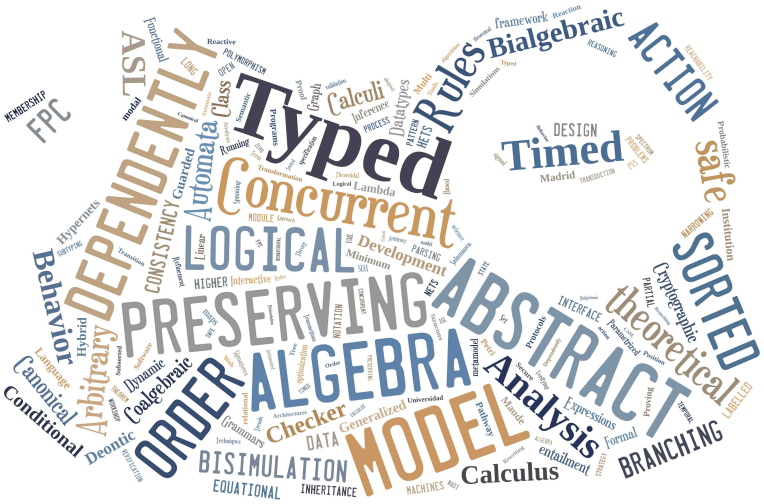
WADT 2012

21st International Workshop on Algebraic Development Techniques

June 7-10, 2012, Salamanca, Spain



Editors: Narciso Martí-Oliet Miguel Palomino

Technical report TR-08/12

Universidad Complutense de Madrid
Departamento de Sistemas Informáticos y Computación

WADT 2012
Preliminary Proceedings

21st International Workshop on Algebraic Development Techniques

Narciso Martí-Oliet Miguel Palomino

Technical report TR-08/12

Universidad Complutense de Madrid
Departamento de Sistemas Informáticos y Computación

June 2012

Preface

The 21st International Workshop on Algebraic Development Techniques (WADT 2012) was held in Salamanca, Spain, from the 7th to the 10th of June 2012.

The algebraic approach to system specification encompasses many aspects of the formal design of software systems. Originally born as a formal method for reasoning about abstract data types, it now covers new specification frameworks and programming paradigms (such as object-oriented, aspect-oriented, agent-oriented, logic and higher-order functional programming) as well as a wide range of application areas (including information systems, concurrent, distributed and mobile systems).

The WADT workshop series focuses on the algebraic approach to the specification and development of systems and aims at providing a platform for presenting recent and ongoing work, to meet colleagues, and to discuss new ideas and future trends. Typical, but not exclusive topics of interest are:

- Foundations of algebraic specification.
- Other approaches to formal specification, including process calculi and models of concurrent, distributed and mobile computing.
- Specification languages, methods, and environments.
- Semantics of conceptual modelling methods and techniques.
- Model-driven development.
- Graph transformations, term rewriting and proof systems.
- Integration of formal specification techniques.
- Formal testing and quality assurance.
- Validation and verification.

This technical report contains the 36 abstracts presented during the workshop. In addition to the presentations of ongoing research results, the program included three invited lectures by Roberto Bruni (Università di Pisa, Italy), Francisco Durán (Universidad de Málaga, Spain), and Kim G. Larsen (Aalborg University, Denmark).

As for previous WADT workshops, after the meeting authors will be invited to submit full papers for the refereed proceedings, which will be published as a volume of Springer's Lecture Notes in Computer Science.

