ (w2', w3') \in V))$
$)$
($\because \alpha_{w1} \in \mathcal{I}_{w3}$ remains in scope and remains in conjunction)

$\implies \forall \alpha_{w1} \in \mathcal{I}_{w1}$
$\forall w1' \in \mathcal{P}$
$(w1 \xrightarrow{\alpha_{w1}} w1' \implies$
$\alpha_{w1} \in \mathcal{I}_{w3}$
$\wedge$
$\exists w2' \in \mathcal{P}$
$(\exists w3' \in \mathcal{P}(w3 \xrightarrow{\alpha_{w1}} w3' \ \wedge \ (w1', w2') \in U \ \wedge \ (w2', w3') \in V))$
$)$
($\because w1', w2'$ are not restricted in the predicate quantified by $\exists w3' \in \mathcal{P}$;
and (therefore) the truth value of $(w1', w2') \in U$ is not affected by the predicate)

$\implies \forall \alpha_{w1} \in \mathcal{I}_{w1}$
$\forall w1' \in \mathcal{P}$
$(w1 \xrightarrow{\alpha_{w1}} w1' \implies$
$\alpha_{w1} \in \mathcal{I}_{w3}$
$\wedge$
$\exists w3' \in \mathcal{P}(w3 \xrightarrow{\alpha_{w1}} w3' \ \wedge \ (w1', w3') \in W)$
$)$
(by composition of binary relations and definition of W).

$\therefore$ The *Observation* condition of strong of-simulation on $\mathcal{P}$ is satisfied for $(w1, w3) \in W \ \wedge \ W \neq \emptyset$ Q.E.D.

Proof the *Fraction* condition of strong of-simulation on $\mathcal{P}$ is satisfied for $(w1, w3) \in W \ \wedge \ W \neq \emptyset$:

The similarity between the two conditions of strong of-simulation on $\mathcal{P}$ implies the proof of the *Fraction* condition is identical to the proof of the *Observation* condition with the following substitutions:
α_{w1} is replaced with τ_{r_X}

$\mathcal{I}_{w1}$ with $\mathcal{R}_{w1}$
$w1'$ with $w1''$
α_{w2} with τ_{r_Y}
$\mathcal{I}_{w2}$ with $\mathcal{R}_{w2}$
$w2'$ with $w2''$
$\mathcal{I}_{w3}$ with $\mathcal{R}_{w3}$
$w3'$ with $w3''$
and any reference to the *Observation* condition of strong of-simulation on $\mathcal{P}$ in the justifications is replaced with a corresponding reference to the *Fraction* condition.

Thus, the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied for $(w1, w3) \in W \ \wedge \ W \neq \emptyset$.
$\therefore \forall (w1, w3) \in W$ the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied
($\because (w1, w3) \in W$ is arbitrary). Q.E.D.
$\therefore W$ is a strong of-simulation on $\mathcal{P}$
(by definition of strong of-simulation on $\mathcal{P}$, $\because W \subseteq \mathcal{P}$ x $\mathcal{P}$)
$\implies UV$ is a strong of-simulation on $\mathcal{P}$ (by definition of W)
$\implies \forall$ strong of-simulations U, V on $\mathcal{P}$ (UV is a strong of-simulation on $\mathcal{P}$)
($\because U, V$ are arbitrary strong of-simulations on $\mathcal{P}$). Q.E.D.

A.4 Lemma 4.3.5 $\forall p \in \mathcal{P}^+(\mathcal{I}_p \cup \mathcal{R}_p \neq \emptyset)$

Proof: uses complete induction on the depth of inference of the applications of the $\mathcal{P}^+$ production rules.

For $n \in \mathbb{N}^+$, let $Prop(n)$ be this lemma for p obtained from applications of the $\mathcal{P}^+$ production rules with depth of inference n.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(1)$
2. $\vdash \forall n \in \mathbb{N}^+ \ (\forall m \in [1, n] Prop(m) \implies Prop(n + 1))$

Base Case: Proof of Prop(1)

For $r \in \mathcal{P}^+$ produced from a 1^{st} application of the $\mathcal{P}^+$ production rules, only rules 1 and 5 apply.

If rule 1 is applied
then $\exists \alpha \in \mathcal{I}\ \exists p \in \mathcal{P}\ (r = \alpha.p)$ (by definition of rule 1 of $\mathcal{P}^+$)
$\implies \alpha \in \mathcal{I}_r$ (by the *Sum* rule)
$\implies \alpha \in \mathcal{I}_r \cup \mathcal{R}_r$ (by set theory)
$\implies \mathcal{I}_r \cup \mathcal{R}_r \neq \emptyset$ (by set theory).

If rule 5 is applied
then $\exists \beta \in \mathcal{I}\ \exists X \in \mathcal{PN}\ (r = \beta.X)$ (by definition of rule 5 of $\mathcal{P}^+$)
$\implies \beta \in \mathcal{I}_r$ (by the *Sum* rule)
$\implies \beta \in \mathcal{I}_r \cup \mathcal{R}_r$ (by set theory)
$\implies \mathcal{I}_r \cup \mathcal{R}_r \neq \emptyset$ (by set theory).

$\therefore$ *Prop*(1) holds
($\because$ $r \in \mathcal{P}^+$ produced from a 1^{st} application of the $\mathcal{P}^+$ production rules is arbitrary).
Q.E.D.

Induction Step: Proof of $\forall \mathbf{n} \in \mathbb{N}^+\ (\forall \mathbf{m} \in [\mathbf{1}, \mathbf{n}]\mathbf{Prop(m)} \implies \mathbf{Prop(n+1)})$

For $n \in \mathbb{N}^+$, assume $\forall m \in [1, n]Prop(m)$ holds (inductive hypothesis).
For $r \in \mathcal{P}^+$ produced from applications of the $\mathcal{P}^+$ production rules with depth of inference $n + 1$, all 5 rules apply.

If rule 1 is applied, the proof of $\mathcal{I}_r \cup \mathcal{R}_r \neq \emptyset$ is identical to that in *Prop*(1) with rule 1 applied.

If rule 2 is applied
then $\exists p, q \in \mathcal{P}\ (p + q \in \mathcal{P}\ \wedge\ (p \in \mathcal{P}^+ \vee q \in \mathcal{P}^+)\ \wedge\ r = p + q)$
(by definition of rule 2 of $\mathcal{P}^+$).

If $p \in \mathcal{P}^+$
then p is a result of applications of the $\mathcal{P}^+$ production rules with inference depth m_p, with $m_p \in [1, n]$
(by definition of r)

$\Longrightarrow$ $Prop(m_p)$ holds (by the inductive hypothesis)
$\Longrightarrow$ $\mathcal{I}_p \cup \mathcal{R}_p \neq \emptyset$ (by definition of $Prop(m_p)$).
Now p, q are terms in a summation ($\because r = p + q$)
$\Longrightarrow$ $\mathcal{I}_{p+q} = \mathcal{I}_p \cup \mathcal{I}_q \wedge \mathcal{R}_p = \emptyset \wedge \mathcal{R}_q = \emptyset \wedge \mathcal{R}_{p+q} = \emptyset$
(by the syntax of processes in a summation and the *Sum* rule)
$\Longrightarrow$ $\mathcal{I}_{p+q} \cup \mathcal{R}_{p+q} = \mathcal{I}_p \cup \mathcal{R}_p \cup \mathcal{I}_q$ (by set theory)
$\Longrightarrow$ $\mathcal{I}_{p+q} \cup \mathcal{R}_{p+q} \neq \emptyset$ ($\because \mathcal{I}_p \cup \mathcal{R}_p \neq \emptyset$, and by set theory)
$\Longrightarrow$ $\mathcal{I}_r \cup \mathcal{R}_r \neq \emptyset$ ($\because r = p + q$).

If $q \in \mathcal{P}^+$
then q is a result of applications of the $\mathcal{P}^+$ production rules with inference depth m_q, with $m_q \in [1, n]$
(by definition of r)
$\Longrightarrow$ $Prop(m_q)$ holds (by the inductive hypothesis)
$\Longrightarrow$ $\mathcal{I}_q \cup \mathcal{R}_q \neq \emptyset$ (by definition of $Prop(m_q)$).
Now p, q are terms in a summation ($\because r = p + q$)
$\Longrightarrow$ $\mathcal{I}_{p+q} = \mathcal{I}_p \cup \mathcal{I}_q \wedge \mathcal{R}_p = \emptyset \wedge \mathcal{R}_q = \emptyset \wedge \mathcal{R}_{p+q} = \emptyset$
(by the syntax of processes in a summation and the *Sum* rule)
$\Longrightarrow$ $\mathcal{I}_{p+q} \cup \mathcal{R}_{p+q} = \mathcal{I}_p \cup \mathcal{I}_q \cup \mathcal{R}_q$ (by set theory)
$\Longrightarrow$ $\mathcal{I}_{p+q} \cup \mathcal{R}_{p+q} \neq \emptyset$ ($\because \mathcal{I}_q \cup \mathcal{R}_q \neq \emptyset$, and by set theory)
$\Longrightarrow$ $\mathcal{I}_r \cup \mathcal{R}_r \neq \emptyset$ ($\because r = p + q$).

If rule 3 is applied
then $\exists p, q \in \mathcal{P}$ $((p \in \mathcal{P}^+ \vee q \in \mathcal{P}^+) \wedge r = p|q)$ (by definition of rule 3 of $\mathcal{P}^+$).

If $p \in \mathcal{P}^+$
then p is a result of applications of the $\mathcal{P}^+$ production rules with inference depth m_p, with $m_p \in [1, n]$
(by definition of r)
$\Longrightarrow$ $Prop(m_p)$ holds (by the inductive hypothesis)
$\Longrightarrow$ $\mathcal{I}_p \cup \mathcal{R}_p \neq \emptyset$ (by definition of $Prop(m_p)$)
$\Longrightarrow$ $\mathcal{I}_{p|q} \cup \mathcal{R}_{p|q} \neq \emptyset$ (by the $L - Par$ rule)
$\Longrightarrow$ $\mathcal{I}_r \cup \mathcal{R}_r \neq \emptyset$ ($\because r = p|q$).

If $q \in \mathcal{P}^+$
then q is a result of applications of the $\mathcal{P}^+$ production rules with inference depth m_q, with $m_q \in [1, n]$

(by definition of r)
$\Longrightarrow$ $Prop(m_q)$ holds (by the inductive hypothesis)
$\Longrightarrow$ $\mathcal{I}_q \cup \mathcal{R}_q \neq \emptyset$ (by definition of $Prop(m_q)$)
$\Longrightarrow$ $\mathcal{I}_{p|q} \cup \mathcal{R}_{p|q} \neq \emptyset$ (by the $R - Par$ rule)
$\Longrightarrow$ $\mathcal{I}_r \cup \mathcal{R}_r \neq \emptyset$ ($\because r = p|q$).

If rule 4 is applied
then $\exists p \in \mathcal{P}\ \exists q \in \mathcal{P}^+\ (r = \frac{p}{q})$ (by definition of rule 4 of $\mathcal{P}^+$)
$\Longrightarrow$ $\tau_{r_q} \in \mathcal{R}_{\frac{p}{q}}$ (by the *Creat* rule, $\because q \in \mathcal{P}^+$ and $q \sim_{of} q$ (by Lemma 4.3.1))
$\Longrightarrow$ $\tau_{r_q} \in \mathcal{I}_{\frac{p}{q}} \cup \mathcal{R}_{\frac{p}{q}}$ (by set theory)
$\Longrightarrow$ $\mathcal{I}_{\frac{p}{q}} \cup \mathcal{R}_{\frac{p}{q}} \neq \emptyset$ (by set theory)
$\Longrightarrow$ $\mathcal{I}_r \cup \mathcal{R}_r \neq \emptyset$ ($\because r = \frac{p}{q}$).

If rule 5 is applied, the proof of $\mathcal{I}_r \cup \mathcal{R}_r \neq \emptyset$ is identical to that in $Prop(1)$ with rule 5 applied.
$\therefore$ $Prop(n+1)$ holds
($\because$ $r \in \mathcal{P}^+$ produced from applications of the $\mathcal{P}^+$ production rules with depth of inference $n+1$ is arbitrary).
$\therefore$ $\forall n \in \mathbb{N}^+$ $(\forall m \in [1, n] Prop(m) \Longrightarrow Prop(n+1))$ holds ($\because n \in \mathbb{N}^+$ is arbitrary). Q.E.D.

$\therefore$ $\forall n \in \mathbb{N}^+$ $Prop(n)$ holds (by complete induction).
$\therefore$ $\forall p \in \mathcal{P}^+$ $(\mathcal{I}_p \cup \mathcal{R}_p \neq \emptyset)$
($\because$ every p in $\mathcal{P}^+$ is the result of applications of the $\mathcal{P}^+$ production rules with finite depth of inference). Q.E.D.

A.5 Lemma 4.3.6 $\forall p \in \mathcal{P}(p \in \mathcal{P}^+ \iff \overline{\mathcal{R}}_p \neq \emptyset)$

Proof: consists of discharging the following two proof obligations. The proof obligation stating $\overline{\mathcal{R}}_p \neq \emptyset$ is necessary for $p \in \mathcal{P}^+$ is discharged by proving $p \in \mathcal{P}^+$ has a transition in $\overline{\mathcal{R}}$ defined by the *Delet* rule. The proof obligation stating $\overline{\mathcal{R}}_p \neq \emptyset$ is sufficient for $p \in \mathcal{P}^+$ is discharged using complete induction on the depth of inference of the applications of the LTS rules that determine the transitions of p in $\overline{\mathcal{R}}$.

1. $\vdash \forall p \in \mathcal{P}\,(p \in \mathcal{P}^+ \Longrightarrow \overline{\mathcal{R}}_p \neq \emptyset)$
2. $\vdash \forall p \in \mathcal{P}\,(\overline{\mathcal{R}}_p \neq \emptyset \Longrightarrow p \in \mathcal{P}^+)$

A.5.1 $\forall p \in \mathcal{P}(p \in \mathcal{P}^+ \implies \overline{\mathcal{R}}_p \neq \emptyset)$

Proof: If $p \in \mathcal{P}(p \in \mathcal{P}^+)$
then $p \in \mathcal{P}$ (by set theory and definition of $\mathcal{P}$)
$\implies p \sim_{of} p$ (by Lemma 4.3.1)
$\implies p \xrightarrow{\overline{\tau}_{r_p}} 0$ (by the *Delet* rule, $\because p \in \mathcal{P}^+$)
$\implies \overline{\tau}_{r_p} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$)
$\implies \overline{\mathcal{R}}_p \neq \emptyset$ (by set theory).
$\therefore \forall p \in \mathcal{P}(p \in \mathcal{P}^+ \implies \overline{\mathcal{R}}_p \neq \emptyset)$ ($\because p \in \mathcal{P}$ with $p \in \mathcal{P}^+$ is arbitrary). Q.E.D.

A.5.2 $\forall p \in \mathcal{P}(\overline{\mathcal{R}}_p \neq \emptyset \implies p \in \mathcal{P}^+)$

Proof: For $n \in \mathbb{N}^+$, let $Prop(n)$ be the proposition:
$\forall p, p'' \in \mathcal{P}\ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p\ (\overline{\mathcal{R}}_p \neq \emptyset \wedge p \xrightarrow{\overline{\tau}_{r_X}} p'' \implies p \in \mathcal{P}^+)$
for $p \xrightarrow{\overline{\tau}_{r_X}} p''$ determined by applications of LTS rules with depth of inference n.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(1)$
2. $\vdash \forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \implies Prop(n+1))$

Base Case: Proof of Prop(1)

For $p, p'' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$,
$\overline{\mathcal{R}}_p \neq \emptyset \ \wedge$ the transition $p \xrightarrow{\overline{\tau}_{r_X}} p''$ has depth of inference 1
(by the hypothesis of $Prop(1)$)
$\implies$ only the *Delet* rule determines the transition $p \xrightarrow{\overline{\tau}_{r_X}} p''$
(by definitions of the LTS rules):

If $p \xrightarrow{\overline{\tau}_{r_X}} p''$ (by the *Delet* rule)
then $p \in \mathcal{P}^+$ (by the hypothesis of *Delet*).
$\therefore \forall p, p'' \in \mathcal{P}\ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p\ (\overline{\mathcal{R}}_p \neq \emptyset \wedge p \xrightarrow{\overline{\tau}_{r_X}} p'' \implies p \in \mathcal{P}^+)$
for $p \xrightarrow{\overline{\tau}_{r_X}} p''$ determined by applications of LTS rules with depth of inference 1
($\because p, p'' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$ with $\overline{\mathcal{R}}_p \neq \emptyset$ and transition $p \xrightarrow{\overline{\tau}_{r_X}} p''$ with depth of inference 1 are arbitrary)
$\implies Prop(1)$ holds (by definition of $Prop(1)$). Q.E.D.

Induction Step: Proof of $\forall \mathbf{n} \in \mathbb{N}^+$ ($\forall \mathbf{m} \in [\mathbf{1}, \mathbf{n}]\mathbf{Prop}(\mathbf{m}) \implies \mathbf{Prop}(\mathbf{n+1})$)

For $n \in \mathbb{N}^+$, assume $\forall m \in [1, n] Prop(m)$ holds (inductive hypothesis).
For $p, p'' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$,
$\overline{\mathcal{R}}_p \neq \emptyset \ \wedge$ the transition $p \xrightarrow{\overline{\tau}_{r_X}} p''$ has depth of inference $n+1$
(by the hypothesis of $Prop(n+1)$)
$\implies n+1 \geq 2$ ($\because n \in \mathbb{N}^+$)
$\implies$ only the $L-Par$, $R-Par$ or $CompDelet$ rules determine the transition $p \xrightarrow{\overline{\tau}_{r_X}} p''$
(by definitions of the LTS rules):

If the $L-Par$ rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p''$
then $\exists u, u'', v \in \mathcal{P}$ ($p = u|v \ \wedge \ p'' = u''|v \ \wedge \ u|v \xrightarrow{\overline{\tau}_{r_X}} u''|v$) (by the $L-Par$ rule)
$\implies u \xrightarrow{\overline{\tau}_{r_X}} u''$ (by the hypothesis of $L-Par$)
$\implies \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_u$ (by definition of $\overline{\mathcal{R}}_u$)
$\implies \overline{\mathcal{R}}_u \neq \emptyset$ (by set theory).
Now the transition $u|v \xrightarrow{\overline{\tau}_{r_X}} u''|v$ has depth of inference $n+1$
($\because p = u|v \ \wedge \ p'' = u''|v \ \wedge \ p \xrightarrow{\overline{\tau}_{r_X}} p''$ has depth of inference $n+1$)
$\implies$ the transition $u \xrightarrow{\overline{\tau}_{r_X}} u''$ has depth of inference m_1, with $m_1 \in [1, n]$
($\because$ the transition $u|v \xrightarrow{\overline{\tau}_{r_X}} u''|v$ is inferred from the transition $u \xrightarrow{\overline{\tau}_{r_X}} u''$ using the $L-Par$ rule)
$\implies Prop(m_1)$ holds (by the inductive hypothesis)
$\implies u \in \mathcal{P}^+$
(by modus ponens, $\because u, u'' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_u \ \wedge \ \overline{\mathcal{R}}_u \neq \emptyset \ \wedge \ u \xrightarrow{\overline{\tau}_{r_X}} u''$ with depth of inference m_1)
$\implies u|v \in \mathcal{P}^+$ (by production rule 3 of $\mathcal{P}^+$, $\because u, v \in \mathcal{P}$)
$\implies p \in \mathcal{P}^+$ ($\because p = u|v$).

If the $R-Par$ rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p''$
then $\exists u, v, v'' \in \mathcal{P}$ ($p = u|v \ \wedge \ p'' = u|v'' \ \wedge \ u|v \xrightarrow{\overline{\tau}_{r_X}} u|v''$) (by the $R-Par$ rule)
$\implies v \xrightarrow{\overline{\tau}_{r_X}} v''$ (by the hypothesis of $R-Par$)
$\implies \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_v$ (by definition of $\overline{\mathcal{R}}_v$)
$\implies \overline{\mathcal{R}}_v \neq \emptyset$ (by set theory).
Now the transition $u|v \xrightarrow{\overline{\tau}_{r_X}} u|v''$ has depth of inference $n+1$
($\because p = u|v \ \wedge \ p'' = u|v'' \ \wedge \ p \xrightarrow{\overline{\tau}_{r_X}} p''$ has depth of inference $n+1$)
$\implies$ the transition $v \xrightarrow{\overline{\tau}_{r_X}} v''$ has depth of inference m_2, with $m_2 \in [1, n]$

($\because$ the transition $u|v \xrightarrow{\overline{\tau}_{r_X}} u|v''$ is inferred from the transition $v \xrightarrow{\overline{\tau}_{r_X}} v''$ using the $R - Par$ rule)
$\implies Prop(m_2)$ holds (by the inductive hypothesis)
$\implies v \in \mathcal{P}^+$
(by modus ponens, $\because v, v'' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_v \ \wedge \ \overline{\mathcal{R}}_v \neq \emptyset \ \wedge \ v \xrightarrow{\overline{\tau}_{r_X}} v''$ with depth of inference m_2)
$\implies u|v \in \mathcal{P}^+$ (by production rule 3 of $\mathcal{P}^+$, $\because u, v \in \mathcal{P}$)
$\implies p \in \mathcal{P}^+$ ($\because p = u|v$).

If the *CompDelet* rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p''$
then $\exists\, \overline{\tau}_{r_{X_1}}, \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}\ \exists p' \in \mathcal{P}\ (X \sim_{of} X_1|X_2 \ \wedge \ p \xrightarrow{\overline{\tau}_{r_{X_1}}} p' \ \wedge \ p' \xrightarrow{\overline{\tau}_{r_{X_2}}} p'')$
(by the hypothesis of *CompDelet*)
$\implies \overline{\tau}_{r_{X_1}} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$)
$\implies \overline{\mathcal{R}}_p \neq \emptyset$ (by set theory).
Now the transition $p \xrightarrow{\overline{\tau}_{r_X}} p''$ has depth of inference $n + 1$
(by the hypothesis of $Prop(n + 1)$)
$\implies$ the transition $p \xrightarrow{\overline{\tau}_{r_{X_1}}} p'$ has depth of inference m_3, with $m_3 \in [1, n]$
($\because$ the transition $p \xrightarrow{\overline{\tau}_{r_X}} p''$ is inferred from the transition $p \xrightarrow{\overline{\tau}_{r_{X_1}}} p'$ using the *CompDelet* rule)
$\implies Prop(m_3)$ holds (by the inductive hypothesis)
$\implies p \in \mathcal{P}^+$
(by modus ponens, $\because p, p' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_{X_1}} \in \overline{\mathcal{R}}_p \ \wedge \ \overline{\mathcal{R}}_p \neq \emptyset \ \wedge \ p \xrightarrow{\overline{\tau}_{r_{X_1}}} p'$ with depth of inference m_3).

$\therefore \forall p, p'' \in \mathcal{P}\ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p\ (\overline{\mathcal{R}}_p \neq \emptyset \ \wedge \ p \xrightarrow{\overline{\tau}_{r_X}} p'' \implies p \in \mathcal{P}^+)$
for $p \xrightarrow{\overline{\tau}_{r_X}} p''$ determined by applications of LTS rules with depth of inference $n + 1$
($\because p, p'' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$ with $\overline{\mathcal{R}}_p \neq \emptyset$ and transition $p \xrightarrow{\overline{\tau}_{r_X}} p''$ with depth of inference $n + 1$ are arbitrary)
$\implies Prop(n + 1)$ holds (by definition of $Prop(n + 1)$).
$\therefore \forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \implies Prop(n + 1))$ holds ($\because n \in \mathbb{N}^+$ is arbitrary). Q.E.D.

$\therefore \forall n \in \mathbb{N}^+\ Prop(n)$ holds (by complete induction).
$\therefore\ \forall p \in \mathcal{P}\ (\overline{\mathcal{R}}_p \neq \emptyset \implies p \in \mathcal{P}^+)$
($\because$ every transition of every $p \in \mathcal{P}$ is a result of one or more applications of the LTS semantic rules with finite depth of inference). Q.E.D.

$\therefore\ \forall p \in \mathcal{P}\,(p \in \mathcal{P}^+ \iff \overline{\mathcal{R}}_p \neq \emptyset)$
$(\because (predicate_1 \implies predicate_2) \wedge (predicate_2 \implies predicate_1) \iff$
$(predicate_1 \iff predicate_2))$. Q.E.D.

A.6 Lemma 4.3.7 $\forall p \in \mathcal{P}^0(\mathcal{I}_p \cup \mathcal{R}_p = \emptyset)$

Proof: uses complete induction on the depth of inference of the applications of the $\mathcal{P}^0$ production rules.

For $n \in \mathbb{N}^+$, let $Prop(n)$ be this lemma for p obtained from applications of the $\mathcal{P}^0$ production rules with depth of inference n.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(1)$
2. $\vdash \forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \implies Prop(n+1))$

Base Case: Proof of Prop(1)

For $r \in \mathcal{P}^0$ produced from a 1^{st} application of the $\mathcal{P}^0$ production rules, only rule 1 applies.

If rule 1 is applied
then $r = 0$ (by definition of rule 1 of $\mathcal{P}^0$).
Now $\mathcal{I}_0 = \emptyset \ \wedge\ \mathcal{R}_0 = \emptyset$
(by the LTS rules of basic CCS^{dp}, and because the transitions of the processes in $\mathcal{P}$ is the least relation on $\mathcal{P}$ that is defined by the LTS rules of basic CCS^{dp})
$\implies \mathcal{I}_0 \cup \mathcal{R}_0 = \emptyset$ (by set theory)
$\implies \mathcal{I}_r \cup \mathcal{R}_r = \emptyset$ $(\because r = 0)$.

$\therefore Prop(1)$ holds $(\because r$ can be only 0). Q.E.D.

Induction Step: Proof of $\forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \implies Prop(n+1))$

For $n \in \mathbb{N}^+$, assume $\forall m \in [1, n] Prop(m)$ holds (inductive hypothesis).
For $r \in \mathcal{P}^0$ produced from applications of the $\mathcal{P}^0$ production rules with depth of

inference $n + 1$, only rules 2, 3 and 4 rules apply.

If rule 2 is applied
then $\exists p, q \in \mathcal{P}^0$ $(p + q \in \mathcal{P} \ \wedge \ r = p + q)$ (by definition of rule 2 of $\mathcal{P}^0$)
$\Longrightarrow$ p, q are results of applications of the $\mathcal{P}^0$ production rules with depths of inference m_p, m_q (respectively), with $m_p, m_q \in [1, n]$
(by definition of r)
$\Longrightarrow$ $Prop(m_p)$ and $Prop(m_q)$ hold (by the inductive hypothesis)
$\Longrightarrow \mathcal{I}_p \cup \mathcal{R}_p = \emptyset \ \wedge \ \mathcal{I}_q \cup \mathcal{R}_q = \emptyset$ (by definitions of $Prop(m_p)$ and $Prop(m_q)$)
$\Longrightarrow \mathcal{I}_p = \emptyset \ \wedge \ \mathcal{I}_q = \emptyset$ (by set theory).
Now $p + q \in \mathcal{P}$
$\Longrightarrow \mathcal{I}_{p+q} = \mathcal{I}_p \cup \mathcal{I}_q \ \wedge \ \mathcal{R}_{p+q} = \emptyset$
(by the syntax of processes in a summation and the *Sum* rule)
$\Longrightarrow \mathcal{I}_{p+q} \cup \mathcal{R}_{p+q} = \mathcal{I}_p \cup \mathcal{I}_q$ (by set theory)
$\Longrightarrow \mathcal{I}_{p+q} \cup \mathcal{R}_{p+q} = \emptyset$ ($\because \mathcal{I}_p = \emptyset \ \wedge \ \mathcal{I}_q = \emptyset$, and by set theory)
$\Longrightarrow \mathcal{I}_r \cup \mathcal{R}_r = \emptyset$ ($\because r = p + q$).

If rule 3 is applied
then $\exists p, q \in \mathcal{P}^0$ $(r = p|q)$ (by definition of rule 3 of $\mathcal{P}^0$)
$\Longrightarrow$ p, q are results of applications of the $\mathcal{P}^0$ production rules with depths of inference m_p, m_q (respectively), with $m_p, m_q \in [1, n]$
(by definition of r)
$\Longrightarrow$ $Prop(m_p)$ and $Prop(m_q)$ hold (by the inductive hypothesis)
$\Longrightarrow \mathcal{I}_p \cup \mathcal{R}_p = \emptyset \ \wedge \ \mathcal{I}_q \cup \mathcal{R}_q = \emptyset$ (by definitions of $Prop(m_p)$ and $Prop(m_q)$).
If $p|q$ has a transition in $\mathcal{I} \cup \mathcal{R}$
then only the $L - Par$, $R - Par$, $React$, $L - React$ or $R - React$ rules determine the transition
(by definitions of the LTS rules).

If the $L - Par$ rule defines a transition of $p|q$ in $\mathcal{I} \cup \mathcal{R}$
then p has a transition in $\mathcal{I} \cup \mathcal{R}$ (by the hypothesis of $L - Par$)
$\Longrightarrow \mathcal{I}_p \cup \mathcal{R}_p \neq \emptyset$ (by set theory).
But $\mathcal{I}_p \cup \mathcal{R}_p = \emptyset$ ($\because Prop(m_p)$ holds; which is a contradiction).
$\therefore$ The $L - Par$ rule does not define a transition of $p|q$ in $\mathcal{I} \cup \mathcal{R}$.

If the $R - Par$ rule defines a transition of $p|q$ in $\mathcal{I} \cup \mathcal{R}$
then q has a transition in $\mathcal{I} \cup \mathcal{R}$ (by the hypothesis of $R - Par$)

$\Longrightarrow \mathcal{I}_q \cup \mathcal{R}_q \neq \emptyset$ (by set theory).
But $\mathcal{I}_q \cup \mathcal{R}_q = \emptyset$ ($\because$ $Prop(m_q)$ holds; which is a contradiction).
$\therefore$ The $R-Par$ rule does not define a transition of $p|q$ in $\mathcal{I} \cup \mathcal{R}$.

If the *React* rule defines a transition of $p|q$ in $\mathcal{I} \cup \mathcal{R}$
then p and q have complementary transitions in $\mathcal{L} \cup C$
(by the hypothesis of *React*)
$\Longrightarrow p$ has a transition in $\mathcal{I} \cup \mathcal{R}$ or q has a transition in $\mathcal{R}$
(by definitions of $\mathcal{I}$, C and complementary actions)
$\Longrightarrow \mathcal{I}_p \cup \mathcal{R}_p \neq \emptyset \ \vee \ \mathcal{I}_q \cup \mathcal{R}_q \neq \emptyset$ (by set theory).
But $\mathcal{I}_p \cup \mathcal{R}_p = \emptyset \ \wedge \ \mathcal{I}_q \cup \mathcal{R}_q = \emptyset$
($\because$ $Prop(m_p)$ and $Prop(m_q)$ hold; which is a contradiction).
$\therefore$ The *React* rule does not define a transition of $p|q$ in $\mathcal{I} \cup \mathcal{R}$.

If the $L-React$ rule defines a transition of $p|q$ in $\mathcal{I} \cup \mathcal{R}$
then $\exists\, \overline{\tau}_{r_{X_1}} \in \overline{\mathcal{R}}\ \exists p' \in \mathcal{P}\ (p \xrightarrow{\overline{\tau}_{r_{X_1}}} p')$ (by the hypothesis of $L-React$)
$\Longrightarrow \overline{\tau}_{r_{X_1}} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$)
$\Longrightarrow \overline{\mathcal{R}}_p \neq \emptyset$ (by set theory)
$\Longrightarrow p \in \mathcal{P}^+$ (by Lemma 4.3.6, $\because p \in \mathcal{P}$ (by definitions of p and $\mathcal{P}$, and set theory))
$\Longrightarrow \mathcal{I}_p \cup \mathcal{R}_p \neq \emptyset$ (by Lemma 4.3.5).
But $\mathcal{I}_p \cup \mathcal{R}_p = \emptyset$ ($\because$ $Prop(m_p)$ holds; which is a contradiction).
$\therefore$ The $L-React$ rule does not define a transition of $p|q$ in $\mathcal{I} \cup \mathcal{R}$.

If the $R-React$ rule defines a transition of $p|q$ in $\mathcal{I} \cup \mathcal{R}$
then $\exists\, \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}\ \exists q' \in \mathcal{P}\ (q \xrightarrow{\overline{\tau}_{r_{X_2}}} q')$ (by the hypothesis of $R-React$)
$\Longrightarrow \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}_q$ (by definition of $\overline{\mathcal{R}}_q$)
$\Longrightarrow \overline{\mathcal{R}}_q \neq \emptyset$ (by set theory)
$\Longrightarrow q \in \mathcal{P}^+$ (by Lemma 4.3.6, $\because q \in \mathcal{P}$ (by definitions of q and $\mathcal{P}$, and set theory))
$\Longrightarrow \mathcal{I}_q \cup \mathcal{R}_q \neq \emptyset$ (by Lemma 4.3.5).
But $\mathcal{I}_q \cup \mathcal{R}_q = \emptyset$ ($\because$ $Prop(m_q)$ holds; which is a contradiction).
$\therefore$ The $R-React$ rule does not define a transition of $p|q$ in $\mathcal{I} \cup \mathcal{R}$.

$\therefore \mathcal{I}_{p|q} \cup \mathcal{R}_{p|q} = \emptyset$
(by the LTS rules of basic CCS^{dp}, and because the transitions of the processes in $\mathcal{P}$ is the least relation on $\mathcal{P}$ that is defined by the LTS rules of basic CCS^{dp})
$\Longrightarrow \mathcal{I}_r \cup \mathcal{R}_r = \emptyset$ ($\because r = p|q$).

If rule 4 is applied
then $\exists p \in \mathcal{P}\ \exists q \in \mathcal{P}^0\ (r = \frac{p}{q})$ (by definition of rule 4 of $\mathcal{P}^0$)
$\Longrightarrow q$ is a result of applications of the $\mathcal{P}^0$ production rules with inference depth m_q, with $m_q \in [1, n]$
(by definition of r)
$\Longrightarrow Prop(m_q)$ holds (by the inductive hypothesis)
$\Longrightarrow \mathcal{I}_q \cup \mathcal{R}_q = \emptyset$ (by definition of $Prop(m_q)$)
$\Longrightarrow q \notin \mathcal{P}^+$ ($\because q \in \mathcal{P}^+ \Longrightarrow \mathcal{I}_q \cup \mathcal{R}_q \neq \emptyset$ (by Lemma 4.3.5); which is a contradiction)
$\Longrightarrow \mathcal{R}_{\frac{p}{q}} = \emptyset$ (by the *Creat* rule)
$\Longrightarrow \mathcal{I}_{\frac{p}{q}} \cup \mathcal{R}_{\frac{p}{q}} = \emptyset$
($\because$ a fraction process does not have an input or an output or a τ transition (by the LTS rules of basic CCS^{dp}), and by set theory)
$\Longrightarrow \mathcal{I}_r \cup \mathcal{R}_r = \emptyset$ ($\because r = \frac{p}{q}$).

$\therefore Prop(n+1)$ holds
($\because\ r \in \mathcal{P}^0$ produced from applications of the $\mathcal{P}^0$ production rules with depth of inference $n+1$ is arbitrary).
$\therefore \forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \Longrightarrow Prop(n+1))$ holds ($\because n \in \mathbb{N}^+$ is arbitrary). Q.E.D.

$\therefore \forall n \in \mathbb{N}^+\ Prop(n)$ holds (by complete induction).
$\therefore \forall p \in \mathcal{P}^0\ (\mathcal{I}_p \cup \mathcal{R}_p = \emptyset)$
($\because$ every p in $\mathcal{P}^0$ is the result of applications of the $\mathcal{P}^0$ production rules with finite depth of inference). Q.E.D.

A.7 Theorem 4.3.3 $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{of} p \ \wedge\ p \sim_{of} z|p)$

Proof: consists of discharging the following two proof obligations. Each proof obligation is discharged by defining a binary relation S on $\mathcal{P}$ which contains the pair of processes that are required to be strongly of-bisimilar, proving S is a strong of-simulation on $\mathcal{P}$, then proving S is a strong of-bisimulation on $\mathcal{P}$.

1. $\vdash \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{of} p)$
2. $\vdash \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p \sim_{of} z|p)$

A.7.1 $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{of} p)$

Proof: If $\exists$ strong of-bisimulation S on $\mathcal{P}$ with $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 ((p|z, p) \in S)$
then $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{of} p)$ (by definition of $p|z \sim_{of} p$).
Therefore, we find such an S.

Let $S \triangleq \{(p|z, p) \mid p \in \mathcal{P}\ \wedge\ z \in \mathcal{P}^0\}$.

A.7.1.1 S is a strong of-simulation on $\mathcal{P}$

$S \subseteq \mathcal{P} \times \mathcal{P}\ \wedge\ \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 ((p|z, p) \in S)$ (by definition of S).

We prove for $(p|z, p)$ in S that the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied.

$(p|z, p)$ satisfies the Observation and Fraction conditions

If $p|z$ has a transition in $\mathcal{I} \cup \mathcal{R}$
then only the $L - Par$, $R - Par$, $React$, $L - React$ or $R - React$ rules determine the transition
(by the syntax of $p|z$ and definitions of the LTS rules).

If the $L - Par$ rule defines a transition δ of $p|z$ in $\mathcal{I} \cup \mathcal{R}$
then $p|z \xrightarrow{\delta} p'|z$ (by the $L - Par$ rule)
$\Longrightarrow p \xrightarrow{\delta} p'$ (by the hypothesis of $L - Par$).
And $(p'|z, p') \in S$ (by definition of S).

$z \in \mathcal{P}^0$ (by definition of z)
$\Longrightarrow \mathcal{I}_z \cup \mathcal{R}_z = \emptyset$ (by Lemma 4.3.7)
$\Longrightarrow z$ has no transition in $\mathcal{I} \cup \mathcal{R}$ (by set theory and definitions of $\mathcal{I}_z$ and $\mathcal{R}_z$)
$\Longrightarrow$ the $R - Par$ rule does not define a transition of $p|z$ in $\mathcal{I} \cup \mathcal{R}$
($\because$ the hypothesis of $R - Par$ does not hold).

If the $React$ rule defines a transition of $p|z$ in $\mathcal{I} \cup \mathcal{R}$
then p, z have complementary transitions in $\mathcal{L} \cup \mathcal{C}$
(by the hypothesis of $React$, and definitions of complementary transitions, $\mathcal{L}$ and $\mathcal{C}$)
$\Longrightarrow \mathcal{L}_z \cup \mathcal{C}_z \neq \emptyset$ (by set theory and definitions of $\mathcal{L}_z$ and $\mathcal{C}_z$)
$\Longrightarrow \mathcal{I}_z \cup \mathcal{C}_z \neq \emptyset$ (by set theory and definition of $\mathcal{I}_z$).

But $\mathcal{I}_z \cup \mathcal{R}_z = \emptyset$ (by Lemma 4.3.7) $\wedge$ $\overline{\mathcal{R}}_z = \emptyset$ (by Lemma 4.3.8)
$\Longrightarrow \mathcal{I}_z \cup \mathcal{R}_z \cup \overline{\mathcal{R}}_z = \emptyset$ (by set theory)
$\Longrightarrow \mathcal{I}_z \cup C_z = \emptyset$ (by definition of C_z; which is a contradiction).
$\therefore$ The *React* rule does not define a transition of $p|z$ in $\mathcal{I} \cup \mathcal{R}$.

If the $L-React$ rule or the $R-React$ rule defines a transition of $p|z$ in $\mathcal{I} \cup \mathcal{R}$
then z has a transition in $\overline{\mathcal{R}}$
(by the hypotheses of $L-React$ and $R-React$)
$\Longrightarrow \overline{\mathcal{R}}_z \neq \emptyset$ (by set theory and definition of $\overline{\mathcal{R}}_z$).
But $\overline{\mathcal{R}}_z = \emptyset$ (by Lemma 4.3.8; which is a contradiction).
$\therefore$ Neither the $L-React$ rule nor the $R-React$ rule defines a transition of $p|z$ in $\mathcal{I} \cup \mathcal{R}$.

$\therefore \forall (p|z, p) \in S$ the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied
($\because (p|z, p) \in S$ is arbitrary)
$\Longrightarrow S$ is a strong of-simulation on $\mathcal{P}$
(by definition of strong of-simulation on $\mathcal{P}$). Q.E.D.

A.7.1.2 S is a strong of-bisimulation on $\mathcal{P}$

We prove S is a strong of-bisimulation on $\mathcal{P}$ by proving S^{-1} is a strong of-simulation on $\mathcal{P}$.
$S^{-1} = \{(p, p|z) \mid p \in \mathcal{P}\ \wedge\ z \in \mathcal{P}^0\}$ (by definitions of S and inverse binary relations)
$\Longrightarrow S^{-1} \subseteq \mathcal{P} \times \mathcal{P}\ \wedge\ \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 ((p, p|z) \in S^{-1})$ (by definition of S^{-1}).

For $(p, p|z) \in S^{-1}$, if $p \xrightarrow{\delta} p'$ with $\delta \in \mathcal{I} \cup \mathcal{R}$
then $p|z \xrightarrow{\delta} p'|z$ (by the $L-Par$ rule) $\wedge$ $(p', p'|z) \in S^{-1}$ (by definition of S^{-1}).

$\therefore \forall (p, p|z) \in S^{-1}$ the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied
($\because (p, p|z) \in S^{-1}$ is arbitrary)
$\Longrightarrow S^{-1}$ is a strong of-simulation on $\mathcal{P}$
(by definition of strong of-simulation on $\mathcal{P}$).
$\therefore S$ is a strong of-bisimulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$).
$\therefore \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{of} p)$ (by definition of $p|z \sim_{of} p$). Q.E.D.

A.7.2 $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p \sim_{of} z|p)$

Proof: If $\exists$ strong of-bisimulation S on $\mathcal{P}$ with $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0((p, z|p) \in S)$
then $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p \sim_{of} z|p)$ (by definition of $p \sim_{of} z|p$).
Therefore, we find such an S.

Let $S \triangleq \{(p, z|p) \mid p \in \mathcal{P}\ \wedge\ z \in \mathcal{P}^0\}$.

A.7.2.1 S is a strong of-simulation on $\mathcal{P}$

$S \subseteq \mathcal{P}$ x $\mathcal{P}$ $\wedge$ $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0((p, z|p) \in S)$ (by definition of S).

We prove for $(p, z|p)$ in S that the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied.

$(\mathbf{p}, \mathbf{z|p})$ satisfies the Observation and Fraction conditions

If $p \xrightarrow{\delta} p'$ with $\delta \in \mathcal{I} \cup \mathcal{R}$
then $z|p \xrightarrow{\delta} z|p'$ (by the $R - Par$ rule) $\wedge$ $(p', z|p') \in S$ (by definition of S).

$\therefore \forall (p, z|p) \in S$ the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied
($\because (p, z|p) \in S$ is arbitrary)
$\Longrightarrow S$ is a strong of-simulation on $\mathcal{P}$
(by definition of strong of-simulation on $\mathcal{P}$). Q.E.D.

A.7.2.2 S is a strong of-bisimulation on $\mathcal{P}$

We prove S is a strong of-bisimulation on $\mathcal{P}$ by proving S^{-1} is a strong of-simulation on $\mathcal{P}$.
$S^{-1} = \{(z|p, p) \mid p \in \mathcal{P}\ \wedge\ z \in \mathcal{P}^0\}$ (by definitions of S and inverse binary relations)
$\Longrightarrow S^{-1} \subseteq \mathcal{P}$ x $\mathcal{P}$ $\wedge$ $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0((z|p, p) \in S^{-1})$ (by definition of S^{-1}).

For $(z|p, p) \in S^{-1}$, if $z|p$ has a transition in $\mathcal{I} \cup \mathcal{R}$
then only the $L - Par$, $R - Par$, $React$, $L - React$ or $R - React$ rules determine the transition
(by the syntax of $z|p$ and definitions of the LTS rules).

$z \in \mathcal{P}^0$ (by definition of z)
$\Longrightarrow \mathcal{I}_z \cup \mathcal{R}_z = \emptyset$ (by Lemma 4.3.7)
$\Longrightarrow z$ has no transition in $\mathcal{I} \cup \mathcal{R}$ (by set theory and definitions of $\mathcal{I}_z$ and $\mathcal{R}_z$)
$\Longrightarrow$ the $L - Par$ rule does not define a transition of $z|p$ in $\mathcal{I} \cup \mathcal{R}$
($\because$ the hypothesis of $L - Par$ does not hold).

If the $R - Par$ rule defines a transition δ of $z|p$ in $\mathcal{I} \cup \mathcal{R}$
then $z|p \xrightarrow{\delta} z|p'$ (by the $R - Par$ rule)
$\Longrightarrow p \xrightarrow{\delta} p'$ (by the hypothesis of $R - Par$).
And $(z|p', p') \in S^{-1}$ (by definition of S^{-1}).

If the *React* rule defines a transition of $z|p$ in $\mathcal{I} \cup \mathcal{R}$
then z has a transition in $\mathcal{L} \cup C$ (by the hypothesis of *React*)
$\Longrightarrow \mathcal{L}_z \cup C_z \neq \emptyset$ (by set theory and definitions of $\mathcal{L}_z$ and C_z)
$\Longrightarrow \mathcal{I}_z \cup C_z \neq \emptyset$ (by set theory and definition of $\mathcal{I}_z$).
But $\mathcal{I}_z \cup \mathcal{R}_z = \emptyset$ (by Lemma 4.3.7) $\ \wedge\ \overline{\mathcal{R}}_z = \emptyset$ (by Lemma 4.3.8)
$\Longrightarrow \mathcal{I}_z \cup \mathcal{R}_z \cup \overline{\mathcal{R}}_z = \emptyset$ (by set theory)
$\Longrightarrow \mathcal{I}_z \cup C_z = \emptyset$ (by definition of C_z; which is a contradiction).
$\therefore$ The *React* rule does not define a transition of $z|p$ in $\mathcal{I} \cup \mathcal{R}$.