The Steering Committee consists of:

- Michel Bidoit (France)
- Andrea Corradini (Italy)
- Jose Fiadeiro (UK)
- Rolf Hennicker (Germany)
- Hans-Jörg Kreowski (Germany)
- Till Mossakowski (chair, Germany)
- Fernando Orejas (Spain)
- Francesco Parisi-Presicce (Italy)

– Grigore Roşu (United States)
– Andrzej Tarlecki (Poland)

The local Organising Committee included:

– Narciso Martí-Oliet (cochair, Universidad Complutense de Madrid)
– Miguel Palomino Tarjuelo (cochair, Universidad Complutense de Madrid)
– Gustavo Santos (Universidad de Salamanca)
– Ignacio Fábregas (Universidad Complutense de Madrid)
– Isabel Pita (Universidad Complutense de Madrid)
– Adrián Riesco (Universidad Complutense de Madrid)
– David Romero (Universidad Complutense de Madrid)
– Fernando Rosa (Universidad Complutense de Madrid)

The workshop took place under the auspices of IFIP WG 1.3, and it was organized by the Departmento de Sistemas Informáticos y Computación at Universidad Complutense de Madrid. We gratefully acknowledge the sponsorship by the Spanish Ministerio de Economía y Competitividad, IFIP TC1, Facultad de Informática of Universidad Complutense de Madrid, Caja España–Duero Obra Social, Universidad de Salamanca, and IMDEA Software Institute.

May 29, 2012
Madrid

Narciso Martí-Oliet
Miguel Palomino

Table of Contents

Invited talks

Regular presentations

Open Multiparty Interaction*

Chiara Bodei[1], Linda Brodo[2], and Roberto Bruni[1]

[1] University of Pisa, Italy
[2] University of Sassari, Italy

Extended Abstract

An *interaction* is an action by which communicating processes can influence each other. Interactions in the time of Web are something more than input and output between two entities. Actually, the word itself can be misleading, by suggesting a reciprocal or mutual kind of actions. Instead, interactions more and more often involve many parties and actions are difficult to classify under output and input primitives. In the field of process calculi, the best studied, well-understood and popular form of interaction is dyadic, i.e., where only two processes are involved. As networks have become part of the critical infrastructure of our daily activities (for business, home, school, social, health, news, government, etc.) and a large variety of loosely coupled processes have been offered over global networks, as services, more sophisticated forms of interactions have become common, for which convenient formal abstractions are under study. For example, one important trend in networking is moving towards architectures where the infrastructure itself can be manipulated by the software, like in the Software Defined Networking (SDN) approach, where the control plane is remotely accessible and modifiable by software clients, using open protocols such as OpenFlow, making it possible to decouple the network control from the network topology and to provide Infrastructure as a Service (IaaS) over data-centers and cloud systems.

In this talk, we propose a process calculus abstraction based on Open Multiparty Interaction instead of ordinary Dyadic Interaction. An interaction is *multiparty* when it involves two or more processes and it is *open* when the number of involved processes is not fixed or known a priori. Despite the inherent complexity of representing more sophisticated forms of interaction, we show that the underlying synchronisation algebra and name handling primitives are quite simple and straightforward generalisation of dyadic ones. This is witnessed by the operational semantics rules of our calculus, that in the simpler version (i.e., without message passing) resemble the SOS rules of CCS, while in the full one they resemble the SOS rules of pi-calculus (early variant). As a main result, we show how the new calculus can be used to encode Cardelli and Gordon's Mobile Ambients [5] in a natural way. We prove a tight correspondence at the level of reduction semantics and we provide a bisimilarity semantics for Mobile Ambients as a side result.

* Research partially supported by the EU through the FP7-ICT Integrated Project 257414 ASCENS (Autonomic Service-Component Ensembles).

Related Work Our work originated during the study of pi-calculus encodings of Mobile Ambients [6, 2] and it has been mainly inspired and influenced by the Network Conscious pi-calculus [14] and by previous graph-based encodings of Mobile Ambients [8, 9].

Among related work on extending CCS and pi-calculus with more sophisticated forms of interactions, it seems our approach is closest in spirit to [4, 3, 10, 11] (among many others).

Among related work on providing abstract semantics for Mobile Ambients, we mention [13, 15, 1]. The paper [13] is the first work that studies a reduction barbed congruence characterised by a rather ad hoc weak bisimilarity. This bisimilarity relies on a labelled transition system (LTS) where transitions for processes are distinguished from the transitions for systems. The execution of a capability is then derived in two steps: first, transitions for processes are performed for allowing a capability prefix to fire, and the target process contains a hole for holding the sub-process that will change its location; then, once transitions for systems are performed, the involved sub-process will fill the "hole" and the capability is completed.

In [15] the authors build on the theory of *reactive systems* [17, 12, 16] to derive the LTS directly from the reduction semantics and define a transition system composed of three types of transitions. In order to derive the execution of a capability, they first apply process-view transitions, then context-view transitions, and finally they suitably combine and apply the two previous kinds of transitions.

The work in [1] studies a graphical encoding over which the authors introduce a graph transformation system for simulating process reduction. Furthermore the technique of *borrowed contexts* [7] is exploited to derive an LTS for graphs that are images of Mobile Ambients processes. It is conjectured that the three semantics in [13, 15, 1] do coincide.

References

1. F. Bonchi, F. Gadducci, and G.V. Monreale. Labelled transitions for mobile ambients (as synthesized via a graphical encoding). *Electron. Notes Theor. Comput. Sci.*, 242(1):73–98, July 2009.
2. L. Brodo. On the expressiveness of the *pi*-calculus and the mobile ambients. In M. Johnson and D. Pavlovic, editors, *AMAST*, volume 6486 of *Lecture Notes in Computer Science*, pages 44–59. Springer, 2010.
3. R. Bruni and I. Lanese. Parametric synchronizations in mobile nominal calculi. *Theor. Comput. Sci.*, 402(2-3):102–119, 2008.
4. M. Carbone and S. Maffeis. On the expressive power of polyadic synchronisation in pi-calculus. *Nord. J. Comput.*, 10(2):70–98, 2003.
5. L. Cardelli and A.D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.
6. G. Ciobanu and V.A. Zakharov. Encoding mobile ambients into the *pi* -calculus. In I. Virbitskaite and A. Voronkov, editors, *Ershov Memorial Conference*, volume 4378 of *Lecture Notes in Computer Science*, pages 148–165. Springer, 2006.

7. H. Ehrig and B. König. Deriving bisimulation congruences in the dpo approach to graph rewriting with borrowed contexts. *Mathematical Structures in Computer Science*, 16(6):1133–1163, 2006.
8. G.L. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In *Proceedings of the 7th Italian Conference on Theoretical Computer Science*, ICTCS '01, pages 1–16, London, UK, 2001. Springer-Verlag.
9. F. Gadducci and G.V. Monreale. A decentralised graphical implementation of mobile ambients. *J. Log. Algebr. Program.*, 80(2):113–136, 2011.
10. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In G.C. Necula and P. Wadler, editors, *POPL*, pages 273–284. ACM, 2008.
11. C. Laneve and A. Vitale. The expressive power of synchronizations. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, UK*, pages 382–391. IEEE Computer Society, 2010.
12. J.J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In C. Palamidessi, editor, *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 243–258. Springer, 2000.
13. M. Merro and F. Zappa Nardelli. Behavioral theory for mobile ambients. *J. ACM*, 52(6):961–1023, November 2005.
14. U. Montanari and M. Sammartino. Network conscious pi-calculus. Technical Report TR-12-01, Computer Science Department, University of Pisa, 2012.
15. J. Rathke and P. Sobociński. Deriving structural labelled transitions for mobile ambients. *Inf. Comput.*, 208(10):1221–1242, October 2010.
16. V. Sassone and P. Sobocinski. Deriving bisimulation congruences: 2-categories vs precategories. In A.D. Gordon, editor, *FoSSaCS*, volume 2620 of *Lecture Notes in Computer Science*, pages 409–424. Springer, 2003.
17. P. Sewell. From rewrite rules to bisimulation congruences. *Theor. Comput. Sci.*, 274(1-2):183–230, 2002.

On the modularity and reusability of the rule-based specification of QoS properties of systems

Francisco Durán[1]* and Steffen Zschaler[2]

[1] Universidad de Málaga, Spain
[2] King's College London, United Kingdom

Quality of service (QoS) constraints, such as performance, reliability, etc., are essential characteristics of any non-trivial system. Thus, as for other technologies and development processes, in Model-Driven Engineering (MDE), and inside it in the community of Domain-Specific Visual Languages (DSVLs), the modelling, simulation and reasoning about the non-functional properties of systems play a crucial role.