If the $L - React$ rule or the $R - React$ rule defines a transition of $z|p$ in $\mathcal{I} \cup \mathcal{R}$
then z has a transition in $\overline{\mathcal{R}}$ (by the hypotheses of $L - React$ and $R - React$)
$\Longrightarrow \overline{\mathcal{R}}_z \neq \emptyset$ (by set theory and definition of $\overline{\mathcal{R}}_z$).
But $\overline{\mathcal{R}}_z = \emptyset$ (by Lemma 4.3.8; which is a contradiction).
$\therefore$ Neither the $L - React$ rule nor the $R - React$ rule defines a transition of $z|p$ in $\mathcal{I} \cup \mathcal{R}$.

$\therefore\ \forall (z|p, p) \in S^{-1}$ the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied
($\because (z|p, p) \in S^{-1}$ is arbitrary)
$\Longrightarrow S^{-1}$ is a strong of-simulation on $\mathcal{P}$
(by definition of strong of-simulation on $\mathcal{P}$).
$\therefore S$ is a strong of-bisimulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$).
$\therefore \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p \sim_{of} z|p)$ (by definition of $p \sim_{of} z|p$). Q.E.D.

$\therefore \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{of} p \ \wedge\ p \sim_{of} z|p)$. Q.E.D.

A.8 Theorem 4.3.4

$\sim_{of}$ Preserves the Elementary Contexts $\alpha.[\cdot] + M$, $\frac{[\cdot]}{P}$ and $\frac{P}{[\cdot]}$

Proof: consists of discharging the following three proof obligations. Each proof obligation is discharged by defining a binary relation S on $\mathcal{P}$ which contains the pair of processes that are required to be strongly of-bisimilar, proving S is a strong of-simulation on $\mathcal{P}$, then proving S is a strong of-bisimulation on $\mathcal{P}$.

1. $\vdash \forall p,q \in \mathcal{P}\,(p \sim_{of} q \implies \forall \alpha \in \mathcal{I}\,(\alpha.p + M \sim_{of} \alpha.q + M)$
   where M is any summation in $\mathcal{P}$)
2. $\vdash \forall p,q \in \mathcal{P}\,(p \sim_{of} q \implies \forall r \in \mathcal{P}\,(\frac{p}{r} \sim_{of} \frac{q}{r}))$
3. $\vdash \forall p,q \in \mathcal{P}\,(p \sim_{of} q \implies \forall r \in \mathcal{P}\,(\frac{r}{p} \sim_{of} \frac{r}{q}))$

A.8.1 $\forall p,q \in \mathcal{P}$ (if $p \sim_{of} q$ then $\forall \alpha \in \mathcal{I}\ (\alpha.p + M \sim_{of} \alpha.q + M)$ where M is any summation in $\mathcal{P}$)

Proof: If $\exists$ strong of-bisimulation S on $\mathcal{P}$ with $\forall \alpha \in \mathcal{I}\ ((\alpha.p + M, \alpha.q + M) \in S)$ where M is any summation in $\mathcal{P}$ and p,q are any processes in $\mathcal{P}$ such that $p \sim_{of} q$
then $\forall p,q \in \mathcal{P}$ (if $p \sim_{of} q$ then $\forall \alpha \in \mathcal{I}\ (\alpha.p + M \sim_{of} \alpha.q + M)$ where M is any summation in $\mathcal{P}$)
(by definition of $\alpha.p + M \sim_{of} \alpha.q + M$).
Therefore, we find such an S.

Let $S \triangleq \{(\alpha.p + M, \alpha.q + M), (p,q), (r,r) \mid$
$\alpha \in \mathcal{I}\ \wedge\ p,q \in \mathcal{P}\,(p \sim_{of} q)\ \wedge\ M$ is any summation in $\mathcal{P}\ \wedge\ r \in \mathcal{P}\}$.

We prove S is a strong of-simulation on $\mathcal{P}$.
$S \subseteq \mathcal{P} \times \mathcal{P}\ \wedge$
$\forall \alpha \in \mathcal{I}$ and $\forall p,q \in \mathcal{P}$ such that $p \sim_{of} q$ and $\forall$summation $M \in \mathcal{P}\ (\alpha.p + M, \alpha.q + M) \in S$
(by definition of S).

The transitions of $\alpha.p + M$ in $\mathcal{I} \cup \mathcal{R}$ are defined by the *Sum* rule only
(by the syntax of $\alpha.p + M$ and definitions of the LTS rules):
$\alpha.p + M \xrightarrow{\alpha} p$ and $\alpha.q + M \xrightarrow{\alpha} q$ (by the *Sum* rule),

and $p \sim_{of} q$ (by definition of p and q)
$\Longrightarrow (p, q) \in S$ (by definition of S).
If $M \xrightarrow{\delta} s$ for some $\delta \in \mathcal{I} \cup \mathcal{R}$ and some $s \in \mathcal{P}$
then $\alpha.p + M \xrightarrow{\delta} s$ and $\alpha.q + M \xrightarrow{\delta} s$ (by the *Sum* rule),
and $(s, s) \in S$ ($\because \forall r \in \mathcal{P}\ ((r, r) \in S)$, by definition of S).

If $p \xrightarrow{\beta} p'$ for some $\beta \in \mathcal{I} \cup \mathcal{R}$ and some $p' \in \mathcal{P}$
then $q \xrightarrow{\beta} q'$ for some $q' \in \mathcal{P}\ \wedge\ p' \sim_{of} q'$ ($\because p \sim_{of} q$, by definition of p and q)
$\Longrightarrow (p', q') \in S$ (by definition of S).

If $r \xrightarrow{\gamma} r'$ for some $\gamma \in \mathcal{I} \cup \mathcal{R}$ and some $r' \in \mathcal{P}$
then $r \xrightarrow{\gamma} r'$
and $(r', r') \in S$ (by definition of S).

$\therefore S$ is a strong of-simulation on $\mathcal{P}$ (by definition of strong of-simulation on $\mathcal{P}$).

We prove S is a strong of-bisimulation on $\mathcal{P}$ by proving S^{-1} is a strong of-simulation on $\mathcal{P}$.
$S^{-1} = \{(\alpha.q + M, \alpha.p + M), (q, p), (r, r) \mid$
$\alpha \in \mathcal{I}\ \wedge\ p, q \in \mathcal{P}\ (p \sim_{of} q)\ \wedge\ M$ is any summation in $\mathcal{P}\ \wedge\ r \in \mathcal{P}\}$
(by definitions of S and inverse binary relations)
$\Longrightarrow S^{-1} = \{(\alpha.q + M, \alpha.p + M), (q, p), (r, r) \mid$
$\alpha \in \mathcal{I}\ \wedge\ q, p \in \mathcal{P}\ (q \sim_{of} p)\ \wedge\ M$ is any summation in $\mathcal{P}\ \wedge\ r \in \mathcal{P}\}$
($\because \sim_{of}$ is symmetric, by Lemma 4.3.2)
$\Longrightarrow S^{-1} \subseteq \mathcal{P} \times \mathcal{P}\ \wedge$
$\forall \alpha \in \mathcal{I}$ and $\forall p, q \in \mathcal{P}$ such that $p \sim_{of} q$ and $\forall$summation $M \in \mathcal{P}\ (\alpha.q + M, \alpha.p + M) \in S^{-1}$
(by definition of S^{-1}).
The proof that S^{-1} is a strong of-simulation on $\mathcal{P}$ is identical to the proof that S is a strong of-simulation on $\mathcal{P}$ with the following substitutions:
p is replaced with q
q with p
p' with q'
q' with p'
S with S^{-1}.
Thus, S^{-1} is a strong of-simulation on $\mathcal{P}$
(by definition of strong of-simulation on $\mathcal{P}$).
$\therefore S$ is a strong of-bisimulation on $\mathcal{P}$

(by definition of strong of-bisimulation on $\mathcal{P}$).
$\therefore \forall p, q \in \mathcal{P}$ (if $p \sim_{of} q$ then $\forall \alpha \in \mathcal{I}$ $(\alpha.p + M \sim_{of} \alpha.q + M)$ where M is any summation in $\mathcal{P}$)
(by definition of $\alpha.p + M \sim_{of} \alpha.q + M$). Q.E.D.

A.8.2 $\forall p, q \in \mathcal{P}$ (if $p \sim_{of} q$ then $\forall r \in \mathcal{P}$ $(\frac{p}{r} \sim_{of} \frac{q}{r})$)

Proof: If $\exists$ strong of-bisimulation S on $\mathcal{P}$ with $\forall r \in \mathcal{P}$ $((\frac{p}{r}, \frac{q}{r}) \in S)$ for any processes p, q in $\mathcal{P}$ such that $p \sim_{of} q$
then $\forall p, q \in \mathcal{P}$ (if $p \sim_{of} q$ then $\forall r \in \mathcal{P}$ $(\frac{p}{r} \sim_{of} \frac{q}{r})$) (by definition of $\frac{p}{r} \sim_{of} \frac{q}{r}$).
Therefore, we find such an S.

Let $S \triangleq \{(\frac{p}{r}, \frac{q}{r}), (p, q) \mid p, q, r \in \mathcal{P} \ \wedge \ p \sim_{of} q\}$.

We prove S is a strong of-simulation on $\mathcal{P}$.
$S \subseteq \mathcal{P} \times \mathcal{P} \ \wedge \ \forall p, q, r \in \mathcal{P}$ $(p \sim_{of} q \Longrightarrow (\frac{p}{r}, \frac{q}{r}) \in S)$ (by definition of S).

The transitions of $\frac{p}{r}$ in $\mathcal{I} \cup \mathcal{R}$ are defined by the *Creat* rule only
(by the syntax of $\frac{p}{r}$ and definitions of the LTS rules):
If $r \in \mathcal{P}^0$
then $\frac{p}{r} \in \mathcal{P}^0$ (by definition of rule 4 of $\mathcal{P}^0$)
$\Longrightarrow \mathcal{I}_{\frac{p}{r}} \cup \mathcal{R}_{\frac{p}{r}} = \emptyset$ (by Lemma 4.3.7)
$\Longrightarrow \frac{p}{r}$ has no transition in $\mathcal{I} \cup \mathcal{R}$ (by set theory).
If $r \notin \mathcal{P}^0$
then $r \in \mathcal{P}^+$ (by Theorem 4.3.2),
and if $r \sim_{of} X$
then $\frac{p}{r} \xrightarrow{\tau_{r_X}} p \ \wedge \ \frac{q}{r} \xrightarrow{\tau_{r_X}} q$ (by the *Creat* rule).
$p \sim_{of} q$ (by definition of p and q)
$\Longrightarrow (p, q) \in S$ (by definition of S).

If $p \xrightarrow{\beta} p'$ for some $\beta \in \mathcal{I} \cup \mathcal{R}$ and some $p' \in \mathcal{P}$
then $q \xrightarrow{\beta} q'$ for some $q' \in \mathcal{P} \ \wedge \ p' \sim_{of} q'$ ($\because p \sim_{of} q$, by definition of p and q)
$\Longrightarrow (p', q') \in S$ (by definition of S).

$\therefore S$ is a strong of-simulation on $\mathcal{P}$ (by definition of strong of-simulation on $\mathcal{P}$).

We prove S is a strong of-bisimulation on $\mathcal{P}$ by proving S^{-1} is a strong of-simulation

on $\mathcal{P}$.
$S^{-1} = \{(\frac{q}{r}, \frac{p}{r}), (q,p) \mid p,q,r \in \mathcal{P} \ \wedge \ p \sim_{of} q\}$
(by definitions of S and inverse binary relations)
$\Longrightarrow S^{-1} = \{(\frac{q}{r}, \frac{p}{r}), (q,p) \mid q,p,r \in \mathcal{P} \ \wedge \ q \sim_{of} p\}$ ($\because \sim_{of}$ is symmetric, by Lemma 4.3.2)
$\Longrightarrow S^{-1} \subseteq \mathcal{P} \times \mathcal{P} \ \wedge \ \forall p,q,r \in \mathcal{P}$ such that $p \sim_{of} q$ $(\frac{q}{r}, \frac{p}{r}) \in S^{-1}$ (by definition of S^{-1}).
The proof that S^{-1} is a strong of-simulation on $\mathcal{P}$ is identical to the proof that S is a strong of-simulation on $\mathcal{P}$ with the following substitutions:
p is replaced with q
q with p
p' with q'
q' with p'
S with S^{-1}.
Thus, S^{-1} is a strong of-simulation on $\mathcal{P}$
(by definition of strong of-simulation on $\mathcal{P}$).
$\therefore S$ is a strong of-bisimulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$).
$\therefore \forall p,q \in \mathcal{P}$ (if $p \sim_{of} q$ then $\forall r \in \mathcal{P}$ $(\frac{p}{r} \sim_{of} \frac{q}{r})$) (by definition of $\frac{p}{r} \sim_{of} \frac{q}{r}$). Q.E.D.

A.8.3 $\forall p,q \in \mathcal{P}$ (if $p \sim_{of} q$ then $\forall r \in \mathcal{P}$ $(\frac{r}{p} \sim_{of} \frac{r}{q})$)

Proof: If $\exists$ strong of-bisimulation S on $\mathcal{P}$ with $\forall r \in \mathcal{P}$ $((\frac{r}{p}, \frac{r}{q}) \in S)$ for any processes p,q in $\mathcal{P}$ such that $p \sim_{of} q$
then $\forall p,q \in \mathcal{P}$ (if $p \sim_{of} q$ then $\forall r \in \mathcal{P}$ $(\frac{r}{p} \sim_{of} \frac{r}{q})$) (by definition of $\frac{r}{p} \sim_{of} \frac{r}{q}$).
Therefore, we find such an S.

Let $S \triangleq \{(\frac{r}{p}, \frac{r}{q}), (r,r) \mid p,q,r \in \mathcal{P} \ \wedge \ p \sim_{of} q\}$.

We prove S is a strong of-simulation on $\mathcal{P}$.
$S \subseteq \mathcal{P} \times \mathcal{P} \ \wedge \ \forall p,q,r \in \mathcal{P}$ $(p \sim_{of} q \Longrightarrow (\frac{r}{p}, \frac{r}{q}) \in S)$ (by definition of S).

The transitions of $\frac{r}{p}$ in $\mathcal{I} \cup \mathcal{R}$ are defined by the *Creat* rule only
(by the syntax of $\frac{r}{p}$ and definitions of the LTS rules):
If $p \in \mathcal{P}^0$
then $\frac{r}{p} \in \mathcal{P}^0$ (by definition of rule 4 of $\mathcal{P}^0$)
$\Longrightarrow \mathcal{I}_{\frac{r}{p}} \cup \mathcal{R}_{\frac{r}{p}} = \emptyset$ (by Lemma 4.3.7)
$\Longrightarrow \frac{r}{p}$ has no transition in $\mathcal{I} \cup \mathcal{R}$ (by set theory).
If $p \notin \mathcal{P}^0$

then $p \in \mathcal{P}^+$ (by Theorem 4.3.2)
$\implies q \in \mathcal{P}^+$ (by Lemma 4.3.9, $\because p \sim_{of} q \ \wedge \ p \notin \mathcal{P}^0$),
and if $p \sim_{of} X$
then $q \sim_{of} X$ ($\because p \sim_{of} q \ \wedge \ \sim_{of}$ is an equivalence relation, by Theorem 4.3.1).
$\therefore \frac{r}{p} \xrightarrow{\tau_{r_X}} r \implies \frac{r}{q} \xrightarrow{\tau_{r_X}} r$ (by the *Creat* rule),
and $(r, r) \in S$ (by definition of S).

If $r \xrightarrow{\gamma} r'$ for some $\gamma \in \mathcal{I} \cup \mathcal{R}$ and some $r' \in \mathcal{P}$
then $r \xrightarrow{\gamma} r'$
and $(r', r') \in S$ (by definition of S).

$\therefore S$ is a strong of-simulation on $\mathcal{P}$ (by definition of strong of-simulation on $\mathcal{P}$).

We prove S is a strong of-bisimulation on $\mathcal{P}$ by proving S^{-1} is a strong of-simulation on $\mathcal{P}$.
$S^{-1} = \{(\frac{r}{q}, \frac{r}{p}), (r, r) \mid p, q, r \in \mathcal{P} \ \wedge \ p \sim_{of} q\}$
(by definitions of S and inverse binary relations)
$\implies S^{-1} = \{(\frac{r}{q}, \frac{r}{p}), (r, r) \mid q, p, r \in \mathcal{P} \ \wedge \ q \sim_{of} p\}$ ($\because \sim_{of}$ is symmetric, by Lemma 4.3.2)
$\implies S^{-1} \subseteq \mathcal{P} \times \mathcal{P} \ \wedge \ \forall p, q, r \in \mathcal{P}$ such that $p \sim_{of} q$ $(\frac{r}{q}, \frac{r}{p}) \in S^{-1}$ (by definition of S^{-1}).
The proof that S^{-1} is a strong of-simulation on $\mathcal{P}$ is identical to the proof that S is a strong of-simulation on $\mathcal{P}$ with the following substitutions:
p is replaced with q
q with p
S with S^{-1}.
Thus, S^{-1} is a strong of-simulation on $\mathcal{P}$
(by definition of strong of-simulation on $\mathcal{P}$).
$\therefore S$ is a strong of-bisimulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$).
$\therefore \forall p, q \in \mathcal{P}$ (if $p \sim_{of} q$ then $\forall r \in \mathcal{P}$ $(\frac{r}{p} \sim_{of} \frac{r}{q})$) (by definition of $\frac{r}{p} \sim_{of} \frac{r}{q}$). Q.E.D.

A.9 Lemma 4.3.13 ∀strong dp-simulations U, V on $\mathcal{P}$ (UV is a strong dp-simulation on $\mathcal{P}$)

Proof: For strong dp-simulations U, V on $\mathcal{P}$, let $W \triangleq UV$.
W is a strong dp-simulation on $\mathcal{P}$ iff $W \subseteq \mathcal{P} \times \mathcal{P}$ and the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ hold $\forall (w1, w3) \in W$

(by definition of strong dp-simulation on $\mathcal{P}$).
Therefore, we prove $W \subseteq \mathcal{P}$ x $\mathcal{P}$, then prove the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ hold $\forall(w1, w3) \in W$.

U, V are strong dp-simulations on $\mathcal{P}$ (by definitions of U, V)
$\Longrightarrow$ U, V are strong of-simulations on $\mathcal{P}$ (by Lemma 4.3.10)
$\Longrightarrow$ UV is a strong of-simulation on $\mathcal{P}$ (by Lemma 4.3.3)
$\Longrightarrow$ W is a strong of-simulation on $\mathcal{P}$ (by definition of W)
$\Longrightarrow$ $W \subseteq \mathcal{P}$ x $\mathcal{P}$ $\wedge$
for all elements of W the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied
(by definition of strong of-simulation on $\mathcal{P}$)
$\Longrightarrow$ for all elements of W the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
($\because$ the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are the same as the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$, respectively).

It remains to prove that for all elements of W the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied.

Proof the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ holds for $(w1, w3) \in W$:

The similarity between the three conditions of strong dp-simulation on $\mathcal{P}$ implies that the proof of the *Deletion* condition of strong dp-simulation is identical to the proof of the *Observation* condition of strong of-simulation in Lemma 4.3.3 (see A.3) with the following substitutions:
α_{w1} is replaced with $\overline{\tau}_{r_Y}$
$\mathcal{I}_{w1}$ with $\overline{\mathcal{R}}_{w1}$
$w1'$ with $w1'''$
α_{w2} with $\overline{\tau}_{r_Z}$
$\mathcal{I}_{w2}$ with $\overline{\mathcal{R}}_{w2}$
$w2'$ with $w2'''$
$\mathcal{I}_{w3}$ with $\overline{\mathcal{R}}_{w3}$
$w3'$ with $w3'''$
and in the justifications: U, V are strong dp-simulations on $\mathcal{P}$, and any reference to the *Observation* condition of strong of-simulation on $\mathcal{P}$ is replaced with a

corresponding reference to the *Deletion* condition of strong dp-simulation on $\mathcal{P}$.

$\therefore$ The *Deletion* condition of strong dp-simulation on $\mathcal{P}$ holds $\forall (w1, w3) \in W$
($\because (w1, w3) \in W$ is arbitrary)
$\implies W$ is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$, $\because W \subseteq \mathcal{P}$ x $\mathcal{P}$ and for all elements of W the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied)
$\implies UV$ is a strong dp-simulation on $\mathcal{P}$ (by definition of W)
$\implies \forall$strong dp-simulations U, V on $\mathcal{P}$ (UV is a strong dp-simulation on $\mathcal{P}$)
($\because U, V$ are arbitrary strong dp-simulations on $\mathcal{P}$). Q.E.D.

A.10 Lemma 4.3.15

$$\forall p, q \in \mathcal{P}\ (p|q \in \mathcal{P}^+ \implies factors^+_m(p|q) \neq \emptyset_m)$$

Proof: uses complete induction on the depth of the factor tree of $p|q$.

For $n \in (\mathbb{N}^+ - \{1\})$, let $Prop(n)$ be this lemma for $p|q$ with factor tree of depth n.
Depth of the factor tree of $p|q \geq 2$ (by Definition 4.3.3 and Definition 4.3.4).
Therefore, the proof by complete induction on the depth of the factor tree of $p|q$ involves discharging the following two proof obligations:

1. $\vdash Prop(2)$
2. $\vdash \forall n \in (\mathbb{N}^+ - \{1\})\ (\forall m \in [2, n] Prop(m) \implies Prop(n+1))$

Base Case: Proof of Prop(2)

For $p, q \in \mathcal{P}$, $p|q \in \mathcal{P}^+$ with factor tree of depth 2 (by the hypothesis of $Prop(2)$)
$\implies factors_m(p) = \emptyset_m \ \wedge \ factors_m(q) = \emptyset_m$
(by definitions of $factors_m(p)$ and $factors_m(q)$)
$\implies factors_m(p|q) = \{p\}_m \uplus \{q\}_m$ (by definition of $factors_m(p|q)$ and set theory)
$\implies factors_m(p|q) = \{p, q\}_m$ (by multiset theory).
Now $p|q \in \mathcal{P}^+$ (by the hypothesis of $Prop(2)$)
$\implies p \in \mathcal{P}^+ \ \vee \ q \in \mathcal{P}^+$ (by the hypothesis of production rule 3 of $\mathcal{P}^+$, $\because p, q \in \mathcal{P}$)
$\implies factors_m(p) = \emptyset_m \ \wedge \ p \in \mathcal{P}^+ \ \vee \ factors_m(q) = \emptyset_m \ \wedge \ q \in \mathcal{P}^+$
($\because factors_m(p) = \emptyset_m \ \wedge \ factors_m(q) = \emptyset_m$)
$\implies p \in factors^+_m(p|q) \ \vee \ q \in factors^+_m(p|q)$
(by definition of $factors^+_m(p|q)$, $\because factors_m(p|q) = \{p, q\}_m$)

$\implies factors_m^+(p|q)\neq\emptyset_m$ (by set theory).
$\therefore \forall p,q\in\mathcal{P}\,(p|q\in\mathcal{P}^+ \implies factors_m^+(p|q)\neq\emptyset_m)$ for $p|q$ with factor tree of depth 2
($\because p,q\in\mathcal{P}$ with $p|q\in\mathcal{P}^+$ and factor tree of $p|q$ of depth 2 are arbitrary)
$\implies Prop(2)$ holds (by definition of $Prop(2)$). Q.E.D.

Induction Step:

Proof of $\forall \mathbf{n}\in(\mathbb{N}^+-\{\mathbf{1}\})\,(\forall \mathbf{m}\in[\mathbf{2},\mathbf{n}]\mathbf{Prop(m)} \implies \mathbf{Prop(n+1)})$

For $n\in(\mathbb{N}^+-\{1\})$, assume $\forall m\in[2,n]Prop(m)$ holds (inductive hypothesis).
For $p,q\in\mathcal{P}$, $p|q\in\mathcal{P}^+$ with factor tree of depth $n+1$
(by the hypothesis of $Prop(n+1)$)
$\implies factors_m(p|q)=\{p\}_m \uplus \{q\}_m \uplus factors_m(p) \uplus factors_m(q)$
(by definition of $factors_m(p|q)$)
$\implies factors_m(p|q)=\{p,q\}_m \uplus factors_m(p) \uplus factors_m(q)$ (by multiset theory)
$\implies \{p,q\}_m\subseteq factors_m(p|q)\ \wedge$
$factors_m(p)\subseteq factors_m(p|q)\ \wedge\ factors_m(q)\subseteq factors_m(p|q)$
(by set theory).
Now $p|q\in\mathcal{P}^+$ (by the hypothesis of $Prop(n+1)$)
$\implies p\in\mathcal{P}^+\ \vee\ q\in\mathcal{P}^+$ (by the hypothesis of production rule 3 of $\mathcal{P}^+$, $\because p,q\in\mathcal{P}$)
$\implies (factors_m(p)=\emptyset_m\ \vee\ factors_m(p)\neq\emptyset_m)\ \wedge\ p\in\mathcal{P}^+\ \vee$
$(factors_m(q)=\emptyset_m\ \vee\ factors_m(q)\neq\emptyset_m)\ \wedge\ q\in\mathcal{P}^+$
($\because (factors_m(p)=\emptyset_m\ \vee\ factors_m(p)\neq\emptyset_m)\ \wedge\ (factors_m(q)=\emptyset_m\ \vee\ factors_m(q)\neq\emptyset_m)$)
$\implies factors_m(p)=\emptyset_m\ \wedge\ p\in\mathcal{P}^+\ \vee\ factors_m(p)\neq\emptyset_m\ \wedge\ p\in\mathcal{P}^+\ \vee$
$factors_m(q)=\emptyset_m\ \wedge\ q\in\mathcal{P}^+\ \vee\ factors_m(q)\neq\emptyset_m\ \wedge\ q\in\mathcal{P}^+$
($\because (predicate_1\ \vee\ predicate_2)\ \wedge\ predicate_3 \iff$
$predicate_1\ \wedge\ predicate_3\ \vee\ predicate_2\ \wedge\ predicate_3$).

If $factors_m(p)=\emptyset_m\ \wedge\ p\in\mathcal{P}^+$
then $p\in factors_m^+(p|q)$ (by definition of $factors_m^+(p|q)$, $\because \{p,q\}_m\subseteq factors_m(p|q)$)
$\implies factors_m^+(p|q)\neq\emptyset_m$ (by set theory).

If $factors_m(p)\neq\emptyset_m\ \wedge\ p\in\mathcal{P}^+$
then $\exists r,s\in\mathcal{P}\,(p=r|s)$ (by definition of $factors_m(p)$)
$\implies r|s\in\mathcal{P}^+$ ($\because p\in\mathcal{P}^+$) $\wedge$
$r|s$ has factor tree of depth m_p, with $m_p\in[2,n]$
($\because$ the factor tree of $p|q$ has depth $n+1$ and p is a subnode of $p|q$ in the factor tree of $p|q$)

$\implies Prop(m_p)$ holds (by the inductive hypothesis)
$\implies factors^+_m(r|s)\neq\emptyset_m$
(by modus ponens, $\because r,s\in\mathcal{P}$ with $r|s\in\mathcal{P}^+$ and $r|s$ has factor tree of depth m_p).
Now $factors^+_m(r|s)\subseteq factors_m(r|s)$ (by definition of $factors^+_m(r|s)$) $\wedge$
$factors_m(r|s)\subseteq factors_m(p|q)$ ($\because p=r|s\ \wedge\ factors_m(p)\subseteq factors_m(p|q)$)
$\implies factors^+_m(r|s)\subseteq factors_m(p|q)$ ($\because\subseteq$ is transitive, by multiset theory)
$\implies \exists f\in factors_m(p|q)\,(factors_m(f)=\emptyset_m\ \wedge\ f\in\mathcal{P}^+)$
($\because factors^+_m(r|s)\neq\emptyset_m$, and by set theory and definition of $factors^+_m(r|s)$)
$\implies f\in factors^+_m(p|q)$ (by definition of $factors^+_m(p|q)$)
$\implies factors^+_m(p|q)\neq\emptyset_m$ (by set theory).

If $factors_m(q)=\emptyset_m\ \wedge\ q\in\mathcal{P}^+$
then $q\in factors^+_m(p|q)$ (by definition of $factors^+_m(p|q)$, $\because \{p,q\}_m\subseteq factors_m(p|q)$)
$\implies factors^+_m(p|q)\neq\emptyset_m$ (by set theory).

If $factors_m(q)\neq\emptyset_m\ \wedge\ q\in\mathcal{P}^+$
then $\exists u,v\in\mathcal{P}\,(q=u|v)$ (by definition of $factors_m(q)$)
$\implies u|v\in\mathcal{P}^+$ ($\because q\in\mathcal{P}^+$) $\wedge$
$u|v$ has factor tree of depth m_q, with $m_q\in[2,n]$
($\because$ the factor tree of $p|q$ has depth $n+1$ and q is a subnode of $p|q$ in the factor tree of $p|q$)
$\implies Prop(m_q)$ holds (by the inductive hypothesis)
$\implies factors^+_m(u|v)\neq\emptyset_m$
(by modus ponens, $\because u,v\in\mathcal{P}$ with $u|v\in\mathcal{P}^+$ and $u|v$ has factor tree of depth m_q).
Now $factors^+_m(u|v)\subseteq factors_m(u|v)$ (by definition of $factors^+_m(u|v)$) $\wedge$
$factors_m(u|v)\subseteq factors_m(p|q)$ ($\because q=u|v\ \wedge\ factors_m(q)\subseteq factors_m(p|q)$)
$\implies factors^+_m(u|v)\subseteq factors_m(p|q)$ ($\because\subseteq$ is transitive, by multiset theory)
$\implies \exists g\in factors_m(p|q)\,(factors_m(g)=\emptyset_m\ \wedge\ g\in\mathcal{P}^+)$
($\because factors^+_m(u|v)\neq\emptyset_m$, and by set theory and definition of $factors^+_m(u|v)$)
$\implies g\in factors^+_m(p|q)$ (by definition of $factors^+_m(p|q)$)
$\implies factors^+_m(p|q)\neq\emptyset_m$ (by set theory).

$\therefore \forall p,q\in\mathcal{P}\,(p|q\in\mathcal{P}^+ \implies factors^+_m(p|q)\neq\emptyset_m)$ for $p|q$ with factor tree of depth $n+1$
($\because p,q\in\mathcal{P}$ with $p|q\in\mathcal{P}^+$ and factor tree of $p|q$ of depth $n+1$ are arbitrary)
$\implies Prop(n+1)$ holds (by definition of $Prop(n+1)$).
$\therefore \forall n\in(\mathbb{N}^+-\{1\})\,(\forall m\in[2,n]Prop(m) \implies Prop(n+1))$ holds
($\because n\in(\mathbb{N}^+-\{1\})$ is arbitrary). Q.E.D.

$\therefore \forall n \in (\mathbb{N}^+ - \{1\})\ Prop(n)$ holds (by complete induction).
$\therefore \forall p, q \in \mathcal{P}\ (p|q \in \mathcal{P}^+ \implies factors_m^+(p|q) \neq \emptyset_m)$
($\because$ for every $p, q \in \mathcal{P}$ with $p|q \in \mathcal{P}^+$, the depth of the factor tree of $p|q$ is finite). Q.E.D.

A.11 Lemma 4.3.16 $\forall p \in \mathcal{P}\ (p \in \mathcal{P}^0 \implies factors_m^+(p) = \emptyset_m)$

Proof: uses complete induction on the depth of the factor tree of p.

For $n \in \mathbb{N}^+$, let $Prop(n)$ be this lemma for p with factor tree of depth n.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(1)$
2. $\vdash \forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \implies Prop(n+1))$

Base Case: Proof of Prop(1)

For $p \in \mathcal{P}$, $p \in \mathcal{P}^0$ with factor tree of depth 1 (by the hypothesis of $Prop(1)$)
$\implies factors_m(p) = \emptyset_m$
(by Definition 4.3.3, Definition 4.3.4 and definition of $factors_m(p)$)
$\implies factors_m^+(p) \subseteq factors_m(p)$ (by definition of $factors_m^+(p)$) $\wedge$ $factors_m(p) = \emptyset_m$
$\implies factors_m^+(p) = \emptyset_m$ (by set theory).
$\therefore \forall p \in \mathcal{P}\ (p \in \mathcal{P}^0 \implies factors_m^+(p) = \emptyset_m)$ for p with factor tree of depth 1
($\because p \in \mathcal{P}$ with $p \in \mathcal{P}^0$ and factor tree of depth 1 is arbitrary)
$\implies Prop(1)$ holds (by definition of $Prop(1)$). Q.E.D.

Induction Step: Proof of $\forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \implies Prop(n+1))$

For $n \in \mathbb{N}^+$, assume $\forall m \in [1, n] Prop(m)$ holds (inductive hypothesis).
For $p \in \mathcal{P}$, $p \in \mathcal{P}^0$ with factor tree of depth $n+1$ (by the hypothesis of $Prop(n+1)$)
$\implies p$ has factor tree of depth ≥ 2 ($\because n + 1 \geq 2$ ($\because n \in \mathbb{N}^+$))
$\implies \exists r, s \in \mathcal{P}\ (p = r|s)$ (by Definition 4.3.3 and Definition 4.3.4)
$\implies r|s \in \mathcal{P}^0$ ($\because p \in \mathcal{P}^0$) $\wedge$
r, s have factor trees of depths m_r, m_s respectively, with $m_r, m_s \in [1, n]$
($\because p$ has factor tree of depth $n+1$)
$\implies r, s \in \mathcal{P}^0$ (by production rule 3 of $\mathcal{P}^0$) $\wedge$

$Prop(m_r)$ and $Prop(m_s)$ hold (by the inductive hypothesis)
$\implies factors_m^+(r) = \emptyset_m \ \wedge \ factors_m^+(s) = \emptyset_m$ (by modus ponens, $\because r, s \in \mathcal{P}$).
Now $factors_m(p) = \{r\}_m \uplus \{s\}_m \uplus factors_m(r) \uplus factors_m(s)$
(by definition of $factors_m(p)$, $\because p = r|s$).
If $factors_m^+(p) \neq \emptyset_m$
then $\exists f \in factors_m^+(p)$ (by set theory)
$\implies f \in factors_m(p) \ \wedge \ factors_m(f) = \emptyset_m \ \wedge \ f \in \mathcal{P}^+$ (by definition of $factors_m^+(p)$)
$\implies f \in \{r\}_m \ \vee \ f \in \{s\}_m \ \vee \ f \in factors_m(r) \ \vee \ f \in factors_m(s)$
(by definition of $factors_m(p)$).

If $f \in \{r\}_m$
then $f = r$ (by set theory)
$\implies f \in \mathcal{P}^0$ ($\because r \in \mathcal{P}^0$)
$\implies f \notin \mathcal{P}^+$ (by Theorem 4.3.2; which is a contradiction).
$\therefore f \notin \{r\}_m$.

If $f \in \{s\}_m$
then $f = s$ (by set theory)
$\implies f \in \mathcal{P}^0$ ($\because s \in \mathcal{P}^0$)
$\implies f \notin \mathcal{P}^+$ (by Theorem 4.3.2; which is a contradiction).
$\therefore f \notin \{s\}_m$.

If $f \in factors_m(r)$
then $f \in factors_m(r) \ \wedge \ factors_m(f) = \emptyset_m \ \wedge \ f \in \mathcal{P}^+$ (by definition of f)
$\implies f \in factors_m^+(r)$ (by definition of $factors_m^+(r)$)
$\implies factors_m^+(r) \neq \emptyset_m$ (by set theory; which is a contradiction).
$\therefore f \notin factors_m(r)$.

If $f \in factors_m(s)$
then $f \in factors_m(s) \ \wedge \ factors_m(f) = \emptyset_m \ \wedge \ f \in \mathcal{P}^+$ (by definition of f)
$\implies f \in factors_m^+(s)$ (by definition of $factors_m^+(s)$)
$\implies factors_m^+(s) \neq \emptyset_m$ (by set theory; which is a contradiction).
$\therefore f \notin factors_m(s)$.

$\therefore factors_m^+(p) = \emptyset_m$ (by contradiction).
$\therefore \forall p \in \mathcal{P}\ (p \in \mathcal{P}^0 \implies factors_m^+(p) = \emptyset_m)$ for p with factor tree of depth $n + 1$
($\because p \in \mathcal{P}$ with $p \in \mathcal{P}^0$ and factor tree of depth $n + 1$ is arbitrary)

$\Longrightarrow$ $Prop(n+1)$ holds (by definition of $Prop(n+1)$).
$\therefore \forall n \in \mathbb{N}^+ \; (\forall m \in [1,n] Prop(m) \Longrightarrow Prop(n+1))$ holds ($\because n \in \mathbb{N}^+$ is arbitrary). Q.E.D.

$\therefore \forall n \in \mathbb{N}^+ \; Prop(n)$ holds (by complete induction).
$\therefore \forall p \in \mathcal{P} \; (p \in \mathcal{P}^0 \Longrightarrow factors^+_m(p) = \emptyset_m)$
($\because$ every $p \in \mathcal{P}$ with $p \in \mathcal{P}^0$ has a factor tree of finite depth). Q.E.D.

A.12 Lemma 4.3.17 $\forall p, p' \in \mathcal{P}$

$$\forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p \; (p \xrightarrow{\overline{\tau}_{r_X}} p' \Longrightarrow p' = 0 \; \vee \; factors_m(p') \neq \emptyset_m)$$

Proof: uses complete induction on the depth of inference of the applications of the LTS rules that determine the transitions of p in $\overline{\mathcal{R}}$.

For $n \in \mathbb{N}^+$, let $Prop(n)$ be this lemma for $p \xrightarrow{\overline{\tau}_{r_X}} p'$ determined by applications of LTS rules with depth of inference n.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(1)$
2. $\vdash \forall n \in \mathbb{N}^+ \; (\forall m \in [1,n] Prop(m) \Longrightarrow Prop(n+1))$

Base Case: Proof of Prop(1)

For $p, p' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$, the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference 1
(by the hypothesis of $Prop(1)$)
$\Longrightarrow$ only the *Delet* rule determines the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
(by definitions of the LTS rules):

If the *Delet* rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
then $p' = 0$ (by the *Delet* rule)
$\Longrightarrow p' = 0 \; \vee \; factors_m(p') \neq \emptyset_m$ (by definition of $\vee$)
$\Longrightarrow \forall p, p' \in \mathcal{P} \; \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p \; (p \xrightarrow{\overline{\tau}_{r_X}} p' \Longrightarrow p' = 0 \; \vee \; factors_m(p') \neq \emptyset_m)$
for $p \xrightarrow{\overline{\tau}_{r_X}} p'$ determined by applications of LTS rules with depth of inference 1
($\because$ $p, p' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$ with transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ and depth of inference 1 are

arbitrary)
$\Longrightarrow$ *Prop*(1) holds (by definition of *Prop*(1)). Q.E.D.

Induction Step: Proof of $\forall \mathbf{n} \in \mathbb{N}^+ \, (\forall \mathbf{m} \in [\mathbf{1}, \mathbf{n}] \mathbf{Prop(m)} \Longrightarrow \mathbf{Prop(n+1)})$

For $n \in \mathbb{N}^+$, assume $\forall m \in [1, n] Prop(m)$ holds (inductive hypothesis).
For $p, p' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$, the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference $n+1$
(by the hypothesis of $Prop(n+1)$)
$\Longrightarrow p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference ≥ 2 ($\because n+1 \geq 2$ ($\because n \in \mathbb{N}^+$))
$\Longrightarrow$ only the $L-Par$, $R-Par$ or *CompDelet* rules determine the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
(by definitions of the LTS rules):

If the $L-Par$ rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
then $\exists r, r', s \in \mathcal{P} \, (p = r|s \ \wedge \ p' = r'|s \ \wedge \ r|s \xrightarrow{\overline{\tau}_{r_X}} r'|s)$ (by the $L-Par$ rule)
$\Longrightarrow factors_m(p') = \{r'\}_m \uplus \{s\}_m \uplus factors_m(r') \uplus factors_m(s)$
(by definition of $factors_m(r'|s)$)
$\Longrightarrow r' \in factors_m(p')$ (by set theory)
$\Longrightarrow factors_m(p') \neq \emptyset_m$ (by set theory)
$\Longrightarrow p' = 0 \ \vee \ factors_m(p') \neq \emptyset_m$ (by definition of $\vee$).

If the $R-Par$ rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
then $\exists r, s, s' \in \mathcal{P} \, (p = r|s \ \wedge \ p' = r|s' \ \wedge \ r|s \xrightarrow{\overline{\tau}_{r_X}} r|s')$ (by the $R-Par$ rule)
$\Longrightarrow factors_m(p') = \{r\}_m \uplus \{s'\}_m \uplus factors_m(r) \uplus factors_m(s')$
(by definition of $factors_m(r|s')$)
$\Longrightarrow r \in factors_m(p')$ (by set theory)
$\Longrightarrow factors_m(p') \neq \emptyset_m$ (by set theory)
$\Longrightarrow p' = 0 \ \vee \ factors_m(p') \neq \emptyset_m$ (by definition of $\vee$).

If the *CompDelet* rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
then $\exists \, \overline{\tau}_{r_{X_1}}, \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}} \, \exists u \in \mathcal{P} \, (X \sim_{of} X_1|X_2 \ \wedge \ p \xrightarrow{\overline{\tau}_{r_{X_1}}} u \ \wedge \ u \xrightarrow{\overline{\tau}_{r_{X_2}}} p')$
(by the hypothesis of *CompDelet*)
$\Longrightarrow \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}_u$ (by definition of $\overline{\mathcal{R}}_u$).
Now the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference $n+1$
(by the hypothesis of $Prop(n+1)$)
$\Longrightarrow$ the transition $u \xrightarrow{\overline{\tau}_{r_{X_2}}} p'$ has depth of inference m_u, with $m_u \in [1, n]$
($\because$ the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ is inferred from the transition $u \xrightarrow{\overline{\tau}_{r_{X_2}}} p'$ using the *CompDelet*

rule)
$\Longrightarrow$ $Prop(m_u)$ holds (by the inductive hypothesis)
$\Longrightarrow p' = 0 \ \vee \ factors_m(p') \neq \emptyset_m$
(by modus ponens, $\because u, p' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}_u \ \wedge \ u \xrightarrow{\overline{\tau}_{r_{X_2}}} p'$ with depth of inference m_u).

$\therefore \forall p, p' \in \mathcal{P} \ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p \ (p \xrightarrow{\overline{\tau}_{r_X}} p' \Longrightarrow p' = 0 \ \vee \ factors_m(p') \neq \emptyset_m)$
for $p \xrightarrow{\overline{\tau}_{r_X}} p'$ determined by applications of LTS rules with depth of inference $n+1$
($\because p, p' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$ with transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ and depth of inference $n+1$ are arbitrary)
$\Longrightarrow$ $Prop(n+1)$ holds (by definition of $Prop(n+1)$).
$\therefore \forall n \in \mathbb{N}^+ \ (\forall m \in [1, n] Prop(m) \Longrightarrow Prop(n+1))$ holds ($\because n \in \mathbb{N}^+$ is arbitrary). Q.E.D.

$\therefore \forall n \in \mathbb{N}^+ \ Prop(n)$ holds (by complete induction).
$\therefore \ \forall p, p' \in \mathcal{P} \ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p \ (p \xrightarrow{\overline{\tau}_{r_X}} p' \Longrightarrow p' = 0 \ \vee \ factors_m(p') \neq \emptyset_m)$
($\because$ every transition of every $p \in \mathcal{P}$ is a result of one or more applications of the LTS semantic rules with finite depth of inference). Q.E.D.

A.13 Lemma 4.3.18

$$\forall p, p' \in \mathcal{P} \ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$$
$$(p \xrightarrow{\overline{\tau}_{r_X}} p' \Longrightarrow p' = 0 \ \vee \ factors_m^+(p') \subset factors_m^+(p))$$

Proof: uses complete induction on the depth of inference of the applications of the LTS rules that determine the transitions of p in $\overline{\mathcal{R}}$.

For $n \in \mathbb{N}^+$, let $Prop(n)$ be this lemma for $p \xrightarrow{\overline{\tau}_{r_X}} p'$ determined by applications of LTS rules with depth of inference n.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(1)$
2. $\vdash \forall n \in \mathbb{N}^+ \ (\forall m \in [1, n] Prop(m) \Longrightarrow Prop(n+1))$

Base Case: Proof of Prop(1)

For $p, p' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$, the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference 1
(by the hypothesis of *Prop*(1))
$\Longrightarrow$ only the *Delet* rule determines the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
(by definitions of the LTS rules):

If the *Delet* rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
then $p' = 0$ (by the *Delet* rule)
$\Longrightarrow p' = 0 \ \vee \ factors_m^+(p') \subset factors_m^+(p)$ (by definition of $\vee$)
$\Longrightarrow \forall p, p' \in \mathcal{P} \ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p \ (p \xrightarrow{\overline{\tau}_{r_X}} p' \Longrightarrow p' = 0 \ \vee \ factors_m^+(p') \subset factors_m^+(p))$
for $p \xrightarrow{\overline{\tau}_{r_X}} p'$ determined by applications of LTS rules with depth of inference 1
($\because \ p, p' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$ with transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ and depth of inference 1 are arbitrary)
$\Longrightarrow$ *Prop*(1) holds (by definition of *Prop*(1)). Q.E.D.

Induction Step: Proof of $\forall \mathbf{n} \in \mathbb{N}^+ \ (\forall \mathbf{m} \in [\mathbf{1}, \mathbf{n}] \mathbf{Prop(m)} \Longrightarrow \mathbf{Prop(n+1)})$

For $n \in \mathbb{N}^+$, assume $\forall m \in [1, n] Prop(m)$ holds (inductive hypothesis).
For $p, p' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$, the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference $n+1$
(by the hypothesis of *Prop*$(n+1)$)
$\Longrightarrow p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference ≥ 2 ($\because n+1 \geq 2$ ($\because n \in \mathbb{N}^+$))
$\Longrightarrow$ only the $L - Par$, $R - Par$ or *CompDelet* rules determine the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
(by definitions of the LTS rules):

If the $L - Par$ rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
then $\exists r, r', s \in \mathcal{P} \ (p = r|s \ \wedge \ p' = r'|s \ \wedge \ r|s \xrightarrow{\overline{\tau}_{r_X}} r'|s)$ (by the $L - Par$ rule)
$\Longrightarrow r \xrightarrow{\overline{\tau}_{r_X}} r'$ (by the hypothesis of $L - Par$).
Now $r'|s \neq 0$ (by the syntax of $r'|s$ and 0)
$\Longrightarrow p' \neq 0$ ($\because p' = r'|s$).
Therefore, we prove $factors_m^+(p') \subset factors_m^+(p)$.

$\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_r$ ($\because r \xrightarrow{\overline{\tau}_{r_X}} r'$, and by definition of $\overline{\mathcal{R}}_r$)
$\Longrightarrow \overline{\mathcal{R}}_r \neq \emptyset$ (by set theory)
$\Longrightarrow r \in \mathcal{P}^+$ (by Lemma 4.3.6, $\because r \in \mathcal{P}$)
$\Longrightarrow r|s \in \mathcal{P}^+$ (by production rule 3 of $\mathcal{P}^+$, $\because r, s \in \mathcal{P}$)

$\implies factors^{+}_{m}(r|s) \neq \emptyset_m$ (by Lemma 4.3.15, $\because r, s \in \mathcal{P}$)
$\implies factors^{+}_{m}(p) \neq \emptyset_m$ ($\because p = r|s$).
Now $factors^{+}_{m}(p') = \emptyset_m \ \vee \ factors^{+}_{m}(p') \neq \emptyset_m$ (by set theory).

If $factors^{+}_{m}(p') = \emptyset_m$
then $factors^{+}_{m}(p') \subset factors^{+}_{m}(p)$ (by set theory, $\because factors^{+}_{m}(p) \neq \emptyset_m$).

If $factors^{+}_{m}(p') \neq \emptyset_m$
then $factors^{+}_{m}(r'|s) \neq \emptyset_m$ ($\because p' = r'|s$)
$\implies \exists f \in factors^{+}_{m}(r'|s)$ (by set theory)
$\implies f \in factors_m(r'|s)(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^{+})$ (by definition of $factors^{+}_{m}(r'|s)$)
$\implies f \in \{r'\}_m \uplus \{s\}_m \uplus factors_m(r') \uplus factors_m(s)(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^{+})$
(by definition of $factors_m(r'|s)$)
$\implies f \in \{r'\}_m(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^{+}) \ \vee \ f \in \{s\}_m(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^{+}) \ \vee$
$f \in factors_m(r')(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^{+}) \ \vee$
$f \in factors_m(s)(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^{+})$
(by set theory).
Now $factors_m(r|s) = \{r\}_m \uplus \{s\}_m \uplus factors_m(r) \uplus factors_m(s)$
(by definition of $factors_m(r|s)$).
And the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference $n + 1$
(by the hypothesis of $Prop(n + 1)$)
$\implies$ the transition $r|s \xrightarrow{\overline{\tau}_{r_X}} r'|s$ has depth of inference $n + 1$ ($\because p = r|s \ \wedge \ p' = r'|s$)
$\implies$ the transition $r \xrightarrow{\overline{\tau}_{r_X}} r'$ has depth of inference m_r, with $m_r \in [1, n]$
($\because$ the transition $r|s \xrightarrow{\overline{\tau}_{r_X}} r'|s$ is inferred from the transition $r \xrightarrow{\overline{\tau}_{r_X}} r'$ using the $L - Par$ rule)
$\implies Prop(m_r)$ holds (by the inductive hypothesis)
$\implies r' = 0 \ \vee \ factors^{+}_{m}(r') \subset factors^{+}_{m}(r)$
(by modus ponens, $\because r, r' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_r \ \wedge \ r \xrightarrow{\overline{\tau}_{r_X}} r'$ with depth of inference m_r).

If $f \in \{r'\}_m(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^{+})$
then $f = r' \ \wedge \ factors_m(r') = \emptyset_m \ \wedge \ r' \in \mathcal{P}^{+}$ (by set theory).
Now $r' = 0 \ \vee \ r' \neq 0$ (by definition of $\vee$).
If $r' = 0$
then $r' \in \mathcal{P}^0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$)
$\implies r' \notin \mathcal{P}^{+}$ (by Theorem 4.3.2; which is a contradiction).
$\therefore r' \neq 0$
$\implies factors_m(r') \neq \emptyset_m$

(by Lemma 4.3.17, $\because r, r' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_r \ \wedge \ r \xrightarrow{\overline{\tau}_{r_X}} r'$; which is a contradiction).
$\therefore f \notin \{r'\}_m$.

If $f \in \{s\}_m (factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$
then $f \in factors_m(r|s) (factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$
($\because \{s\}_m \subseteq factors_m(r|s)$ (by definition of $factors_m(r|s)$), and by set theory)
$\implies f \in factors_m^+(r|s)$ (by definition of $factors_m^+(r|s)$).

If $f \in factors_m(r') (factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$
then $factors_m(r') \neq \emptyset_m$ (by set theory) $\wedge$
$f \in factors_m^+(r')$ (by definition of $factors_m^+(r')$).
Now $r' = 0 \ \vee \ r' \neq 0$ (by definition of $\vee$).
If $r' = 0$
then $factors_m(r') = \emptyset_m$ (by definition of $factors_m(0)$; which is a contradiction).
$\therefore r' \neq 0$
$\implies factors_m^+(r') \subset factors_m^+(r)$ ($\because r' = 0 \ \vee \ factors_m^+(r') \subset factors_m^+(r)$) $\wedge$
$factors_m^+(r) \subseteq factors_m(r)$ (by definition of $factors_m^+(r)$) $\wedge$
$factors_m(r) \subseteq factors_m(r|s)$ (by definition of $factors_m(r|s)$)
$\implies factors_m^+(r') \subset factors_m(r|s)$ (by set theory)
$\implies f \in factors_m(r|s)$ ($\because f \in factors_m^+(r')$, and by set theory)
$\implies f \in factors_m^+(r|s)$
($\because factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+$, and by definition of $factors_m^+(r|s)$).

If $f \in factors_m(s) (factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$
then $f \in factors_m(r|s) (factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$
($\because factors_m(s) \subseteq factors_m(r|s)$ (by definition of $factors_m(r|s)$), and by set theory)
$\implies f \in factors_m^+(r|s)$ (by definition of $factors_m^+(r|s)$).

$\therefore \forall f \in factors_m^+(r'|s) (f \in factors_m^+(r|s))$ ($\because f \in factors_m^+(r'|s)$ is arbitrary)
$\implies factors_m^+(r'|s) \subseteq factors_m^+(r|s)$ (by definition of $\subseteq$)
$\implies factors_m^+(r'|s) \subset factors_m^+(r|s) \ \vee \ factors_m^+(r'|s) = factors_m^+(r|s)$
(by multiset theory).
We prove $factors_m^+(r'|s) \subset factors_m^+(r|s)$.

Let $F_{r'}^+ \triangleq \{g \in \{r'\}_m \uplus factors_m(r') \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
and let $F_r^+ \triangleq \{g \in \{r\}_m \uplus factors_m(r) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$.
If $factors_m^+(r'|s) = factors_m^+(r|s)$

then $\{g \in factors_m(r'|s) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m =$
$\{g \in factors_m(r|s) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
(by definitions of $factors^+_m(r'|s)$ and $factors^+_m(r|s)$)
$\Longrightarrow \{g \in \{r'\}_m \uplus \{s\}_m \uplus factors_m(r') \uplus factors_m(s) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m =$
$\{g \in \{r\}_m \uplus \{s\}_m \uplus factors_m(r) \uplus factors_m(s) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
(by definitions of $factors_m(r'|s)$ and $factors_m(r|s)$)
$\Longrightarrow \{g \in \{r'\}_m \uplus factors_m(r') \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m =$
$\{g \in \{r\}_m \uplus factors_m(r) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
(by multiset theory)
$\Longrightarrow F^+_{r'} = F^+_r$ (by definitions of $F^+_{r'}$ and F^+_r).
Now $r' = 0 \vee r' \neq 0$ (by definition of $\vee$).

If $r' = 0$
then $r' \in \mathcal{P}^0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$) $\wedge$
$factors_m(r') = \emptyset_m$ ($\because factors_m(0) = \emptyset_m$, by definition of $factors_m(0)$)
$\Longrightarrow r' \notin \mathcal{P}^+$ (by Theorem 4.3.2) $\wedge$ $factors_m(r') = \emptyset_m$
$\Longrightarrow \{g \in \{r'\}_m \uplus factors_m(r') \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m = \emptyset_m$
(by multiset theory)
$\Longrightarrow F^+_{r'} = \emptyset_m$ (by definition of $F^+_{r'}$).
Now $factors_m(r) = \emptyset_m \vee factors_m(r) \neq \emptyset_m$ (by set theory).
If $factors_m(r) = \emptyset_m$
then $factors_m(r) = \emptyset_m \wedge r \in \mathcal{P}^+$ ($\because r \in \mathcal{P}^+$)
$\Longrightarrow r \in F^+_r$ (by definition of F^+_r and by multiset theory)
$\Longrightarrow F^+_r \neq \emptyset_m$ (by set theory)
$\Longrightarrow F^+_{r'} \neq \emptyset_m$ ($\because F^+_{r'} = F^+_r$; which is a contradiction).
$\therefore factors_m(r) \neq \emptyset_m$
$\Longrightarrow \exists u, v \in \mathcal{P}(r = u|v)$ (by definition of $factors_m(r)$)
$\Longrightarrow u|v \in \mathcal{P}^+$ ($\because r \in \mathcal{P}^+$)
$\Longrightarrow factors^+_m(u|v) \neq \emptyset_m$ (by Lemma 4.3.15, $\because u, v \in \mathcal{P}$)
$\Longrightarrow factors^+_m(r) \neq \emptyset_m$ ($\because r = u|v$)
$\Longrightarrow \exists g_1 \in factors^+_m(r)$ (by set theory)
$\Longrightarrow g_1 \in factors_m(r)(factors_m(g_1) = \emptyset_m \wedge g_1 \in \mathcal{P}^+)$ (by definition of $factors^+_m(r)$)
$\Longrightarrow g_1 \in F^+_r$ (by definition of F^+_r and by multiset theory)
$\Longrightarrow F^+_r \neq \emptyset_m$ (by set theory)
$\Longrightarrow F^+_{r'} \neq \emptyset_m$ ($\because F^+_{r'} = F^+_r$; which is a contradiction).

$\therefore r' \neq 0$

$\Longrightarrow factors_m(r') \neq \emptyset_m$ (by Lemma 4.3.17, $\because r, r' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_r \ \wedge \ r \xrightarrow{\overline{\tau}_{r_X}} r'$) $\wedge$
$factors^+_m(r') \subset factors^+_m(r)$ ($\because r' = 0 \ \vee \ factors^+_m(r') \subset factors^+_m(r)$)
$\Longrightarrow r' \notin F^+_{r'}$ (by definition of $F^+_{r'}$ and by set theory) $\wedge$ $factors^+_m(r') \subset factors^+_m(r)$
$\Longrightarrow F^+_{r'} = \{g \in factors_m(r') \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
(by definition of $F^+_{r'}$ and by multiset theory) $\wedge$
$factors^+_m(r') \subset factors^+_m(r)$
$\Longrightarrow F^+_{r'} = factors^+_m(r')$ (by definition of $factors^+_m(r')$) $\wedge$ $factors^+_m(r') \subset factors^+_m(r)$
$\Longrightarrow F^+_{r'} \subset factors^+_m(r)$ (by multiset theory)
$\Longrightarrow F^+_{r'} \subset factors^+_m(r)$ $\wedge$
$factors^+_m(r) \subseteq F^+_r$ (by definitions of $factors^+_m(r)$ and F^+_r, and by multiset theory)
$\Longrightarrow F^+_{r'} \subset F^+_r$ (by multiset theory)
$\Longrightarrow F^+_{r'} \neq F^+_r$ (by multiset theory; which is a contradiction).
$\therefore factors^+_m(r'|s) \neq factors^+_m(r|s)$
$\Longrightarrow factors^+_m(r'|s) \subset factors^+_m(r|s)$
($\because factors^+_m(r'|s) \subset factors^+_m(r|s) \ \vee \ factors^+_m(r'|s) = factors^+_m(r|s)$)
$\Longrightarrow factors^+_m(p') \subset factors^+_m(p)$ ($\because p' = r'|s \ \wedge \ p = r|s$).
$\therefore p' = 0 \ \vee \ factors^+_m(p') \subset factors^+_m(p)$ (by definition of $\vee$).

If the $R - Par$ rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
then $\exists r, s, s' \in \mathcal{P}$ ($p = r|s \ \wedge \ p' = r|s' \ \wedge \ r|s \xrightarrow{\overline{\tau}_{r_X}} r|s'$) (by the $R - Par$ rule)
$\Longrightarrow s \xrightarrow{\overline{\tau}_{r_X}} s'$ (by the hypothesis of $R - Par$).
Now $r|s' \neq 0$ (by the syntax of $r|s'$ and 0)
$\Longrightarrow p' \neq 0$ ($\because p' = r|s'$).
Therefore, we prove $factors^+_m(p') \subset factors^+_m(p)$.

$\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_s$ ($\because s \xrightarrow{\overline{\tau}_{r_X}} s'$, and by definition of $\overline{\mathcal{R}}_s$)
$\Longrightarrow \overline{\mathcal{R}}_s \neq \emptyset$ (by set theory)
$\Longrightarrow s \in \mathcal{P}^+$ (by Lemma 4.3.6, $\because s \in \mathcal{P}$)
$\Longrightarrow r|s \in \mathcal{P}^+$ (by production rule 3 of $\mathcal{P}^+$, $\because r, s \in \mathcal{P}$)
$\Longrightarrow factors^+_m(r|s) \neq \emptyset_m$ (by Lemma 4.3.15, $\because r, s \in \mathcal{P}$)
$\Longrightarrow factors^+_m(p) \neq \emptyset_m$ ($\because p = r|s$).
Now $factors^+_m(p') = \emptyset_m \ \vee \ factors^+_m(p') \neq \emptyset_m$ (by set theory).

If $factors^+_m(p') = \emptyset_m$
then $factors^+_m(p') \subset factors^+_m(p)$ (by set theory, $\because factors^+_m(p) \neq \emptyset_m$).

If $factors^+_m(p') \neq \emptyset_m$

then $factors_m^+(r|s') \neq \emptyset_m$ ($\because p' = r|s'$)
$\implies \exists f \in factors_m^+(r|s')$ (by set theory)
$\implies f \in factors_m(r|s')(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$ (by definition of $factors_m^+(r|s')$)
$\implies f \in \{r\}_m \uplus \{s'\}_m \uplus factors_m(r) \uplus factors_m(s')(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$
(by definition of $factors_m(r|s')$)
$\implies f \in \{r\}_m(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+) \ \vee \ f \in \{s'\}_m(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+) \ \vee$
$f \in factors_m(r)(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+) \ \vee$
$f \in factors_m(s')(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$
(by set theory).
Now $factors_m(r|s) = \{r\}_m \uplus \{s\}_m \uplus factors_m(r) \uplus factors_m(s)$
(by definition of $factors_m(r|s)$).
And the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference $n+1$
(by the hypothesis of $Prop(n+1)$)
$\implies$ the transition $r|s \xrightarrow{\overline{\tau}_{r_X}} r|s'$ has depth of inference $n+1$ ($\because p = r|s \ \wedge \ p' = r|s'$)
$\implies$ the transition $s \xrightarrow{\overline{\tau}_{r_X}} s'$ has depth of inference m_s, with $m_s \in [1, n]$
($\because$ the transition $r|s \xrightarrow{\overline{\tau}_{r_X}} r|s'$ is inferred from the transition $s \xrightarrow{\overline{\tau}_{r_X}} s'$ using the $R - Par$ rule)
$\implies Prop(m_s)$ holds (by the inductive hypothesis)
$\implies s' = 0 \ \vee \ factors_m^+(s') \subset factors_m^+(s)$
(by modus ponens, $\because s, s' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_s \ \wedge \ s \xrightarrow{\overline{\tau}_{r_X}} s'$ with depth of inference m_s).

If $f \in \{r\}_m(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$
then $f \in factors_m(r|s)(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$
($\because \{r\}_m \subseteq factors_m(r|s)$ (by definition of $factors_m(r|s)$), and by set theory)
$\implies f \in factors_m^+(r|s)$ (by definition of $factors_m^+(r|s)$).

If $f \in \{s'\}_m(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$
then $f = s' \ \wedge \ factors_m(s') = \emptyset_m \ \wedge \ s' \in \mathcal{P}^+$ (by set theory).
Now $s' = 0 \ \vee \ s' \neq 0$ (by definition of $\vee$).
If $s' = 0$
then $s' \in \mathcal{P}^0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$)
$\implies s' \notin \mathcal{P}^+$ (by Theorem 4.3.2; which is a contradiction).
$\therefore s' \neq 0$
$\implies factors_m(s') \neq \emptyset_m$
(by Lemma 4.3.17, $\because s, s' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_s \ \wedge \ s \xrightarrow{\overline{\tau}_{r_X}} s'$; which is a contradiction).
$\therefore f \notin \{s'\}_m$.

If $f \in factors_m(r)(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$
then $f \in factors_m(r|s)(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$
($\because$ $factors_m(r) \subseteq factors_m(r|s)$ (by definition of $factors_m(r|s)$), and by set theory)
$\implies f \in factors_m^+(r|s)$ (by definition of $factors_m^+(r|s)$).

If $f \in factors_m(s')(factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+)$
then $factors_m(s') \neq \emptyset_m$ (by set theory) $\wedge$
$f \in factors_m^+(s')$ (by definition of $factors_m^+(s')$).
Now $s' = 0 \vee s' \neq 0$ (by definition of $\vee$).
If $s' = 0$
then $factors_m(s') = \emptyset_m$ (by definition of $factors_m(0)$; which is a contradiction).
$\therefore s' \neq 0$
$\implies factors_m^+(s') \subset factors_m^+(s)$ ($\because$ $s' = 0 \vee factors_m^+(s') \subset factors_m^+(s)$) $\wedge$
$factors_m^+(s) \subseteq factors_m(s)$ (by definition of $factors_m^+(s)$) $\wedge$
$factors_m(s) \subseteq factors_m(r|s)$ (by definition of $factors_m(r|s)$)
$\implies factors_m^+(s') \subset factors_m(r|s)$ (by set theory)
$\implies f \in factors_m(r|s)$ ($\because$ $f \in factors_m^+(s')$, and by set theory)
$\implies f \in factors_m^+(r|s)$
($\because$ $factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+$, and by definition of $factors_m^+(r|s)$).

$\therefore \forall f \in factors_m^+(r|s')(f \in factors_m^+(r|s))$ ($\because$ $f \in factors_m^+(r|s')$ is arbitrary)
$\implies factors_m^+(r|s') \subseteq factors_m^+(r|s)$ (by definition of $\subseteq$)
$\implies factors_m^+(r|s') \subset factors_m^+(r|s) \vee$
$factors_m^+(r|s') = factors_m^+(r|s)$ (by multiset theory).
We prove $factors_m^+(r|s') \subset factors_m^+(r|s)$.

Let $F_{s'}^+ \triangleq \{g \in \{s'\}_m \uplus factors_m(s') \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
and let $F_s^+ \triangleq \{g \in \{s\}_m \uplus factors_m(s) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$.
If $factors_m^+(r|s') = factors_m^+(r|s)$
then $\{g \in factors_m(r|s') \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m =$
$\{g \in factors_m(r|s) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
(by definitions of $factors_m^+(r|s')$ and $factors_m^+(r|s)$)
$\implies \{g \in \{r\}_m \uplus \{s'\}_m \uplus factors_m(r) \uplus factors_m(s') \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m =$
$\{g \in \{r\}_m \uplus \{s\}_m \uplus factors_m(r) \uplus factors_m(s) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
(by definitions of $factors_m(r|s')$ and $factors_m(r|s)$)
$\implies \{g \in \{s'\}_m \uplus factors_m(s') \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m =$
$\{g \in \{s\}_m \uplus factors_m(s) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$

(by multiset theory)
$\Longrightarrow F^+_{s'} = F^+_s$ (by definitions of $F^+_{s'}$ and F^+_s).
Now $s' = 0 \vee s' \neq 0$ (by definition of $\vee$).

If $s' = 0$
then $s' \in \mathcal{P}^0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$) $\wedge$
$factors_m(s') = \emptyset_m$ ($\because factors_m(0) = \emptyset_m$, by definition of $factors_m(0)$)
$\Longrightarrow s' \notin \mathcal{P}^+$ (by Theorem 4.3.2) $\wedge$ $factors_m(s') = \emptyset_m$
$\Longrightarrow \{g \in \{s'\}_m \uplus factors_m(s') \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m = \emptyset_m$
(by multiset theory)
$\Longrightarrow F^+_{s'} = \emptyset_m$ (by definition of $F^+_{s'}$).
Now $factors_m(s) = \emptyset_m \vee factors_m(s) \neq \emptyset_m$ (by set theory).
If $factors_m(s) = \emptyset_m$
then $factors_m(s) = \emptyset_m \wedge s \in \mathcal{P}^+$ ($\because s \in \mathcal{P}^+$)
$\Longrightarrow s \in F^+_s$ (by definition of F^+_s and by multiset theory)
$\Longrightarrow F^+_s \neq \emptyset_m$ (by set theory)
$\Longrightarrow F^+_{s'} \neq \emptyset_m$ ($\because F^+_{s'} = F^+_s$; which is a contradiction).
$\therefore factors_m(s) \neq \emptyset_m$
$\Longrightarrow \exists u, v \in \mathcal{P}(s = u|v)$ (by definition of $factors_m(s)$)
$\Longrightarrow u|v \in \mathcal{P}^+$ ($\because s \in \mathcal{P}^+$)
$\Longrightarrow factors^+_m(u|v) \neq \emptyset_m$ (by Lemma 4.3.15, $\because u, v \in \mathcal{P}$)
$\Longrightarrow factors^+_m(s) \neq \emptyset_m$ ($\because s = u|v$)
$\Longrightarrow \exists g_1 \in factors^+_m(s)$ (by set theory)
$\Longrightarrow g_1 \in factors_m(s)(factors_m(g_1) = \emptyset_m \wedge g_1 \in \mathcal{P}^+)$ (by definition of $factors^+_m(s)$)
$\Longrightarrow g_1 \in F^+_s$ (by definition of F^+_s and by multiset theory)
$\Longrightarrow F^+_s \neq \emptyset_m$ (by set theory)
$\Longrightarrow F^+_{s'} \neq \emptyset_m$ ($\because F^+_{s'} = F^+_s$; which is a contradiction).

$\therefore s' \neq 0$
$\Longrightarrow factors_m(s') \neq \emptyset_m$ (by Lemma 4.3.17, $\because s, s' \in \mathcal{P} \wedge \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_s \wedge s \xrightarrow{\overline{\tau}_{r_X}} s'$) $\wedge$
$factors^+_m(s') \subset factors^+_m(s)$ ($\because s' = 0 \vee factors^+_m(s') \subset factors^+_m(s)$)
$\Longrightarrow s' \notin F^+_{s'}$ (by definition of $F^+_{s'}$ and by set theory) $\wedge$ $factors^+_m(s') \subset factors^+_m(s)$
$\Longrightarrow F^+_{s'} = \{g \in factors_m(s') \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
(by definition of $F^+_{s'}$ and by multiset theory) $\wedge$
$factors^+_m(s') \subset factors^+_m(s)$
$\Longrightarrow F^+_{s'} = factors^+_m(s')$ (by definition of $factors^+_m(s')$) $\wedge$ $factors^+_m(s') \subset factors^+_m(s)$
$\Longrightarrow F^+_{s'} \subset factors^+_m(s)$ (by multiset theory)

$\implies F^+_{s'} \subset factors^+_m(s) \ \wedge$
$factors^+_m(s) \subseteq F^+_s$ (by definitions of $factors^+_m(s)$ and F^+_s, and by multiset theory)
$\implies F^+_{s'} \subset F^+_s$ (by multiset theory)
$\implies F^+_{s'} \neq F^+_s$ (by multiset theory; which is a contradiction).
$\therefore factors^+_m(r|s') \neq factors^+_m(r|s)$
$\implies factors^+_m(r|s') \subset factors^+_m(r|s)$
$(\because factors^+_m(r|s') \subset factors^+_m(r|s) \ \vee \ factors^+_m(r|s') = factors^+_m(r|s))$
$\implies factors^+_m(p') \subset factors^+_m(p)$ $(\because p' = r|s' \ \wedge \ p = r|s)$.
$\therefore p' = 0 \ \vee \ factors^+_m(p') \subset factors^+_m(p)$ (by definition of $\vee$).

If the *CompDelet* rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
then $\exists\, \overline{\tau}_{r_{X_1}}, \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}\ \exists u \in \mathcal{P}\ (X \sim_{of} X_1|X_2 \ \wedge \ p \xrightarrow{\overline{\tau}_{r_{X_1}}} u \ \wedge \ u \xrightarrow{\overline{\tau}_{r_{X_2}}} p')$
(by the hypothesis of *CompDelet*)
$\implies \overline{\tau}_{r_{X_1}} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\ \wedge \ \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}_u$ (by definition of $\overline{\mathcal{R}}_u$)
$\implies \overline{\mathcal{R}}_u \neq \emptyset$ (by set theory).
Now the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference $n+1$
(by the hypothesis of $Prop(n+1)$)
$\implies$ the transition $p \xrightarrow{\overline{\tau}_{r_{X_1}}} u$ has depth of inference m_p, with $m_p \in [1, n]$ $\ \wedge$
the transition $u \xrightarrow{\overline{\tau}_{r_{X_2}}} p'$ has depth of inference m_u, with $m_u \in [1, n]$
($\because$ the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ is inferred from the transitions $p \xrightarrow{\overline{\tau}_{r_{X_1}}} u$ and $u \xrightarrow{\overline{\tau}_{r_{X_2}}} p'$ using the *CompDelet* rule)
$\implies Prop(m_p)$ and $Prop(m_u)$ hold (by the inductive hypothesis)
$\implies u = 0 \ \vee \ factors^+_m(u) \subset factors^+_m(p)$
(by modus ponens, $\because p, u \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_{X_1}} \in \overline{\mathcal{R}}_p \ \wedge \ p \xrightarrow{\overline{\tau}_{r_{X_1}}} u$ with depth of inference m_p) $\ \wedge$
$p' = 0 \ \vee \ factors^+_m(p') \subset factors^+_m(u)$
(by modus ponens, $\because u, p' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}_u \ \wedge \ u \xrightarrow{\overline{\tau}_{r_{X_2}}} p'$ with depth of inference m_u).
Now $u = 0 \ \vee \ u \neq 0$ (by definition of $\vee$).
If $u = 0$
then $u \in \mathcal{P}^0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$)
$\implies \overline{\mathcal{R}}_u = \emptyset$ (by Lemma 4.3.8; which is a contradiction).
$\therefore u \neq 0$
$\implies factors^+_m(u) \subset factors^+_m(p)$ $(\because u = 0 \ \vee \ factors^+_m(u) \subset factors^+_m(p))$.
Now $p' = 0 \ \vee \ p' \neq 0$ (by definition of $\vee$).
If $p' = 0$
then $p' = 0 \ \vee \ factors^+_m(p') \subset factors^+_m(p)$ (by definition of $\vee$).

If $p' \neq 0$
then $factors_m^+(p') \subset factors_m^+(u)$ ($\because p' = 0 \ \vee \ factors_m^+(p') \subset factors_m^+(u)$)
$\Longrightarrow factors_m^+(p') \subset factors_m^+(u) \ \wedge$
$factors_m^+(u) \subset factors_m^+(p)$ ($\because factors_m^+(u) \subset factors_m^+(p)$)
$\Longrightarrow factors_m^+(p') \subset factors_m^+(p)$ ($\because \subset$ is transitive, by multiset theory)
$\Longrightarrow p' = 0 \ \vee \ factors_m^+(p') \subset factors_m^+(p)$ (by definition of $\vee$).

$\therefore \forall p, p' \in \mathcal{P} \ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p \ (p \xrightarrow{\overline{\tau}_{r_X}} p' \Longrightarrow p' = 0 \ \vee \ factors_m^+(p') \subset factors_m^+(p))$
for $p \xrightarrow{\overline{\tau}_{r_X}} p'$ determined by applications of LTS rules with depth of inference $n+1$
($\because p, p' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$ with transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ and depth of inference $n+1$ are arbitrary)
$\Longrightarrow Prop(n+1)$ holds (by definition of $Prop(n+1)$).
$\therefore \forall n \in \mathbb{N}^+ \ (\forall m \in [1, n] Prop(m) \Longrightarrow Prop(n+1))$ holds ($\because n \in \mathbb{N}^+$ is arbitrary). Q.E.D.

$\therefore \forall n \in \mathbb{N}^+ \ Prop(n)$ holds (by complete induction).
$\therefore \ \forall p, p' \in \mathcal{P} \ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p \ (p \xrightarrow{\overline{\tau}_{r_X}} p' \Longrightarrow p' = 0 \ \vee \ factors_m^+(p') \subset factors_m^+(p))$
($\because$ every transition of every $p \in \mathcal{P}$ is a result of one or more applications of the LTS semantic rules with finite depth of inference). Q.E.D.

A.14 Lemma 4.3.19

$$\forall p \in \mathcal{P} \ \forall z \in \mathcal{P}^0 \ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{p|z} \ \forall (p|z)' \in \mathcal{P}^+$$
$$(p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)' \Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \wedge \exists p' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (p|z)' = p'|z))$$

Proof: uses complete induction on the number of positive singleton factors of $p|z$ deleted by $\overline{\tau}_{r_Y}$.

For $p \in \mathcal{P}$, $z \in \mathcal{P}^0$, $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{p|z}$ and $(p|z)' \in \mathcal{P}^+$, $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$
(by the hypothesis of the lemma)
$\Longrightarrow$ only the *Delet*, $L-Par$, $R-Par$ or *CompDelet* rules determine the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$
(by definitions of the LTS rules).

If the *Delet* rule defines the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$
then $(p|z)' = 0$ (by the *Delet* rule)

$\implies (p|z)' \in \mathcal{P}^0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$)
$\implies (p|z)' \notin \mathcal{P}^+$ (by Theorem 4.3.2 and set theory; which is a contradiction).
$\therefore$ The *Delet* rule does not define the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$.

And $z \in \mathcal{P}^0$ (by the hypothesis of the lemma)
$\implies \overline{\mathcal{R}}_z = \emptyset$ (by Lemma 4.3.8)
$\implies \overline{\tau}_{r_Y} \notin \overline{\mathcal{R}}_z$ (by set theory)
$\implies \neg(\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}\ \exists z' \in \mathcal{P}\ (z \xrightarrow{\overline{\tau}_{r_Y}} z'))$ (by definition of $\overline{\mathcal{R}}_z$)
$\implies$ the $R-Par$ rule does not define the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$
($\because$ the hypothesis of $R-Par$ does not hold)
$\implies$ neither the *Delet* rule nor the $R-Par$ rule defines the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$
($\because$ the *Delet* rule does not define the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$)
$\implies$ only the $L-Par$ or *CompDelet* rules determine the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$
($\because$ only the *Delet*, $L-Par$, $R-Par$ or *CompDelet* rules determine the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$).

Now $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{p|z}\ \wedge\ p \in \mathcal{P}\ \wedge\ z \in \mathcal{P}^0$ (by the hypothesis of the lemma)
$\implies \overline{\mathcal{R}}_{p|z} \neq \emptyset$ (by set theory) $\wedge$
$p|z \in \mathcal{P}$
(by Theorem 4.3.2, production rule 3 of $\mathcal{P}^+$, production rule 3 of $\mathcal{P}^0$ and set theory)
$\implies p|z \in \mathcal{P}^+$ (by Lemma 4.3.6) $\wedge$
$z \in \mathcal{P}$ ($\because z \in \mathcal{P}^0$, and by Theorem 4.3.2 and set theory)
$\implies factors^+_m(p|z) \neq \emptyset_m$ (by Lemma 4.3.15, $\because p \in \mathcal{P}$).
And $(p|z)' \in \mathcal{P}$ ($\because (p|z)' \in \mathcal{P}^+$, and by Theorem 4.3.2 and set theory)
$\implies (p|z)' = 0\ \vee\ factors^+_m((p|z)') \subset factors^+_m(p|z)$
(by Lemma 4.3.18, $\because p|z \in \mathcal{P}\ \wedge\ \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{p|z}\ \wedge\ p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$)
$\implies factors^+_m((p|z)') = \emptyset_m$ ($\because factors^+_m(0) = \emptyset_m$, by definition of $factors^+_m(0)$) $\vee$
$factors^+_m((p|z)') \subset factors^+_m(p|z)$
$\implies factors^+_m((p|z)') \subset factors^+_m(p|z)$ (by set theory, and because $factors^+_m(p|z) \neq \emptyset_m$)
$\implies |factors^+_m((p|z)')| < |factors^+_m(p|z)|$ (by multiset theory)
$\implies 1 \leq |factors^+_m(p|z)| - |factors^+_m((p|z)')|$
($\because |factors^+_m(p|z)|, |factors^+_m((p|z)')| \in \mathbb{N}$, and by algebra of inequalities)
$\implies 1 \leq |factors^+_m(p|z)| - |factors^+_m((p|z)')|\ \wedge\ p|z \in \mathcal{P}^+\ \wedge$
only the $L-Par$ or *CompDelet* rules determine the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$
($\because\ p|z \in \mathcal{P}^+\ \wedge$ only the $L-Par$ or *CompDelet* rules determine the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$)

$\implies \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{p|z}\ \forall (p|z)' \in \mathcal{P}^+$ with $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$
$(1 \leq |factors^+_m(p|z)| - |factors^+_m((p|z)')|\ \wedge\ p|z \in \mathcal{P}^+\ \wedge$
only the $L - Par$ or $CompDelet$ rules determine the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)')$
($\because p \in \mathcal{P}$ and $z \in \mathcal{P}^0$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{p|z}$ and $(p|z)' \in \mathcal{P}^+$ with $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$ are arbitrary).

Therefore, we use complete induction on $|factors^+_m(p|z)| - |factors^+_m((p|z)')|$
and use only the $L - Par$ or $CompDelet$ rules to determine $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$.

For $n \in \mathbb{N}^+$, let $Prop(n)$ be the proposition:
$\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{p|z}\ \forall (p|z)' \in \mathcal{P}^+$
$(p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'\ \wedge\ |factors^+_m(p|z)| - |factors^+_m((p|z)')| = n \implies$
$\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \wedge \exists p' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_Y}} p'\ \wedge (p|z)' = p'|z))$.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(1)$
2. $\vdash \forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \implies Prop(n+1))$

Base Case: Proof of Prop(1)

For $p \in \mathcal{P}$ and $z \in \mathcal{P}^0$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{p|z}$ and $(p|z)' \in \mathcal{P}^+$,
$p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'\ \wedge\ |factors^+_m(p|z)| - |factors^+_m((p|z)')| = 1$ (by the hypothesis of $Prop(1)$).

If the $L - Par$ rule defines the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$
then $\exists\, p' \in \mathcal{P}\, (p \xrightarrow{\overline{\tau}_{r_Y}} p')$ (by the hypothesis of $L - Par$) $\wedge$
$(p|z)' = p'|z$ (by the $L - Par$ rule)
$\implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\wedge\ p'|z \in \mathcal{P}^+$ ($\because (p|z)' \in \mathcal{P}^+$)
$\implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p\ \wedge\ p' \in \mathcal{P}^+$ (by production rule 3 of $\mathcal{P}^+$ and Theorem 4.3.2, $\because z \in \mathcal{P}^0$)
$\implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \wedge \exists p' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (p|z)' = p'|z)$ ($\because p \xrightarrow{\overline{\tau}_{r_Y}} p'\ \wedge (p|z)' = p'|z$).

If the $CompDelet$ rule defines the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$
then $\exists\, \overline{\tau}_{r_{Y_1}}, \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}\ \exists (p|z)'' \in \mathcal{P}\ (Y \sim_{of} Y_1|Y_2\ \wedge\ p|z \xrightarrow{\overline{\tau}_{r_{Y_1}}} (p|z)''\ \wedge\ (p|z)'' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (p|z)')$
(by the hypothesis of $CompDelet$)
$\implies \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{p|z}$ (by definition of $\overline{\mathcal{R}}_{p|z}$) $\wedge\ \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(p|z)''}$ (by definition of $\overline{\mathcal{R}}_{(p|z)''}$)

$\implies \overline{\mathcal{R}}_{(p|z)''} \neq \emptyset$ (by set theory)
$\implies (p|z)'' \in \mathcal{P}$ (by definition of $p|z \xrightarrow{\overline{\tau}_{r_{Y_1}}} (p|z)'')$ $\wedge$ $\overline{\mathcal{R}}_{(p|z)''} \neq \emptyset$
$\implies (p|z)'' \in \mathcal{P}^+$ (by Lemma 4.3.6)
$\implies (p|z)'' \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because (p|z)'' \in \mathcal{P}$)
$\implies (p|z)'' \neq 0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$).
Now $p|z \in \mathcal{P}$
(by Theorem 4.3.2, production rule 3 of $\mathcal{P}^+$, production rule 3 of $\mathcal{P}^0$ and set theory, $\because p \in \mathcal{P} \wedge z \in \mathcal{P}^0$)
$\implies (p|z)'' = 0 \vee factors_m^+((p|z)'') \subset factors_m^+(p|z)$
(by Lemma 4.3.18, $\because (p|z)'' \in \mathcal{P} \wedge \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{p|z} \wedge p|z \xrightarrow{\overline{\tau}_{r_{Y_1}}} (p|z)''$)
$\implies factors_m^+((p|z)'') \subset factors_m^+(p|z)$ ($\because (p|z)'' \neq 0$)
$\implies |factors_m^+((p|z)'')| < |factors_m^+(p|z)|$ (by multiset theory).

And $(p|z)' \in \mathcal{P}^+$ (by the hypothesis of *Prop*(1))
$\implies (p|z)' \in \mathcal{P}$ (by Theorem 4.3.2 and set theory)
$\implies (p|z)' \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because (p|z)' \in \mathcal{P}^+$)
$\implies (p|z)' \neq 0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$).
Now $(p|z)' = 0 \vee factors_m^+((p|z)') \subset factors_m^+((p|z)'')$
(by Lemma 4.3.18, $\because (p|z)'' \in \mathcal{P} \wedge (p|z)' \in \mathcal{P} \wedge \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(p|z)''} \wedge (p|z)'' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (p|z)'$)
$\implies factors_m^+((p|z)') \subset factors_m^+((p|z)'')$ ($\because (p|z)' \neq 0$)
$\implies |factors_m^+((p|z)')| < |factors_m^+((p|z)'')|$ (by multiset theory)
$\implies |factors_m^+((p|z)')| < |factors_m^+((p|z)'')| < |factors_m^+(p|z)|$
(by algebra of inequalities, $\because |factors_m^+((p|z)'')| < |factors_m^+(p|z)|$)
$\implies 2 \leq |factors_m^+(p|z)| - |factors_m^+((p|z)')|$
(by algebra of inequalities and
because $|factors_m^+(p|z)|, |factors_m^+((p|z)'')|, |factors_m^+((p|z)')| \in \mathbb{N}$).
But $|factors_m^+(p|z)| - |factors_m^+((p|z)')| = 1$
(by the hypothesis of *Prop*(1); which is a contradiction).
$\therefore$ The *CompDelet* rule does not define the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$.

$\therefore \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{p|z}\ \forall (p|z)' \in \mathcal{P}^+$
$(p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)' \wedge |factors_m^+(p|z)| - |factors_m^+((p|z)')| = 1 \implies$
$\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \wedge \exists p' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (p|z)' = p'|z))$
($\because p \in \mathcal{P}$ and $z \in \mathcal{P}^0$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{p|z}$ and $(p|z)' \in \mathcal{P}^+$ with
$p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$ and $|factors_m^+(p|z)| - |factors_m^+((p|z)')| = 1$ are arbitrary)
$\implies$ *Prop*(1) holds (by definition of *Prop*(1)). Q.E.D.

Induction Step: Proof of $\forall \mathbf{n} \in \mathbb{N}^+ \ (\forall \mathbf{m} \in [\mathbf{1}, \mathbf{n}] \mathbf{Prop(m)} \implies \mathbf{Prop(n+1)})$

For $n \in \mathbb{N}^+$, assume $\forall m \in [1, n] Prop(m)$ holds (inductive hypothesis).
For $p \in \mathcal{P}$ and $z \in \mathcal{P}^0$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{p|z}$ and $(p|z)' \in \mathcal{P}^+$,
$p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)' \ \wedge \ |factors^+_m(p|z)| - |factors^+_m((p|z)')| = n+1$
(by the hypothesis of $Prop(n+1)$).

If the $L - Par$ rule defines the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$
then $\exists p' \in \mathcal{P} (p \xrightarrow{\overline{\tau}_{r_Y}} p')$ (by the hypothesis of $L - Par$) $\wedge$
$(p|z)' = p'|z$ (by the $L - Par$ rule)
$\implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\wedge \ p'|z \in \mathcal{P}^+$ ($\because (p|z)' \in \mathcal{P}^+$)
$\implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \ \wedge \ p' \in \mathcal{P}^+$ (by production rule 3 of $\mathcal{P}^+$ and Theorem 4.3.2, $\because z \in \mathcal{P}^0$)
$\implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \wedge \exists p' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (p|z)' = p'|z)$ ($\because p \xrightarrow{\overline{\tau}_{r_Y}} p' \ \wedge \ (p|z)' = p'|z$).

If the *CompDelet* rule defines the transition $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$
then $\exists \overline{\tau}_{r_{Y_1}}, \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}} \ \exists (p|z)'' \in \mathcal{P} \ (Y \sim_{of} Y_1|Y_2 \ \wedge \ p|z \xrightarrow{\overline{\tau}_{r_{Y_1}}} (p|z)'' \ \wedge \ (p|z)'' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (p|z)')$
(by the hypothesis of *CompDelet*)
$\implies \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{p|z}$ (by definition of $\overline{\mathcal{R}}_{p|z}$) $\ \wedge \ \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(p|z)''}$ (by definition of $\overline{\mathcal{R}}_{(p|z)''}$)
$\implies \overline{\mathcal{R}}_{(p|z)''} \neq \emptyset$ (by set theory)
$\implies (p|z)'' \in \mathcal{P}$ (by definition of $p|z \xrightarrow{\overline{\tau}_{r_{Y_1}}} (p|z)''$) $\ \wedge \ \overline{\mathcal{R}}_{(p|z)''} \neq \emptyset$
$\implies (p|z)'' \in \mathcal{P}^+$ (by Lemma 4.3.6)
$\implies (p|z)'' \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because (p|z)'' \in \mathcal{P}$)
$\implies (p|z)'' \neq 0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$).
Now $p|z \in \mathcal{P}$
(by Theorem 4.3.2, production rule 3 of $\mathcal{P}^+$, production rule 3 of $\mathcal{P}^0$ and set theory,
$\because p \in \mathcal{P} \ \wedge \ z \in \mathcal{P}^0$)
$\implies (p|z)'' = 0 \ \vee \ factors^+_m((p|z)'') \subset factors^+_m(p|z)$
(by Lemma 4.3.18, $\because (p|z)'' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{p|z} \ \wedge \ p|z \xrightarrow{\overline{\tau}_{r_{Y_1}}} (p|z)''$)
$\implies factors^+_m((p|z)'') \subset factors^+_m(p|z)$ ($\because (p|z)'' \neq 0$)
$\implies |factors^+_m((p|z)'')| < |factors^+_m(p|z)|$ (by multiset theory)
$\implies 1 \leq |factors^+_m(p|z)| - |factors^+_m((p|z)'')|$
(by algebra of inequalities and because $|factors^+_m(p|z)|, |factors^+_m((p|z)'')| \in \mathbb{N}$).

And $(p|z)' \in \mathcal{P}^+$ (by the hypothesis of $Prop(1)$)
$\implies (p|z)' \in \mathcal{P}$ (by Theorem 4.3.2 and set theory)
$\implies (p|z)' \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because (p|z)' \in \mathcal{P}^+$)

$\implies (p|z)' \neq 0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$).
Now $(p|z)' = 0 \ \vee \ factors_m^+((p|z)') \subset factors_m^+((p|z)'')$
(by Lemma 4.3.18, $\because (p|z)'' \in \mathcal{P} \ \wedge \ (p|z)' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(p|z)''} \ \wedge \ (p|z)'' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (p|z)'$)
$\implies factors_m^+((p|z)') \subset factors_m^+((p|z)'')$ ($\because (p|z)' \neq 0$)
$\implies |factors_m^+((p|z)')| < |factors_m^+((p|z)'')|$ (by multiset theory)
$\implies 1 \leq |factors_m^+((p|z)'')| - |factors_m^+((p|z)')|$
(by algebra of inequalities and because $|factors_m^+((p|z)'')|, |factors_m^+((p|z)')| \in \mathbb{N}$).