Most DSVLs already allow the specification of the structure and behavior of systems. However, little attention has been paid to the QoS constraints usage and management. Indeed, just a few DSVLs currently provide some support for the modelling of QoS properties. In most of these exceptional cases very specialized notations are used, requiring skilled users to specify and manage such constraints. Very recently, Troya and Vallecillo have proposed in [2, 3] an alternative approach to specify, simulate and reason about QoS properties of systems specified using rule-based DSVLs in a high-level and platform-independent way.

Basically, they propose the use of alien dedicated objects, which they call *observers*, to enable the monitorization of the execution of systems, and, specifically, of the actions that take place along the simulations of the system and of the states their objects pass through. These observers have been effectively used to achieve testing, performance analysis of systems, and automatic reconfiguration of systems depending on the values of the observed properties.

Their proposal is actually quite attractive, and generic, since although they provide a wide variety of observers for measuring performance parameters such as throughput, mean time between failures, idle time, etc., other properties could similarly be specified and analysed. The kind of analysis they can carry on is also quite powerful, obtaining, with relatively very little effort, results that with other approaches require a rather big amount of knowledge and effort.

The methods proposed for DSVLs lack however appropriate mechanisms for modularity and reusability. On the other hand, concepts such as *measurement* [4] or *characteristic* [1] have been used in different contexts to represent non-functional dimensions of systems. The idea in all of them is to use these measurements to express non-functional specifications as constraints over them. Based on ideas for the formalisation of non-functional properties introduced by Zschaler in the context of component-based software engineering (CBSE), we

* Partially supported by Spanish Research Projects TIN2008-031087 and TIN2011-23795.

suggest the use of *context models* to specify measurements independently of the concrete applications on which they are to be used.

In our proposal, measurements are defined relative a partial functional characterisation of the system to be constrained. Specifically, this characterization contains the elements, structures, and behaviours on which the measurement definition relies. This is the reason by which Zschaler calls them *context models*, since they define the context of a measurement, explicitly stating its assumptions on the environment to which it will be applied. This way, context models allow us to specify measurements independently of their usage in concrete applications. In fact, Zschaler defines his notion of *measurement* with a clear idea in his mind: "The formal specifications used to define measurements must not influence the behaviour of the systems to which they are applied."

Once measurements are specified, these definitions can be applied to concrete application specifications. This is accomplished by saying how the application model instantiates the measurement's context model, or, more specifically, by mapping the elements (concepts) in the measurement's context model to the elements (structures and behaviours) in the application model. This mapping will, of course, then have to be checked valid, but will tell us how to bind the parameters.

On appropriate definitions of DSVL and DSVL morphism, we develop a formal framework in which observer specifications can be developed independently, and which, with appropriate bindings to specific systems, can be combined with these, thus giving place to instrumentalized systems which can then be used for simulation and reasoning. On appropriate conditions on the specification of observers and bindings, the framework can ensure the preservation of the system semantics, which is key for the correctness of the analysis on the instrumentalized system.

References

1. Information technology - quality of service: Framework, 1998. ISO/IEC 13236:1998, ITU-T X.641.
2. J. Troya, J. E. Rivera, and A. Vallecillo. On the specification of non-functional properties of systems by observation. In S. Ghosh, ed., *Models in Software Engineering*, vol. 6002 of *LNCS*, pp. 296–309. Springer, 2009.
3. J. Troya and A. Vallecillo. On the performance analysis of rule-based domain specific visual models. Submitted for publication, 2011.
4. S. Zschaler. Formal specification of non-functional properties of component-based software systems. *Software and Systems Modeling*, 9:161–201, 2010.