Let $m_1 \triangleq |factors_m^+(p|z)| - |factors_m^+((p|z)'')|$ and
$m_2 \triangleq |factors_m^+((p|z)'')| - |factors_m^+((p|z)')|$.
$m_1 \in \mathbb{N}^+$
(by definition of m_1, and
because $1 \leq |factors_m^+(p|z)| - |factors_m^+((p|z)'')| \ \wedge$
$|factors_m^+(p|z)|, |factors_m^+((p|z)'')| \in \mathbb{N}) \ \wedge$
$m_2 \in \mathbb{N}^+$
(by definition of m_2, and
because $1 \leq |factors_m^+((p|z)'')| - |factors_m^+((p|z)')| \ \wedge$
$|factors_m^+((p|z)'')|, |factors_m^+((p|z)')| \in \mathbb{N}$).
And $m_1 + m_2 =$
$(|factors_m^+(p|z)| - |factors_m^+((p|z)'')|) + (|factors_m^+((p|z)'')| - |factors_m^+((p|z)')|)$
(by definitions of m_1 and m_2)
$\implies m_1 + m_2 = |factors_m^+(p|z)| - |factors_m^+((p|z)')|$ (by arithmetic)
$\implies m_1 + m_2 = n + 1$ (by the hypothesis of $Prop(n+1)$)
$\implies m_1, m_2 \in [1, n]$ ($\because m_1, m_2 \in \mathbb{N}^+$, and by algebra of inequalities)
$\implies Prop(m_1)$ and $Prop(m_2)$ hold (by the inductive hypothesis)
$\implies \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_p \ \wedge \ \exists p'' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_{Y_1}}} p'' \ \wedge \ (p|z)'' = p''|z)$
(by modus ponens, $\because p \in \mathcal{P} \ \wedge \ z \in \mathcal{P}^0 \ \wedge \ \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{p|z} \ \wedge \ (p|z)'' \in \mathcal{P}^+ \ \wedge \ p|z \xrightarrow{\overline{\tau}_{r_{Y_1}}} (p|z)'' \ \wedge$
$|factors_m^+(p|z)| - |factors_m^+((p|z)'')| = m_1$)
$\implies p'' \in \mathcal{P}$ (by Theorem 4.3.2 and set theory) $\ \wedge \ (p|z)'' = p''|z \ \wedge$
$\overline{\mathcal{R}}_{p''|z} = \overline{\mathcal{R}}_{(p|z)''}$ (by definitions of $\overline{\mathcal{R}}_{p''|z}$ and $\overline{\mathcal{R}}_{(p|z)''}$) $\ \wedge$
$factors_m^+(p''|z) = factors_m^+((p|z)'')$
(by definitions of $factors_m^+(p''|z)$ and $factors_m^+((p|z)'')$)
$\implies p'' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{p''|z}$ ($\because \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(p|z)''}$) $\ \wedge \ p''|z \xrightarrow{\overline{\tau}_{r_{Y_2}}} (p|z)'$ ($\because (p|z)'' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (p|z)'$) $\ \wedge$
$|factors_m^+(p''|z)| - |factors_m^+((p|z)')| = m_2$
(by set theory and because $|factors_m^+((p|z)'')| - |factors_m^+((p|z)')| = m_2$)
$\implies \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{p''} \ \wedge \ \exists p' \in \mathcal{P}^+ (p'' \xrightarrow{\overline{\tau}_{r_{Y_2}}} p' \ \wedge \ (p|z)' = p'|z)$

(by modus ponens, $\because Prop(m_2)$ holds $\wedge\ z{\in}\mathcal{P}^0\ \wedge\ (p|z)'{\in}\mathcal{P}^+$)
$\Longrightarrow p \xrightarrow{\overline{\tau}_{r_Y}} p'$ (by the *CompDelet* rule, $\because Y \sim_{of} Y_1|Y_2\ \wedge\ p \xrightarrow{\overline{\tau}_{r_{Y_1}}} p''$) $\wedge$
$p'|z{\in}\mathcal{P}^+$ ($\because (p|z)'{\in}\mathcal{P}^+$)
$\Longrightarrow \overline{\tau}_{r_Y}{\in}\overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\wedge$
$p'{\in}\mathcal{P}^+$ (by production rule 3 of $\mathcal{P}^+$ and Theorem 4.3.2, $\because z{\in}\mathcal{P}^0$)
$\Longrightarrow \overline{\tau}_{r_Y}{\in}\overline{\mathcal{R}}_p \wedge \exists p'{\in}\mathcal{P}^+(p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (p|z)' = p'|z)$ ($\because p \xrightarrow{\overline{\tau}_{r_Y}} p'\ \wedge\ (p|z)' = p'|z$).

$\therefore \forall p{\in}\mathcal{P}\ \forall z{\in}\mathcal{P}^0\ \forall\overline{\tau}_{r_Y}{\in}\overline{\mathcal{R}}_{p|z}\ \forall(p|z)'{\in}\mathcal{P}^+$
$(p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'\ \wedge\ |factors^+_m(p|z)| - |factors^+_m((p|z)')| = n+1 \Longrightarrow$
$\overline{\tau}_{r_Y}{\in}\overline{\mathcal{R}}_p \wedge \exists p'{\in}\mathcal{P}^+(p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (p|z)' = p'|z))$
($\because p{\in}\mathcal{P}$ and $z{\in}\mathcal{P}^0$ and $\overline{\tau}_{r_Y}{\in}\overline{\mathcal{R}}_{p|z}$ and $(p|z)'{\in}\mathcal{P}^+$ with $p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)'$ and
$|factors^+_m(p|z)| - |factors^+_m((p|z)')| = n+1$ are arbitrary)
$\Longrightarrow Prop(n+1)$ holds (by definition of $Prop(n+1)$).
$\therefore \forall n{\in}\mathbb{N}^+$ $(\forall m{\in}[1,n]Prop(m) \Longrightarrow Prop(n+1))$ holds ($\because n{\in}\mathbb{N}^+$ is arbitrary). Q.E.D.

$\therefore \forall n{\in}\mathbb{N}^+\ Prop(n)$ holds (by complete induction).
$\therefore \forall p{\in}\mathcal{P}\ \forall z{\in}\mathcal{P}^0\ \forall\overline{\tau}_{r_Y}{\in}\overline{\mathcal{R}}_{p|z}\ \forall(p|z)'{\in}\mathcal{P}^+$
$(p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)' \Longrightarrow \overline{\tau}_{r_Y}{\in}\overline{\mathcal{R}}_p \wedge \exists p'{\in}\mathcal{P}^+(p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (p|z)' = p'|z))$
($\because \forall p{\in}\mathcal{P}\ \forall z{\in}\mathcal{P}^0\ \forall\overline{\tau}_{r_Y}{\in}\overline{\mathcal{R}}_{p|z}\ \forall(p|z)'{\in}\mathcal{P}^+$
$(p|z \xrightarrow{\overline{\tau}_{r_Y}} (p|z)' \Longrightarrow |factors^+_m(p|z)| - |factors^+_m((p|z)')|{\in}\mathbb{N}^+)$). Q.E.D.

A.15 Lemma 4.3.20

$$\forall p{\in}\mathcal{P}\ \forall z{\in}\mathcal{P}^0\ \forall\overline{\tau}_{r_Y}{\in}\overline{\mathcal{R}}_{z|p}\ \forall(z|p)'{\in}\mathcal{P}^+$$
$$(z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)' \Longrightarrow \overline{\tau}_{r_Y}{\in}\overline{\mathcal{R}}_p \wedge \exists p'{\in}\mathcal{P}^+(p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (z|p)' = z|p'))$$

Proof: uses complete induction on the number of positive singleton factors of $z|p$ deleted by $\overline{\tau}_{r_Y}$.

For $p{\in}\mathcal{P}$, $z{\in}\mathcal{P}^0$, $\overline{\tau}_{r_Y}{\in}\overline{\mathcal{R}}_{z|p}$ and $(z|p)'{\in}\mathcal{P}^+$, $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$
(by the hypothesis of the lemma)
$\Longrightarrow$ only the *Delet*, *L − Par*, *R − Par* or *CompDelet* rules determine the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$
(by definitions of the LTS rules).

If the *Delet* rule defines the transition $z|p \xrightarrow{\overline{\tau}_{r_\gamma}} (z|p)'$
then $(z|p)' = 0$ (by the *Delet* rule)
$\implies (z|p)' \in \mathcal{P}^0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$)
$\implies (z|p)' \notin \mathcal{P}^+$ (by Theorem 4.3.2 and set theory; which is a contradiction).
$\therefore$ The *Delet* rule does not define the transition $z|p \xrightarrow{\overline{\tau}_{r_\gamma}} (z|p)'$.

And $z \in \mathcal{P}^0$ (by the hypothesis of the lemma)
$\implies \overline{\mathcal{R}}_z = \emptyset$ (by Lemma 4.3.8)
$\implies \overline{\tau}_{r_\gamma} \notin \overline{\mathcal{R}}_z$ (by set theory)
$\implies \neg(\overline{\tau}_{r_\gamma} \in \overline{\mathcal{R}}\ \exists z' \in \mathcal{P}\,(z \xrightarrow{\overline{\tau}_{r_\gamma}} z'))$ (by definition of $\overline{\mathcal{R}}_z$)
$\implies$ the $L - Par$ rule does not define the transition $z|p \xrightarrow{\overline{\tau}_{r_\gamma}} (z|p)'$
($\because$ the hypothesis of $L - Par$ does not hold)
$\implies$ neither the *Delet* rule nor the $L - Par$ rule defines the transition $z|p \xrightarrow{\overline{\tau}_{r_\gamma}} (z|p)'$
($\because$ the *Delet* rule does not define the transition $z|p \xrightarrow{\overline{\tau}_{r_\gamma}} (z|p)'$)
$\implies$ only the $R - Par$ or *CompDelet* rules determine the transition $z|p \xrightarrow{\overline{\tau}_{r_\gamma}} (z|p)'$
($\because$ only the *Delet*, $L - Par$, $R - Par$ or *CompDelet* rules determine the transition $z|p \xrightarrow{\overline{\tau}_{r_\gamma}} (z|p)'$).

Now $\overline{\tau}_{r_\gamma} \in \overline{\mathcal{R}}_{z|p} \ \wedge\ p \in \mathcal{P} \ \wedge\ z \in \mathcal{P}^0$ (by the hypothesis of the lemma)
$\implies \overline{\mathcal{R}}_{z|p} \neq \emptyset$ (by set theory) $\ \wedge$
$z|p \in \mathcal{P}$
(by Theorem 4.3.2, production rule 3 of $\mathcal{P}^+$, production rule 3 of $\mathcal{P}^0$ and set theory)
$\implies z|p \in \mathcal{P}^+$ (by Lemma 4.3.6) $\ \wedge$
$z \in \mathcal{P}$ ($\because z \in \mathcal{P}^0$, and by Theorem 4.3.2 and set theory)
$\implies factors^+_m(z|p) \neq \emptyset_m$ (by Lemma 4.3.15, $\because p \in \mathcal{P}$).
And $(z|p)' \in \mathcal{P}$ ($\because (z|p)' \in \mathcal{P}^+$, and by Theorem 4.3.2 and set theory)
$\implies (z|p)' = 0 \ \vee\ factors^+_m((z|p)') \subset factors^+_m(z|p)$
(by Lemma 4.3.18, $\because z|p \in \mathcal{P} \ \wedge\ \overline{\tau}_{r_\gamma} \in \overline{\mathcal{R}}_{z|p} \ \wedge\ z|p \xrightarrow{\overline{\tau}_{r_\gamma}} (z|p)'$)
$\implies factors^+_m((z|p)') = \emptyset_m$ ($\because factors^+_m(0) = \emptyset_m$, by definition of $factors^+_m(0)$) $\vee$
$factors^+_m((z|p)') \subset factors^+_m(z|p)$
$\implies factors^+_m((z|p)') \subset factors^+_m(z|p)$ (by set theory, and because $factors^+_m(z|p) \neq \emptyset_m$)
$\implies |factors^+_m((z|p)')| < |factors^+_m(z|p)|$ (by multiset theory)
$\implies 1 \leq |factors^+_m(z|p)| - |factors^+_m((z|p)')|$
($\because |factors^+_m(z|p)|, |factors^+_m((z|p)')| \in \mathbb{N}$, and by algebra of inequalities)
$\implies 1 \leq |factors^+_m(z|p)| - |factors^+_m((z|p)')| \ \wedge\ z|p \in \mathcal{P}^+ \ \wedge$

only the $R-Par$ or $CompDelet$ rules determine the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$
($\because$ $z|p \in \mathcal{P}^+ \ \wedge$ only the $R-Par$ or $CompDelet$ rules determine the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$)
$\implies \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{z|p}\ \forall (z|p)' \in \mathcal{P}^+$ with $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$
$(1 \leq |factors_m^+(z|p)| - |factors_m^+((z|p)')| \ \wedge\ z|p \in \mathcal{P}^+ \ \wedge$
only the $R-Par$ or $CompDelet$ rules determine the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$)
($\because$ $p \in \mathcal{P}$ and $z \in \mathcal{P}^0$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{z|p}$ and $(z|p)' \in \mathcal{P}^+$ with $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$ are arbitrary).

Therefore, we use complete induction on $|factors_m^+(z|p)| - |factors_m^+((z|p)')|$
and use only the $R-Par$ or $CompDelet$ rules to determine $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$.

For $n \in \mathbb{N}^+$, let $Prop(n)$ be the proposition:
$\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{z|p}\ \forall (z|p)' \in \mathcal{P}^+$
$(z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)' \ \wedge\ |factors_m^+(z|p)| - |factors_m^+((z|p)')| = n \implies$
$\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \wedge \exists p' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (z|p)' = z|p'))$.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(1)$
2. $\vdash \forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \implies Prop(n+1))$

Base Case: Proof of Prop(1)

For $p \in \mathcal{P}$ and $z \in \mathcal{P}^0$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{z|p}$ and $(z|p)' \in \mathcal{P}^+$,
$z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)' \ \wedge\ |factors_m^+(z|p)| - |factors_m^+((z|p)')| = 1$ (by the hypothesis of $Prop(1)$).

If the $R-Par$ rule defines the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$
then $\exists\, p' \in \mathcal{P}\ (p \xrightarrow{\overline{\tau}_{r_Y}} p')$ (by the hypothesis of $R-Par$) $\wedge$
$(z|p)' = z|p'$ (by the $R-Par$ rule)
$\implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\wedge\ z|p' \in \mathcal{P}^+$ ($\because (z|p)' \in \mathcal{P}^+$)
$\implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \ \wedge\ p' \in \mathcal{P}^+$ (by production rule 3 of $\mathcal{P}^+$ and Theorem 4.3.2, $\because z \in \mathcal{P}^0$)
$\implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \wedge \exists p' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (z|p)' = z|p')$ ($\because p \xrightarrow{\overline{\tau}_{r_Y}} p' \ \wedge\ (z|p)' = z|p'$).

If the $CompDelet$ rule defines the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$
then $\exists\, \overline{\tau}_{r_{Y_1}}, \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}\ \exists (z|p)'' \in \mathcal{P}\ (Y \sim_{of} Y_1|Y_2 \ \wedge\ z|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (z|p)'' \ \wedge\ (z|p)'' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (z|p)')$

(by the hypothesis of *CompDelet*)
$\implies \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{z|p}$ (by definition of $\overline{\mathcal{R}}_{z|p}$) $\wedge$ $\overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(z|p)''}$ (by definition of $\overline{\mathcal{R}}_{(z|p)''}$)
$\implies \overline{\mathcal{R}}_{(z|p)''} \neq \emptyset$ (by set theory)
$\implies (z|p)'' \in \mathcal{P}$ (by definition of $z|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (z|p)''$) $\wedge$ $\overline{\mathcal{R}}_{(z|p)''} \neq \emptyset$
$\implies (z|p)'' \in \mathcal{P}^+$ (by Lemma 4.3.6)
$\implies (z|p)'' \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because (z|p)'' \in \mathcal{P}$)
$\implies (z|p)'' \neq 0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$).
Now $z|p \in \mathcal{P}$
(by Theorem 4.3.2, production rule 3 of $\mathcal{P}^+$, production rule 3 of $\mathcal{P}^0$ and set theory, $\because p \in \mathcal{P} \wedge z \in \mathcal{P}^0$)
$\implies (z|p)'' = 0 \vee factors_m^+((z|p)'') \subset factors_m^+(z|p)$
(by Lemma 4.3.18, $\because (z|p)'' \in \mathcal{P} \wedge \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{z|p} \wedge z|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (z|p)''$)
$\implies factors_m^+((z|p)'') \subset factors_m^+(z|p)$ ($\because (z|p)'' \neq 0$)
$\implies |factors_m^+((z|p)'')| < |factors_m^+(z|p)|$ (by multiset theory).

And $(z|p)' \in \mathcal{P}^+$ (by the hypothesis of *Prop*(1))
$\implies (z|p)' \in \mathcal{P}$ (by Theorem 4.3.2 and set theory)
$\implies (z|p)' \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because (z|p)' \in \mathcal{P}^+$)
$\implies (z|p)' \neq 0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$).
Now $(z|p)' = 0 \vee factors_m^+((z|p)') \subset factors_m^+((z|p)'')$
(by Lemma 4.3.18, $\because (z|p)'' \in \mathcal{P} \wedge (z|p)' \in \mathcal{P} \wedge \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(z|p)''} \wedge (z|p)'' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (z|p)'$)
$\implies factors_m^+((z|p)') \subset factors_m^+((z|p)'')$ ($\because (z|p)' \neq 0$)
$\implies |factors_m^+((z|p)')| < |factors_m^+((z|p)'')|$ (by multiset theory)
$\implies |factors_m^+((z|p)')| < |factors_m^+((z|p)'')| < |factors_m^+(z|p)|$
(by algebra of inequalities, $\because |factors_m^+((z|p)'')| < |factors_m^+(z|p)|$)
$\implies 2 \leq |factors_m^+(z|p)| - |factors_m^+((z|p)')|$
(by algebra of inequalities and
because $|factors_m^+(z|p)|, |factors_m^+((z|p)'')|, |factors_m^+((z|p)')| \in \mathbb{N}$).
But $|factors_m^+(z|p)| - |factors_m^+((z|p)')| = 1$
(by the hypothesis of *Prop*(1); which is a contradiction).
$\therefore$ The *CompDelet* rule does not define the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$.

$\therefore \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{z|p}\ \forall (z|p)' \in \mathcal{P}^+$
$(z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)' \wedge |factors_m^+(z|p)| - |factors_m^+((z|p)')| = 1 \implies$
$\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \wedge \exists p' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (z|p)' = z|p'))$
($\because p \in \mathcal{P}$ and $z \in \mathcal{P}^0$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{z|p}$ and $(z|p)' \in \mathcal{P}^+$ with

$z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$ and $|factors^+_m(z|p)| - |factors^+_m((z|p)')| = 1$ are arbitrary)
$\Longrightarrow$ *Prop*(1) holds (by definition of *Prop*(1)). Q.E.D.

Induction Step: Proof of $\forall n \in \mathbb{N}^+ \ (\forall m \in [1,n] Prop(m) \Longrightarrow Prop(n+1))$

For $n \in \mathbb{N}^+$, assume $\forall m \in [1,n] Prop(m)$ holds (inductive hypothesis).
For $p \in \mathcal{P}$ and $z \in \mathcal{P}^0$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{z|p}$ and $(z|p)' \in \mathcal{P}^+$,
$z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)' \ \wedge \ |factors^+_m(z|p)| - |factors^+_m((z|p)')| = n+1$
(by the hypothesis of $Prop(n+1)$).

If the $R-Par$ rule defines the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$
then $\exists p' \in \mathcal{P} (p \xrightarrow{\overline{\tau}_{r_Y}} p')$ (by the hypothesis of $R-Par$) $\wedge$
$(z|p)' = z|p'$ (by the $R-Par$ rule)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\wedge \ z|p' \in \mathcal{P}^+$ ($\because (z|p)' \in \mathcal{P}^+$)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \ \wedge \ p' \in \mathcal{P}^+$ (by production rule 3 of $\mathcal{P}^+$ and Theorem 4.3.2, $\because z \in \mathcal{P}^0$)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p \wedge \exists p' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (z|p)' = z|p')$ ($\because p \xrightarrow{\overline{\tau}_{r_Y}} p' \ \wedge \ (z|p)' = z|p'$).

If the *CompDelet* rule defines the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$
then $\exists \overline{\tau}_{r_{Y_1}}, \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}} \ \exists (z|p)'' \in \mathcal{P} \ (Y \sim_{of} Y_1|Y_2 \ \wedge \ z|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (z|p)'' \ \wedge \ (z|p)'' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (z|p)')$
(by the hypothesis of *CompDelet*)
$\Longrightarrow \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{z|p}$ (by definition of $\overline{\mathcal{R}}_{z|p}$) $\ \wedge \ \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(z|p)''}$ (by definition of $\overline{\mathcal{R}}_{(z|p)''}$)
$\Longrightarrow \overline{\mathcal{R}}_{(z|p)''} \neq \emptyset$ (by set theory)
$\Longrightarrow (z|p)'' \in \mathcal{P}$ (by definition of $z|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (z|p)''$) $\ \wedge \ \overline{\mathcal{R}}_{(z|p)''} \neq \emptyset$
$\Longrightarrow (z|p)'' \in \mathcal{P}^+$ (by Lemma 4.3.6)
$\Longrightarrow (z|p)'' \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because (z|p)'' \in \mathcal{P}$)
$\Longrightarrow (z|p)'' \neq 0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$).
Now $z|p \in \mathcal{P}$
(by Theorem 4.3.2, production rule 3 of $\mathcal{P}^+$, production rule 3 of $\mathcal{P}^0$ and set theory,
$\because p \in \mathcal{P} \ \wedge \ z \in \mathcal{P}^0$)
$\Longrightarrow (z|p)'' = 0 \ \vee \ factors^+_m((z|p)'') \subset factors^+_m(z|p)$
(by Lemma 4.3.18, $\because (z|p)'' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{z|p} \ \wedge \ z|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (z|p)''$)
$\Longrightarrow factors^+_m((z|p)'') \subset factors^+_m(z|p)$ ($\because (z|p)'' \neq 0$)
$\Longrightarrow |factors^+_m((z|p)'')| < |factors^+_m(z|p)|$ (by multiset theory)
$\Longrightarrow 1 \leq |factors^+_m(z|p)| - |factors^+_m((z|p)'')|$
(by algebra of inequalities and because $|factors^+_m(z|p)|, |factors^+_m((z|p)'')| \in \mathbb{N}$).

And $(z|p)' \in \mathcal{P}^+$ (by the hypothesis of $Prop(1)$)
$\implies (z|p)' \in \mathcal{P}$ (by Theorem 4.3.2 and set theory)
$\implies (z|p)' \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because (z|p)' \in \mathcal{P}^+$)
$\implies (z|p)' \neq 0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$).
Now $(z|p)' = 0 \ \vee \ factors_m^+((z|p)') \subset factors_m^+((z|p)'')$
(by Lemma 4.3.18, $\because (z|p)'' \in \mathcal{P} \ \wedge \ (z|p)' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(z|p)''} \ \wedge \ (z|p)'' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (z|p)'$)
$\implies factors_m^+((z|p)') \subset factors_m^+((z|p)'')$ ($\because (z|p)' \neq 0$)
$\implies |factors_m^+((z|p)')| < |factors_m^+((z|p)'')|$ (by multiset theory)
$\implies 1 \leq |factors_m^+((z|p)'')| - |factors_m^+((z|p)')|$
(by algebra of inequalities and because $|factors_m^+((z|p)'')|, |factors_m^+((z|p)')| \in \mathbb{N}$).

Let $m_1 \triangleq |factors_m^+(z|p)| - |factors_m^+((z|p)'')|$ and
$m_2 \triangleq |factors_m^+((z|p)'')| - |factors_m^+((z|p)')|$.
$m_1 \in \mathbb{N}^+$
(by definition of m_1, and
because $1 \leq |factors_m^+(z|p)| - |factors_m^+((z|p)'')| \ \wedge$
$|factors_m^+(z|p)|, |factors_m^+((z|p)'')| \in \mathbb{N}) \ \wedge$
$m_2 \in \mathbb{N}^+$
(by definition of m_2, and
because $1 \leq |factors_m^+((z|p)'')| - |factors_m^+((z|p)')| \ \wedge$
$|factors_m^+((z|p)'')|, |factors_m^+((z|p)')| \in \mathbb{N}$).
And $m_1 + m_2 =$
$(|factors_m^+(z|p)| - |factors_m^+((z|p)'')|) + (|factors_m^+((z|p)'')| - |factors_m^+((z|p)')|)$
(by definitions of m_1 and m_2)
$\implies m_1 + m_2 = |factors_m^+(z|p)| - |factors_m^+((z|p)')|$ (by arithmetic)
$\implies m_1 + m_2 = n + 1$ (by the hypothesis of $Prop(n+1)$)
$\implies m_1, m_2 \in [1, n]$ ($\because m_1, m_2 \in \mathbb{N}^+$, and by algebra of inequalities)
$\implies Prop(m_1)$ and $Prop(m_2)$ hold (by the inductive hypothesis)
$\implies \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_p \ \wedge \ \exists p'' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_{Y_1}}} p'' \ \wedge \ (z|p)'' = z|p'')$
(by modus ponens, $\because p \in \mathcal{P} \ \wedge \ z \in \mathcal{P}^0 \ \wedge \ \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{z|p} \ \wedge \ (z|p)'' \in \mathcal{P}^+ \ \wedge \ z|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (z|p)'' \ \wedge$
$|factors_m^+(z|p)| - |factors_m^+((z|p)'')| = m_1$)
$\implies p'' \in \mathcal{P}$ (by Theorem 4.3.2 and set theory) $\ \wedge \ (z|p)'' = z|p'' \ \wedge$
$\overline{\mathcal{R}}_{z|p''} = \overline{\mathcal{R}}_{(z|p)''}$ (by definitions of $\overline{\mathcal{R}}_{z|p''}$ and $\overline{\mathcal{R}}_{(z|p)''}$) $\ \wedge$
$factors_m^+(z|p'') = factors_m^+((z|p)'')$
(by definitions of $factors_m^+(z|p'')$ and $factors_m^+((z|p)'')$)
$\implies p'' \in \mathcal{P} \ \wedge \ \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{z|p''}$ ($\because \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(z|p)''}$) $\ \wedge \ z|p'' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (z|p)'$ ($\because (z|p)'' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (z|p)'$) $\ \wedge$
$|factors_m^+(z|p'')| - |factors_m^+((z|p)')| = m_2$

(by set theory and because $|factors^+_m((z|p)'')| - |factors^+_m((z|p)')| = m_2$)
$\Longrightarrow \overline{\tau}_{r_{Y_2}} {\in} \overline{\mathcal{R}}_{p''}\ \wedge\ \exists p' {\in} \mathcal{P}^+(p'' \xrightarrow{\overline{\tau}_{r_{Y_2}}} p'\ \wedge\ (z|p)' = z|p')$
(by modus ponens, $\because$ $Prop(m_2)$ holds $\wedge$ $z{\in}\mathcal{P}^0\ \wedge\ (z|p)' {\in} \mathcal{P}^+$)
$\Longrightarrow p \xrightarrow{\overline{\tau}_{r_Y}} p'$ (by the *CompDelet* rule, $\because$ $Y \sim_{of} Y_1|Y_2\ \wedge\ p \xrightarrow{\overline{\tau}_{r_{Y_1}}} p'') \wedge$
$z|p' {\in} \mathcal{P}^+$ ($\because$ $(z|p)' {\in} \mathcal{P}^+$)
$\Longrightarrow \overline{\tau}_{r_Y} {\in} \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\wedge$
$p' {\in} \mathcal{P}^+$ (by production rule 3 of $\mathcal{P}^+$ and Theorem 4.3.2, $\because$ $z{\in}\mathcal{P}^0$)
$\Longrightarrow \overline{\tau}_{r_Y} {\in} \overline{\mathcal{R}}_p \wedge \exists p' {\in} \mathcal{P}^+(p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (z|p)' = z|p')$ ($\because$ $p \xrightarrow{\overline{\tau}_{r_Y}} p'\ \wedge\ (z|p)' = z|p'$).

$\therefore$ $\forall p{\in}\mathcal{P}\ \forall z{\in}\mathcal{P}^0\ \forall \overline{\tau}_{r_Y} {\in} \overline{\mathcal{R}}_{z|p}\ \forall (z|p)' {\in} \mathcal{P}^+$
$(z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'\ \wedge\ |factors^+_m(z|p)| - |factors^+_m((z|p)')| = n+1 \Longrightarrow$
$\overline{\tau}_{r_Y} {\in} \overline{\mathcal{R}}_p \wedge \exists p' {\in} \mathcal{P}^+(p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (z|p)' = z|p'))$
($\because$ $p{\in}\mathcal{P}$ and $z{\in}\mathcal{P}^0$ and $\overline{\tau}_{r_Y} {\in} \overline{\mathcal{R}}_{z|p}$ and $(z|p)' {\in} \mathcal{P}^+$ with $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'$ and
$|factors^+_m(z|p)| - |factors^+_m((z|p)')| = n+1$ are arbitrary)
$\Longrightarrow$ $Prop(n+1)$ holds (by definition of $Prop(n+1)$).
$\therefore$ $\forall n {\in} \mathbb{N}^+$ ($\forall m {\in} [1,n] Prop(m) \Longrightarrow Prop(n+1)$) holds ($\because$ $n {\in} \mathbb{N}^+$ is arbitrary). Q.E.D.

$\therefore$ $\forall n {\in} \mathbb{N}^+$ $Prop(n)$ holds (by complete induction).
$\therefore$ $\forall p{\in}\mathcal{P}\ \forall z{\in}\mathcal{P}^0\ \forall \overline{\tau}_{r_Y} {\in} \overline{\mathcal{R}}_{z|p}\ \forall (z|p)' {\in} \mathcal{P}^+$
$(z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)' \Longrightarrow \overline{\tau}_{r_Y} {\in} \overline{\mathcal{R}}_p \wedge \exists p' {\in} \mathcal{P}^+(p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge (z|p)' = z|p'))$
($\because$ $\forall p{\in}\mathcal{P}\ \forall z{\in}\mathcal{P}^0\ \forall \overline{\tau}_{r_Y} {\in} \overline{\mathcal{R}}_{z|p}\ \forall (z|p)' {\in} \mathcal{P}^+$
$(z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)' \Longrightarrow |factors^+_m(z|p)| - |factors^+_m((z|p)')| {\in} \mathbb{N}^+)$). Q.E.D.

A.16 Theorem 4.3.7 $\forall p{\in}\mathcal{P}\ \forall z{\in}\mathcal{P}^0(p|z \sim_{dp} p\ \wedge\ p \sim_{dp} z|p)$

Proof: consists of discharging the following two proof obligations. Each proof obligation is discharged by defining a binary relation T on $\mathcal{P}$ which contains the pair of processes that are required to be strongly dp-bisimilar, proving T is a strong dp-simulation on $\mathcal{P}$, then proving T is a strong dp-bisimulation on $\mathcal{P}$.

1. $\vdash \forall p{\in}\mathcal{P}\ \forall z{\in}\mathcal{P}^0(p|z \sim_{dp} p)$

2. $\vdash \forall p{\in}\mathcal{P}\ \forall z{\in}\mathcal{P}^0(p \sim_{dp} z|p)$

A.16.1 $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{dp} p)$

Proof: If $\exists$ strong dp-bisimulation T on $\mathcal{P}$ with $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 ((p|z, p) \in T)$
then $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{dp} p)$ (by definition of $p|z \sim_{dp} p$).
Therefore, we find such a T.

Let $S \triangleq \{(p|z, p) \mid p \in \mathcal{P}\ \wedge\ z \in \mathcal{P}^0\}$.
$S \subseteq \mathcal{P} \times \mathcal{P}$
(by definition of S, Theorem 4.3.2, production rule 3 of $\mathcal{P}^+$, production rule 3 of $\mathcal{P}^0$ and set theory) $\wedge$
$\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 ((p|z, p) \in S)$ (by definition of S).
Let $Z \triangleq \{(z_1, z_2) \mid z_1, z_2 \in \mathcal{P}^0\}$.
$Z \subseteq \mathcal{P} \times \mathcal{P}$ (by definition of Z, Theorem 4.3.2 and set theory).
Let $T \triangleq S \cup Z$.
$\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 ((p|z, p) \in T)$ (by set theory and definitions of S and T).

T is a strong dp-bisimulation on $\mathcal{P}$
$\iff T, T^{-1}$ are strong dp-simulations on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$)
$\iff T, T^{-1}$ are binary relations on $\mathcal{P}$ $\wedge$
for all elements of T, T^{-1} the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
(by definition of strong dp-simulation on $\mathcal{P}$).
We prove T, T^{-1} are binary relations on $\mathcal{P}$ and for all elements of T, T^{-1} the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied.

A.16.1.1 $\mathbf{T, T^{-1}}$ satisfy the Observation and Fraction conditions

S is a strong of-bisimulation on $\mathcal{P}$
(by the proof of Theorem 4.3.3, see A.7) $\wedge$
Z is a strong of-bisimulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$, $\because \mathcal{I}_{z_1} \cup \mathcal{R}_{z_1} = \emptyset\ \wedge\ \mathcal{I}_{z_2} \cup \mathcal{R}_{z_2} = \emptyset$ (by Lemma 4.3.7))
$\implies T$ is a strong of-bisimulation on $\mathcal{P}$
($\because$ the union of strong of-bisimulations on $\mathcal{P}$ is a strong of-bisimulation on $\mathcal{P}$, and by definition of T)
$\implies T, T^{-1}$ are strong of-simulations on $\mathcal{P}$

(by definition of strong of-bisimulation on $\mathcal{P}$)
$\Longrightarrow T \subseteq \mathcal{P}\ \mathrm{x}\ \mathcal{P}\ \wedge\ T^{-1} \subseteq \mathcal{P}\ \mathrm{x}\ \mathcal{P}\ \wedge$ for all elements of T, T^{-1} the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied
(by definition of strong of-simulation on $\mathcal{P}$)
$\Longrightarrow$ for all elements of T, T^{-1} the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
($\because$ the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are the same as the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$, respectively). Q.E.D.

It remains to prove that for all elements of T, T^{-1} the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied.

A.16.1.2 T satisfies the Deletion condition

We use complete induction on $|factors^+_m(u)|$ for $u \in dom(T)$.

For $n \in \mathbb{N}$, let $Prop(n)$ be the proposition:
$\forall (u, v) \in T\ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_u\ \forall u'' \in \mathcal{P}$
$(|factors^+_m(u)| = n\ \wedge\ u \xrightarrow{\overline{\tau}_{r_X}} u'' \Longrightarrow \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_v \wedge \exists v'' \in \mathcal{P}\ (v \xrightarrow{\overline{\tau}_{r_X}} v''\ \wedge\ (u'', v'') \in T))$.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(0)$
2. $\vdash \forall n \in \mathbb{N}\ (\forall m \in [0, n] Prop(m) \Longrightarrow Prop(n+1))$

Base Case: Proof of Prop(0)

For $(u, v) \in T$, $u \in \mathcal{P}$ ($\because T \subseteq \mathcal{P}\ \mathrm{x}\ \mathcal{P}$ and by set theory) $\wedge$
$|factors^+_m(u)| = 0$ (by the hypothesis of $Prop(0)$)
$\Longrightarrow factors^+_m(u) = \emptyset_m$ (by set theory).
Now $u \in \mathcal{P}^+\ \vee\ u \in \mathcal{P}^0$ (by Theorem 4.3.2, $\because u \in \mathcal{P}$).
If $u \in \mathcal{P}^+$
then $u \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because u \in \mathcal{P}$)
$\Longrightarrow (u, v) \notin Z$ (by definition of Z)
$\Longrightarrow (u, v) \in S$ ($\because (u, v) \in T\ \wedge\ T = S \cup Z$)
$\Longrightarrow \exists p \in \mathcal{P}\ \exists z \in \mathcal{P}^0\ (u = p|z\ \wedge\ v = p)$ (by definition of S)

$\Longrightarrow p|z \in \mathcal{P}^+$ ($\because u \in \mathcal{P}^+$) $\wedge$ $z \in \mathcal{P}$ (by Theorem 4.3.2 and set theory)
$\Longrightarrow factors_m^+(p|z) \neq \emptyset_m$ (by Lemma 4.3.15, $\because p \in \mathcal{P}$)
$\Longrightarrow factors_m^+(u) \neq \emptyset_m$ ($\because u = p|z$; which is a contradiction).
$\therefore u \notin \mathcal{P}^+$
$\Longrightarrow u \in \mathcal{P}^0$ ($\because u \in \mathcal{P}^+ \vee u \in \mathcal{P}^0$)
$\Longrightarrow \overline{\mathcal{R}}_u = \emptyset$ (by Lemma 4.3.8)
$\Longrightarrow$ consequent of $Prop(0)$ holds ($\because \emptyset$ satisfies all conditions)
$\Longrightarrow Prop(0)$ holds
(by definition of $Prop(0)$, $\because (u, v) \in T$ with $|factors_m^+(u)| = 0$ is arbitrary). Q.E.D.

Induction Step: Proof of $\forall \mathbf{n} \in \mathbb{N}$ ($\forall \mathbf{m} \in [\mathbf{0}, \mathbf{n}]\mathbf{Prop(m)} \Longrightarrow \mathbf{Prop(n+1)}$)

For $n \in \mathbb{N}$, assume $\forall m \in [0, n] Prop(m)$ holds (inductive hypothesis).
For $(u, v) \in T$, $u \in \mathcal{P}$ ($\because T \subseteq \mathcal{P}$ x $\mathcal{P}$ and by set theory) $\wedge$
$|factors_m^+(u)| = n + 1$ (by the hypothesis of $Prop(n+1)$)
$\Longrightarrow |factors_m^+(u)| \geq 1$ (by algebra of inequalities, $\because n \geq 0$)
$\Longrightarrow factors_m^+(u) \neq \emptyset_m$ (by set theory).
Now $u \in \mathcal{P}^+ \vee u \in \mathcal{P}^0$ (by Theorem 4.3.2, $\because u \in \mathcal{P}$).
If $u \in \mathcal{P}^0$
then $factors_m^+(u) = \emptyset_m$ (by Lemma 4.3.16, $\because u \in \mathcal{P}$; which is a contradiction).
$\therefore u \notin \mathcal{P}^0$
$\Longrightarrow u \in \mathcal{P}^+$ ($\because u \in \mathcal{P}^+ \vee u \in \mathcal{P}^0$) $\wedge$ $(u, v) \notin Z$ (by definition of Z)
$\Longrightarrow (u, v) \in S$ ($\because (u, v) \in T\ \wedge\ T = S \cup Z$)
$\Longrightarrow \exists p \in \mathcal{P}\ \exists z \in \mathcal{P}^0$ ($u = p|z\ \wedge\ v = p$) (by definition of S)
$\Longrightarrow p|z \in \mathcal{P}$ ($\because u \in \mathcal{P}$) $\wedge$ $p|z \in \mathcal{P}^+$ ($\because u \in \mathcal{P}^+$) $\wedge$
$z \in \mathcal{P}$ (by Theorem 4.3.2 and set theory)
$\Longrightarrow p \in \mathcal{P}^+$ (by production rule 3 of $\mathcal{P}^+$ and Theorem 4.3.2, $\because p \in \mathcal{P} \wedge z \in \mathcal{P}^0$) $\wedge$
$\overline{\mathcal{R}}_{p|z} \neq \emptyset$ (by Lemma 4.3.6)
$\Longrightarrow \exists\, \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_{p|z}$ (by set theory)
$\Longrightarrow \exists\, (p|z)'' \in \mathcal{P}\, (p|z \xrightarrow{\overline{\tau}_{r_X}} (p|z)'')$ (by definition of $\overline{\mathcal{R}}_{p|z}$)
$\Longrightarrow$ only the *Delet*, $L - Par$, $R - Par$ or *CompDelet* rules determine the transition $p|z \xrightarrow{\overline{\tau}_{r_X}} (p|z)''$
(by definitions of the LTS rules).
Now $z \in \mathcal{P}^0$ (by definition of z)
$\Longrightarrow \overline{\mathcal{R}}_z = \emptyset$ (by Lemma 4.3.8)
$\Longrightarrow \overline{\tau}_{r_X} \notin \overline{\mathcal{R}}_z$ (by set theory)
$\Longrightarrow \neg(\overline{\tau}_{r_X} \in \overline{\mathcal{R}}\ \exists z'' \in \mathcal{P}\, (z \xrightarrow{\overline{\tau}_{r_X}} z''))$ (by definition of $\overline{\mathcal{R}}_z$)

$\Longrightarrow$ the $R-Par$ rule does not define the transition $p|z \xrightarrow{\overline{\tau}_{r_X}} (p|z)''$
($\because$ the hypothesis of $R-Par$ does not hold)
$\Longrightarrow$ only the *Delet*, $L-Par$ or *CompDelet* rules determine the transition $p|z \xrightarrow{\overline{\tau}_{r_X}} (p|z)''$
($\because$ only the *Delet*, $L-Par$, $R-Par$ or *CompDelet* rules determine the transition $p|z \xrightarrow{\overline{\tau}_{r_X}} (p|z)''$).

If the *Delet* rule defines the transition $p|z \xrightarrow{\overline{\tau}_{r_X}} (p|z)''$
then $(p|z)'' = 0$ (by the conclusion of *Delet*) $\wedge$
$p|z \sim_{of} X$ (by the hypothesis of *Delet*)
$\Longrightarrow p|z \sim_{of} p$ (by Theorem 4.3.3, $\because p \in \mathcal{P} \wedge z \in \mathcal{P}^0$) $\wedge$ $p|z \sim_{of} X$
$\Longrightarrow p \sim_{of} p|z$ ($\because \sim_{of}$ is symmetric, by Lemma 4.3.2) $\wedge$ $p|z \sim_{of} X$
$\Longrightarrow p \sim_{of} X$ ($\because \sim_{of}$ is transitive, by Lemma 4.3.4)
$\Longrightarrow p \xrightarrow{\overline{\tau}_{r_X}} 0$ (by the *Delet* rule, $\because p \in \mathcal{P}^+$)
$\Longrightarrow \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\wedge$
$0 \in \mathcal{P}$ (by production rule 1 of $\mathcal{P}^0$, set theory and Theorem 4.3.2) $\wedge$ $p \xrightarrow{\overline{\tau}_{r_X}} 0$ $\wedge$
$(0,0) \in T$ (by set theory and definitions of Z and T)
$\Longrightarrow \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_v \wedge 0 \in \mathcal{P} \wedge v \xrightarrow{\overline{\tau}_{r_X}} 0 \wedge (0,0) \in T$ ($\because v = p$).

If the $L-Par$ rule defines the transition $p|z \xrightarrow{\overline{\tau}_{r_X}} (p|z)''$
then $\exists p'' \in \mathcal{P} (p \xrightarrow{\overline{\tau}_{r_X}} p'' \wedge (p|z)'' = p''|z)$ (by definition of $L-Par$)
$\Longrightarrow p'' \in \mathcal{P} \wedge p \xrightarrow{\overline{\tau}_{r_X}} p'' \wedge ((p|z)'', p'') = (p''|z, p'')$
(by definition of p'', and by algebra of binary relations)
$\Longrightarrow \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\wedge$ $p'' \in \mathcal{P}$ $\wedge$ $p \xrightarrow{\overline{\tau}_{r_X}} p''$ $\wedge$
$((p|z)'', p'') \in T$ ($\because (p''|z, p'') \in T$, by set theory and definitions of S and T)
$\Longrightarrow \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_v \wedge p'' \in \mathcal{P} \wedge v \xrightarrow{\overline{\tau}_{r_X}} p'' \wedge ((p|z)'', p'') \in T$ ($\because v = p$).

If the *CompDelet* rule defines the transition $p|z \xrightarrow{\overline{\tau}_{r_X}} (p|z)''$
then $\exists \overline{\tau}_{r_{X_1}}, \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}\ \exists (p|z)' \in \mathcal{P} (X \sim_{of} X_1|X_2 \wedge p|z \xrightarrow{\overline{\tau}_{r_{X_1}}} (p|z)' \wedge (p|z)' \xrightarrow{\overline{\tau}_{r_{X_2}}} (p|z)'')$
(by the hypothesis of *CompDelet*)
$\Longrightarrow \overline{\tau}_{r_{X_1}} \in \overline{\mathcal{R}}_{p|z}$ (by definition of $\overline{\mathcal{R}}_{p|z}$) $\wedge$ $\overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}_{(p|z)'}$ (by definition of $\overline{\mathcal{R}}_{(p|z)'}$)
$\Longrightarrow \overline{\mathcal{R}}_{(p|z)'} \neq \emptyset$ (by set theory).
$\Longrightarrow (p|z)' \in \mathcal{P}^+$ (by Lemma 4.3.6, $\because (p|z)' \in \mathcal{P}$)
$\Longrightarrow (p|z)' \in \mathcal{P}^+ \wedge (p|z)' \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because (p|z)' \in \mathcal{P}$)
$\Longrightarrow \overline{\tau}_{r_{X_1}} \in \overline{\mathcal{R}}_p \wedge \exists p' \in \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_{X_1}}} p' \wedge (p|z)' = p'|z)$
(by Lemma 4.3.19, $\because p \in \mathcal{P} \wedge z \in \mathcal{P}^0 \wedge \overline{\tau}_{r_{X_1}} \in \overline{\mathcal{R}}_{p|z} \wedge p|z \xrightarrow{\overline{\tau}_{r_{X_1}}} (p|z)'$)

$\Longrightarrow ((p|z)', p') = (p'|z, p')$ (by algebra of binary relations)
$\Longrightarrow ((p|z)', p') \in S$ ($\because (p'|z, p') \in S$, by definition of S)
$\Longrightarrow ((p|z)', p') \in T$ (by set theory and definition of T).

Now $(p|z)' = 0\ \vee\ factors^+_m((p|z)') \subset factors^+_m(p|z)$
(by Lemma 4.3.18, $\because p|z, (p|z)' \in \mathcal{P}\ \wedge\ \overline{\tau}_{r_{X_1}} \in \overline{\mathcal{R}}_{p|z}\ \wedge\ p|z \xrightarrow{\overline{\tau}_{r_{X_1}}} (p|z)'$)
$\Longrightarrow (p|z)' \in \mathcal{P}^0$ ($\because 0 \in \mathcal{P}^0$) $\vee$ $factors^+_m((p|z)') \subset factors^+_m(p|z)$
$\Longrightarrow factors^+_m((p|z)') \subset factors^+_m(p|z)$ ($\because (p|z)' \notin \mathcal{P}^0$)
$\Longrightarrow |factors^+_m((p|z)')| < |factors^+_m(p|z)|$ (by set theory)
$\Longrightarrow |factors^+_m((p|z)')| < n + 1$ ($\because u = p|z\ \wedge\ |factors^+_m(u)| = n + 1$)
$\Longrightarrow |factors^+_m((p|z)')| \in [0, n]$ (by set theory and algebra of inequalities)
$\Longrightarrow Prop(|factors^+_m((p|z)')|)$ holds (by the inductive hypothesis)
$\Longrightarrow \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}_{p'} \wedge \exists p'' \in \mathcal{P}\ (p' \xrightarrow{\overline{\tau}_{r_{X_2}}} p''\ \wedge\ ((p|z)'', p'') \in T)$
(by modus ponens,
$\because ((p|z)', p') \in T\ \wedge\ \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}_{(p|z)'}\ \wedge\ (p|z)'' \in \mathcal{P}\ \wedge\ (p|z)' \xrightarrow{\overline{\tau}_{r_{X_2}}} (p|z)''$)
$\Longrightarrow p \xrightarrow{\overline{\tau}_{r_X}} p''\ \wedge\ ((p|z)'', p'') \in T$ (by the *CompDelet* rule, $\because X \sim_{of} X_1|X_2\ \wedge\ p \xrightarrow{\overline{\tau}_{r_{X_1}}} p'$)
$\Longrightarrow \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\wedge$ $p'' \in \mathcal{P}$ (by definition of p'') $\wedge$ $p \xrightarrow{\overline{\tau}_{r_X}} p''$ $\wedge$
$((p|z)'', p'') \in T$
$\Longrightarrow \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_v\ \wedge\ p'' \in \mathcal{P}\ \wedge\ v \xrightarrow{\overline{\tau}_{r_X}} p''\ \wedge\ ((p|z)'', p'') \in T$ ($\because v = p$).

$\therefore Prop(n + 1)$ holds
($\because u = p|z\ \wedge\ v = p$, and
because $(u, v) \in T$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_{p|z}$ and $(p|z)'' \in \mathcal{P}$
with $|factors^+_m(u)| = n + 1$ and $p|z \xrightarrow{\overline{\tau}_{r_X}} (p|z)''$ are arbitrary).
$\therefore \forall n \in \mathbb{N}\ (\forall m \in [0, n] Prop(m) \Longrightarrow Prop(n + 1))$ holds ($\because n$ is arbitrary). Q.E.D.

$\therefore \forall n \in \mathbb{N}\ Prop(n)$ holds (by complete induction).
$\therefore$ For all elements of T the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied
($\because$ every process in $dom(T)$ has a finite number of positive singleton factors). Q.E.D.
$\therefore T$ is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$, $\because T \subseteq \mathcal{P}$ x $\mathcal{P}$ and for all elements of T the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied).

A.16.1.3 $\mathbf{T^{-1}}$ satisfies the Deletion condition

$T = S \cup Z$ (by definition of T)
$\Longrightarrow T^{-1} = S^{-1} \cup Z^{-1}$ (by algebra of binary relations)
$\Longrightarrow T^{-1} = \{(p, p|z), (z_2, z_1) \mid p \in \mathcal{P} \wedge z, z_1, z_2 \in \mathcal{P}^0\}$
(by definitions of S, Z and inverse binary relations, and by set theory).

If $p \xrightarrow{\overline{\tau}_{r_Y}} p'$ with $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}$
then $p|z \xrightarrow{\overline{\tau}_{r_Y}} p'|z$ (by the $L - Par$ rule).
And $(p', p'|z) \in T$ (by definition of T^{-1}).

And $z_2 \in \mathcal{P}^0$ (by definition of Z)
$\Longrightarrow \overline{\mathcal{R}}_{z_2} = \emptyset$ (by Lemma 4.3.8).

$\therefore$ For all elements of T^{-1} the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied
($\because p \in \mathcal{P}$ and $z, z_2 \in \mathcal{P}^0$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}$ are arbitrary). Q.E.D.
$\therefore T^{-1}$ is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$, $\because T^{-1} \subseteq \mathcal{P}$ x $\mathcal{P}$ and for all elements of T^{-1} the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied).
$\therefore T$ is a strong dp-bisimulation on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$).
$\therefore \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{dp} p)$ (by definition of $p|z \sim_{dp} p$). Q.E.D.

A.16.2 $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p \sim_{dp} z|p)$

Proof: If $\exists$ strong dp-bisimulation T on $\mathcal{P}$ with $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 ((p, z|p) \in T)$
then $\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p \sim_{dp} z|p)$ (by definition of $p \sim_{dp} z|p$).
Therefore, we find such a T.

Let $S \triangleq \{(p, z|p) \mid p \in \mathcal{P}\ \wedge\ z \in \mathcal{P}^0\}$.
$S \subseteq \mathcal{P}$ x $\mathcal{P}$
(by definition of S, Theorem 4.3.2, production rule 3 of $\mathcal{P}^+$, production rule 3 of $\mathcal{P}^0$ and set theory) $\wedge$
$\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 ((p, z|p) \in S)$ (by definition of S).
Let $Z \triangleq \{(z_1, z_2) \mid z_1, z_2 \in \mathcal{P}^0\}$.
$Z \subseteq \mathcal{P}$ x $\mathcal{P}$ (by definition of Z, Theorem 4.3.2 and set theory).

Let $T \triangleq S \cup Z$.
$\forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0((p, z|p) \in T)$ (by set theory and definitions of S and T).

T is a strong dp-bisimulation on $\mathcal{P}$
$\iff T, T^{-1}$ are strong dp-simulations on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$)
$\iff T, T^{-1}$ are binary relations on $\mathcal{P}$ $\wedge$
for all elements of T, T^{-1} the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
(by definition of strong dp-simulation on $\mathcal{P}$).
We prove T, T^{-1} are binary relations on $\mathcal{P}$ and for all elements of T, T^{-1} the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied.

A.16.2.1 T, T^{-1} satisfy the Observation and Fraction conditions

S is a strong of-bisimulation on $\mathcal{P}$
(by the proof of Theorem 4.3.3, see A.7) $\wedge$
Z is a strong of-bisimulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$, $\because \mathcal{I}_{z_1} \cup \mathcal{R}_{z_1} = \emptyset\ \wedge\ \mathcal{I}_{z_2} \cup \mathcal{R}_{z_2} = \emptyset$ (by Lemma 4.3.7))
$\implies T$ is a strong of-bisimulation on $\mathcal{P}$
($\because$ the union of strong of-bisimulations on $\mathcal{P}$ is a strong of-bisimulation on $\mathcal{P}$, and by definition of T)
$\implies T, T^{-1}$ are strong of-simulations on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$)
$\implies T \subseteq \mathcal{P}\,\mathrm{x}\,\mathcal{P}\ \wedge\ T^{-1} \subseteq \mathcal{P}\,\mathrm{x}\,\mathcal{P}\ \wedge$ for all elements of T, T^{-1} the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied
(by definition of strong of-simulation on $\mathcal{P}$)
$\implies$ for all elements of T, T^{-1} the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
($\because$ the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are the same as the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$, respectively). Q.E.D.

It remains to prove that for all elements of T, T^{-1} the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied.

A.16.2.2 T satisfies the Deletion condition

$T = S \cup Z$ (by definition of T)
$\implies T = \{(p, z|p), (z_1, z_2) \mid p \in \mathcal{P} \wedge z, z_1, z_2 \in \mathcal{P}^0\}$
(by definitions of S and Z, and by set theory).

If $p \xrightarrow{\overline{\tau}_{r_X}} p'$ with $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}$
then $z|p \xrightarrow{\overline{\tau}_{r_X}} z|p'$ (by the $R - Par$ rule).
And $(p', z|p') \in T$ (by definition of T).

And $z_1 \in \mathcal{P}^0$ (by definition of Z)
$\implies \overline{\mathcal{R}}_{z_1} = \emptyset$ (by Lemma 4.3.8).

$\therefore$ For all elements of T the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied
($\because p \in \mathcal{P}$ and $z, z_1 \in \mathcal{P}^0$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}$ are arbitrary). Q.E.D.
$\therefore T$ is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$, $\because T \subseteq \mathcal{P}$ x $\mathcal{P}$ and for all elements of T the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied).

A.16.2.3 T^{-1} satisfies the Deletion condition

Now $S^{-1} = \{(z|p, p) \mid p \in \mathcal{P}\ \wedge\ z \in \mathcal{P}^0\}$
(by definitions of S and inverse binary relations).
And $Z^{-1} = \{(z_2, z_1) \mid z_1, z_2 \in \mathcal{P}^0\}$ (by definitions of Z and inverse binary relations).
And $T = S \cup Z$ (by definition of T)
$\implies T^{-1} = S^{-1} \cup Z^{-1}$ (by algebra of binary relations).
We use complete induction on $|factors^+_m(u)|$ for $u \in dom(T^{-1})$.

For $n \in \mathbb{N}$, let $Prop(n)$ be the proposition:
$\forall (u, v) \in T^{-1}\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u\ \forall u'' \in \mathcal{P}$
$(|factors^+_m(u)| = n\ \wedge\ u \xrightarrow{\overline{\tau}_{r_Y}} u'' \implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \wedge \exists v'' \in \mathcal{P}\ (v \xrightarrow{\overline{\tau}_{r_Y}} v''\ \wedge\ (u'', v'') \in T^{-1}))$.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(0)$
2. $\vdash \forall n \in \mathbb{N}\ (\forall m \in [0, n] Prop(m) \implies Prop(n + 1))$

Base Case: Proof of Prop(0)

For $(u,v) \in T^{-1}$, $u \in \mathcal{P}$ ($\because T^{-1} \subseteq \mathcal{P}$ x $\mathcal{P}$ and by set theory) $\wedge$
$|factors_m^+(u)| = 0$ (by the hypothesis of $Prop(0)$)
$\implies factors_m^+(u) = \emptyset_m$ (by set theory).
Now $u \in \mathcal{P}^+ \vee u \in \mathcal{P}^0$ (by Theorem 4.3.2, $\because u \in \mathcal{P}$).
If $u \in \mathcal{P}^+$
then $u \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because u \in \mathcal{P}$)
$\implies (u,v) \notin Z^{-1}$ (by definition of Z^{-1})
$\implies (u,v) \in S^{-1}$ ($\because (u,v) \in T^{-1}\ \wedge\ T^{-1} = S^{-1} \cup Z^{-1}$)
$\implies \exists p \in \mathcal{P}\ \exists z \in \mathcal{P}^0\ (u = z|p\ \wedge\ v = p)$ (by definition of S^{-1})
$\implies z|p \in \mathcal{P}^+$ ($\because u \in \mathcal{P}^+$) $\wedge$ $z \in \mathcal{P}$ (by Theorem 4.3.2 and set theory)
$\implies factors_m^+(z|p) \neq \emptyset_m$ (by Lemma 4.3.15, $\because p \in \mathcal{P}$)
$\implies factors_m^+(u) \neq \emptyset_m$ ($\because u = z|p$; which is a contradiction).
$\therefore u \notin \mathcal{P}^+$
$\implies u \in \mathcal{P}^0$ ($\because u \in \mathcal{P}^+ \vee u \in \mathcal{P}^0$)
$\implies \overline{\mathcal{R}}_u = \emptyset$ (by Lemma 4.3.8)
$\implies$ consequent of $Prop(0)$ holds ($\because \emptyset$ satisfies all conditions)
$\implies Prop(0)$ holds
(by definition of $Prop(0)$, $\because (u,v) \in T^{-1}$ with $|factors_m^+(u)| = 0$ is arbitrary). Q.E.D.

Induction Step: Proof of $\forall \mathbf{n} \in \mathbb{N}$ ($\forall \mathbf{m} \in [\mathbf{0}, \mathbf{n}]$ Prop(m) $\implies$ Prop(n + 1))

For $n \in \mathbb{N}$, assume $\forall m \in [0, n] Prop(m)$ holds (inductive hypothesis).
For $(u,v) \in T^{-1}$, $u \in \mathcal{P}$ ($\because T^{-1} \subseteq \mathcal{P}$ x $\mathcal{P}$ and by set theory) $\wedge$
$|factors_m^+(u)| = n + 1$ (by the hypothesis of $Prop(n+1)$)
$\implies |factors_m^+(u)| \geq 1$ (by algebra of inequalities, $\because n \geq 0$)
$\implies factors_m^+(u) \neq \emptyset_m$ (by set theory).
Now $u \in \mathcal{P}^+ \vee u \in \mathcal{P}^0$ (by Theorem 4.3.2, $\because u \in \mathcal{P}$).
If $u \in \mathcal{P}^0$
then $factors_m^+(u) = \emptyset_m$ (by Lemma 4.3.16, $\because u \in \mathcal{P}$; which is a contradiction).
$\therefore u \notin \mathcal{P}^0$
$\implies u \in \mathcal{P}^+$ ($\because u \in \mathcal{P}^+ \vee u \in \mathcal{P}^0$) $\wedge$ $(u,v) \notin Z^{-1}$ (by definition of Z^{-1})
$\implies (u,v) \in S^{-1}$ ($\because (u,v) \in T^{-1}\ \wedge\ T^{-1} = S^{-1} \cup Z^{-1}$)
$\implies \exists p \in \mathcal{P}\ \exists z \in \mathcal{P}^0\ (u = z|p\ \wedge\ v = p)$ (by definition of S^{-1})
$\implies z|p \in \mathcal{P}$ ($\because u \in \mathcal{P}$) $\wedge$ $z|p \in \mathcal{P}^+$ ($\because u \in \mathcal{P}^+$) $\wedge$
$z \in \mathcal{P}$ (by Theorem 4.3.2 and set theory)
$\implies p \in \mathcal{P}^+$ (by production rule 3 of $\mathcal{P}^+$ and Theorem 4.3.2, $\because p \in \mathcal{P} \wedge z \in \mathcal{P}^0$) $\wedge$

$\overline{\mathcal{R}}_{z|p} \neq \emptyset$ (by Lemma 4.3.6)
$\Longrightarrow \exists\, \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{z|p}$ (by set theory)
$\Longrightarrow \exists\, (z|p)'' \in \mathcal{P}\, (z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)'')$ (by definition of $\overline{\mathcal{R}}_{z|p}$)
$\Longrightarrow$ only the *Delet*, $L-Par$, $R-Par$ or *CompDelet* rules determine the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)''$
(by definitions of the LTS rules).
Now $z \in \mathcal{P}^0$ (by definition of z)
$\Longrightarrow \overline{\mathcal{R}}_z = \emptyset$ (by Lemma 4.3.8)
$\Longrightarrow \overline{\tau}_{r_Y} \notin \overline{\mathcal{R}}_z$ (by set theory)
$\Longrightarrow \neg(\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}\ \exists z'' \in \mathcal{P}\, (z \xrightarrow{\overline{\tau}_{r_Y}} z''))$ (by definition of $\overline{\mathcal{R}}_z$)
$\Longrightarrow$ the $L-Par$ rule does not define the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)''$
($\because$ the hypothesis of $L-Par$ does not hold)
$\Longrightarrow$ only the *Delet*, $R-Par$ or *CompDelet* rules determine the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)''$
($\because$ only the *Delet*, $L-Par$, $R-Par$ or *CompDelet* rules determine the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)''$).

If the *Delet* rule defines the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)''$
then $(z|p)'' = 0$ (by the conclusion of *Delet*) $\ \wedge$
$z|p \sim_{of} Y$ (by the hypothesis of *Delet*)
$\Longrightarrow p \sim_{of} z|p$ (by Theorem 4.3.3, $\because p \in \mathcal{P} \ \wedge\ z \in \mathcal{P}^0$) $\ \wedge\ z|p \sim_{of} Y$
$\Longrightarrow p \sim_{of} Y$ ($\because \sim_{of}$ is transitive, by Lemma 4.3.4)
$\Longrightarrow p \xrightarrow{\overline{\tau}_{r_Y}} 0$ (by the *Delet* rule, $\because p \in \mathcal{P}^+$)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\ \wedge$
$0 \in \mathcal{P}$ (by production rule 1 of $\mathcal{P}^0$, set theory and Theorem 4.3.2) $\ \wedge\ p \xrightarrow{\overline{\tau}_{r_Y}} 0\ \wedge$
$(0,0) \in T^{-1}$ (by set theory and definitions of Z^{-1} and T^{-1})
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \ \wedge\ 0 \in \mathcal{P} \ \wedge\ v \xrightarrow{\overline{\tau}_{r_Y}} 0 \ \wedge\ (0,0) \in T^{-1}$ ($\because v = p$).

If the $R-Par$ rule defines the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)''$
then $\exists\, p'' \in \mathcal{P}\, (p \xrightarrow{\overline{\tau}_{r_Y}} p'' \ \wedge\ (z|p)'' = z|p'')$ (by definition of $R-Par$)
$\Longrightarrow p'' \in \mathcal{P} \ \wedge\ p \xrightarrow{\overline{\tau}_{r_Y}} p'' \ \wedge\ ((z|p)'', p'') = (z|p'', p'')$
(by definition of p'', and by algebra of binary relations)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\ \wedge\ \ p'' \in \mathcal{P}\ \ \wedge\ \ p \xrightarrow{\overline{\tau}_{r_Y}} p''\ \ \wedge$
$((z|p)'', p'') \in T^{-1}$ ($\because (z|p'', p'') \in T^{-1}$, by set theory and definitions of S^{-1} and T^{-1})
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \ \wedge\ p'' \in \mathcal{P} \ \wedge\ v \xrightarrow{\overline{\tau}_{r_Y}} p'' \ \wedge\ ((z|p)'', p'') \in T^{-1}$ ($\because v = p$).

If the *CompDelet* rule defines the transition $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)''$
then $\exists\, \overline{\tau}_{r_{Y_1}}, \overline{\tau}_{r_{Y_2}} {\in} \overline{\mathcal{R}}\ \exists (z|p)' {\in} \mathcal{P}\,(Y \sim_{of} Y_1|Y_2 \ \wedge\ z|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (z|p)' \ \wedge\ (z|p)' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (z|p)'')$
(by the hypothesis of *CompDelet*)
$\implies \overline{\tau}_{r_{Y_1}} {\in} \overline{\mathcal{R}}_{z|p}$ (by definition of $\overline{\mathcal{R}}_{z|p}$) $\ \wedge\ \overline{\tau}_{r_{Y_2}} {\in} \overline{\mathcal{R}}_{(z|p)'}$ (by definition of $\overline{\mathcal{R}}_{(z|p)'}$)
$\implies \overline{\mathcal{R}}_{(z|p)'} \neq \emptyset$ (by set theory)
$\implies (z|p)' {\in} \mathcal{P}^+$ (by Lemma 4.3.6, $\because (z|p)' {\in} \mathcal{P}$)
$\implies (z|p)' {\in} \mathcal{P}^+ \ \wedge\ (z|p)' \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because (z|p)' {\in} \mathcal{P}$)
$\implies \overline{\tau}_{r_{Y_1}} {\in} \overline{\mathcal{R}}_p \wedge \exists p' {\in} \mathcal{P}^+ (p \xrightarrow{\overline{\tau}_{r_{Y_1}}} p' \wedge (z|p)' = z|p')$
(by Lemma 4.3.20, $\because p {\in} \mathcal{P} \wedge z {\in} \mathcal{P}^0 \wedge \overline{\tau}_{r_{Y_1}} {\in} \overline{\mathcal{R}}_{z|p} \wedge z|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (z|p)'$)
$\implies ((z|p)', p') = (z|p', p')$ (by algebra of binary relations)
$\implies ((z|p)', p') {\in} S^{-1}$ ($\because (z|p', p') {\in} S^{-1}$, by definition of S^{-1})
$\implies ((z|p)', p') {\in} T^{-1}$ (by set theory and definition of T^{-1}).

Now $(z|p)' = 0 \ \vee\ factors^+_m((z|p)') \subset factors^+_m(z|p)$
(by Lemma 4.3.18, $\because z|p, (z|p)' {\in} \mathcal{P} \ \wedge\ \overline{\tau}_{r_{Y_1}} {\in} \overline{\mathcal{R}}_{z|p} \ \wedge\ z|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (z|p)'$)
$\implies (z|p)' {\in} \mathcal{P}^0$ ($\because 0 {\in} \mathcal{P}^0$) $\ \vee\ factors^+_m((z|p)') \subset factors^+_m(z|p)$
$\implies factors^+_m((z|p)') \subset factors^+_m(z|p)$ ($\because (z|p)' \notin \mathcal{P}^0$)
$\implies |factors^+_m((z|p)')| < |factors^+_m(z|p)|$ (by set theory)
$\implies |factors^+_m((z|p)')| < n+1$ ($\because u = z|p \ \wedge\ |factors^+_m(u)| = n+1$)
$\implies |factors^+_m((z|p)')| \in [0, n]$ (by set theory and algebra of inequalities)
$\implies Prop(|factors^+_m((z|p)')|)$ holds (by the inductive hypothesis)
$\implies \overline{\tau}_{r_{Y_2}} {\in} \overline{\mathcal{R}}_{p'} \wedge \exists p'' {\in} \mathcal{P}\ (p' \xrightarrow{\overline{\tau}_{r_{Y_2}}} p'' \ \wedge\ ((z|p)'', p'') {\in} T^{-1})$
(by modus ponens,
$\because ((z|p)', p') {\in} T^{-1} \ \wedge\ \overline{\tau}_{r_{Y_2}} {\in} \overline{\mathcal{R}}_{(z|p)'} \ \wedge\ (z|p)'' {\in} \mathcal{P} \ \wedge\ (z|p)' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (z|p)''$)
$\implies p \xrightarrow{\overline{\tau}_{r_Y}} p'' \ \wedge\ ((z|p)'', p'') {\in} T^{-1}$ (by the *CompDelet* rule, $\because Y \sim_{of} Y_1|Y_2 \ \wedge\ p \xrightarrow{\overline{\tau}_{r_{Y_1}}} p'$)
$\implies \overline{\tau}_{r_Y} {\in} \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\ \wedge\ p'' {\in} \mathcal{P}$ (by definition of p'') $\ \wedge\ p \xrightarrow{\overline{\tau}_{r_Y}} p'' \ \wedge$
$((z|p)'', p'') {\in} T^{-1}$
$\implies \overline{\tau}_{r_Y} {\in} \overline{\mathcal{R}}_v \ \wedge\ p'' {\in} \mathcal{P} \ \wedge\ v \xrightarrow{\overline{\tau}_{r_Y}} p'' \ \wedge\ ((z|p)'', p'') {\in} T^{-1}$ ($\because v = p$).

$\therefore Prop(n+1)$ holds
($\because u = z|p \ \wedge\ v = p$, and
because $(u, v) {\in} T^{-1}$ and $\overline{\tau}_{r_Y} {\in} \overline{\mathcal{R}}_{z|p}$ and $(z|p)'' {\in} \mathcal{P}$
with $|factors^+_m(u)| = n+1$ and $z|p \xrightarrow{\overline{\tau}_{r_Y}} (z|p)''$ are arbitrary).
$\therefore \forall n {\in} \mathbb{N}\ (\forall m {\in} [0, n] Prop(m) \implies Prop(n+1))$ holds ($\because n$ is arbitrary). Q.E.D.

$\therefore \forall n \in \mathbb{N}\ Prop(n)$ holds (by complete induction).
$\therefore$ For all elements of T^{-1} the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied
($\because$ every process in $dom(T^{-1})$ has a finite number of positive singleton factors). Q.E.D.
$\therefore T^{-1}$ is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$, $\because T^{-1} \subseteq \mathcal{P}$ x $\mathcal{P}$ and for all elements of T^{-1} the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied).
$\therefore T$ is a strong dp-bisimulation on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$).
$\therefore \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p \sim_{dp} z|p)$ (by definition of $p \sim_{dp} z|p$). Q.E.D.

$\therefore \forall p \in \mathcal{P}\ \forall z \in \mathcal{P}^0 (p|z \sim_{dp} p\ \wedge\ p \sim_{dp} z|p)$. Q.E.D.

A.17 Lemma 4.3.21

$$\forall p \in \mathcal{P}\ \forall f \in factors_m^+(p)$$
$$(\exists \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p\ \exists p' \in \mathcal{P}$$
$$(p \xrightarrow{\overline{\tau}_{r_f}} p'\ \wedge\ factors_m^+(p) = factors_m^+(p') \uplus \{f\}_m))$$

Proof: uses complete induction on the depth of the factor tree of p.

For $n \in (\mathbb{N}^+ - \{1\})$, let $Prop(n)$ be this lemma for p with factor tree of depth n.
For $p \in \mathcal{P}$ and $f \in factors_m^+(p)$, $f \in factors_m(p)$ (by definition of $factors_m^+(p)$)
$\Longrightarrow factors_m(p) \neq \emptyset_m$ (by set theory)
$\Longrightarrow$ depth of the factor tree of $p \geq 2$
(by definition of $factors_m(p)$, Definition 4.3.3 and Definition 4.3.4).
Therefore, the proof by complete induction on the depth of the factor tree of p involves discharging the following two proof obligations:

1. $\vdash Prop(2)$
2. $\vdash \forall n \in (\mathbb{N}^+ - \{1\})\ (\forall m \in [2, n] Prop(m) \Longrightarrow Prop(n+1))$

Base Case: Proof of Prop(2)

For $p \in \mathcal{P}$ and $f \in factors_m^+(p)$, p has factor tree of depth 2
(by the hypothesis of *Prop*(2))
$\Longrightarrow \exists r, s \in \mathcal{P}\,(p = r|s)$ (by definition of factor tree of p)
$\Longrightarrow r, s$ have factor trees of depth 1
($\because$ p has factor tree of depth 2, and by definition of depth of factor tree)
$\Longrightarrow factors_m(r) = \emptyset_m \ \wedge \ factors_m(s) = \emptyset_m$
(by definitions of $factors_m(r)$ and $factors_m(s)$)
$\Longrightarrow factors_m(r|s) = \{r\}_m \uplus \{s\}_m$ (by definition of $factors_m(r|s)$)
$\Longrightarrow factors_m(p) = \{r, s\}_m$ ($\because$ $p = r|s$, and by multiset theory)
$\Longrightarrow factors_m^+(p) = \{g \in \{r, s\}_m \mid factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m$
(by definition of $factors_m^+(p)$)
$\Longrightarrow f = r \ \vee \ f = s$ ($\because$ $f \in factors_m^+(p)$).
Now $f \in \mathcal{P}^+$ ($\because$ $f \in factors_m^+(p)$, and by definition of $factors_m^+(p)$)
$\Longrightarrow \exists\, \overline{\tau}_{r_f} \in \overline{\mathcal{R}}\,(f \xrightarrow{\overline{\tau}_{r_f}} 0)$ (by the *Delet* rule, $\because$ $f \sim_{of} f$ (by Lemma 4.3.1))
$\Longrightarrow f|s \xrightarrow{\overline{\tau}_{r_f}} 0|s$ (by the $L - Par$ rule) $\ \wedge \ r|f \xrightarrow{\overline{\tau}_{r_f}} r|0$ (by the $R - Par$ rule).

If $f = r$
then $p = f|s$ ($\because$ $p = r|s$)
$\Longrightarrow p \xrightarrow{\overline{\tau}_{r_f}} 0|s$ ($\because$ $f|s \xrightarrow{\overline{\tau}_{r_f}} 0|s$)
$\Longrightarrow \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\ \wedge \ 0|s \in \mathcal{P}$ (by definition of $p \xrightarrow{\overline{\tau}_{r_f}} 0|s$).
Let $p' \triangleq 0|s$.
Now $factors_m^+(p) = \{g \in \{f, s\}_m \mid factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m$
($\because$ $factors_m^+(p) = \{g \in \{r, s\}_m \mid factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m \ \wedge \ f = r$)
$\Longrightarrow factors_m^+(p) = \{f\}_m \uplus \{g \in \{s\}_m \mid factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m$
($\because$ $f \in factors_m^+(p)$, and by multiset theory)
$\Longrightarrow factors_m^+(p) = \{f\}_m \uplus \{g \in \{0\}_m \uplus \{s\}_m \uplus factors_m(0) \uplus factors_m(s) \mid$
$factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m$
($\because$ $0 \notin \mathcal{P}^+$ (by production rule 1 of $\mathcal{P}^0$ and Theorem 4.3.2) $\wedge$ $factors_m(0) = \emptyset_m$ (by definition of $factors_m(0)$) $\wedge$ $factors_m(s) = \emptyset_m$)
$\Longrightarrow factors_m^+(p) = \{f\}_m \uplus \{g \in factors_m(0|s) \mid factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m$
(by definition of $factors_m(0|s)$)
$\Longrightarrow factors_m^+(p) = \{f\}_m \uplus factors_m^+(0|s)$ (by definition of $factors_m^+(0|s)$)
$\Longrightarrow factors_m^+(p) = factors_m^+(p') \uplus \{f\}_m$
($\because$ $\uplus$ is commutative, and by definition of p')

$\implies \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p \ \wedge\ p' \in \mathcal{P} \ \wedge\ p \xrightarrow{\overline{\tau}_{r_f}} p' \ \wedge\ factors^+_m(p) = factors^+_m(p') \uplus \{f\}_m$
$(\because \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p \ \wedge\ p' = 0|s \ \wedge\ 0|s \in \mathcal{P} \ \wedge\ p \xrightarrow{\overline{\tau}_{r_f}} 0|s).$

If $f = s$
then $p = r|f$ $(\because p = r|s)$
$\implies p \xrightarrow{\overline{\tau}_{r_f}} r|0$ $(\because r|f \xrightarrow{\overline{\tau}_{r_f}} r|0)$
$\implies \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\ \wedge\ $ $r|0 \in \mathcal{P}$ (by definition of $p \xrightarrow{\overline{\tau}_{r_f}} r|0$).
Let $p' \triangleq r|0$.
Now $factors^+_m(p) = \{g \in \{r, f\}_m \mid factors_m(g) = \emptyset_m \ \wedge\ g \in \mathcal{P}^+\}_m$
$(\because factors^+_m(p) = \{g \in \{r, s\}_m \mid factors_m(g) = \emptyset_m \ \wedge\ g \in \mathcal{P}^+\}_m \ \wedge\ f = s)$
$\implies factors^+_m(p) = \{f\}_m \uplus \{g \in \{r\}_m \mid factors_m(g) = \emptyset_m \ \wedge\ g \in \mathcal{P}^+\}_m$
$(\because f \in factors^+_m(p)$, and by multiset theory)
$\implies factors^+_m(p) = \{f\}_m \uplus \{g \in \{r\}_m \uplus \{0\}_m \uplus factors_m(r) \uplus factors_m(0) \mid$
$factors_m(g) = \emptyset_m \ \wedge\ g \in \mathcal{P}^+\}_m$
$(\because 0 \notin \mathcal{P}^+$ (by production rule 1 of $\mathcal{P}^0$ and Theorem 4.3.2) $\wedge\ factors_m(r) = \emptyset_m \ \wedge$
$factors_m(0) = \emptyset_m$ (by definition of $factors_m(0)$))
$\implies factors^+_m(p) = \{f\}_m \uplus \{g \in factors_m(r|0) \mid factors_m(g) = \emptyset_m \ \wedge\ g \in \mathcal{P}^+\}_m$
(by definition of $factors_m(r|0)$)
$\implies factors^+_m(p) = \{f\}_m \uplus factors^+_m(r|0)$ (by definition of $factors^+_m(r|0)$)
$\implies factors^+_m(p) = factors^+_m(p') \uplus \{f\}_m$
$(\because \uplus$ is commutative, and by definition of $p')$
$\implies \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p \ \wedge\ p' \in \mathcal{P} \ \wedge\ p \xrightarrow{\overline{\tau}_{r_f}} p' \ \wedge\ factors^+_m(p) = factors^+_m(p') \uplus \{f\}_m$
$(\because \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p \ \wedge\ p' = r|0 \ \wedge\ r|0 \in \mathcal{P} \ \wedge\ p \xrightarrow{\overline{\tau}_{r_f}} r|0).$

$\therefore \forall p \in \mathcal{P}\ \forall f \in factors^+_m(p)$
$(\exists \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p\ \exists p' \in \mathcal{P}\ (p \xrightarrow{\overline{\tau}_{r_f}} p' \ \wedge\ factors^+_m(p) = factors^+_m(p') \uplus \{f\}_m))$
for p with factor tree of depth 2
$(\because p \in \mathcal{P}$ with factor tree of depth 2 and $f \in factors^+_m(p)$ are arbitrary)
$\implies Prop(2)$ holds (by definition of $Prop(2)$). Q.E.D.

Induction Step:

Proof of $\forall \mathbf{n} \in (\mathbb{N}^+ - \{\mathbf{1}\})\ (\forall \mathbf{m} \in \mathbf{[2, n]Prop(m)} \implies \mathbf{Prop(n+1)})$

For $n \in (\mathbb{N}^+ - \{1\})$, assume $\forall m \in [2, n] Prop(m)$ holds (inductive hypothesis).
For $p \in \mathcal{P}$ and $f \in factors^+_m(p)$, p has factor tree of depth $n + 1$

(by the hypothesis of $Prop(n+1)$)
$\implies p$ has factor tree of depth ≥ 3 ($\because n+1 \geq 3$ ($\because n \in (\mathbb{N}^+ - \{1\})$))
$\implies \exists r, s \in \mathcal{P}\,(p = r|s)$ (by definition of factor tree of p)
$\implies r$ has factor tree of depth d_r, with $d_r \in [1, n]$ $\wedge$
s has factor tree of depth d_s, with $d_s \in [1, n]$
($\because p$ has factor tree of depth $n+1$, and by definition of depth of factor tree).
Now $factors_m(p) = factors_m(r|s)$ ($\because p = r|s$)
$\implies factors_m(p) = \{r\}_m \uplus \{s\}_m \uplus factors_m(r) \uplus factors_m(s)$
(by definition of $factors_m(r|s)$)
$\implies factors^+_m(p) = \{g \in \{r\}_m \uplus \{s\}_m \uplus factors_m(r) \uplus factors_m(s) \mid$
$factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
(by definition of $factors^+_m(p)$)
$\implies f \in \{g \in \{r\}_m \uplus \{s\}_m \uplus factors_m(r) \uplus factors_m(s) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
($\because f \in factors^+_m(p)$)
$\implies f \in \{r\}_m \vee f \in \{s\}_m \vee f \in factors_m(r) \vee f \in factors_m(s)$ (by multiset theory)
$\implies f = r \vee f = s \vee f \in factors_m(r) \vee f \in factors_m(s)$ (by set theory).
Now $factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+$
($\because f \in factors^+_m(p)$, and by definition of $factors^+_m(p)$)
$\implies \exists\, \overline{\tau}_{r_f} \in \overline{\mathcal{R}}\,(f \xrightarrow{\overline{\tau}_{r_f}} 0)$ (by the *Delet* rule, $\because f \sim_{of} f$ (by Lemma 4.3.1))
$\implies f|s \xrightarrow{\overline{\tau}_{r_f}} 0|s$ (by the $L - Par$ rule) $\wedge$ $r|f \xrightarrow{\overline{\tau}_{r_f}} r|0$ (by the $R - Par$ rule).

If $f = r$
then $p = f|s$ ($\because p = r|s$)
$\implies p \xrightarrow{\overline{\tau}_{r_f}} 0|s$ ($\because f|s \xrightarrow{\overline{\tau}_{r_f}} 0|s$)
$\implies \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\wedge$ $0|s \in \mathcal{P}$ (by definition of $p \xrightarrow{\overline{\tau}_{r_f}} 0|s$).
Let $p' \triangleq 0|s$.
Now $factors^+_m(p) = \{g \in \{f\}_m \uplus \{s\}_m \uplus factors_m(f) \uplus factors_m(s) \mid$
$factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
($\because factors^+_m(p) = \{g \in \{r\}_m \uplus \{s\}_m \uplus factors_m(r) \uplus factors_m(s) \mid$
$factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m \wedge f = r$)
$\implies factors^+_m(p) = \{f\}_m \uplus \{g \in \{s\}_m \uplus factors_m(s) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
($\because f \in factors^+_m(p) \wedge factors_m(f) = \emptyset_m$, and by multiset theory)
$\implies factors^+_m(p) = \{f\}_m \uplus \{g \in \{0\}_m \uplus \{s\}_m \uplus factors_m(0) \uplus factors_m(s) \mid$
$factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
($\because 0 \notin \mathcal{P}^+$ (by production rule 1 of $\mathcal{P}^0$ and Theorem 4.3.2) $\wedge$ $factors_m(0) = \emptyset_m$ (by definition of $factors_m(0)$))

$\implies factors_m^+(p) = \{f\}_m \uplus \{g \in factors_m(0|s) \mid factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m$
(by definition of $factors_m(0|s)$)
$\implies factors_m^+(p) = \{f\}_m \uplus factors_m^+(0|s)$ (by definition of $factors_m^+(0|s)$)
$\implies factors_m^+(p) = factors_m^+(p') \uplus \{f\}_m$
($\because \uplus$ is commutative, and by definition of p')
$\implies \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p \ \wedge \ p' \in \mathcal{P} \ \wedge \ p \xrightarrow{\overline{\tau}_{r_f}} p' \ \wedge \ factors_m^+(p) = factors_m^+(p') \uplus \{f\}_m$
($\because \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p \ \wedge \ p' = 0|s \ \wedge \ 0|s \in \mathcal{P} \ \wedge \ p \xrightarrow{\overline{\tau}_{r_f}} 0|s$).

If $f = s$
then $p = r|f$ ($\because p = r|s$)
$\implies p \xrightarrow{\overline{\tau}_{r_f}} r|0$ ($\because r|f \xrightarrow{\overline{\tau}_{r_f}} r|0$)
$\implies \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\ \wedge \ r|0 \in \mathcal{P}$ (by definition of $p \xrightarrow{\overline{\tau}_{r_f}} r|0$).
Let $p' \triangleq r|0$.
Now $factors_m^+(p) = \{g \in \{r\}_m \uplus \{f\}_m \uplus factors_m(r) \uplus factors_m(f) \mid$
$factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m$
($\because factors_m^+(p) = \{g \in \{r\}_m \uplus \{s\}_m \uplus factors_m(r) \uplus factors_m(s) \mid$
$factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m \ \wedge \ f = s$)
$\implies factors_m^+(p) = \{f\}_m \uplus \{g \in \{r\}_m \uplus factors_m(r) \mid factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m$
($\because f \in factors_m^+(p) \ \wedge \ factors_m(f) = \emptyset_m$, and by multiset theory)
$\implies factors_m^+(p) = \{f\}_m \uplus \{g \in \{r\}_m \uplus \{0\}_m \uplus factors_m(r) \uplus factors_m(0) \mid$
$factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m$
($\because 0 \notin \mathcal{P}^+$ (by production rule 1 of $\mathcal{P}^0$ and Theorem 4.3.2) $\ \wedge \ factors_m(0) = \emptyset_m$ (by definition of $factors_m(0)$))
$\implies factors_m^+(p) = \{f\}_m \uplus \{g \in factors_m(r|0) \mid factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m$
(by definition of $factors_m(r|0)$)
$\implies factors_m^+(p) = \{f\}_m \uplus factors_m^+(r|0)$ (by definition of $factors_m^+(r|0)$)
$\implies factors_m^+(p) = factors_m^+(p') \uplus \{f\}_m$
($\because \uplus$ is commutative, and by definition of p')
$\implies \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p \ \wedge \ p' \in \mathcal{P} \ \wedge \ p \xrightarrow{\overline{\tau}_{r_f}} p' \ \wedge \ factors_m^+(p) = factors_m^+(p') \uplus \{f\}_m$
($\because \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p \ \wedge \ p' = r|0 \ \wedge \ r|0 \in \mathcal{P} \ \wedge \ p \xrightarrow{\overline{\tau}_{r_f}} r|0$).

If $f \in factors_m(r)$
then $factors_m(r) \neq \emptyset_m$ (by set theory) $\ \wedge$
$f \in factors_m^+(r)$ (by definition of $factors_m^+(r)$, $\because factors_m(f) = \emptyset_m \ \wedge \ f \in \mathcal{P}^+$)
$\implies \exists u, v \in \mathcal{P}\,(r = u|v)$ (by definition of $factors_m(r)$)

$\implies r$ has factor tree of depth ≥ 2 (by definition of depth of factor tree of r)
$\implies d_r \in [2, n]$ ($\because d_r \in [1, n]$, and by definition of d_r)
$\implies Prop(d_r)$ holds (by the inductive hypothesis)
$\implies \exists \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_r \; \exists r' \in \mathcal{P} \, (r \xrightarrow{\overline{\tau}_{r_f}} r' \; \wedge \; factors^+_m(r) = factors^+_m(r') \uplus \{f\}_m)$
(by modus ponens, $\because r \in \mathcal{P} \; \wedge \; f \in factors^+_m(r)$)
$\implies r|s \xrightarrow{\overline{\tau}_{r_f}} r'|s$ (by the $L - Par$ rule) $\;\wedge$
$(r' = 0 \; \vee \; factors_m(r') \neq \emptyset_m)$ (by Lemma 4.3.17, $\because r \in \mathcal{P}$)
$\implies p \xrightarrow{\overline{\tau}_{r_f}} r'|s$ ($\because p = r|s$) $\;\wedge$
$(r' \in \mathcal{P}^0$ (by production rule 1 of $\mathcal{P}^0$) $\;\vee\;$ $factors_m(r') \neq \emptyset_m)$
$\implies \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\;\wedge\;$ $r'|s \in \mathcal{P}$ (by definition of $p \xrightarrow{\overline{\tau}_{r_f}} r'|s$) $\;\wedge$
$(r' \notin \mathcal{P}^+$ (by Theorem 4.3.2, $\because r' \in \mathcal{P}$) $\;\vee\;$ $factors_m(r') \neq \emptyset_m)$.

Let $p' \triangleq r'|s$.
Now $factors^+_m(p) = factors^+_m(r|s)$ ($\because p = r|s$)
$\implies factors^+_m(p) = \{g \in factors_m(r|s) \mid factors_m(g) = \emptyset_m \; \wedge \; g \in \mathcal{P}^+\}_m$
(by definition of $factors^+_m(r|s)$)
$\implies factors^+_m(p) = \{g \in \{r\}_m \uplus \{s\}_m \uplus factors_m(r) \uplus factors_m(s) \mid$
$factors_m(g) = \emptyset_m \; \wedge \; g \in \mathcal{P}^+\}_m$
(by definition of $factors_m(r|s)$)
$\implies factors^+_m(p) = \{g \in \{s\}_m \uplus factors_m(s) \mid$
$factors_m(g) = \emptyset_m \; \wedge \; g \in \mathcal{P}^+\}_m \uplus factors^+_m(r)$
($\because factors_m(r) \neq \emptyset_m$, and by multiset theory and definition of $factors^+_m(r)$)
$\implies factors^+_m(p) = \{g \in \{s\}_m \uplus factors_m(s) \mid$
$factors_m(g) = \emptyset_m \; \wedge \; g \in \mathcal{P}^+\}_m \uplus factors^+_m(r') \uplus \{f\}_m$
($\because factors^+_m(r) = factors^+_m(r') \uplus \{f\}_m$)
$\implies factors^+_m(p) = \{g \in \{s\}_m \uplus factors_m(r') \uplus factors_m(s) \mid$
$factors_m(g) = \emptyset_m \; \wedge \; g \in \mathcal{P}^+\}_m \uplus \{f\}_m$
(by definition of $factors^+_m(r')$ and by multiset theory)
$\implies factors^+_m(p) = \{g \in \{r'\}_m \uplus \{s\}_m \uplus factors_m(r') \uplus factors_m(s) \mid$
$factors_m(g) = \emptyset_m \; \wedge \; g \in \mathcal{P}^+\}_m \uplus \{f\}_m$
($\because r' \notin \mathcal{P}^+ \; \vee \; factors_m(r') \neq \emptyset_m$)
$\implies factors^+_m(p) = factors^+_m(r'|s) \uplus \{f\}_m$ (by definition of $factors^+_m(r'|s)$)
$\implies factors^+_m(p) = factors^+_m(p') \uplus \{f\}_m$ ($\because p' = r'|s$)
$\implies \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p \; \wedge \; p' \in \mathcal{P} \; \wedge \; p \xrightarrow{\overline{\tau}_{r_f}} p' \; \wedge \; factors^+_m(p) = factors^+_m(p') \uplus \{f\}_m$
($\because \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p \; \wedge \; p' = r'|s \; \wedge \; r'|s \in \mathcal{P} \; \wedge \; p \xrightarrow{\overline{\tau}_{r_f}} r'|s$).

If $f \in factors_m(s)$
then $factors_m(s) \neq \emptyset_m$ (by set theory) $\wedge$
$f \in factors^+_m(s)$ (by definition of $factors^+_m(s)$, $\because factors_m(f) = \emptyset_m \wedge f \in \mathcal{P}^+$)
$\implies \exists u, v \in \mathcal{P}\, (s = u|v)$ (by definition of $factors_m(s)$)
$\implies s$ has factor tree of depth ≥ 2 (by definition of depth of factor tree of s)
$\implies d_s \in [2, n]$ ($\because d_s \in [1, n]$, and by definition of d_s)
$\implies Prop(d_s)$ holds (by the inductive hypothesis)
$\implies \exists \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_s \; \exists s' \in \mathcal{P}\, (s \xrightarrow{\overline{\tau}_{r_f}} s' \wedge factors^+_m(s) = factors^+_m(s') \uplus \{f\}_m)$
(by modus ponens, $\because s \in \mathcal{P} \wedge f \in factors^+_m(s)$)
$\implies r|s \xrightarrow{\overline{\tau}_{r_f}} r|s'$ (by the $R - Par$ rule) $\wedge$
$(s' = 0 \vee factors_m(s') \neq \emptyset_m)$ (by Lemma 4.3.17, $\because s \in \mathcal{P}$)
$\implies p \xrightarrow{\overline{\tau}_{r_f}} r|s'$ ($\because p = r|s$) $\wedge$
($s' \in \mathcal{P}^0$ (by production rule 1 of $\mathcal{P}^0$) $\vee$ $factors_m(s') \neq \emptyset_m$)
$\implies \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\wedge$ $r|s' \in \mathcal{P}$ (by definition of $p \xrightarrow{\overline{\tau}_{r_f}} r|s'$) $\wedge$
($s' \notin \mathcal{P}^+$ (by Theorem 4.3.2, $\because s' \in \mathcal{P}$) $\vee$ $factors_m(s') \neq \emptyset_m$).
Let $p' \triangleq r|s'$.
Now $factors^+_m(p) = factors^+_m(r|s)$ ($\because p = r|s$)
$\implies factors^+_m(p) = \{g \in factors_m(r|s) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
(by definition of $factors^+_m(r|s)$)
$\implies factors^+_m(p) = \{g \in \{r\}_m \uplus \{s\}_m \uplus factors_m(r) \uplus factors_m(s) \mid$
$factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m$
(by definition of $factors_m(r|s)$)
$\implies factors^+_m(p) = \{g \in \{r\}_m \uplus factors_m(r) \mid factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m \uplus factors^+_m(s)$
($\because factors_m(s) \neq \emptyset_m$, and by multiset theory and definition of $factors^+_m(s)$)
$\implies factors^+_m(p) = \{g \in \{r\}_m \uplus factors_m(r) \mid$
$factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m \uplus factors^+_m(s') \uplus \{f\}_m$
($\because factors^+_m(s) = factors^+_m(s') \uplus \{f\}_m$)
$\implies factors^+_m(p) = \{g \in \{r\}_m \uplus factors_m(r) \uplus factors_m(s') \mid$
$factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m \uplus \{f\}_m$
(by definition of $factors^+_m(s')$ and by multiset theory)
$\implies factors^+_m(p) = \{g \in \{r\}_m \uplus \{s'\}_m \uplus factors_m(r) \uplus factors_m(s') \mid$
$factors_m(g) = \emptyset_m \wedge g \in \mathcal{P}^+\}_m \uplus \{f\}_m$
($\because s' \notin \mathcal{P}^+ \vee factors_m(s') \neq \emptyset_m$)
$\implies factors^+_m(p) = factors^+_m(r|s') \uplus \{f\}_m$ (by definition of $factors^+_m(r|s')$)
$\implies factors^+_m(p) = factors^+_m(p') \uplus \{f\}_m$ ($\because p' = r|s'$)
$\implies \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_p \wedge p' \in \mathcal{P} \wedge p \xrightarrow{\overline{\tau}_{r_f}} p' \wedge factors^+_m(p) = factors^+_m(p') \uplus \{f\}_m$

$(\because \overline{\tau}_{r_f}\in\overline{\mathcal{R}}_p \ \wedge\ p' = r|s' \ \wedge\ r|s'\in\mathcal{P} \ \wedge\ p \xrightarrow{\overline{\tau}_{r_f}} r|s')$.

$\therefore \forall p\in\mathcal{P}\ \forall f\in factors_m^+(p)$
$(\exists\overline{\tau}_{r_f}\in\overline{\mathcal{R}}_p\ \exists p'\in\mathcal{P}\,(p \xrightarrow{\overline{\tau}_{r_f}} p' \ \wedge\ factors_m^+(p) = factors_m^+(p')\uplus\{f\}_m))$
for p with factor tree of depth $n+1$
$(\because p\in\mathcal{P}$ with factor tree of depth $n+1$ and $f\in factors_m^+(p)$ are arbitrary)
$\implies Prop(n+1)$ holds (by definition of $Prop(n+1)$).
$\therefore \forall n\in(\mathbb{N}^+-\{1\})\ (\forall m\in[2,n]Prop(m) \implies Prop(n+1))$ holds
$(\because n\in(\mathbb{N}^+-\{1\})$ is arbitrary) Q.E.D.

$\therefore \forall n\in(\mathbb{N}^+-\{1\})\ Prop(n)$ holds (by complete induction).
$\therefore \forall p\in\mathcal{P}\ \forall f\in factors_m^+(p)$
$(\exists\overline{\tau}_{r_f}\in\overline{\mathcal{R}}_p\ \exists p'\in\mathcal{P}\,(p \xrightarrow{\overline{\tau}_{r_f}} p' \ \wedge\ factors_m^+(p) = factors_m^+(p')\uplus\{f\}_m))$
$(\because$ for every $p\in\mathcal{P}$, the depth of the factor tree of p is finite). Q.E.D.

A.18 Theorem 4.3.8 $\forall p,q\in\mathcal{P}$

$$(p\sim_{dp} q \implies |factors_m^+(p|0)| = |factors_m^+(q|0)|)$$

Proof: assumes the theorem is *false* and uses a process with the minimum number of positive singleton factors to produce a contradiction.

The theorem is *true* $\vee$ the theorem is *false*
(by definition of $\vee$, and because the logic used is 2-valued).

Let $S \triangleq \{(p,q)\in\mathcal{P} \mid p\sim_{dp} q \ \wedge\ |factors_m^+(p|0)| \neq |factors_m^+(q|0)|\}$.

If the theorem is *false*
then $\exists p,q\in\mathcal{P}\,(p\sim_{dp} q \ \wedge\ |factors_m^+(p|0)| \neq |factors_m^+(q|0)|)$
(by definition of $\neg$ and by set theory)
$\implies S\neq\emptyset$ (by definition of S)
$\implies \exists(r,s)\in S\ \forall(p,q)\in S$
$(|factors_m^+(s|0)| \leq |factors_m^+(p|0)| \ \wedge\ |factors_m^+(s|0)| \leq |factors_m^+(q|0)|)$
$(\because$ for every $p,q\in\mathcal{P}$, the depths of the factor trees of p,q are in $\mathbb{N}^+$, $|$ is a binary operator, and $\sim_{dp}$ is symmetric on $\mathcal{P}$).
Now $r,s\in\mathcal{P} \ \wedge\ r\sim_{dp} s \ \wedge\ |factors_m^+(r|0)| \neq |factors_m^+(s|0)|$ $(\because (r,s)\in S)$
$\implies r|0, s|0\in\mathcal{P}$ $(\because 0\in\mathcal{P})$ $\wedge$ $r\sim_{of} s$ (by Corollary 4.3.5)

$\implies s|0 \in \mathcal{P}^+ \ \vee \ s|0 \in \mathcal{P}^0$ (by Theorem 4.3.2).

If $s|0 \in \mathcal{P}^0$
then $s \in \mathcal{P}^0$ (by the hypothesis of production rule 3 of $\mathcal{P}^0$) $\ \wedge$
$factors^+_m(s|0) = \emptyset_m$ (by Lemma 4.3.16, $\because s|0 \in \mathcal{P}$)
$\implies s \notin \mathcal{P}^+$ (by Theorem 4.3.2) $\ \wedge \ |factors^+_m(s|0)| = 0$ (by multiset theory)
$\implies r, s \in \mathcal{P}^0$ (by Lemma 4.3.9, $\because r, s \in \mathcal{P} \ \wedge \ r \sim_{of} s$)
$\implies r|0 \in \mathcal{P}^0$ (by production rule 3 of $\mathcal{P}^0$, $\because 0 \in \mathcal{P}^0$)
$\implies factors^+_m(r|0) = \emptyset_m$ (by Lemma 4.3.16, $\because r|0 \in \mathcal{P}$)
$\implies |factors^+_m(r|0)| = 0$ (by multiset theory)
$\implies |factors^+_m(r|0)| = |factors^+_m(s|0)|$
($\because |factors^+_m(s|0)| = 0$; which is a contradiction).

$\therefore s|0 \notin \mathcal{P}^0$
$\implies s|0 \in \mathcal{P}^+$ (by Theorem 4.3.2, $\because s|0 \in \mathcal{P}$)
$\implies factors^+_m(s|0) \neq \emptyset_m$ (by Lemma 4.3.15, $\because s, 0 \in \mathcal{P}$)
$\implies 0 < |factors^+_m(s|0)|$ (by multiset theory)
$\implies 0 < |factors^+_m(s|0)| \ \wedge$
$|factors^+_m(s|0)| < |factors^+_m(r|0)|$ (by definitions of (r, s) and S)
$\implies 0 < |factors^+_m(r|0)|$ ($\because <$ is transitive)
$\implies \exists f \in factors^+_m(r|0)$ (by set theory).
Now $r \sim_{dp} s$ ($\because (r, s) \in S$)
$\implies r|0 \sim_{dp} r$ (by Theorem 4.3.7, $\because r \in \mathcal{P} \ \wedge \ 0 \in \mathcal{P}^0$) $\ \wedge \ r \sim_{dp} s \ \wedge$
$s|0 \sim_{dp} s$ (by Theorem 4.3.7, $\because s \in \mathcal{P} \ \wedge \ 0 \in \mathcal{P}^0$)
$\implies r|0 \sim_{dp} s$ ($\because \sim_{dp}$ is transitive on $\mathcal{P}$, by Lemma 4.3.14) $\ \wedge$
$s \sim_{dp} s|0$ ($\because \sim_{dp}$ is symmetric on $\mathcal{P}$, by Lemma 4.3.12)
$\implies r|0 \sim_{dp} s|0$ ($\because \sim_{dp}$ is transitive on $\mathcal{P}$, by Lemma 4.3.14).
Now $\exists \overline{\tau}_{r_f} \in \overline{\mathcal{R}}_{r|0} \ \exists r_l \in \mathcal{P} \ (r|0 \xrightarrow{\overline{\tau}_{r_f}} r_l \ \wedge \ factors^+_m(r|0) = factors^+_m(r_l) \uplus \{f\}_m)$
(by Lemma 4.3.21, $\because r|0 \in \mathcal{P} \ \wedge \ f \in factors^+_m(r|0)$)
$\implies (r_l = 0 \ \vee \ factors_m(r_l) \neq \emptyset_m)$ (by Lemma 4.3.17, $\because r|0 \in \mathcal{P}$) $\ \wedge$
$\overline{\tau}_{r_f} \in \overline{\mathcal{R}}_{s|0} \ \wedge \ \exists s_l \in \mathcal{P} \ (s|0 \xrightarrow{\overline{\tau}_{r_f}} s_l \ \wedge \ r_l \sim_{dp} s_l)$ ($\because r|0 \sim_{dp} s|0$) $\ \wedge$
$|factors^+_m(r|0)| = |factors^+_m(r_l)| + 1$ (by multiset theory)
$\implies (r_l = 0 \ \vee \ factors_m(r_l) \neq \emptyset_m) \ \wedge$
$(s_l = 0 \ \vee \ factors_m(s_l) \neq \emptyset_m)$ (by Lemma 4.3.17, $\because s|0 \in \mathcal{P}$) $\ \wedge$
$(s_l = 0 \ \vee \ factors^+_m(s_l) \subset factors^+_m(s|0))$ (by Lemma 4.3.18, $\because s|0 \in \mathcal{P}$) $\ \wedge$
$|factors^+_m(r|0)| = |factors^+_m(r_l)| + 1$
$\implies |factors^+_m(s_l)| < |factors^+_m(s|0)|$

($\because |factors^+_m(0)| = 0 \ \wedge \ 0 < |factors^+_m(s|0)|$ and by multiset theory) $\wedge$
$|factors^+_m(s|0)| < |factors^+_m(r|0)|$ ($\because |factors^+_m(s|0)| < |factors^+_m(r|0)|$) $\wedge$
$|factors^+_m(r|0)| = |factors^+_m(r_l)| + 1$
$\implies |factors^+_m(s_l)| < |factors^+_m(s|0)| \ \wedge \ |factors^+_m(s|0)| \leq |factors^+_m(r_l)|$
(by algebra of inequalities on $\mathbb{Z}$)
$\implies |factors^+_m(s_l)| < |factors^+_m(r_l)|$ (by algebra of inequalities)
$\implies |factors^+_m(s_l)| \neq |factors^+_m(r_l)|$ (by algebra of inequalities).

Now suppose $s_l = 0 \ \vee \ r_l = 0$
then $s_l \in \mathcal{P}^0 \ \vee \ r_l \in \mathcal{P}^0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$)
$\implies s_l \notin \mathcal{P}^+ \ \vee \ r_l \notin \mathcal{P}^+$ (by Theorem 4.3.2, $\because r_l, s_l \in \mathcal{P}$) $\wedge$
$r_l \sim_{of} s_l$ (by Corollary 4.3.5, $\because r_l, s_l \in \mathcal{P} \ \wedge \ r_l \sim_{dp} s_l$)
$\implies s_l \in \mathcal{P}^0 \ \wedge \ r_l \in \mathcal{P}^0$ (by Lemma 4.3.9, $\because r_l, s_l \in \mathcal{P}$)
$\implies factors^+_m(s_l) = \emptyset_m \ \wedge \ factors^+_m(r_l) = \emptyset_m$ (by Lemma 4.3.16, $\because r_l, s_l \in \mathcal{P}$)
$\implies |factors^+_m(s_l)| = 0 \ \wedge \ |factors^+_m(r_l)| = 0$ (by set theory)
$\implies |factors^+_m(s_l)| = |factors^+_m(r_l)|$
($\because =$ is symmetic and transitive; which is a contradiction).

$\therefore s_l \neq 0 \ \wedge \ r_l \neq 0$ (by definitions of $\neg$ and $\vee$)
$\implies factors_m(s_l) \neq \emptyset_m$ ($\because s_l = 0 \ \vee \ factors_m(s_l) \neq \emptyset_m$) $\wedge$
$factors_m(r_l) \neq \emptyset_m$ ($\because r_l = 0 \ \vee \ factors_m(r_l) \neq \emptyset_m$)
$\implies \{g \in \{s_l\}_m \uplus factors_m(s_l) \mid factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m = factors^+_m(s_l)$
(by definition of $factors^+_m(s_l)$) $\wedge$
$\{h \in \{r_l\}_m \uplus factors_m(r_l) \mid factors_m(h) = \emptyset_m \ \wedge \ h \in \mathcal{P}^+\}_m = factors^+_m(r_l)$
(by definition of $factors^+_m(r_l)$)
$\implies \{g \in \{s_l\}_m \uplus \{0\}_m \uplus factors_m(s_l) \uplus factors_m(0) \mid factors_m(g) = \emptyset_m \ \wedge \ g \in \mathcal{P}^+\}_m = factors^+_m(s_l)$ $\wedge$
$\{h \in \{r_l\}_m \uplus \{0\}_m \uplus factors_m(r_l) \uplus factors_m(0) \mid factors_m(h) = \emptyset_m \ \wedge \ h \in \mathcal{P}^+\}_m = factors^+_m(r_l)$
($\because 0 \notin \mathcal{P}^+$ (by production rule 1 of $\mathcal{P}^0$ and Theorem 4.3.2) $\wedge$ $factors_m(0) = \emptyset_m$ (by definition of $factors_m(0)$))
$\implies factors^+_m(s_l|0) = factors^+_m(s_l)$ (by definition of $factors^+_m(s_l|0)$) $\wedge$
$factors^+_m(r_l|0) = factors^+_m(r_l)$ (by definition of $factors^+_m(r_l|0)$)
$\implies |factors^+_m(s_l|0)| = |factors^+_m(s_l)| \ \wedge \ |factors^+_m(r_l|0)| = |factors^+_m(r_l)|$
(by multiset theory)
$\implies |factors^+_m(s_l|0)| \neq |factors^+_m(r_l|0)|$ ($\because |factors^+_m(s_l)| \neq |factors^+_m(r_l)|$)
$\implies (r_l, s_l) \in S$ ($\because r_l, s_l \in \mathcal{P} \ \wedge \ r_l \sim_{dp} s_l$)
$\implies |factors^+_m(s|0)| \leq |factors^+_m(s_l|0)|$ (by definition of (r, s))

$\implies |factors^{+}_{m}(s_l|0)| \not< |factors^{+}_{m}(s|0)|$ (by algebra of inequalities).
But $|factors^{+}_{m}(s_l|0)| < |factors^{+}_{m}(s|0)|$
($\because |factors^{+}_{m}(s_l|0)| = |factors^{+}_{m}(s_l)| \ \wedge \ |factors^{+}_{m}(s_l)| < |factors^{+}_{m}(s|0)|$; which is a contradiction).

$\therefore$ The theorem is not *false*
$\implies$ the theorem is *true* ($\because$ the theorem is *true* $\vee$ the theorem is *false*). Q.E.D.

A.19 Theorem 4.3.9 $\forall p,q \in \mathcal{P}$

$$(p \sim_{dp} q \implies \forall r \in \mathcal{P}\,(p|r \sim_{of} q|r) \ \wedge \ \forall r \in \mathcal{P}\,(r|p \sim_{of} r|q))$$

Proof: consists of discharging the following two proof obligations. Each proof obligation is discharged by defining a binary relation S on $\mathcal{P}$ which contains the pair of processes that are required to be strongly of-bisimilar, proving S is a strong of-simulation on $\mathcal{P}$, then proving S is a strong of-bisimulation on $\mathcal{P}$.

1. $\vdash \forall p,q \in \mathcal{P}\,(p \sim_{dp} q \implies \forall r \in \mathcal{P}\,(p|r \sim_{of} q|r))$
2. $\vdash \forall p,q \in \mathcal{P}\,(p \sim_{dp} q \implies \forall r \in \mathcal{P}\,(r|p \sim_{of} r|q))$

A.19.1 $\forall p,q \in \mathcal{P}\,(p \sim_{dp} q \implies \forall r \in \mathcal{P}\,(p|r \sim_{of} q|r))$

Proof: If $\exists$ strong of-bisimulation S on $\mathcal{P}$ with $\forall r \in \mathcal{P}\ ((p|r, q|r) \in S)$ for any processes p,q in $\mathcal{P}$ such that $p \sim_{dp} q$
then $\forall p,q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall r \in \mathcal{P}\,(p|r \sim_{of} q|r)$) (by definition of $p|r \sim_{of} q|r$).
Therefore, we find such an S.

Let $S \triangleq \{(p|r, q|r) \mid p,q,r \in \mathcal{P} \ \wedge \ p \sim_{dp} q\}$.

A.19.1.1 S is a strong of-simulation on $\mathcal{P}$

$S \subseteq \mathcal{P}\ \mathrm{x}\ \mathcal{P} \ \wedge \ \forall p,q,r \in \mathcal{P}\,(p \sim_{dp} q \implies (p|r, q|r) \in S)$ (by definition of S).

We prove for $(p|r, q|r)$ in S that the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied.

$(\mathbf{p}|\mathbf{r},\mathbf{q}|\mathbf{r})$ satisfies the Observation and Fraction conditions

The transitions of $p|r$ in $\mathcal{I}\cup\mathcal{R}$ are defined by the $L-Par$, $R-Par$, $React$, $L-React$ and $R-React$ rules only (by the syntax of $p|r$ and definitions of the LTS rules):

If the $L-Par$ rule defines a transition $p|r \xrightarrow{\delta} (p|r)'$ with $\delta\in\mathcal{I}\cup\mathcal{R}$
then $(p|r)'\in\mathcal{P}$ (by definition of $p|r \xrightarrow{\delta} (p|r)'$) $\wedge$ $(p|r)' = p'|r$ (by the $L-Par$ rule) $\wedge$
$p \xrightarrow{\delta} p'$ (by the hypothesis of $L-Par$)
$\Longrightarrow q \xrightarrow{\delta} q'\ \wedge\ p'\sim_{dp} q'$ ($\because p\sim_{dp} q$)
$\Longrightarrow q|r \xrightarrow{\delta} q'|r$ (by the $L-Par$ rule) $\wedge$
$(p'|r, q'|r)\in S$ (by definition of S, $\because p', q', r\in\mathcal{P}$).

If the $R-Par$ rule defines a transition $p|r \xrightarrow{\delta} (p|r)'$ with $\delta\in\mathcal{I}\cup\mathcal{R}$
then $(p|r)'\in\mathcal{P}$ (by definition of $p|r \xrightarrow{\delta} (p|r)'$) $\wedge$ $(p|r)' = p|r'$ (by the $R-Par$ rule) $\wedge$
$r \xrightarrow{\delta} r'$ (by the hypothesis of $R-Par$)
$\Longrightarrow q|r \xrightarrow{\delta} q|r'$ (by the $R-Par$ rule) $\wedge$
$(p|r', q|r')\in S$ (by definition of S, $\because p, q, r'\in\mathcal{P}\ \wedge\ p\sim_{dp} q$).

If the $React$ rule defines a transition $p|r \xrightarrow{\tau} (p|r)'$
then $(p|r)'\in\mathcal{P}$ (by definition of $p|r \xrightarrow{\tau} (p|r)'$) $\wedge$ $(p|r)' = p'|r'$ (by the $React$ rule) $\wedge$
$\exists\lambda\in\mathcal{L}\cup\mathcal{C}\ (p \xrightarrow{\lambda} p'\ \wedge\ r \xrightarrow{\overline{\lambda}} r')$ (by the hypothesis of $React$)
$\Longrightarrow q \xrightarrow{\lambda} q'\ \wedge\ p'\sim_{dp} q'$ ($\because p\sim_{dp} q$) $\wedge$ $r \xrightarrow{\overline{\lambda}} r'$
$\Longrightarrow q|r \xrightarrow{\tau} q'|r'$ (by the $React$ rule, $\because \lambda\in\mathcal{L}\cup\mathcal{C}$) $\wedge$
$(p'|r', q'|r')\in S$ (by definition of S, $\because p', q', r'\in\mathcal{P}$).

If the $L-React$ rule defines a transition $p|r \xrightarrow{\tau} (p|r)'$
then $(p|r)'\in\mathcal{P}$ (by definition of $p|r \xrightarrow{\tau} (p|r)'$) $\wedge$
$(p|r)' = p''|r'$ (by the $L-React$ rule) $\wedge$
$\exists\,\tau_{r_X}\in\mathcal{R}\ \exists\,\overline{\tau}_{r_{X_1}}, \overline{\tau}_{r_{X_2}}\in\overline{\mathcal{R}}\ \exists p'\in\mathcal{P}\,(X\sim_{of} X_1|X_2\ \wedge\ p \xrightarrow{\overline{\tau}_{r_{X_1}}} p'\ \wedge\ p' \xrightarrow{\tau_{r_X}} p''\ \wedge\ r \xrightarrow{\overline{\tau}_{r_{X_2}}} r')$
(by the hypothesis of $L-React$)
$\Longrightarrow \exists q', q''\in\mathcal{P}\,(q \xrightarrow{\overline{\tau}_{r_{X_1}}} q'\ \wedge\ q' \xrightarrow{\tau_{r_X}} q''\ \wedge\ p'\sim_{dp} q'\ \wedge\ p''\sim_{dp} q'')(\because p\sim_{dp} q)\ \wedge$
$r \xrightarrow{\overline{\tau}_{r_{X_2}}} r'\ \wedge\ X\sim_{of} X_1|X_2$
$\Longrightarrow q|r \xrightarrow{\tau} q''|r'$ (by the $L-React$ rule) $\wedge$
$(p''|r', q''|r')\in S$ (by definition of S, $\because p'', q'', r'\in\mathcal{P}$).

If the $R-React$ rule defines a transition $p|r \xrightarrow{\tau} (p|r)'$

then $(p|r)'\in\mathcal{P}$ (by definition of $p|r\xrightarrow{\tau}(p|r)'$) $\wedge$
$(p|r)' = p'|r''$ (by the $R-React$ rule) $\wedge$
$\exists\,\tau_{r_X}\in\mathcal{R}\,\exists\,\overline{\tau}_{r_{X_1}},\overline{\tau}_{r_{X_2}}\in\overline{\mathcal{R}}\,\exists\,r'\in\mathcal{P}\,(X\sim_{of} X_1|X_2\ \wedge\ p\xrightarrow{\overline{\tau}_{r_{X_1}}}p'\ \wedge\ r\xrightarrow{\overline{\tau}_{r_{X_2}}}r'\ \wedge\ r'\xrightarrow{\tau_{r_X}}r'')$
(by the hypothesis of $R-React$)
$\implies\exists\,q'\in\mathcal{P}\,(q\xrightarrow{\overline{\tau}_{r_{X_1}}}q'\ \wedge\ p'\sim_{dp} q')(\because p\sim_{dp} q)\ \wedge\ r\xrightarrow{\overline{\tau}_{r_{X_2}}}r'\ \wedge\ r'\xrightarrow{\tau_{r_X}}r''\ \wedge\ X\sim_{of} X_1|X_2$
$\implies q|r\xrightarrow{\tau}q'|r''$ (by the $R-React$ rule) $\wedge$
$(p'|r'',q'|r'')\in S$ (by definition of S, $\because p',q',r''\in\mathcal{P}$).

$\therefore$ S is a strong of-simulation on $\mathcal{P}$ (by definition of strong of-simulation on $\mathcal{P}$). Q.E.D.

A.19.1.2 S is a strong of-bisimulation on $\mathcal{P}$

We prove S is a strong of-bisimulation on $\mathcal{P}$ by proving S^{-1} is a strong of-simulation on $\mathcal{P}$.
Let $S'\triangleq\{(q|r,p|r)\mid q,p,r\in\mathcal{P}\ \wedge\ q\sim_{dp} p\}$.
$S^{-1}=\{(q|r,p|r)\mid p,q,r\in\mathcal{P}\ \wedge\ p\sim_{dp} q\}$
(by definitions of S and inverse binary relations)
$\Rightarrow S^{-1}=\{(q|r,p|r)\mid q,p,r\in\mathcal{P}\ \wedge\ q\sim_{dp} p\}$ ($\because\sim_{dp}$ is symmetric, by Lemma 4.3.12)
$\Rightarrow S^{-1}=S'$ (by definition of S').
The proof that S^{-1} is a strong of-simulation on $\mathcal{P}$ is identical to the proof that S is a strong of-simulation on $\mathcal{P}$ (see A.19.1.1) with the following substitutions:
p is replaced with q
q with p
p' with q'
q' with p'
p'' with q''
q'' with p''
S with S'.
Thus, S' is a strong of-simulation on $\mathcal{P}$
(by definition of strong of-simulation on $\mathcal{P}$)
$\Rightarrow S^{-1}$ is a strong of-simulation on $\mathcal{P}$ ($\because S^{-1}=S'$)
$\Rightarrow S$ is a strong of-bisimulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$, $\because S$ is a strong of-simulation on $\mathcal{P}$).
Q.E.D.
$\therefore\ \forall p,q\in\mathcal{P}\,(p\sim_{dp} q\implies\forall r\in\mathcal{P}\,(p|r\sim_{of} q|r))$ (by definitions of $p|r\sim_{of} q|r$ and S). Q.E.D.

A.19.2 $\forall p,q\in\mathcal{P}\,(p\sim_{dp} q \implies \forall r\in\mathcal{P}\,(r|p\sim_{of} r|q))$

Proof: If $\exists$ strong of-bisimulation S on $\mathcal{P}$ with $\forall r\in\mathcal{P}\,((r|p,r|q)\in S)$ for any processes p,q in $\mathcal{P}$ such that $p\sim_{dp} q$
then $\forall p,q\in\mathcal{P}$ (if $p\sim_{dp} q$ then $\forall r\in\mathcal{P}\,(r|p\sim_{of} r|q)$) (by definition of $r|p\sim_{of} r|q$).
Therefore, we find such an S.

Let $S\triangleq\{(r|p,r|q)\mid p,q,r\in\mathcal{P}\ \wedge\ p\sim_{dp} q\}$.

A.19.2.1 S is a strong of-simulation on $\mathcal{P}$

$S\subseteq\mathcal{P}\times\mathcal{P}\ \wedge\ \forall p,q,r\in\mathcal{P}\,(p\sim_{dp} q\implies(r|p,r|q)\in S)$ (by definition of S).

We prove for $(r|p,r|q)$ in S that the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied.

$(\mathbf{r|p},\mathbf{r|q})$ satisfies the Observation and Fraction conditions

The transitions of $r|p$ in $\mathcal{I}\cup\mathcal{R}$ are defined by the $L-Par$, $R-Par$, $React$, $L-React$ and $R-React$ rules only (by the syntax of $r|p$ and definitions of the LTS rules):

If the $L-Par$ rule defines a transition $r|p\xrightarrow{\delta}(r|p)'$ with $\delta\in\mathcal{I}\cup\mathcal{R}$
then $(r|p)'\in\mathcal{P}$ (by definition of $r|p\xrightarrow{\delta}(r|p)'$) $\wedge$ $(r|p)'=r'|p$ (by the $L-Par$ rule) $\wedge$
$r\xrightarrow{\delta}r'$ (by the hypothesis of $L-Par$)
$\implies r|q\xrightarrow{\delta}r'|q$ (by the $L-Par$ rule) $\wedge$
$(r'|p,r'|q)\in S$ (by definition of S, $\because p,q,r'\in\mathcal{P}\ \wedge\ p\sim_{dp} q$).

If the $R-Par$ rule defines a transition $r|p\xrightarrow{\delta}(r|p)'$ with $\delta\in\mathcal{I}\cup\mathcal{R}$
then $(r|p)'\in\mathcal{P}$ (by definition of $r|p\xrightarrow{\delta}(r|p)'$) $\wedge$ $(r|p)'=r|p'$ (by the $R-Par$ rule) $\wedge$
$p\xrightarrow{\delta}p'$ (by the hypothesis of $R-Par$)
$\implies q\xrightarrow{\delta}q'\ \wedge\ p'\sim_{dp} q'$ ($\because p\sim_{dp} q$)
$\implies r|q\xrightarrow{\delta}r|q'$ (by the $R-Par$ rule) $\wedge$
$(r|p',r|q')\in S$ (by definition of S, $\because p',q',r\in\mathcal{P}$).

If the $React$ rule defines a transition $r|p\xrightarrow{\tau}(r|p)'$
then $(r|p)'\in\mathcal{P}$ (by definition of $r|p\xrightarrow{\tau}(r|p)'$) $\wedge$ $(r|p)'=r'|p'$ (by the $React$ rule) $\wedge$
$\exists\lambda\in\mathcal{L}\cup\mathcal{C}\,(r\xrightarrow{\lambda}r'\ \wedge\ p\xrightarrow{\overline{\lambda}}p')$ (by the hypothesis of $React$)
$\implies q\xrightarrow{\overline{\lambda}}q'\ \wedge\ p'\sim_{dp} q'$ ($\because p\sim_{dp} q$) $\wedge$ $r\xrightarrow{\lambda}r'$
$\implies r|q\xrightarrow{\tau}r'|q'$ (by the $React$ rule, $\because\lambda\in\mathcal{L}\cup\mathcal{C}$) $\wedge$

$(r'|p', r'|q')\in S$ (by definition of S, $\because p', q', r'\in\mathcal{P}$).

If the $L-React$ rule defines a transition $r|p \xrightarrow{\tau} (r|p)'$
then $(r|p)'\in\mathcal{P}$ (by definition of $r|p \xrightarrow{\tau} (r|p)'$) $\wedge$
$(r|p)' = r''|p'$ (by the $L-React$ rule) $\wedge$
$\exists\,\tau_{r_X}\in\mathcal{R}\ \exists\,\overline{\tau}_{r_{X_1}}, \overline{\tau}_{r_{X_2}}\in\overline{\mathcal{R}}\ \exists\,r'\in\mathcal{P}\,(X\sim_{of} X_1|X_2\ \wedge\ r \xrightarrow{\overline{\tau}_{r_{X_1}}} r'\ \wedge\ r' \xrightarrow{\tau_{r_X}} r''\ \wedge\ p \xrightarrow{\overline{\tau}_{r_{X_2}}} p')$
(by the hypothesis of $L-React$)
$\implies \exists\,q'\in\mathcal{P}\,(q \xrightarrow{\overline{\tau}_{r_{X_2}}} q'\ \wedge\ p'\sim_{dp} q')(\because p\sim_{dp} q)\ \wedge\ r \xrightarrow{\overline{\tau}_{r_{X_1}}} r'\ \wedge\ r' \xrightarrow{\tau_{r_X}} r''\ \wedge\ X\sim_{of} X_1|X_2$
$\implies r|q \xrightarrow{\tau} r''|q'$ (by the $L-React$ rule) $\wedge$
$(r''|p', r''|q')\in S$ (by definition of S, $\because p', q', r''\in\mathcal{P}$).

If the $R-React$ rule defines a transition $r|p \xrightarrow{\tau} (r|p)'$
then $(r|p)'\in\mathcal{P}$ (by definition of $r|p \xrightarrow{\tau} (r|p)'$) $\wedge$
$(r|p)' = r'|p''$ (by the $R-React$ rule) $\wedge$
$\exists\,\tau_{r_X}\in\mathcal{R}\ \exists\,\overline{\tau}_{r_{X_1}}, \overline{\tau}_{r_{X_2}}\in\overline{\mathcal{R}}\ \exists\,p'\in\mathcal{P}\,(X\sim_{of} X_1|X_2\ \wedge\ r \xrightarrow{\overline{\tau}_{r_{X_1}}} r'\ \wedge\ p \xrightarrow{\overline{\tau}_{r_{X_2}}} p'\ \wedge\ p' \xrightarrow{\tau_{r_X}} p'')$
(by the hypothesis of $R-React$)
$\implies \exists\,q', q''\in\mathcal{P}\,(q \xrightarrow{\overline{\tau}_{r_{X_2}}} q'\ \wedge\ q' \xrightarrow{\tau_{r_X}} q''\ \wedge\ p'\sim_{dp} q'\ \wedge\ p''\sim_{dp} q'')(\because p\sim_{dp} q)\ \wedge$
$r \xrightarrow{\overline{\tau}_{r_{X_1}}} r'\ \wedge\ X\sim_{of} X_1|X_2$
$\implies r|q \xrightarrow{\tau} r'|q''$ (by the $R-React$ rule) $\wedge$
$(r'|p'', r'|q'')\in S$ (by definition of S, $\because p'', q'', r'\in\mathcal{P}$).

$\therefore$ S is a strong of-simulation on $\mathcal{P}$ (by definition of strong of-simulation on $\mathcal{P}$). Q.E.D.

A.19.2.2 S is a strong of-bisimulation on $\mathcal{P}$

We prove S is a strong of-bisimulation on $\mathcal{P}$ by proving S^{-1} is a strong of-simulation on $\mathcal{P}$.
Let $S' \triangleq \{(r|q, r|p)\mid q,p,r\in\mathcal{P}\ \wedge\ q\sim_{dp} p\}$.
$S^{-1} = \{(r|q, r|p)\mid p,q,r\in\mathcal{P}\ \wedge\ p\sim_{dp} q\}$
(by definitions of S and inverse binary relations)
$\Rightarrow S^{-1} = \{(r|q, r|p)\mid q,p,r\in\mathcal{P}\ \wedge\ q\sim_{dp} p\}$ ($\because \sim_{dp}$ is symmetric, by Lemma 4.3.12)
$\Rightarrow S^{-1} = S'$ (by definition of S').
The proof that S^{-1} is a strong of-simulation on $\mathcal{P}$ is identical to the proof that S is a strong of-simulation on $\mathcal{P}$ (see A.19.2.1) with the following substitutions:
p is replaced with q

q with p
p' with q'
q' with p'
p'' with q''
q'' with p''
S with S'.
Thus, S' is a strong of-simulation on $\mathcal{P}$
(by definition of strong of-simulation on $\mathcal{P}$)
$\Rightarrow S^{-1}$ is a strong of-simulation on $\mathcal{P}$ ($\because S^{-1} = S'$)
$\Rightarrow S$ is a strong of-bisimulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$, $\because S$ is a strong of-simulation on $\mathcal{P}$).
Q.E.D.
$\therefore \forall p, q \in \mathcal{P}\,(p \sim_{dp} q \Longrightarrow \forall r \in \mathcal{P}\,(r|p \sim_{of} r|q))$ (by definitions of $r|p \sim_{of} r|q$ and S). Q.E.D.

$\therefore \forall p, q \in \mathcal{P}\,(p \sim_{dp} q \Longrightarrow \forall r \in \mathcal{P}\,(p|r \sim_{of} q|r) \ \wedge \ \forall r \in \mathcal{P}\,(r|p \sim_{of} r|q))$
($\because \Longrightarrow$ is transitive and
$(predicate_1 \Longrightarrow predicate_2) \wedge (predicate_1 \Longrightarrow predicate_3) \Longrightarrow$
$(predicate_1 \Longrightarrow predicate_2 \wedge predicate_3)$). Q.E.D.

A.20 Theorem 4.3.10
$\sim_{dp}$ Preserves all Elementary Contexts

Proof: consists of discharging the following five proof obligations. Each proof obligation is discharged by defining a binary relation T on $\mathcal{P}$ which contains the pair of processes that are required to be strongly dp-bisimilar, proving T is a strong dp-simulation on $\mathcal{P}$, then proving T is a strong dp-bisimulation on $\mathcal{P}$.

1. $\vdash \forall p, q \in \mathcal{P}\,(p \sim_{dp} q \Longrightarrow \forall \alpha \in \mathcal{I}\,(\alpha.p + M \sim_{dp} \alpha.q + M)$
   where M is any summation in $\mathcal{P}$)

2. $\vdash \forall p, q \in \mathcal{P}\,(p \sim_{dp} q \Longrightarrow \forall r \in \mathcal{P}\,(p|r \sim_{dp} q|r))$

3. $\vdash \forall p, q \in \mathcal{P}\,(p \sim_{dp} q \Longrightarrow \forall r \in \mathcal{P}\,(r|p \sim_{dp} r|q))$

4. $\vdash \forall p, q \in \mathcal{P}\,(p \sim_{dp} q \Longrightarrow \forall r \in \mathcal{P}\,(\frac{p}{r} \sim_{dp} \frac{q}{r}))$

5. $\vdash \forall p, q \in \mathcal{P}\,(p \sim_{dp} q \Longrightarrow \forall r \in \mathcal{P}\,(\frac{r}{p} \sim_{dp} \frac{r}{q}))$

A.20.1 $\forall p, q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall \alpha \in \mathcal{I}$ $(\alpha.p + M \sim_{dp} \alpha.q + M)$ where M is any summation in $\mathcal{P}$)

Proof: If $\exists$ strong dp-bisimulation T on $\mathcal{P}$ with $\forall \alpha \in \mathcal{I}$ $((\alpha.p + M, \alpha.q + M) \in T)$ where M is any summation in $\mathcal{P}$ and p, q are any processes in $\mathcal{P}$ such that $p \sim_{dp} q$
then $\forall p, q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall \alpha \in \mathcal{I}$ $(\alpha.p + M \sim_{dp} \alpha.q + M)$ where M is any summation in $\mathcal{P}$)
(by definition of $\alpha.p + M \sim_{dp} \alpha.q + M$).
Therefore, we find such a T.

Let $T \triangleq \{(\alpha.p + M, \alpha.q + M), (p, q), (r, r) \mid$
$\alpha \in \mathcal{I} \ \wedge \ p, q \in \mathcal{P}$ $(p \sim_{dp} q) \ \wedge \ M$ is any summation in $\mathcal{P} \ \wedge \ r \in \mathcal{P}\}$.

A.20.1.1 T is a strong dp-simulation on $\mathcal{P}$

$T \subseteq \mathcal{P} \times \mathcal{P} \ \wedge$
$\forall \alpha \in \mathcal{I}$ and $\forall p, q \in \mathcal{P}$ such that $p \sim_{dp} q$ and $\forall$summation $M \in \mathcal{P}$ $((\alpha.p + M, \alpha.q + M) \in T)$
(by definition of T).

We prove for $(\alpha.p + M, \alpha.q + M)$, (p, q) and (r, r) in T that the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied.

$(\alpha.\mathbf{p} + \mathbf{M}, \alpha.\mathbf{q} + \mathbf{M})$ satisfies the Observation and Fraction conditions

The transitions of $\alpha.p + M$ in $\mathcal{I} \cup \mathcal{R}$ are defined by the *Sum* rule only
(by the syntax of $\alpha.p + M$):
$\alpha.p + M \xrightarrow{\alpha} p$ and $\alpha.q + M \xrightarrow{\alpha} q$ (by the *Sum* rule),
and $p \sim_{dp} q$ (by definition of p and q)
$\Longrightarrow (p, q) \in T$ (by definition of T).
If $M \xrightarrow{\delta} s$ for some $\delta \in \mathcal{I} \cup \mathcal{R}$ and some $s \in \mathcal{P}$
then $\alpha.p + M \xrightarrow{\delta} s$ and $\alpha.q + M \xrightarrow{\delta} s$ (by the *Sum* rule),
and $(s, s) \in T$ (by definition of T).

$(\alpha.\mathbf{p} + \mathbf{M}, \alpha.\mathbf{q} + \mathbf{M})$ satisfies the Deletion condition

A transition of $\alpha.p + M$ in $\overline{\mathcal{R}}$ is defined by *Delet* or *CompDelet* only
(by the syntax of $\alpha.p + M$):
If the *Delet* rule defines a transition $\alpha.p + M \xrightarrow{\overline{\tau}_{r_W}} p'$

then $p' = 0$ (by the *Delet* rule) $\wedge$ $\alpha.p + M \sim_{of} W$ (by the hypothesis of *Delet*).
Now $p \sim_{dp} q$ (by definition of p and q)
$\implies p \sim_{of} q$ (by Corollary 4.3.5)
$\implies \alpha.p + M \sim_{of} \alpha.q + M$ (by Theorem 4.3.4, $\because \alpha \in \mathcal{I}$ $\wedge$ M is a summation in $\mathcal{P}$)
$\implies \alpha.q + M \sim_{of} \alpha.p + M$ ($\because \sim_{of}$ is symmetric, by Lemma 4.3.2)
$\implies \alpha.q + M \sim_{of} \alpha.p + M \wedge \alpha.p + M \sim_{of} W$ ($\because \alpha.p + M \sim_{of} W$)
$\implies \alpha.q + M \sim_{of} W$ ($\because \sim_{of}$ is transitive, by Lemma 4.3.4).
And $\alpha.q + M \in \mathcal{P}^+$ (by production rules 1 and 2 of $\mathcal{P}^+$, $\because \alpha \in \mathcal{I}$ $\wedge$ $p, q, M, \alpha.q + M \in \mathcal{P}$)
$\implies \alpha.q + M \xrightarrow{\overline{\tau}_{r_W}} 0$ (by the *Delet* rule, $\because \alpha.q + M \sim_{of} W$),
and $(0, 0) \in T$ (by definition of T).

If the *CompDelet* rule defines a transition $\alpha.p + M \xrightarrow{\overline{\tau}_{r_X}} p''$
then $\exists\, \overline{\tau}_{r_{X_1}}, \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}\ \exists p' \in \mathcal{P}\ (X \sim_{of} X_1 | X_2 \wedge \alpha.p + M \xrightarrow{\overline{\tau}_{r_{X_1}}} p' \wedge p' \xrightarrow{\overline{\tau}_{r_{X_2}}} p'')$
(by the hypothesis of *CompDelet*)
$\implies \overline{\tau}_{r_{X_1}} \in \overline{\mathcal{R}}_{\alpha.p+M}$ (by definition of $\overline{\mathcal{R}}_{\alpha.p+M}$) $\wedge$ $\overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}_{p'}$ (by definition of $\overline{\mathcal{R}}_{p'}$)
$\implies p' = 0 \vee factors^+_m(p') \subset factors^+_m(\alpha.p + M)$
(by Lemma 4.3.18, $\because \alpha.p + M, p' \in \mathcal{P} \wedge \alpha.p + M \xrightarrow{\overline{\tau}_{r_{X_1}}} p'$).
If $p' = 0$
then $\overline{\mathcal{R}}_{p'} = \emptyset$ (by Lemma 4.3.8).
But $\overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}_{p'} \implies \overline{\mathcal{R}}_{p'} \neq \emptyset$ (by set theory; which is a contradiction).
$\therefore p' \neq 0$.
Now $factors^+_m(\alpha.p + M) \subseteq factors_m(\alpha.p + M)$ (by definition of $factors^+_m(\alpha.p + M)$) $\wedge$
$factors_m(\alpha.p + M) = \emptyset$ (by definition of $factors_m(\alpha.p + M)$)
$\implies factors^+_m(\alpha.p + M) \subseteq \emptyset$ (by set theory)
$\implies factors^+_m(\alpha.p + M) = \emptyset$ (by set theory).
$\therefore$ If $factors^+_m(p') \subset factors^+_m(\alpha.p + M)$
then $factors^+_m(p') \subset \emptyset$ (which is false, by set theory).
$\therefore$ The *CompDelet* rule does not define a transition of $\alpha.p + M$ in $\overline{\mathcal{R}}$.

($\mathbf{p}, \mathbf{q}$) satisfies the Observation, Fraction and Deletion conditions

If $p \xrightarrow{\beta} p'$ for some $\beta \in \mathcal{I} \cup \mathcal{R} \cup \overline{\mathcal{R}}$ and some $p' \in \mathcal{P}$
then $q \xrightarrow{\beta} q'$ for some $q' \in \mathcal{P} \wedge p' \sim_{dp} q'$ ($\because p \sim_{dp} q$, by definition of p and q)
$\implies (p', q') \in T$ (by definition of T).

(r, r) satisfies the Observation, Fraction and Deletion conditions

If $r \xrightarrow{\gamma} r'$ for some $\gamma \in \mathcal{I} \cup \mathcal{R} \cup \overline{\mathcal{R}}$ and some $r' \in \mathcal{P}$
then $r \xrightarrow{\gamma} r'$
and $(r', r') \in T$ (by definition of T).

$\therefore$ T is a strong dp-simulation on $\mathcal{P}$ (by definition of strong dp-simulation on $\mathcal{P}$).
Q.E.D.

A.20.1.2 T is a strong dp-bisimulation on $\mathcal{P}$

We prove T is a strong dp-bisimulation on $\mathcal{P}$ by proving T^{-1} is a strong dp-simulation on $\mathcal{P}$.
Let $T' \triangleq \{(\alpha.q + M, \alpha.p + M), (q, p), (r, r) \mid$
$\alpha \in \mathcal{I} \ \wedge \ q, p \in \mathcal{P} \ (q \sim_{dp} p) \ \wedge \ M$ is any summation in $\mathcal{P} \ \wedge \ r \in \mathcal{P}\}$.
$T^{-1} = \{(\alpha.q + M, \alpha.p + M), (q, p), (r, r) \mid$
$\alpha \in \mathcal{I} \ \wedge \ p, q \in \mathcal{P} \ (p \sim_{dp} q) \ \wedge \ M$ is any summation in $\mathcal{P} \ \wedge \ r \in \mathcal{P}\}$
(by definitions of T and inverse binary relations)
$\Rightarrow T^{-1} = \{(\alpha.q + M, \alpha.p + M), (q, p), (r, r) \mid$
$\alpha \in \mathcal{I} \ \wedge \ q, p \in \mathcal{P} \ (q \sim_{dp} p) \ \wedge \ M$ is any summation in $\mathcal{P} \ \wedge \ r \in \mathcal{P}\}$
($\because \sim_{dp}$ is symmetric, by Lemma 4.3.12)
$\Rightarrow T^{-1} = T'$ (by definition of T').
The proof that T^{-1} is a strong dp-simulation on $\mathcal{P}$ is identical to the proof that T is a strong dp-simulation on $\mathcal{P}$ (see A.20.1.1) with the following substitutions:
p is replaced with q
q with p
p' with q'
q' with p'
p'' with q''
T with T'.
Thus, T' is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$)
$\Rightarrow T^{-1}$ is a strong dp-simulation on $\mathcal{P}$ ($\because T^{-1} = T'$)
$\Rightarrow T$ is a strong dp-bisimulation on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$, $\because T$ is a strong dp-simulation on $\mathcal{P}$).
Q.E.D.
$\therefore \forall p, q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall \alpha \in \mathcal{I} \ (\alpha.p + M \sim_{dp} \alpha.q + M)$ where M is any summation in $\mathcal{P}$)

(by definitions of $\alpha.p + M \sim_{dp} \alpha.q + M$ and T). Q.E.D.

A.20.2 $\forall p, q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall r \in \mathcal{P}$ $(p|r \sim_{dp} q|r)$)

Proof: If $\exists$ strong dp-bisimulation T on $\mathcal{P}$ with $\forall r \in \mathcal{P}$ $((p|r, q|r) \in T)$ for any processes p, q in $\mathcal{P}$ such that $p \sim_{dp} q$
then $\forall p, q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall r \in \mathcal{P}$ $(p|r \sim_{dp} q|r)$) (by definition of $p|r \sim_{dp} q|r$).
Therefore, we find such a T.

Let $S \triangleq \{(p|r, q|r) \mid p, q, r \in \mathcal{P} \ \wedge \ p \sim_{dp} q\}$.
$S \subseteq \mathcal{P}$ x $\mathcal{P}$ $\wedge$ $\forall p, q, r \in \mathcal{P}$ $(p \sim_{dp} q \Longrightarrow (p|r, q|r) \in S)$ (by definition of S).
Let $Z \triangleq \{(0, 0)\}$.
$Z \subseteq \mathcal{P}$ x $\mathcal{P}$ (by definition of Z).
Let $T \triangleq S \cup Z$.

T is a strong dp-bisimulation on $\mathcal{P}$
$\Longleftrightarrow$ T, T^{-1} are strong dp-simulations on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$)
$\Longleftrightarrow$ T, T^{-1} are binary relations on $\mathcal{P}$ $\wedge$
for all elements of T, T^{-1} the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
(by definition of strong dp-simulation on $\mathcal{P}$).
We prove T, T^{-1} are binary relations on $\mathcal{P}$ and for all elements of T, T^{-1} the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied.

A.20.2.1 T, T^{-1} satisfy the Observation and Fraction conditions

S is a strong of-bisimulation on $\mathcal{P}$
(by the proof of Theorem 4.3.9, see A.19) $\wedge$
Z is a strong of-bisimulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$, $\because \mathcal{I}_0 \cup \mathcal{R}_0 = \emptyset$ (by Lemma 4.3.7))
$\Longrightarrow$ T is a strong of-bisimulation on $\mathcal{P}$
($\because$ the union of strong of-bisimulations on $\mathcal{P}$ is a strong of-bisimulation on $\mathcal{P}$, and by definition of T)
$\Longrightarrow$ T, T^{-1} are strong of-simulations on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$)
$\Longrightarrow$ $T \subseteq \mathcal{P}$ x $\mathcal{P}$ $\wedge$ $T^{-1} \subseteq \mathcal{P}$ x $\mathcal{P}$ $\wedge$ for all elements of T, T^{-1} the *Observation* and *Fraction*

conditions of strong of-simulation on $\mathcal{P}$ are satisfied
(by definition of strong of-simulation on $\mathcal{P}$)
$\Longrightarrow$ for all elements of T, T^{-1} the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
($\because$ the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are the same as the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$, respectively). Q.E.D.
And $\forall p, q, r \in \mathcal{P}\ (p \sim_{dp} q \Longrightarrow (p|r, q|r) \in T)$ (by set theory and definitions of S and T).

It remains to prove that for all elements of T, T^{-1} the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied.

A.20.2.2 T satisfies the Deletion condition

We use complete induction on the depth of inference of the applications of the LTS rules that determine the transitions of $u \in dom(T)$ in $\overline{\mathcal{R}}$.

For $n \in \mathbb{N}^+$, let $Prop(n)$ be the proposition:
$\forall (u, v) \in T\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u\ \forall u'' \in \mathcal{P}$
$(u \xrightarrow{\overline{\tau}_{r_Y}} u'' \Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \wedge \exists v'' \in \mathcal{P}\ (v \xrightarrow{\overline{\tau}_{r_Y}} v''\ \wedge\ (u'', v'') \in T))$
for $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ determined by applications of LTS rules with depth of inference n.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(1)$
2. $\vdash \forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \Longrightarrow Prop(n+1))$

Base Case: Proof of Prop(1)

For $(u, v) \in T$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u$ and $u'' \in \mathcal{P}$,
the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ has depth of inference 1 (by the hypothesis of $Prop(1)$)
$\Longrightarrow$ only the *Delet* rule determines the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$
(by definitions of the LTS rules):

If $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ (by the *Delet* rule)
then $u'' = 0$ (by the *Delet* rule) $\wedge$ $u \sim_{of} Y$ $\wedge$ $u \in \mathcal{P}^+$ (by the hypothesis of *Delet*)
$\Longrightarrow u \notin \mathcal{P}^0$ (by Theorem 4.3.2)

$\Longrightarrow u \neq 0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$)
$\Longrightarrow (u,v) \notin Z$ (by definition of Z)
$\Longrightarrow (u,v) \in S$ ($\because (u,v) \in T \ \wedge \ T = S \cup Z$)
$\Longrightarrow \exists p,q,r \in \mathcal{P}\ (p \sim_{dp} q \ \wedge \ u = p|r \ \wedge \ v = q|r)$ (by definition of S)
$\Longrightarrow p|r \sim_{of} q|r$ (by Theorem 4.3.9) $\ \wedge \ p|r \sim_{of} Y$ ($\because u \sim_{of} Y$) $\ \wedge \ p|r \in \mathcal{P}^+$ ($\because u \in \mathcal{P}^+$)
$\Longrightarrow q|r \sim_{of} p|r$ ($\because \sim_{of}$ is symmetric, by Lemma 4.3.2) $\ \wedge \ p|r \sim_{of} Y \ \wedge$
$q|r \in \mathcal{P}^+$ (by Lemma 4.3.9)
$\Longrightarrow q|r \sim_{of} Y$ ($\because \sim_{of}$ is transitive, by Lemma 4.3.4) $\ \wedge \ q|r \in \mathcal{P}^+$
$\Longrightarrow q|r \xrightarrow{\overline{\tau}_{r_Y}} 0$ (by the *Delet* rule)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{q|r}$ (by definition of $\overline{\mathcal{R}}_{q|r}$) $\ \wedge$
$0 \in \mathcal{P}$ (by production rule 1 of $\mathcal{P}^0$, set theory and definition of $\mathcal{P}$) $\ \wedge \ q|r \xrightarrow{\overline{\tau}_{r_Y}} 0 \ \wedge$
$(0,0) \in T$ (by set theory and definitions of Z and T)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \ \wedge \ 0 \in \mathcal{P} \ \wedge \ v \xrightarrow{\overline{\tau}_{r_Y}} 0 \ \wedge \ (0,0) \in T$ ($\because v = q|r$).

$\therefore \forall (u,v) \in T \ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u \ \forall u'' \in \mathcal{P}$
$(u \xrightarrow{\overline{\tau}_{r_Y}} u'' \Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \wedge \exists v'' \in \mathcal{P}\ (v \xrightarrow{\overline{\tau}_{r_Y}} v'' \ \wedge \ (u'',v'') \in T))$
for $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ determined by applications of LTS rules with depth of inference 1
($\because (u,v) \in T$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u$ and $u'' \in \mathcal{P}$ with transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ and depth of inference 1 are arbitrary)
$\Longrightarrow$ *Prop*(1) holds (by definition of *Prop*(1)). Q.E.D.

Induction Step: Proof of $\forall \mathbf{n} \in \mathbb{N}^+$ ($\forall \mathbf{m} \in [\mathbf{1}, \mathbf{n}]\mathbf{Prop}(\mathbf{m}) \Longrightarrow \mathbf{Prop}(\mathbf{n}+\mathbf{1})$)

For $n \in \mathbb{N}^+$, assume $\forall m \in [1,n] Prop(m)$ holds (inductive hypothesis).
For $(u,v) \in T$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u$ and $u'' \in \mathcal{P}$,
the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ has depth of inference $n+1$
(by the hypothesis of $Prop(n+1)$)
$\Longrightarrow \overline{\mathcal{R}}_u \neq \emptyset$ (by set theory)
$\Longrightarrow u \in \mathcal{P}^+$ (by Lemma 4.3.6)
$\Longrightarrow u \notin \mathcal{P}^0$ (by Theorem 4.3.2)
$\Longrightarrow u \neq 0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$)
$\Longrightarrow (u,v) \notin Z$ (by definition of Z)
$\Longrightarrow (u,v) \in S$ ($\because (u,v) \in T \ \wedge \ T = S \cup Z$)
$\Longrightarrow \exists p,q,r \in \mathcal{P}\ (p \sim_{dp} q \ \wedge \ u = p|r \ \wedge \ v = q|r)$ (by definition of S)
$\Longrightarrow (p|r, q|r) \in T$ ($\because (u,v) \in T$).
Now $n+1 \geq 2$ ($\because n \in \mathbb{N}^+$)

$\Longrightarrow$ only the $L-Par$, $R-Par$ or $CompDelet$ rules determine the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ (by definitions of the LTS rules):

If the $L-Par$ rule defines a transition $p|r \xrightarrow{\overline{\tau}_{r_Y}} p'|r$
then $p \xrightarrow{\overline{\tau}_{r_Y}} p'$ (by the hypothesis of $L-Par$)
$\Longrightarrow q \xrightarrow{\overline{\tau}_{r_Y}} q' \ \wedge \ p' \sim_{dp} q'$ ($\because p \sim_{dp} q$)
$\Longrightarrow q|r \xrightarrow{\overline{\tau}_{r_Y}} q'|r$ (by the $L-Par$ rule) $\wedge$
$(p'|r, q'|r) \in S$ (by definition of S, $\because p', q', r \in \mathcal{P}$)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{q|r}$ (by definition of $\overline{\mathcal{R}}_{q|r}$) $\wedge$ $q'|r \in \mathcal{P}$ (by definition of $q|r \xrightarrow{\overline{\tau}_{r_Y}} q'|r$) $\wedge$
$q|r \xrightarrow{\overline{\tau}_{r_Y}} q'|r$ $\wedge$ $(p'|r, q'|r) \in T$ (by set theory and definition of T)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \ \wedge \ q'|r \in \mathcal{P} \ \wedge \ v \xrightarrow{\overline{\tau}_{r_Y}} q'|r \ \wedge \ (p'|r, q'|r) \in T$ ($\because v = q|r$).

If the $R-Par$ rule defines a transition $p|r \xrightarrow{\overline{\tau}_{r_Y}} p|r'$
then $r \xrightarrow{\overline{\tau}_{r_Y}} r'$ (by the hypothesis of $R-Par$)
$\Longrightarrow q|r \xrightarrow{\overline{\tau}_{r_Y}} q|r'$ (by the $R-Par$ rule) $\wedge$
$(p|r', q|r') \in S$ (by definition of S, $\because p, q, r' \in \mathcal{P} \ \wedge \ p \sim_{dp} q$)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{q|r}$ (by definition of $\overline{\mathcal{R}}_{q|r}$) $\wedge$ $q|r' \in \mathcal{P}$ (by definition of $q|r \xrightarrow{\overline{\tau}_{r_Y}} q|r'$) $\wedge$
$q|r \xrightarrow{\overline{\tau}_{r_Y}} q|r'$ $\wedge$ $(p|r', q|r') \in T$ (by set theory and definition of T)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \ \wedge \ q|r' \in \mathcal{P} \ \wedge \ v \xrightarrow{\overline{\tau}_{r_Y}} q|r' \ \wedge \ (p|r', q|r') \in T$ ($\because v = q|r$).

If the $CompDelet$ rule defines a transition $p|r \xrightarrow{\overline{\tau}_{r_Y}} (p|r)''$
then $(p|r)'' \in \mathcal{P}$ (by definition of $p|r \xrightarrow{\overline{\tau}_{r_Y}} (p|r)''$) $\wedge$
$\exists\, \overline{\tau}_{r_{Y_1}}, \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}\ \exists (p|r)' \in \mathcal{P}\ (Y \sim_{of} Y_1|Y_2 \ \wedge \ p|r \xrightarrow{\overline{\tau}_{r_{Y_1}}} (p|r)' \ \wedge \ (p|r)' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (p|r)'')$
(by the hypothesis of $CompDelet$)
$\Longrightarrow \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{p|r}$ (by definition of $\overline{\mathcal{R}}_{p|r}$) $\wedge$ $\overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(p|r)'}$ (by definition of $\overline{\mathcal{R}}_{(p|r)'}$).
Now the transition $p|r \xrightarrow{\overline{\tau}_{r_Y}} (p|r)''$ has depth of inference $n+1$
(by the hypothesis of $Prop(n+1)$)
$\Longrightarrow$ the transition $p|r \xrightarrow{\overline{\tau}_{r_{Y_1}}} (p|r)'$ has depth of inference m_1, with $m_1 \in [1, n]$ $\wedge$
the transition $(p|r)' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (p|r)''$ has depth of inference m_2, with $m_2 \in [1, n]$
($\because$ the transition $p|r \xrightarrow{\overline{\tau}_{r_Y}} (p|r)''$ is inferred from the transitions $p|r \xrightarrow{\overline{\tau}_{r_{Y_1}}} (p|r)'$ and $(p|r)' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (p|r)''$ using the $CompDelet$ rule)
$\Longrightarrow Prop(m_1)$ and $Prop(m_2)$ hold (by the inductive hypothesis)

$\implies \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{q|r} \ \wedge \ \exists (q|r)' \in \mathcal{P} \ (q|r \xrightarrow{\overline{\tau}_{r_{Y_1}}} (q|r)' \ \wedge \ ((p|r)', (q|r)') \in T) \ \wedge \ Prop(m_2)$
(by modus ponens,
$\because (p|r, q|r) \in T \ \wedge \ \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{p|r} \ \wedge \ (p|r)' \in \mathcal{P} \ \wedge \ p|r \xrightarrow{\overline{\tau}_{r_{Y_1}}} (p|r)'$ with depth of inference m_1)
$\implies \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(q|r)'} \ \wedge \ \exists (q|r)'' \in \mathcal{P} \ ((q|r)' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (q|r)'' \ \wedge \ ((p|r)'', (q|r)'') \in T)$
(by modus ponens,
$\because \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(p|r)'} \ \wedge \ (p|r)'' \in \mathcal{P} \ \wedge \ (p|r)' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (p|r)''$ with depth of inference m_2)
$\implies (q|r)'' \in \mathcal{P} \ \wedge \ q|r \xrightarrow{\overline{\tau}_{r_Y}} (q|r)'' \ \wedge \ ((p|r)'', (q|r)'') \in T$
(by the *CompDelet* rule, $\because Y \sim_{of} Y_1|Y_2 \ \wedge \ q|r \xrightarrow{\overline{\tau}_{r_{Y_1}}} (q|r)'$)
$\implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{q|r}$ (by definition of $\overline{\mathcal{R}}_{q|r}$) $\wedge \ (q|r)'' \in \mathcal{P} \ \wedge \ q|r \xrightarrow{\overline{\tau}_{r_Y}} (q|r)'' \ \wedge \ ((p|r)'', (q|r)'') \in T$
$\implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \ \wedge \ (q|r)'' \in \mathcal{P} \ \wedge \ v \xrightarrow{\overline{\tau}_{r_Y}} (q|r)'' \ \wedge \ ((p|r)'', (q|r)'') \in T \ (\because v = q|r)$.

$\therefore \forall (u, v) \in T \ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u \ \forall u'' \in \mathcal{P}$
$(u \xrightarrow{\overline{\tau}_{r_Y}} u'' \implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \wedge \exists v'' \in \mathcal{P} \ (v \xrightarrow{\overline{\tau}_{r_Y}} v'' \ \wedge \ (u'', v'') \in T))$
for $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ determined by applications of LTS rules with depth of inference $n + 1$
($\because (u, v) \in T$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u$ and $u'' \in \mathcal{P}$ with transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ and depth of inference $n + 1$ are arbitrary)
$\implies Prop(n + 1)$ holds (by definition of $Prop(n + 1)$).
$\therefore \forall n \in \mathbb{N}^+ \ (\forall m \in [1, n] Prop(m) \implies Prop(n + 1))$ holds ($\because n \in \mathbb{N}^+$ is arbitrary). Q.E.D.

$\therefore \forall n \in \mathbb{N}^+ \ Prop(n)$ holds (by complete induction).
$\therefore$ For all elements of T the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied
($\because$ every transition of $u \in dom(T)$ in $\overline{\mathcal{R}}$ is determined by applications of LTS rules with finite depth of inference). Q.E.D.

$\therefore T$ is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$, $\because T \subseteq \mathcal{P}$ x $\mathcal{P}$ and for all elements of T the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied).

A.20.2.3 T^{-1} satisfies the Deletion condition

Let $S' \triangleq \{(q|r, p|r) \mid q, p, r \in \mathcal{P} \ \wedge \ q \sim_{dp} p\}$.
$S' \subseteq \mathcal{P}$ x $\mathcal{P} \ \wedge \ \forall q, p, r \in \mathcal{P} \ (q \sim_{dp} p \implies (q|r, p|r) \in S')$ (by definition of S').
$S^{-1} = \{(q|r, p|r) \mid p, q, r \in \mathcal{P} \ \wedge \ p \sim_{dp} q\}$
(by definitions of S and inverse binary relations)
$\Rightarrow S^{-1} = \{(q|r, p|r) \mid q, p, r \in \mathcal{P} \ \wedge \ q \sim_{dp} p\}$ ($\because \sim_{dp}$ is symmetric, by Lemma 4.3.12)

$\Rightarrow S^{-1} = S'$ (by definition of S').
$Z^{-1} = Z$ (by definitions of Z and inverse binary relations).
Let $T' \triangleq S' \cup Z$.
$T' \subseteq \mathcal{P}$ x $\mathcal{P}$ $\wedge$ $\forall q, p, r \in \mathcal{P}$ $(q \sim_{dp} p \implies (q|r, p|r) \in T')$
(by set theory and definitions of S', Z and T').
$T^{-1} = S^{-1} \cup Z^{-1}$ ($\because T = S \cup Z$ and by algebra of binary relations)
$\Rightarrow T^{-1} = S' \cup Z$ ($\because S^{-1} = S' \wedge Z^{-1} = Z$)
$\Rightarrow T^{-1} = T'$ (by definition of T').
The remainder of the proof is identical to the proof that for all elements of T the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied (see A.20.2.2) with the following substitutions:
p is replaced with q
q with p
p' with q'
q' with p'
u with v
v with u
u'' with v''
v'' with u''
S with S'
T with T'.
Thus, for all elements of T' the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied
($\because$ every transition of $v \in dom(T')$ in $\overline{\mathcal{R}}$ is determined by applications of LTS rules with finite depth of inference)
$\Rightarrow$ for all elements of T^{-1} the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied
($\because T^{-1} = T'$). Q.E.D.

$\therefore T^{-1}$ is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$, $\because T^{-1} \subseteq \mathcal{P}$ x $\mathcal{P}$ and for all elements of T^{-1} the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied)
$\Rightarrow T$ is a strong dp-bisimulation on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$, $\because T$ is a strong dp-simulation on $\mathcal{P}$).
$\therefore \forall p, q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall r \in \mathcal{P}$ $(p|r \sim_{dp} q|r)$) (by definition of $p|r \sim_{dp} q|r$). Q.E.D.

A.20.3 $\forall p, q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall r \in \mathcal{P}$ $(r|p \sim_{dp} r|q)$)

Proof: If $\exists$ strong dp-bisimulation T on $\mathcal{P}$ with $\forall r \in \mathcal{P}$ $((r|p, r|q) \in T)$ for any processes p, q in $\mathcal{P}$ such that $p \sim_{dp} q$
then $\forall p, q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall r \in \mathcal{P}$ $(r|p \sim_{dp} r|q)$) (by definition of $r|p \sim_{dp} r|q$).
Therefore, we find such a T.

Let $S \triangleq \{(r|p, r|q) \mid p, q, r \in \mathcal{P} \ \wedge \ p \sim_{dp} q\}$.
$S \subseteq \mathcal{P} \times \mathcal{P} \ \wedge \ \forall p, q, r \in \mathcal{P}$ $(p \sim_{dp} q \Longrightarrow (r|p, r|q) \in S)$ (by definition of S).
Let $Z \triangleq \{(0, 0)\}$.
$Z \subseteq \mathcal{P} \times \mathcal{P}$ (by definition of Z).
Let $T \triangleq S \cup Z$.

T is a strong dp-bisimulation on $\mathcal{P}$
$\Longleftrightarrow T, T^{-1}$ are strong dp-simulations on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$)
$\Longleftrightarrow T, T^{-1}$ are binary relations on $\mathcal{P} \ \wedge$
for all elements of T, T^{-1} the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
(by definition of strong dp-simulation on $\mathcal{P}$).
We prove T, T^{-1} are binary relations on $\mathcal{P}$ and for all elements of T, T^{-1} the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied.

A.20.3.1 T, T^{-1} satisfy the Observation and Fraction conditions

S is a strong of-bisimulation on $\mathcal{P}$
(by the proof of Theorem 4.3.9, see A.19) $\wedge$
Z is a strong of-bisimulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$, $\because \mathcal{I}_0 \cup \mathcal{R}_0 = \emptyset$ (by Lemma 4.3.7))
$\Longrightarrow T$ is a strong of-bisimulation on $\mathcal{P}$
($\because$ the union of strong of-bisimulations on $\mathcal{P}$ is a strong of-bisimulation on $\mathcal{P}$, and by definition of T)
$\Longrightarrow T, T^{-1}$ are strong of-simulations on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$)
$\Longrightarrow T \subseteq \mathcal{P} \times \mathcal{P} \ \wedge \ T^{-1} \subseteq \mathcal{P} \times \mathcal{P} \ \wedge$ for all elements of T, T^{-1} the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied
(by definition of strong of-simulation on $\mathcal{P}$)

$\implies$ for all elements of T, T^{-1} the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
($\because$ the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are the same as the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$, respectively). Q.E.D.
And $\forall p, q, r \in \mathcal{P}\ (p \sim_{dp} q \implies (r|p, r|q) \in T)$ (by set theory and definitions of S and T).

It remains to prove that for all elements of T, T^{-1} the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied.

A.20.3.2 T satisfies the Deletion condition

We use complete induction on the depth of inference of the applications of the LTS rules that determine the transitions of $u \in dom(T)$ in $\overline{\mathcal{R}}$.

For $n \in \mathbb{N}^+$, let $Prop(n)$ be the proposition:
$\forall (u, v) \in T\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u\ \forall u'' \in \mathcal{P}$
$(u \xrightarrow{\overline{\tau}_{r_Y}} u'' \implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \wedge \exists v'' \in \mathcal{P}\ (v \xrightarrow{\overline{\tau}_{r_Y}} v''\ \wedge\ (u'', v'') \in T))$
for $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ determined by applications of LTS rules with depth of inference n.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(1)$
2. $\vdash \forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \implies Prop(n+1))$

Base Case: Proof of Prop(1)

For $(u, v) \in T$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u$ and $u'' \in \mathcal{P}$,
the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ has depth of inference 1 (by the hypothesis of $Prop(1)$)
$\implies$ only the *Delet* rule determines the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$
(by definitions of the LTS rules):

If $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ (by the *Delet* rule)
then $u'' = 0$ (by the *Delet* rule) $\wedge$ $u \sim_{of} Y$ $\wedge$ $u \in \mathcal{P}^+$ (by the hypothesis of *Delet*)
$\implies u \notin \mathcal{P}^0$ (by Theorem 4.3.2)
$\implies u \neq 0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$)
$\implies (u, v) \notin Z$ (by definition of Z)

$\implies (u,v)\in S$ ($\because (u,v)\in T \ \wedge \ T = S \cup Z$)
$\implies \exists p,q,r\in\mathcal{P}\ (p \sim_{dp} q \ \wedge \ u = r|p \ \wedge \ v = r|q)$ (by definition of S)
$\implies r|p \sim_{of} r|q$ (by Theorem 4.3.9) $\ \wedge \ r|p \sim_{of} Y$ ($\because u \sim_{of} Y$) $\ \wedge \ r|p\in\mathcal{P}^+$ ($\because u\in\mathcal{P}^+$)
$\implies r|q \sim_{of} r|p$ ($\because \sim_{of}$ is symmetric, by Lemma 4.3.2) $\ \wedge \ r|p \sim_{of} Y \ \wedge$
$r|q\in\mathcal{P}^+$ (by Lemma 4.3.9)
$\implies r|q \sim_{of} Y$ ($\because \sim_{of}$ is transitive, by Lemma 4.3.4) $\ \wedge \ r|q\in\mathcal{P}^+$
$\implies r|q \xrightarrow{\overline{\tau}_{r_Y}} 0$ (by the *Delet* rule)
$\implies \overline{\tau}_{r_Y}\in\overline{\mathcal{R}}_{r|q}$ (by definition of $\overline{\mathcal{R}}_{r|q}$) $\ \wedge$
$0\in\mathcal{P}$ (by production rule 1 of $\mathcal{P}^0$, set theory and definition of $\mathcal{P}$) $\ \wedge \ r|q \xrightarrow{\overline{\tau}_{r_Y}} 0 \ \wedge$
$(0,0)\in T$ (by set theory and definitions of Z and T)
$\implies \overline{\tau}_{r_Y}\in\overline{\mathcal{R}}_v \ \wedge \ 0\in\mathcal{P} \ \wedge \ v \xrightarrow{\overline{\tau}_{r_Y}} 0 \ \wedge \ (0,0)\in T$ ($\because v = r|q$).

$\therefore \forall (u,v)\in T \ \forall \overline{\tau}_{r_Y}\in\overline{\mathcal{R}}_u \ \forall u''\in\mathcal{P}$
$(u \xrightarrow{\overline{\tau}_{r_Y}} u'' \implies \overline{\tau}_{r_Y}\in\overline{\mathcal{R}}_v \wedge \exists v''\in\mathcal{P}\ (v \xrightarrow{\overline{\tau}_{r_Y}} v'' \ \wedge \ (u'',v'')\in T))$
for $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ determined by applications of LTS rules with depth of inference 1
($\because (u,v)\in T$ and $\overline{\tau}_{r_Y}\in\overline{\mathcal{R}}_u$ and $u''\in\mathcal{P}$ with transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ and depth of inference 1 are arbitrary)
$\implies$ *Prop*(1) holds (by definition of *Prop*(1)). Q.E.D.

Induction Step: Proof of $\forall \mathbf{n}\in\mathbb{N}^+$ ($\forall \mathbf{m}\in[\mathbf{1},\mathbf{n}]\mathbf{Prop}(\mathbf{m}) \implies \mathbf{Prop}(\mathbf{n}+\mathbf{1})$)

For $n\in\mathbb{N}^+$, assume $\forall m\in[1,n]Prop(m)$ holds (inductive hypothesis).
For $(u,v)\in T$ and $\overline{\tau}_{r_Y}\in\overline{\mathcal{R}}_u$ and $u''\in\mathcal{P}$,
the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ has depth of inference $n+1$
(by the hypothesis of $Prop(n+1)$)
$\implies \overline{\mathcal{R}}_u \neq \emptyset$ (by set theory)
$\implies u\in\mathcal{P}^+$ (by Lemma 4.3.6)
$\implies u\notin\mathcal{P}^0$ (by Theorem 4.3.2)
$\implies u \neq 0$ ($\because 0\in\mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$)
$\implies (u,v)\notin Z$ (by definition of Z)
$\implies (u,v)\in S$ ($\because (u,v)\in T \ \wedge \ T = S \cup Z$)
$\implies \exists p,q,r\in\mathcal{P}\ (p \sim_{dp} q \ \wedge \ u = r|p \ \wedge \ v = r|q)$ (by definition of S)
$\implies (r|p, r|q)\in T$ ($\because (u,v)\in T$).
Now $n+1 \geq 2$ ($\because n\in\mathbb{N}^+$)
$\implies$ only the $L-Par$, $R-Par$ or *CompDelet* rules determine the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$
(by definitions of the LTS rules):

If the $L-Par$ rule defines a transition $r|p \xrightarrow{\overline{\tau}_{r_Y}} r'|p$
then $r \xrightarrow{\overline{\tau}_{r_Y}} r'$ (by the hypothesis of $L-Par$)
$\Longrightarrow r|q \xrightarrow{\overline{\tau}_{r_Y}} r'|q$ (by the $L-Par$ rule) $\wedge$
$(r'|p, r'|q) \in S$ (by definition of S, $\because p, q, r' \in \mathcal{P} \wedge p \sim_{dp} q$)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{r|q}$ (by definition of $\overline{\mathcal{R}}_{r|q}$) $\wedge$ $r'|q \in \mathcal{P}$ (by definition of $r|q \xrightarrow{\overline{\tau}_{r_Y}} r'|q$) $\wedge$
$r|q \xrightarrow{\overline{\tau}_{r_Y}} r'|q$ $\wedge$ $(r'|p, r'|q) \in T$ (by set theory and definition of T)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \wedge r'|q \in \mathcal{P} \wedge v \xrightarrow{\overline{\tau}_{r_Y}} r'|q \wedge (r'|p, r'|q) \in T$ ($\because v = r|q$).

If the $R-Par$ rule defines a transition $r|p \xrightarrow{\overline{\tau}_{r_Y}} r|p'$
then $p \xrightarrow{\overline{\tau}_{r_Y}} p'$ (by the hypothesis of $R-Par$)
$\Longrightarrow q \xrightarrow{\overline{\tau}_{r_Y}} q' \wedge p' \sim_{dp} q'$ ($\because p \sim_{dp} q$)
$\Longrightarrow r|q \xrightarrow{\overline{\tau}_{r_Y}} r|q'$ (by the $R-Par$ rule) $\wedge$
$(r|p', r|q') \in S$ (by definition of S, $\because p', q', r \in \mathcal{P}$)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{r|q}$ (by definition of $\overline{\mathcal{R}}_{r|q}$) $\wedge$ $r|q' \in \mathcal{P}$ (by definition of $r|q \xrightarrow{\overline{\tau}_{r_Y}} r|q'$) $\wedge$
$r|q \xrightarrow{\overline{\tau}_{r_Y}} r|q'$ $\wedge$ $(r|p', r|q') \in T$ (by set theory and definition of T)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \wedge r|q' \in \mathcal{P} \wedge v \xrightarrow{\overline{\tau}_{r_Y}} r|q' \wedge (r|p', r|q') \in T$ ($\because v = r|q$).

If the *CompDelet* rule defines a transition $r|p \xrightarrow{\overline{\tau}_{r_Y}} (r|p)''$
then $(r|p)'' \in \mathcal{P}$ (by definition of $r|p \xrightarrow{\overline{\tau}_{r_Y}} (r|p)''$) $\wedge$
$\exists\, \overline{\tau}_{r_{Y_1}}, \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}\ \exists (r|p)' \in \mathcal{P}\ (Y \sim_{of} Y_1|Y_2 \wedge r|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (r|p)' \wedge (r|p)' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (r|p)'')$
(by the hypothesis of *CompDelet*)
$\Longrightarrow \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{r|p}$ (by definition of $\overline{\mathcal{R}}_{r|p}$) $\wedge$ $\overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(r|p)'}$ (by definition of $\overline{\mathcal{R}}_{(r|p)'}$).
Now the transition $r|p \xrightarrow{\overline{\tau}_{r_Y}} (r|p)''$ has depth of inference $n+1$
(by the hypothesis of $Prop(n+1)$)
$\Longrightarrow$ the transition $r|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (r|p)'$ has depth of inference m_1, with $m_1 \in [1, n]$ $\wedge$
the transition $(r|p)' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (r|p)''$ has depth of inference m_2, with $m_2 \in [1, n]$
($\because$ the transition $r|p \xrightarrow{\overline{\tau}_{r_Y}} (r|p)''$ is inferred from the transitions $r|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (r|p)'$ and $(r|p)' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (r|p)''$ using the *CompDelet* rule)
$\Longrightarrow Prop(m_1)$ and $Prop(m_2)$ hold (by the inductive hypothesis)
$\Longrightarrow \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{r|q} \wedge \exists (r|q)' \in \mathcal{P}\ (r|q \xrightarrow{\overline{\tau}_{r_{Y_1}}} (r|q)' \wedge ((r|p)', (r|q)') \in T) \wedge Prop(m_2)$
(by modus ponens,
$\because (r|p, r|q) \in T \wedge \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{r|p} \wedge (r|p)' \in \mathcal{P} \wedge r|p \xrightarrow{\overline{\tau}_{r_{Y_1}}} (r|p)'$ with depth of inference m_1)

$\Longrightarrow \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(r|q)'} \ \wedge \ \exists (r|q)'' \in \mathcal{P} \ ((r|q)' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (r|q)'' \ \wedge \ ((r|p)'', (r|q)'') \in T)$
(by modus ponens,
$\because \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{(r|p)'} \ \wedge \ (r|p)'' \in \mathcal{P} \ \wedge \ (r|p)' \xrightarrow{\overline{\tau}_{r_{Y_2}}} (r|p)''$ with depth of inference m_2)
$\Longrightarrow (r|q)'' \in \mathcal{P} \ \wedge \ r|q \xrightarrow{\overline{\tau}_{r_Y}} (r|q)'' \ \wedge \ ((r|p)'', (r|q)'') \in T$
(by the *CompDelet* rule, $\because Y \sim_{of} Y_1|Y_2 \ \wedge \ r|q \xrightarrow{\overline{\tau}_{r_{Y_1}}} (r|q)'$)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{r|q}$ (by definition of $\overline{\mathcal{R}}_{r|q}$) $\wedge \ (r|q)'' \in \mathcal{P} \ \wedge \ r|q \xrightarrow{\overline{\tau}_{r_Y}} (r|q)'' \ \wedge \ ((r|p)'', (r|q)'') \in T$
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \ \wedge \ (r|q)'' \in \mathcal{P} \ \wedge \ v \xrightarrow{\overline{\tau}_{r_Y}} (r|q)'' \ \wedge \ ((r|p)'', (r|q)'') \in T$ ($\because v = r|q$).

$\therefore \forall (u, v) \in T \ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u \ \forall u'' \in \mathcal{P}$
$(u \xrightarrow{\overline{\tau}_{r_Y}} u'' \Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \wedge \exists v'' \in \mathcal{P} \ (v \xrightarrow{\overline{\tau}_{r_Y}} v'' \ \wedge \ (u'', v'') \in T))$
for $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ determined by applications of LTS rules with depth of inference $n+1$
($\because (u, v) \in T$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u$ and $u'' \in \mathcal{P}$ with transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ and depth of inference $n+1$ are arbitrary)
$\Longrightarrow Prop(n+1)$ holds (by definition of $Prop(n+1)$).
$\therefore \forall n \in \mathbb{N}^+ \ (\forall m \in [1, n] Prop(m) \Longrightarrow Prop(n+1))$ holds ($\because n \in \mathbb{N}^+$ is arbitrary). Q.E.D.

$\therefore \forall n \in \mathbb{N}^+ \ Prop(n)$ holds (by complete induction).
$\therefore$ For all elements of T the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied
($\because$ every transition of $u \in dom(T)$ in $\overline{\mathcal{R}}$ is determined by applications of LTS rules with finite depth of inference). Q.E.D.

$\therefore T$ is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$, $\because T \subseteq \mathcal{P} \times \mathcal{P}$ and for all elements of T the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied).

A.20.3.3 T^{-1} satisfies the Deletion condition

Let $S' \triangleq \{(r|q, r|p) \mid q, p, r \in \mathcal{P} \ \wedge \ q \sim_{dp} p\}$.
$S' \subseteq \mathcal{P} \times \mathcal{P} \ \wedge \ \forall q, p, r \in \mathcal{P} \ (q \sim_{dp} p \Longrightarrow (r|q, r|p) \in S')$ (by definition of S').
$S^{-1} = \{(r|q, r|p) \mid p, q, r \in \mathcal{P} \ \wedge \ p \sim_{dp} q\}$
(by definitions of S and inverse binary relations)
$\Rightarrow S^{-1} = \{(r|q, r|p) \mid q, p, r \in \mathcal{P} \ \wedge \ q \sim_{dp} p\}$ ($\because \sim_{dp}$ is symmetric, by Lemma 4.3.12)
$\Rightarrow S^{-1} = S'$ (by definition of S').
$Z^{-1} = Z$ (by definitions of Z and inverse binary relations).
Let $T' \triangleq S' \cup Z$.

$T' \subseteq \mathcal{P}$ x $\mathcal{P}$ $\wedge$ $\forall q,p,r \in \mathcal{P}$ $(q \sim_{dp} p \implies (r|q, r|p) \in T')$
(by set theory and definitions of S', Z and T').
$T^{-1} = S^{-1} \cup Z^{-1}$ ($\because$ $T = S \cup Z$ and by algebra of binary relations)
$\Rightarrow T^{-1} = S' \cup Z$ ($\because$ $S^{-1} = S'$ $\wedge$ $Z^{-1} = Z$)
$\Rightarrow T^{-1} = T'$ (by definition of T').
The remainder of the proof is identical to the proof that for all elements of T the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied (see A.20.3.2) with the following substitutions:
p is replaced with q
q with p
p' with q'
q' with p'
u with v
v with u
u'' with v''
v'' with u''
S with S'
T with T'.
Thus, for all elements of T' the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied
($\because$ every transition of $v \in dom(T')$ in $\overline{\mathcal{R}}$ is determined by applications of LTS rules with finite depth of inference)
$\Rightarrow$ for all elements of T^{-1} the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied
($\because$ $T^{-1} = T'$). Q.E.D.

$\therefore$ T^{-1} is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$, $\because$ $T^{-1} \subseteq \mathcal{P}$ x $\mathcal{P}$ and for all elements of T^{-1} the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied)
$\Rightarrow$ T is a strong dp-bisimulation on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$, $\because$ T is a strong dp-simulation on $\mathcal{P}$).
$\therefore$ $\forall p,q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall r \in \mathcal{P}$ $(r|p \sim_{dp} r|q)$) (by definition of $r|p \sim_{dp} r|q$). Q.E.D.

A.20.4 $\forall p,q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall r \in \mathcal{P}$ $(\frac{p}{r} \sim_{dp} \frac{q}{r})$)

Proof: If $\exists$ strong dp-bisimulation T on $\mathcal{P}$ with $\forall r \in \mathcal{P}$ $((\frac{p}{r}, \frac{q}{r}) \in T)$ for any processes p,q in $\mathcal{P}$ such that $p \sim_{dp} q$

then $\forall p,q\in\mathcal{P}$ (if $p\sim_{dp}q$ then $\forall r\in\mathcal{P}$ $(\frac{p}{r}\sim_{dp}\frac{q}{r})$) (by definition of $\frac{p}{r}\sim_{dp}\frac{q}{r}$).
Therefore, we find such a T.

Let $T\triangleq\{(\frac{p}{r},\frac{q}{r}),(p,q)\mid p,q,r\in\mathcal{P}\ \wedge\ p\sim_{dp}q\}$.

A.20.4.1 T is a strong dp-simulation on $\mathcal{P}$

$T\subseteq\mathcal{P}\times\mathcal{P}\ \wedge\ \forall p,q,r\in\mathcal{P}\ (p\sim_{dp}q\implies(\frac{p}{r},\frac{q}{r})\in T)$ (by definition of T).

We prove for $(\frac{p}{r},\frac{q}{r})$ and (p,q) in T that the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied.

$(\frac{\mathbf{p}}{\mathbf{r}},\frac{\mathbf{q}}{\mathbf{r}})$ satisfies the Observation and Fraction conditions

The transitions of $\frac{p}{r}$ in $\mathcal{I}\cup\mathcal{R}$ are defined by the *Creat* rule only
(by the syntax of $\frac{p}{r}$):
If the *Creat* rule defines a transition $\frac{p}{r}\xrightarrow{\tau_{r_W}}p$
then $r\sim_{of}W\ \wedge\ r\in\mathcal{P}^+$ (by the hypothesis of *Creat*)
$\implies\frac{q}{r}\xrightarrow{\tau_{r_W}}q$ (by the *Creat* rule),
and $p\sim_{dp}q$ (by definition of p and q) $\implies(p,q)\in T$ (by definition of T).

$(\frac{\mathbf{p}}{\mathbf{r}},\frac{\mathbf{q}}{\mathbf{r}})$ satisfies the Deletion condition

A transition of $\frac{p}{r}$ in $\overline{\mathcal{R}}$ is defined by *Delet* or *CompDelet* only (by the syntax of $\frac{p}{r}$):
If the *Delet* rule defines a transition $\frac{p}{r}\xrightarrow{\overline{\tau}_{r_X}}p'$
then $p'=0$ (by the *Delet* rule) $\wedge\ \frac{p}{r}\sim_{of}X\ \wedge\ \frac{p}{r}\in\mathcal{P}^+$ (by the hypothesis of *Delet*)
$\implies r\in\mathcal{P}^+$ (by the hypothesis of production rule 4 of $\mathcal{P}^+$)
$\implies\frac{q}{r}\in\mathcal{P}^+$ (by production rule 4 of $\mathcal{P}^+$, $\because q\in\mathcal{P}$).
Now $p\sim_{dp}q$ (by definition of p and q)
$\implies p\sim_{of}q$ (by Corollary 4.3.5)
$\implies\frac{p}{r}\sim_{of}\frac{q}{r}$ (by Theorem 4.3.4, $\because r\in\mathcal{P}$)
$\implies\frac{q}{r}\sim_{of}\frac{p}{r}$ ($\because\sim_{of}$ is symmetric, by Lemma 4.3.2)
$\implies\frac{q}{r}\sim_{of}\frac{p}{r}\ \wedge\ \frac{p}{r}\sim_{of}X$ ($\because\frac{p}{r}\sim_{of}X$)
$\implies\frac{q}{r}\sim_{of}X$ ($\because\sim_{of}$ is transitive, by Lemma 4.3.4)
$\implies\frac{q}{r}\xrightarrow{\overline{\tau}_{r_X}}0$ (by the *Delet* rule, $\because\frac{q}{r}\in\mathcal{P}^+$),
and $(0,0)\in T$ (by definition of T, $\because\sim_{dp}$ is reflexive (by Lemma 4.3.11)).

If the *CompDelet* rule defines a transition $\frac{p}{r} \xrightarrow{\overline{\tau}_{r_Y}} p''$
then $\exists\, \overline{\tau}_{r_{Y_1}}, \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}\ \exists p' \in \mathcal{P}\ (Y \sim_{of} Y_1|Y_2\ \wedge\ \frac{p}{r} \xrightarrow{\overline{\tau}_{r_{Y_1}}} p'\ \wedge\ p' \xrightarrow{\overline{\tau}_{r_{Y_2}}} p'')$
(by the hypothesis of *CompDelet*)
$\implies \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{\frac{p}{r}}$ (by definition of $\overline{\mathcal{R}}_{\frac{p}{r}}$) $\wedge$ $\overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{p'}$ (by definition of $\overline{\mathcal{R}}_{p'}$)
$\implies p' = 0\ \vee\ factors^+_m(p') \subset factors^+_m(\frac{p}{r})$ (by Lemma 4.3.18, $\because \frac{p}{r}, p' \in \mathcal{P}\ \wedge\ \frac{p}{r} \xrightarrow{\overline{\tau}_{r_{Y_1}}} p'$).
If $p' = 0$ then $\overline{\mathcal{R}}_{p'} = \emptyset$ (by Lemma 4.3.8).
But $\overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{p'} \implies \overline{\mathcal{R}}_{p'} \neq \emptyset$ (by set theory; which is a contradiction).
$\therefore p' \neq 0$.
Now $factors^+_m(\frac{p}{r}) \subseteq factors_m(\frac{p}{r})$ (by definition of $factors^+_m(\frac{p}{r})$) $\wedge$
$factors_m(\frac{p}{r}) = \emptyset$ (by definition of $factors_m(\frac{p}{r})$)
$\implies factors^+_m(\frac{p}{r}) \subseteq \emptyset$ (by set theory)
$\implies factors^+_m(\frac{p}{r}) = \emptyset$ (by set theory).
$\therefore$ If $factors^+_m(p') \subset factors^+_m(\frac{p}{r})$ then $factors^+_m(p') \subset \emptyset$ (which is false, by set theory).
$\therefore$ The *CompDelet* rule does not define a transition of $\frac{p}{r}$ in $\overline{\mathcal{R}}$.

$(\mathbf{p}, \mathbf{q})$ satisfies the Observation, Fraction and Deletion conditions

If $p \xrightarrow{\beta} p'$ for some $\beta \in \mathcal{I} \cup \mathcal{R} \cup \overline{\mathcal{R}}$ and some $p' \in \mathcal{P}$
then $q \xrightarrow{\beta} q'$ for some $q' \in \mathcal{P}\ \wedge\ p' \sim_{dp} q'$ ($\because p \sim_{dp} q$, by definition of p and q)
$\implies (p', q') \in T$ (by definition of T).

$\therefore T$ is a strong dp-simulation on $\mathcal{P}$ (by definition of strong dp-simulation on $\mathcal{P}$).
Q.E.D.

A.20.4.2 T is a strong dp-bisimulation on $\mathcal{P}$

We prove T is a strong dp-bisimulation on $\mathcal{P}$ by proving T^{-1} is a strong dp-simulation on $\mathcal{P}$.
Let $T' \triangleq \{(\frac{q}{r}, \frac{p}{r}), (q, p) \mid q, p, r \in \mathcal{P}\ \wedge\ q \sim_{dp} p\}$.
$T^{-1} = \{(\frac{q}{r}, \frac{p}{r}), (q, p) \mid p, q, r \in \mathcal{P}\ \wedge\ p \sim_{dp} q\}$
(by definitions of T and inverse binary relations)
$\Rightarrow T^{-1} = \{(\frac{q}{r}, \frac{p}{r}), (q, p) \mid q, p, r \in \mathcal{P}\ \wedge\ q \sim_{dp} p\}$ ($\because \sim_{dp}$ is symmetric, by Lemma 4.3.12)
$\Rightarrow T^{-1} = T'$ (by definition of T').
The proof that T^{-1} is a strong dp-simulation on $\mathcal{P}$ is identical to the proof that T is a strong dp-simulation on $\mathcal{P}$ (see A.20.4.1) with the following substitutions:
p is replaced with q
q with p

p' with q'
q' with p'
p'' with q''
T with T'.
Thus, T' is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$)
$\Rightarrow T^{-1}$ is a strong dp-simulation on $\mathcal{P}$ ($\because T^{-1} = T'$)
$\Rightarrow T$ is a strong dp-bisimulation on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$, $\because T$ is a strong dp-simulation on $\mathcal{P}$).
Q.E.D.
$\therefore \forall p, q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall r \in \mathcal{P}$ ($\frac{p}{r} \sim_{dp} \frac{q}{r}$)) (by definitions of $\frac{p}{r} \sim_{dp} \frac{q}{r}$ and T). Q.E.D.

A.20.5 $\forall p, q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall r \in \mathcal{P}$ ($\frac{r}{p} \sim_{dp} \frac{r}{q}$))

Proof: If $\exists$ strong dp-bisimulation T on $\mathcal{P}$ with $\forall r \in \mathcal{P}$ ($(\frac{r}{p}, \frac{r}{q}) \in T$) for any processes p, q in $\mathcal{P}$ such that $p \sim_{dp} q$
then $\forall p, q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall r \in \mathcal{P}$ ($\frac{r}{p} \sim_{dp} \frac{r}{q}$)) (by definition of $\frac{r}{p} \sim_{dp} \frac{r}{q}$).
Therefore, we find such a T.

Let $T \triangleq \{(\frac{r}{p}, \frac{r}{q}), (r, r) \mid p, q, r \in \mathcal{P} \ \wedge \ p \sim_{dp} q\}$.

A.20.5.1 T is a strong dp-simulation on $\mathcal{P}$

$T \subseteq \mathcal{P} \times \mathcal{P} \ \wedge \ \forall p, q, r \in \mathcal{P}$ ($p \sim_{dp} q \Longrightarrow (\frac{r}{p}, \frac{r}{q}) \in T$) (by definition of T).

We prove for $(\frac{r}{p}, \frac{r}{q})$ and (r, r) in T that the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied.

$(\frac{\mathbf{r}}{\mathbf{p}}, \frac{\mathbf{r}}{\mathbf{q}})$ satisfies the Observation and Fraction conditions

The transitions of $\frac{r}{p}$ in $\mathcal{I} \cup \mathcal{R}$ are defined by the *Creat* rule only
(by the syntax of $\frac{r}{p}$):
If the *Creat* rule defines a transition $\frac{r}{p} \xrightarrow{\tau_{r_W}} r$
then $p \sim_{of} W \ \wedge \ p \in \mathcal{P}^+$ (by the hypothesis of *Creat*).
Now $p \sim_{dp} q$ (by definition of p and q)
$\Longrightarrow p \sim_{of} q$ (by Corollary 4.3.5)
$\Longrightarrow q \sim_{of} p$ ($\because \sim_{of}$ is symmetric, by Lemma 4.3.2) $\wedge$
$q \in \mathcal{P}^+$ (by Lemma 4.3.9, $\because p \in \mathcal{P}^+$)

$\implies q \sim_{of} p \ \wedge \ p \sim_{of} W \ (\because p \sim_{of} W) \ \wedge \ q \in \mathcal{P}^+$
$\implies q \sim_{of} W \ (\because \sim_{of}$ is transitive, by Lemma 4.3.4) $\wedge \ q \in \mathcal{P}^+$
$\implies \frac{r}{q} \xrightarrow{\tau_{r_W}} r$ (by the *Creat* rule),
and $(r, r) \in T$ (by definition of T).

$(\frac{\mathbf{r}}{\mathbf{p}}, \frac{\mathbf{r}}{\mathbf{q}})$ satisfies the Deletion condition

A transition of $\frac{r}{p}$ in $\overline{\mathcal{R}}$ is defined by *Delet* or *CompDelet* only (by the syntax of $\frac{r}{p}$):

If the *Delet* rule defines a transition $\frac{r}{p} \xrightarrow{\overline{\tau}_{r_X}} p'$
then $p' = 0$ (by the *Delet* rule) $\wedge$
$\frac{r}{p} \sim_{of} X \ \wedge \ \frac{r}{p} \in \mathcal{P}^+$ (by the hypothesis of *Delet*).
Now $p \sim_{dp} q$ (by definition of p and q)
$\implies p \sim_{of} q$ (by Corollary 4.3.5)
$\implies \frac{r}{p} \sim_{of} \frac{r}{q}$ (by Theorem 4.3.4, $\because r \in \mathcal{P}$)
$\implies \frac{r}{q} \sim_{of} \frac{r}{p}$ ($\because \sim_{of}$ is symmetric, by Lemma 4.3.2) $\wedge$
$\frac{r}{q} \in \mathcal{P}^+$ (by Lemma 4.3.9, $\because \frac{r}{p} \in \mathcal{P}^+$)
$\implies \frac{r}{q} \sim_{of} \frac{r}{p} \ \wedge \ \frac{r}{p} \sim_{of} X \ (\because \frac{r}{p} \sim_{of} X) \ \wedge \ \frac{r}{q} \in \mathcal{P}^+$
$\implies \frac{r}{q} \sim_{of} X \ (\because \sim_{of}$ is transitive, by Lemma 4.3.4) $\wedge \ \frac{r}{q} \in \mathcal{P}^+$
$\implies \frac{r}{q} \xrightarrow{\overline{\tau}_{r_X}} 0$ (by the *Delet* rule),
and $(0, 0) \in T$ (by definition of T).

If the *CompDelet* rule defines a transition $\frac{r}{p} \xrightarrow{\overline{\tau}_{r_Y}} p''$
then $\exists \overline{\tau}_{r_{Y_1}}, \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}} \ \exists p' \in \mathcal{P} \ (Y \sim_{of} Y_1 | Y_2 \ \wedge \ \frac{r}{p} \xrightarrow{\overline{\tau}_{r_{Y_1}}} p' \ \wedge \ p' \xrightarrow{\overline{\tau}_{r_{Y_2}}} p'')$
(by the hypothesis of *CompDelet*)
$\implies \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_{\frac{r}{p}}$ (by definition of $\overline{\mathcal{R}}_{\frac{r}{p}}$) $\wedge \ \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{p'}$ (by definition of $\overline{\mathcal{R}}_{p'}$)
$\implies p' = 0 \ \vee \ factors^+_m(p') \subset factors^+_m(\frac{r}{p})$ (by Lemma 4.3.18, $\because \frac{r}{p}, p' \in \mathcal{P} \ \wedge \ \frac{r}{p} \xrightarrow{\overline{\tau}_{r_{Y_1}}} p'$).
If $p' = 0$
then $\overline{\mathcal{R}}_{p'} = \emptyset$ (by Lemma 4.3.8).
But $\overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{p'}$
$\implies \overline{\mathcal{R}}_{p'} \neq \emptyset$ (by set theory; which is a contradiction).
$\therefore p' \neq 0$.
Now $factors^+_m(\frac{r}{p}) \subseteq factors_m(\frac{r}{p})$ (by definition of $factors^+_m(\frac{r}{p})$) $\wedge$
$factors_m(\frac{r}{p}) = \emptyset$ (by definition of $factors_m(\frac{r}{p})$)
$\implies factors^+_m(\frac{r}{p}) \subseteq \emptyset$ (by set theory)

$\implies factors^{+}_{m}(\frac{r}{p}) = \emptyset$ (by set theory).
$\therefore$ If $factors^{+}_{m}(p') \subset factors^{+}_{m}(\frac{r}{p})$
then $factors^{+}_{m}(p') \subset \emptyset$ (which is false, by set theory).
$\therefore$ The *CompDelet* rule does not define a transition of $\frac{r}{p}$ in $\overline{\mathcal{R}}$.

(r, r) satisfies the Observation, Fraction and Deletion conditions

If $r \xrightarrow{\gamma} r'$ for some $\gamma \in \mathcal{I} \cup \mathcal{R} \cup \overline{\mathcal{R}}$ and some $r' \in \mathcal{P}$
then $r \xrightarrow{\gamma} r'$
and $(r', r') \in T$ (by definition of T).

$\therefore$ T is a strong dp-simulation on $\mathcal{P}$ (by definition of strong dp-simulation on $\mathcal{P}$).
Q.E.D.

A.20.5.2 T is a strong dp-bisimulation on $\mathcal{P}$

We prove T is a strong dp-bisimulation on $\mathcal{P}$ by proving T^{-1} is a strong dp-simulation on $\mathcal{P}$.
Let $T' \triangleq \{(\frac{r}{q}, \frac{r}{p}), (r, r) \mid q, p, r \in \mathcal{P} \ \wedge \ q \sim_{dp} p\}$.
$T^{-1} = \{(\frac{r}{q}, \frac{r}{p}), (r, r) \mid p, q, r \in \mathcal{P} \ \wedge \ p \sim_{dp} q\}$
(by definitions of T and inverse binary relations)
$\Rightarrow T^{-1} = \{(\frac{r}{q}, \frac{r}{p}), (r, r) \mid q, p, r \in \mathcal{P} \ \wedge \ q \sim_{dp} p\}$ ($\because \sim_{dp}$ is symmetric, by Lemma 4.3.12)
$\Rightarrow T^{-1} = T'$ (by definition of T').
The proof that T^{-1} is a strong dp-simulation on $\mathcal{P}$ is identical to the proof that T is a strong dp-simulation on $\mathcal{P}$ (see A.20.5.1) with the following substitutions:
p is replaced with q
q with p
p' with q'
q' with p'
p'' with q''
T with T'.
Thus, T' is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$)
$\Rightarrow T^{-1}$ is a strong dp-simulation on $\mathcal{P}$ ($\because T^{-1} = T'$)
$\Rightarrow T$ is a strong dp-bisimulation on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$, $\because T$ is a strong dp-simulation on $\mathcal{P}$).
Q.E.D.
$\therefore \forall p, q \in \mathcal{P}$ (if $p \sim_{dp} q$ then $\forall r \in \mathcal{P}$ $(\frac{r}{p} \sim_{dp} \frac{r}{q})$) (by definitions of $\frac{r}{p} \sim_{dp} \frac{r}{q}$ and T). Q.E.D.

A.21 Lemma 4.3.22 $\forall z \in \mathcal{P}^0\ \forall p \in \mathcal{P}\ (z \rightsquigarrow_{dp} p)$

Proof: consists of defining a binary relation T on $\mathcal{P}$ with $\forall z \in \mathcal{P}^0\ \forall p \in \mathcal{P}\ ((z,p) \in T)$, then proving T is a strong dp-simulation on $\mathcal{P}$.

Let $T \triangleq \{(z,p) \mid z \in \mathcal{P}^0\ \wedge\ p \in \mathcal{P}\}$.
$T \subseteq \mathcal{P}$ x $\mathcal{P}$ $\wedge$ $\forall z \in \mathcal{P}^0\ \forall p \in \mathcal{P}\ ((z,p) \in T)$ (by definition of T).

For $(z,p) \in T$, $z \in \mathcal{P}^0\ \wedge\ p \in \mathcal{P}$ (by definition of T)
$\Longrightarrow \mathcal{I}_z \cup \mathcal{R}_z = \emptyset$ (by Lemma 4.3.7) $\wedge\ \overline{\mathcal{R}}_z = \emptyset$ (by Lemma 4.3.8)
$\Longrightarrow \mathcal{I}_z = \emptyset \wedge \mathcal{R}_z = \emptyset \wedge \overline{\mathcal{R}}_z = \emptyset$ (by set theory)
$\Longrightarrow$ the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
($\because \emptyset$ satisfies all conditions).
$\therefore \forall (z,p) \in T$ the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
($\because (z,p) \in T$ is arbitrary)
$\Longrightarrow T$ is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$, $\because T \subseteq \mathcal{P}$ x $\mathcal{P}$)
$\Longrightarrow \forall z \in \mathcal{P}^0\ \forall p \in \mathcal{P}\ (z \rightsquigarrow_{dp} p)$ (by definition of $z \rightsquigarrow_{dp} p$, $\because \forall z \in \mathcal{P}^0\ \forall p \in \mathcal{P}\ ((z,p) \in T)$).
Q.E.D.

A.22 Lemma 4.3.23

$$\forall p,q,r,s \in \mathcal{P}\ (p \rightsquigarrow_{dp} q\ \wedge\ r \rightsquigarrow_{dp} s \Longrightarrow p|r \rightsquigarrow_{dp} q|s)$$

Sketch proof: consists of defining a binary relation T on $\mathcal{P}$ with $\forall p,q,r,s \in \mathcal{P}\ ((p|r,q|s) \in T)$ such that $p \rightsquigarrow_{dp} q\ \wedge\ r \rightsquigarrow_{dp} s$, then proving T is a strong dp-simulation on $\mathcal{P}$.

Let $S \triangleq \{(p|r,q|s) \mid p,q,r,s \in \mathcal{P}\ (p \rightsquigarrow_{dp} q\ \wedge\ r \rightsquigarrow_{dp} s)\}$.
Let $Z \triangleq \{(z,t) \mid z \in \mathcal{P}^0\ \wedge\ t \in \mathcal{P}\}$.
Let $T \triangleq S \cup Z$.
$T \subseteq \mathcal{P}$ x $\mathcal{P}$ (by set theory and definitions of S, Z and T).
$\forall p,q,r,s \in \mathcal{P}\ (p \rightsquigarrow_{dp} q\ \wedge\ r \rightsquigarrow_{dp} s \Longrightarrow (p|r,q|s) \in S)$ (by definition of S)
$\Longrightarrow \forall p,q,r,s \in \mathcal{P}\ (p \rightsquigarrow_{dp} q\ \wedge\ r \rightsquigarrow_{dp} s \Longrightarrow (p|r,q|s) \in T)$
(by set theory and definition of T).

It can be shown that T satisfies the *Observation* and *Fraction* conditions, by considering transitions in $\mathcal{I} \cup \mathcal{R}$ defined by the $L-Par$, $R-Par$, $React$, $L-React$ or $R-React$ rules.

It can be shown that T satisfies the *Deletion* condition by complete induction on the depth of inference of the applications of the LTS rules (i.e. *Delet*, $L-Par$, $R-Par$ or *CompDelet*) that determine the transitions of $u \in dom(T)$ in $\overline{\mathcal{R}}$.

A.23 Lemma 4.3.24

$$\forall p, p' \in \mathcal{P}\ \forall \overline{\tau}_{r_X} \in \overline{R}_p\ (p \xrightarrow{\overline{\tau}_{r_X}} p' \implies p' \rightsquigarrow_{dp} p)$$

Proof: uses complete induction on the depth of inference of the applications of the LTS rules that determine the transitions of p in $\overline{\mathcal{R}}$.

For $n \in \mathbb{N}^+$, let $Prop(n)$ be this lemma for $p \xrightarrow{\overline{\tau}_{r_X}} p'$ determined by applications of LTS rules with depth of inference n.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(1)$
2. $\vdash \forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \implies Prop(n+1))$

Base Case: Proof of Prop(1)

For $p, p' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$, the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference 1
(by the hypothesis of $Prop(1)$)
$\implies$ only the *Delet* rule determines the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
(by definitions of the LTS rules):

If the *Delet* rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
then $p' = 0$ (by the *Delet* rule)
$\implies p' \in \mathcal{P}^0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$)
$\implies p' \rightsquigarrow_{dp} p$ (by Lemma 4.3.22, $\because p \in \mathcal{P}$).
$\therefore \forall p, p' \in \mathcal{P}\ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p\ (p \xrightarrow{\overline{\tau}_{r_X}} p' \implies p' \rightsquigarrow_{dp} p)$
for $p \xrightarrow{\overline{\tau}_{r_X}} p'$ determined by applications of LTS rules with depth of inference 1

($\because$ $p, p' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$ with transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ and depth of inference 1 are arbitrary)
$\implies$ *Prop*(1) holds (by definition of *Prop*(1)). Q.E.D.

Induction Step: Proof of $\forall \mathbf{n} \in \mathbb{N}^+\ (\forall \mathbf{m} \in [\mathbf{1}, \mathbf{n}]\mathbf{Prop}(\mathbf{m}) \implies \mathbf{Prop}(\mathbf{n+1}))$

For $n \in \mathbb{N}^+$, assume $\forall m \in [1, n]Prop(m)$ holds (inductive hypothesis).
For $p, p' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$, the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference $n+1$
(by the hypothesis of $Prop(n+1)$)
$\implies p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference ≥ 2 ($\because n+1 \geq 2$ ($\because n \in \mathbb{N}^+$))
$\implies$ only the $L-Par$, $R-Par$ or *CompDelet* rules determine the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
(by definitions of the LTS rules):

If the $L-Par$ rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
then $\exists r, r', s \in \mathcal{P}\ (p = r|s\ \wedge\ p' = r'|s\ \wedge\ r|s \xrightarrow{\overline{\tau}_{r_X}} r'|s)$ (by the $L-Par$ rule)
$\implies r \xrightarrow{\overline{\tau}_{r_X}} r'$ (by the hypothesis of $L-Par$)
$\implies \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_r$ (by definition of $\overline{\mathcal{R}}_r$) $\wedge\ r \xrightarrow{\overline{\tau}_{r_X}} r'$
$\implies r' \rightsquigarrow_{dp} r$ (by Lemma 4.3.24, $\because r, r' \in \mathcal{P}$).
Now $s \sim_{dp} s$ ($\because \sim_{dp}$ is reflexive, by Lemma 4.3.11)
$\implies \exists$ strong dp-bisimulation T on $\mathcal{P}$ with $(s, s) \in T$ (by definition of $s \sim_{dp} s$)
$\implies T$ is a strong dp-simulation on $\mathcal{P}$ with $(s, s) \in T$
(by definition of strong dp-bisimulation on $\mathcal{P}$)
$\implies s \rightsquigarrow_{dp} s$ (by definition of $s \rightsquigarrow_{dp} s$)
$\implies r' \rightsquigarrow_{dp} r\ \wedge\ s \rightsquigarrow_{dp} s$ ($\because r' \rightsquigarrow_{dp} r$)
$\implies r'|s \rightsquigarrow_{dp} r|s$ (by Lemma 4.3.23, $\because r', r, s \in \mathcal{P}$)
$\implies p' \rightsquigarrow_{dp} p$ ($\because p' = r'|s\ \wedge\ p = r|s$).

If the $R-Par$ rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
then $\exists r, s, s' \in \mathcal{P}\ (p = r|s\ \wedge\ p' = r|s'\ \wedge\ r|s \xrightarrow{\overline{\tau}_{r_X}} r|s')$ (by the $R-Par$ rule)
$\implies s \xrightarrow{\overline{\tau}_{r_X}} s'$ (by the hypothesis of $R-Par$)
$\implies \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_s$ (by definition of $\overline{\mathcal{R}}_s$) $\wedge\ s \xrightarrow{\overline{\tau}_{r_X}} s'$
$\implies s' \rightsquigarrow_{dp} s$ (by Lemma 4.3.24, $\because s, s' \in \mathcal{P}$).
Now $r \sim_{dp} r$ ($\because \sim_{dp}$ is reflexive, by Lemma 4.3.11)
$\implies \exists$ strong dp-bisimulation T on $\mathcal{P}$ with $(r, r) \in T$ (by definition of $r \sim_{dp} r$)
$\implies T$ is a strong dp-simulation on $\mathcal{P}$ with $(r, r) \in T$
(by definition of strong dp-bisimulation on $\mathcal{P}$)

$\implies r \rightsquigarrow_{dp} r$ (by definition of $r \rightsquigarrow_{dp} r$)
$\implies r \rightsquigarrow_{dp} r\ \wedge\ s' \rightsquigarrow_{dp} s$ ($\because s' \rightsquigarrow_{dp} s$)
$\implies r|s' \rightsquigarrow_{dp} r|s$ (by Lemma 4.3.23, $\because r, s', s \in \mathcal{P}$)
$\implies p' \rightsquigarrow_{dp} p$ ($\because p' = r|s'\ \wedge\ p = r|s$).

If the *CompDelet* rule defines a transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$
then $\exists\, \overline{\tau}_{r_{X_1}}, \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}\ \exists u \in \mathcal{P}\ (X \sim_{of} X_1|X_2\ \wedge\ p \xrightarrow{\overline{\tau}_{r_{X_1}}} u\ \wedge\ u \xrightarrow{\overline{\tau}_{r_{X_2}}} p')$
(by the hypothesis of *CompDelet*)
$\implies \overline{\tau}_{r_{X_1}} \in \overline{\mathcal{R}}_p$ (by definition of $\overline{\mathcal{R}}_p$) $\wedge$ $\overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}_u$ (by definition of $\overline{\mathcal{R}}_u$).
Now the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ has depth of inference $n+1$
(by the hypothesis of $Prop(n+1)$)
$\implies$ the transition $p \xrightarrow{\overline{\tau}_{r_{X_1}}} u$ has depth of inference m_p, with $m_p \in [1, n]\ \wedge$
the transition $u \xrightarrow{\overline{\tau}_{r_{X_2}}} p'$ has depth of inference m_u, with $m_u \in [1, n]$
($\because$ the transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ is inferred from the transitions $p \xrightarrow{\overline{\tau}_{r_{X_1}}} u$ and $u \xrightarrow{\overline{\tau}_{r_{X_2}}} p'$ using the *CompDelet* rule)
$\implies Prop(m_p)$ and $Prop(m_u)$ hold (by the inductive hypothesis)
$\implies p' \rightsquigarrow_{dp} u$
(by modus ponens, $\because u, p' \in \mathcal{P}\ \wedge\ \overline{\tau}_{r_{X_2}} \in \overline{\mathcal{R}}_u\ \wedge\ u \xrightarrow{\overline{\tau}_{r_{X_2}}} p'$ with depth of inference m_u) $\wedge$
$u \rightsquigarrow_{dp} p$
(by modus ponens, $\because p, u \in \mathcal{P}\ \wedge\ \overline{\tau}_{r_{X_1}} \in \overline{\mathcal{R}}_p\ \wedge\ p \xrightarrow{\overline{\tau}_{r_{X_1}}} u$ with depth of inference m_p)
$\implies p' \rightsquigarrow_{dp} p$ ($\because \rightsquigarrow_{dp}$ is transitive, by Lemma 4.3.6).

$\therefore\ \forall p, p' \in \mathcal{P}\ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p\ (p \xrightarrow{\overline{\tau}_{r_X}} p' \implies p' \rightsquigarrow_{dp} p)$
for $p \xrightarrow{\overline{\tau}_{r_X}} p'$ determined by applications of LTS rules with depth of inference $n+1$
($\because p, p' \in \mathcal{P}$ and $\overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p$ with transition $p \xrightarrow{\overline{\tau}_{r_X}} p'$ and depth of inference $n+1$ are arbitrary)
$\implies Prop(n+1)$ holds (by definition of $Prop(n+1)$).
$\therefore\ \forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \implies Prop(n+1))$ holds ($\because n \in \mathbb{N}^+$ is arbitrary). Q.E.D.

$\therefore\ \forall n \in \mathbb{N}^+\ Prop(n)$ holds (by complete induction).
$\therefore\ \forall p, p' \in \mathcal{P}\ \forall \overline{\tau}_{r_X} \in \overline{\mathcal{R}}_p\ (p \xrightarrow{\overline{\tau}_{r_X}} p' \implies p' \rightsquigarrow_{dp} p)$
($\because$ every transition of every $p \in \mathcal{P}$ is a result of one or more applications of the LTS semantic rules with finite depth of inference). Q.E.D.

A.24 Lemma 4.3.25 $\forall p,q \in \mathcal{P}$ $(sfdrdepth(p|q) = max\{sfdrdepth(p), sfdrdepth(q)\})$

Sketch proof: $sfdrdepth(p|q)$ is determined from a transition in $\mathcal{R}$ that is performed by either p or q; and $sfdrdepth(p)$ is determined by a transition in $\mathcal{R}$ that is performed by $p|q$ (by the $L-Par$ rule); and $sfdrdepth(q)$ is determined by a transition in $\mathcal{R}$ that is performed by $p|q$ (by the $R-Par$ rule). The result follows.

A.25 Theorem 4.3.11 $\forall p,q \in \mathcal{P}$ $(p|q \sim_{of} q|p)$

Sketch proof: uses complete induction on $sfdrdepth(p|q)$.
The rules to consider are $L-Par$, $R-Par$, $React$, $L-React$ and $R-React$.
The transitions in $L-Par$, $R-Par$ and $React$ are straightforward.
The $L-React$ and $R-React$ rule applications require $X_1|X_2 \sim_{of} X_2|X_1$, for which the inductive hypothesis is needed.

A.26 Theorem 4.3.12 $\forall p,q \in \mathcal{P}$ $(p|q \sim_{dp} q|p)$

Proof: If $\exists$ strong dp-bisimulation T on $\mathcal{P}$ with $\forall p,q \in \mathcal{P}$ $((p|q,q|p) \in T)$
then $\forall p,q \in \mathcal{P}$ $(p|q \sim_{dp} q|p)$ (by definition of $p|q \sim_{dp} q|p$).
Therefore, we find such a T.

Let $S \triangleq \{(p|q,q|p) \mid p,q \in \mathcal{P}\}$.
$S \subseteq \mathcal{P} \times \mathcal{P}$
(by definition of S, and by Theorem 4.3.2, production rule 3 of $\mathcal{P}^+$, production rule 3 of $\mathcal{P}^0$ and set theory) $\wedge$
$\forall p,q \in \mathcal{P}$ $((p|q,q|p) \in S)$ (by definition of S).
Let $Z \triangleq \{(0,0)\}$.
$Z \subseteq \mathcal{P} \times \mathcal{P}$
(by definition of Z, and by Theorem 4.3.2, production rule 1 of $\mathcal{P}^0$ and set theory).
Let $T \triangleq S \cup Z$.

T is a strong dp-bisimulation on $\mathcal{P}$
$\Longleftrightarrow$ T, T^{-1} are strong dp-simulations on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$)

$\iff$ T is a binary relation on $\mathcal{P}$ $\wedge$
for all elements of T the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
(by definition of strong dp-simulation on $\mathcal{P}$) $\wedge$
T^{-1} is a strong dp-simulation on $\mathcal{P}$.
We prove T is a binary relation on $\mathcal{P}$ and for all elements of T the *Observation*, *Fraction* and *Deletion* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied, then prove T^{-1} is a strong dp-simulation on $\mathcal{P}$.

A.26.1 T satisfies the Observation and Fraction conditions

S is a strong of-bisimulation on $\mathcal{P}$
(by the proof of Theorem 4.3.11, see A.25) $\wedge$
Z is a strong of-bisimulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$, $\because \mathcal{I}_0 \cup \mathcal{R}_0 = \emptyset$ (by Lemma 4.3.7))
$\implies$ T is a strong of-bisimulation on $\mathcal{P}$
($\because$ the union of strong of-bisimulations on $\mathcal{P}$ is a strong of-bisimulation on $\mathcal{P}$, and by definition of T)
$\implies$ T is a strong of-simulation on $\mathcal{P}$
(by definition of strong of-bisimulation on $\mathcal{P}$)
$\implies$ $T \subseteq \mathcal{P}$ x $\mathcal{P}$ $\wedge$ for all elements of T the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$ are satisfied
(by definition of strong of-simulation on $\mathcal{P}$)
$\implies$ for all elements of T the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied
($\because$ the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are the same as the *Observation* and *Fraction* conditions of strong of-simulation on $\mathcal{P}$, respectively). Q.E.D.
And $\forall p, q \in \mathcal{P}\ ((p|q, q|p) \in T)$ (by set theory and definitions of S and T).

It remains to prove that for all elements of T the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied.

A.26.2 T satisfies the Deletion condition

We use complete induction on the depth of inference of the applications of the LTS rules that determine the transitions of $u \in dom(T)$ in $\overline{\mathcal{R}}$.

For $n \in \mathbb{N}^+$, let $Prop(n)$ be the proposition:
$\forall (u, v) \in T\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u\ \forall u'' \in \mathcal{P}$
$(u \xrightarrow{\overline{\tau}_{r_Y}} u'' \implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \wedge \exists v'' \in \mathcal{P}\ (v \xrightarrow{\overline{\tau}_{r_Y}} v''\ \wedge\ (u'', v'') \in T))$
for $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ determined by applications of LTS rules with depth of inference n.

The proof by complete induction involves discharging the following two proof obligations:

1. $\vdash Prop(1)$
2. $\vdash \forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \implies Prop(n+1))$

Base Case: Proof of Prop(1)

For $(u, v) \in T$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u$ and $u'' \in \mathcal{P}$,
the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ has depth of inference 1 (by the hypothesis of $Prop(1)$)
$\implies$ only the *Delet* rule determines the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$
(by definitions of the LTS rules):

Now $u \in \mathcal{P}\ \wedge\ v \in \mathcal{P}$ ($\because (u, v) \in T\ \wedge\ T \subseteq \mathcal{P}\ \mathrm{x}\ \mathcal{P}$, and by set theory).
If $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ (by the *Delet* rule)
then $u'' = 0$ (by the *Delet* rule) $\wedge$ $u \sim_{of} Y$ $\wedge$ $u \in \mathcal{P}^+$ (by the hypothesis of *Delet*)
$\implies u \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because u \in \mathcal{P}$)
$\implies u \neq 0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$)
$\implies (u, v) \notin Z$ (by definition of Z)
$\implies (u, v) \in S$ ($\because (u, v) \in T\ \wedge\ T = S \cup Z$)
$\implies \exists p, q \in \mathcal{P}\ (u = p|q\ \wedge\ v = q|p)$ (by definition of S)
$\implies p|q \sim_{of} q|p$ (by Theorem 4.3.11) $\wedge$ $p|q \sim_{of} Y$ ($\because u \sim_{of} Y$) $\wedge$
$p|q, q|p \in \mathcal{P}$ ($\because u, v \in \mathcal{P}$) $\wedge$ $p|q \in \mathcal{P}^+$ ($\because u \in \mathcal{P}^+$) $\wedge$ $p|q \notin \mathcal{P}^0$ ($\because u \notin \mathcal{P}^0$)
$\implies q|p \sim_{of} p|q$ ($\because \sim_{of}$ is symmetric, by Lemma 4.3.2) $\wedge$ $p|q \sim_{of} Y$ $\wedge$
$q|p \in \mathcal{P}^+$ (by Lemma 4.3.9, $\because p|q, q|p \in \mathcal{P}\ \wedge\ p|q \notin \mathcal{P}^0$)
$\implies q|p \sim_{of} Y$ ($\because \sim_{of}$ is transitive, by Lemma 4.3.4) $\wedge$ $q|p \in \mathcal{P}^+$
$\implies q|p \xrightarrow{\overline{\tau}_{r_Y}} 0$ (by the *Delet* rule)
$\implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{q|p}$ (by definition of $\overline{\mathcal{R}}_{q|p}$) $\wedge$
$0 \in \mathcal{P}$ (by production rule 1 of $\mathcal{P}^0$, set theory and definition of $\mathcal{P}$) $\wedge$ $q|p \xrightarrow{\overline{\tau}_{r_Y}} 0$ $\wedge$
$(0, 0) \in T$ (by set theory and definitions of Z and T)
$\implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v\ \wedge\ 0 \in \mathcal{P}\ \wedge\ v \xrightarrow{\overline{\tau}_{r_Y}} 0\ \wedge\ (0, 0) \in T$ ($\because v = q|p$).

$\therefore \forall (u,v) \in T\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u\ \forall u'' \in \mathcal{P}$
$(u \xrightarrow{\overline{\tau}_{r_Y}} u'' \Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \wedge \exists v'' \in \mathcal{P}\ (v \xrightarrow{\overline{\tau}_{r_Y}} v''\ \wedge\ (u'', v'') \in T))$
for $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ determined by applications of LTS rules with depth of inference 1
($\because (u,v) \in T$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u$ and $u'' \in \mathcal{P}$ with transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ and depth of inference 1 are arbitrary)
$\Longrightarrow$ *Prop*(1) holds (by definition of *Prop*(1)). Q.E.D.

Induction Step: Proof of $\forall \mathbf{n} \in \mathbb{N}^+\ (\forall \mathbf{m} \in [\mathbf{1}, \mathbf{n}]\mathbf{Prop}(\mathbf{m}) \Longrightarrow \mathbf{Prop}(\mathbf{n+1}))$

For $n \in \mathbb{N}^+$, assume $\forall m \in [1, n] Prop(m)$ holds (inductive hypothesis).
For $(u,v) \in T$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u$ and $u'' \in \mathcal{P}$,
the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ has depth of inference $n+1$
(by the hypothesis of $Prop(n+1)$)
$\Longrightarrow u \in \mathcal{P} \wedge v \in \mathcal{P}$ ($\because T \subseteq \mathcal{P} \times \mathcal{P}$, and by set theory) $\wedge\ \overline{\mathcal{R}}_u \neq \emptyset$ (by set theory)
$\Longrightarrow u \in \mathcal{P}^+$ (by Lemma 4.3.6)
$\Longrightarrow u \notin \mathcal{P}^0$ (by Theorem 4.3.2, $\because u \in \mathcal{P}$)
$\Longrightarrow u \neq 0$ ($\because 0 \in \mathcal{P}^0$, by production rule 1 of $\mathcal{P}^0$)
$\Longrightarrow (u,v) \notin Z$ (by definition of Z)
$\Longrightarrow (u,v) \in S$ ($\because (u,v) \in T \wedge T = S \cup Z$)
$\Longrightarrow \exists p, q \in \mathcal{P}\ (u = p|q \wedge v = q|p)$ (by definition of S)
$\Longrightarrow (p|q, q|p) \in T$ ($\because (u,v) \in T$).
Now $n+1 \geq 2$ ($\because n \in \mathbb{N}^+$)
$\Longrightarrow$ only the $L-Par$, $R-Par$ or *CompDelet* rules determine the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$
(by definitions of the LTS rules):

If the $L-Par$ rule defines the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$
then $\exists\, p' \in \mathcal{P}\ (p \xrightarrow{\overline{\tau}_{r_Y}} p' \wedge u'' = p'|q \wedge p|q \xrightarrow{\overline{\tau}_{r_Y}} p'|q)$
($\because u = p|q$, and by definition of the $L-Par$ rule)
$\Longrightarrow q|p \xrightarrow{\overline{\tau}_{r_Y}} q|p'$ (by the conclusion of the $R-Par$ rule)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{q|p}$ (by definition of $\overline{\mathcal{R}}_{q|p}$) $\wedge$ $q|p' \in \mathcal{P}$ (by definition of $q|p \xrightarrow{\overline{\tau}_{r_Y}} q|p'$) $\wedge$
$q|p \xrightarrow{\overline{\tau}_{r_Y}} q|p'$ $\wedge$
$(p'|q, q|p') \in S$ (by definition of S, $\because p', q \in \mathcal{P}$)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{q|p} \wedge q|p' \in \mathcal{P} \wedge q|p \xrightarrow{\overline{\tau}_{r_Y}} q|p' \wedge$
$(p'|q, q|p') \in T$ (by set theory and definition of T)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \wedge q|p' \in \mathcal{P} \wedge v \xrightarrow{\overline{\tau}_{r_Y}} q|p' \wedge (p'|q, q|p') \in T$ ($\because v = q|p$).

If the $R-Par$ rule defines the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$
then $\exists\, q' \in \mathcal{P}\,(q \xrightarrow{\overline{\tau}_{r_Y}} q' \ \wedge\ u'' = p|q' \ \wedge\ p|q \xrightarrow{\overline{\tau}_{r_Y}} p|q')$
($\because u = p|q$, and by definition of the $R-Par$ rule)
$\Longrightarrow q|p \xrightarrow{\overline{\tau}_{r_Y}} q'|p$ (by the conclusion of the $L-Par$ rule)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{q|p}$ (by definition of $\overline{\mathcal{R}}_{q|p}$) $\wedge$ $q'|p \in \mathcal{P}$ (by definition of $q|p \xrightarrow{\overline{\tau}_{r_Y}} q'|p$) $\wedge$
$q|p \xrightarrow{\overline{\tau}_{r_Y}} q'|p \ \wedge$
$(p|q', q'|p) \in S$ (by definition of S, $\because p, q' \in \mathcal{P}$)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_{q|p} \ \wedge\ q'|p \in \mathcal{P} \ \wedge\ q|p \xrightarrow{\overline{\tau}_{r_Y}} q'|p \ \wedge$
$(p|q', q'|p) \in T$ (by set theory and definition of T)
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \ \wedge\ q'|p \in \mathcal{P} \ \wedge\ v \xrightarrow{\overline{\tau}_{r_Y}} q'|p \ \wedge\ (p|q', q'|p) \in T$ ($\because v = q|p$).

If the *CompDelet* rule defines the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$
then $\exists\, \overline{\tau}_{r_{Y_1}}, \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}\ \exists u' \in \mathcal{P}\,(Y \sim_{of} Y_1|Y_2 \ \wedge\ u \xrightarrow{\overline{\tau}_{r_{Y_1}}} u' \ \wedge\ u' \xrightarrow{\overline{\tau}_{r_{Y_2}}} u'')$
(by the hypothesis of *CompDelet*)
$\Longrightarrow \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_u$ (by definition of $\overline{\mathcal{R}}_u$) $\wedge$ $\overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{u'}$ (by definition of $\overline{\mathcal{R}}_{u'}$).
Now the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ has depth of inference $n+1$
(by the hypothesis of $Prop(n+1)$)
$\Longrightarrow$ the transition $u \xrightarrow{\overline{\tau}_{r_{Y_1}}} u'$ has depth of inference m_1, with $m_1 \in [1, n]$ $\wedge$
the transition $u' \xrightarrow{\overline{\tau}_{r_{Y_2}}} u''$ has depth of inference m_2, with $m_2 \in [1, n]$
($\because$ the transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ is inferred from the transitions $u \xrightarrow{\overline{\tau}_{r_{Y_1}}} u'$ and $u' \xrightarrow{\overline{\tau}_{r_{Y_2}}} u''$ using the *CompDelet* rule)
$\Longrightarrow Prop(m_1)$ and $Prop(m_2)$ hold (by the inductive hypothesis)
$\Longrightarrow \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_v \ \wedge\ \exists v' \in \mathcal{P}\,(v \xrightarrow{\overline{\tau}_{r_{Y_1}}} v' \ \wedge\ (u', v') \in T) \ \wedge\ Prop(m_2)$
(by modus ponens, $\because (u, v) \in T \ \wedge\ \overline{\tau}_{r_{Y_1}} \in \overline{\mathcal{R}}_u \ \wedge\ u' \in \mathcal{P} \ \wedge\ u \xrightarrow{\overline{\tau}_{r_{Y_1}}} u'$ with depth of inference m_1)
$\Longrightarrow \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{v'} \ \wedge\ \exists v'' \in \mathcal{P}\,(v' \xrightarrow{\overline{\tau}_{r_{Y_2}}} v'' \ \wedge\ (u'', v'') \in T)$
(by modus ponens, $\because \overline{\tau}_{r_{Y_2}} \in \overline{\mathcal{R}}_{u'} \ \wedge\ u'' \in \mathcal{P} \ \wedge\ u' \xrightarrow{\overline{\tau}_{r_{Y_2}}} u''$ with depth of inference m_2)
$\Longrightarrow v'' \in \mathcal{P}$ $\wedge$ $v \xrightarrow{\overline{\tau}_{r_Y}} v''$ (by the *CompDelet* rule, $\because Y \sim_{of} Y_1|Y_2 \ \wedge\ v \xrightarrow{\overline{\tau}_{r_{Y_1}}} v'$) $\wedge$
$(u'', v'') \in T$
$\Longrightarrow \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v$ (by definition of $\overline{\mathcal{R}}_v$) $\wedge\, v'' \in \mathcal{P} \ \wedge\ v \xrightarrow{\overline{\tau}_{r_Y}} v'' \ \wedge\ (u'', v'') \in T$.

$\therefore \forall (u, v) \in T\ \forall \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u\ \forall u'' \in \mathcal{P}$

$(u \xrightarrow{\overline{\tau}_{r_Y}} u'' \implies \overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_v \wedge \exists v'' \in \mathcal{P}\ (v \xrightarrow{\overline{\tau}_{r_Y}} v''\ \wedge\ (u'', v'') \in T))$
for $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ determined by applications of LTS rules with depth of inference $n+1$
($\because (u, v) \in T$ and $\overline{\tau}_{r_Y} \in \overline{\mathcal{R}}_u$ and $u'' \in \mathcal{P}$ with transition $u \xrightarrow{\overline{\tau}_{r_Y}} u''$ and depth of inference $n+1$ are arbitrary)
$\implies Prop(n+1)$ holds (by definition of $Prop(n+1)$).
$\therefore \forall n \in \mathbb{N}^+\ (\forall m \in [1, n] Prop(m) \implies Prop(n+1))$ holds ($\because n \in \mathbb{N}^+$ is arbitrary). Q.E.D.

$\therefore \forall n \in \mathbb{N}^+\ Prop(n)$ holds (by complete induction).
$\therefore$ For all elements of T, the *Deletion* condition of strong dp-simulation on $\mathcal{P}$ is satisfied
($\because$ every transition of $u \in dom(T)$ in $\overline{\mathcal{R}}$ is determined by applications of LTS rules with finite depth of inference). Q.E.D.

$\therefore T$ is a strong dp-simulation on $\mathcal{P}$
(by definition of strong dp-simulation on $\mathcal{P}$, $\because T \subseteq \mathcal{P}$ x $\mathcal{P}$ and for all elements of T, the *Observation* and *Fraction* conditions of strong dp-simulation on $\mathcal{P}$ are satisfied).

A.26.3 T^{-1} is a strong dp-simulation on $\mathcal{P}$

Now $T = S \cup Z$ (by definition of T) $\wedge$
$S = \{(p|q, q|p) \mid p, q \in \mathcal{P}\}$ (by definition of S) $\wedge$
$Z = \{(0, 0)\}$ (by definitions of Z)
$\implies T^{-1} = S^{-1} \cup Z^{-1}$ (by algebra of binary relations) $\wedge$
$S^{-1} = \{(q|p, p|q) \mid p, q \in \mathcal{P}\}$ (by definition of inverse binary relations) $\wedge$
$Z^{-1} = \{(0, 0)\}$ (by definition of inverse binary relations)
$\implies S^{-1} = \{(q|p, p|q) \mid q, p \in \mathcal{P}\}$ (by set theory) $\wedge$ $Z^{-1} = Z$ (by definition of Z)
$\implies S^{-1} = S$ (by definition of S) $\wedge$ $Z^{-1} = Z$
$\implies S^{-1} \cup Z^{-1} = S \cup Z$ (by set theory)
$\implies T^{-1} = T$ ($\because T^{-1} = S^{-1} \cup Z^{-1} \wedge T = S \cup Z$)
$\implies T^{-1}$ is a strong dp-simulation on $\mathcal{P}$ ($\because T$ is a strong dp-simulation on $\mathcal{P}$). Q.E.D.

$\therefore T$ is a strong dp-bisimulation on $\mathcal{P}$
(by definition of strong dp-bisimulation on $\mathcal{P}$, $\because T$ is a strong dp-simulation on $\mathcal{P}$).
$\therefore \forall p, q \in \mathcal{P}\ (p|q \sim_{dp} q|p)$ (by definition of $p|q \sim_{dp} q|p$, $\because \forall p, q \in \mathcal{P}\ ((p|q, q|p) \in T)$). Q.E.D.

A.27 Theorem 4.3.13 $\forall p,q,r\in\mathcal{P}\ ((p|q)|r\sim_{of}p|(q|r))$

Sketch proof: Straightforward case by case analysis.
Let $S\triangleq\{((p|q)|r,p|(q|r))\mid p,q,r\in\mathcal{P}\}$.
It can be shown that S,S^{-1} are strong of-simulations on $\mathcal{P}$ by considering transitions in $\mathcal{I}\cup\mathcal{R}$ defined by $L-Par$, $R-Par$, $React$, $L-React$ or $R-React$.

A.28 Lemma 4.3.26

$$\forall p,q\in\mathcal{P}\ \forall\overline{\tau}_{r_X}\in\overline{\mathcal{R}}_{p|q}\ \forall(p|q)'\in\mathcal{P}$$
$$(p|q\xrightarrow{\overline{\tau}_{r_X}}(p|q)'\Longrightarrow$$
$$(p|q)'=0\ \vee$$
$$\exists p'\in\mathcal{P}\,(p\xrightarrow{\overline{\tau}_{r_X}}p'\ \wedge\ (p|q)'=p'|q)\ \vee$$
$$\exists q'\in\mathcal{P}\,(q\xrightarrow{\overline{\tau}_{r_X}}q'\ \wedge\ (p|q)'=p|q')\ \vee$$
$$\exists\overline{\tau}_{r_{X_1}},\overline{\tau}_{r_{X_2}}\in\overline{\mathcal{R}}\ \exists p',q'\in\mathcal{P}$$
$$(X\sim_{of}X_1|X_2\ \wedge\ p\xrightarrow{\overline{\tau}_{r_{X_1}}}p'\ \wedge\ q\xrightarrow{\overline{\tau}_{r_{X_2}}}q'\ \wedge\ (p|q)'=p'|q'))$$

Sketch proof: uses complete induction on the depth of inference of the applications of the LTS rules that determine the transitions of $p|q$ in $\overline{\mathcal{R}}$.

A.29 Theorem 4.3.14 $\forall p,q,r\in\mathcal{P}\ ((p|q)|r\sim_{dp}p|(q|r))$

Sketch proof: Let $S\triangleq\{((p|q)|r,p|(q|r))\mid p,q,r\in\mathcal{P}\}$.
Let $U\triangleq\{(0|u,0|(0|u))\mid u\in\mathcal{P}\}$.
Let $W\triangleq\{(w,w)\mid w\in\mathcal{P}\}$.
Let $V\triangleq\{((v|0)|0,v|0)\mid v\in\mathcal{P}\}$.
Let $T\triangleq S\cup U\cup W\cup V$.

It can be shown that T,T^{-1} are strong dp-simulations on $\mathcal{P}$:

Theorem 4.3.13 is used to prove T,T^{-1} satisfy the *Observation* and *Fraction* conditions.

To prove T, T^{-1} satisfy the *Deletion* condition, we use complete induction on the depth of inference of the applications of the LTS rules that determine transitions in $\overline{\mathcal{R}}$ and the LTS rules *Delet*, $L-Par$, $R-Par$ and *CompDelet*. The transitions of the elements in U, W and V are straightforward.