Quantitative Modal Transition Systems

Kim G. Larsen

CS Department, Aalborg University, Denmark

This talk will offer a comprehensive presentation of the specification formalism of modal transition systems and its recent extensions to the quantitative setting of timed as well as stochastic systems.

Modal transition systems [25] provides a behavioural comositional specification formalism for reactive systems. Modal transition systems grew out of the notion of relativized bisimulation, which allows for simple specifications of components by allowing the notion of bisimulation to take the restricted use that a given context may have in its.

A modal transition system is essentially a (labelled) transition system, but with two types of transitions: so-called *may* transitions, that any implementation may (or may not) have, and *must* transitions, that any implementation must have. In fact, ordinary labelled transition systems (or implementations), are modal transition systems where the set of may- and must-transitions coincide. Modal transition systems come equipped with a bisimulation-like notion of (modal) refinement, reflecting that the the more must-transitions and the fewer may-transitions a modal specification has the more refined and closer to a final implementation it is.

In fact, modal transition systems has all the ingrediences of a complete compositinal specification theory allowing for logical compositions (e.g. conjunction) [22], structural compositions (e.g. parallel) [19] as well as quotienting permitting specifications of composite systems to be transformed into necessary and sufficient specification of components [18]. Thus, modal transition systems has all the benefits of both logical and behavioural specification formalisms [7]. Though modal refinement – like bisimulation – is polynomial-time decidable for finite-state modal transition systems, it only provides a sound but not complete test for the true semantic refinement between modal spefication, in terms set inclusion between their implementation-sets (so-called thorough refinement). For several years, the complexity of thorough refinement – as well as consistency – between modal specifications was an open problem, which after a series of attempts [23, 2] [1] was shown to be EXPTIME-complete [5].

In [20] modal transitions systems were extended into a specification formalism for Markov Chains by the introduction of so-called probabilistic specifications (now known as Interval Markov Chains), where concrete probabilities are replace with intervals and with refinement providing a conservative extension or probabilistic bisimulation [24]. However, Interval Markov Chains lack several of the properties required for a complete compositional specification theory; in particular, they are not closed neither under logical nor structural composition. Recently, the extended notion of Constraint Markov Chains was introduced precisely with the purpose of providing these closure properties [9, 8]. In fact, a

number of decision problems has been settled [17, 16], extions to so-called Abstract Probabilistic Automata has been given [14] with the tool APAC providing support for composition, refinement and consistency checking [15].

Timed extensions of modal transitions, were introduced early on [11] as timed and modal extension of the process algebra CCS. Unfortunately the supporting tool EPSILON was entirely relying on the so-called region-abstraction, making scalability extremely poor. Most recently, taking advantage of the power-full game-theoretical engine of UPPAAL Tiga [3, 10] a "modal-transition system"-like compositional specification theory based on Timed I/O Automata has been proposed [12] with efficient tool support for refinement, consistency and quotienting provided by the tool ECDAR [13].

The work of [6] suggests an alternative timed extension of modal transition systems (though still relying on regions for refinement algorithms). Also, modal transition systems extended with weights [21] has recently been introduced providing a third quantitative extension, also considered in [4].

References

1. Adam Antonik, Michael Huth, Kim G. Larsen, Ulrik Nyman, and Andrzej Wasowski. 20 years of modal and mixed specifications. *Bulletin of the EATCS*, 95:94–129, 2008.
2. Adam Antonik, Michael Huth, Kim G. Larsen, Ulrik Nyman, and Andrzej Wasowski. Modal and mixed specifications: key decision problems and their complexities. *Mathematical Structures in Computer Science*, 20(1):75–103, 2010.
3. Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Uppaal-tiga: Time for playing games! In Werner Damm and Holger Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125. Springer, 2007.
4. Nikola Benes, Jan Kretínský, Kim G. Larsen, Mikael H. Møller, and Jirí Srba. Parametric modal transition systems. In Tevfik Bultan and Pao-Ann Hsiung, editors, *ATVA*, volume 6996 of *Lecture Notes in Computer Science*, pages 275–289. Springer, 2011.
5. Nikola Benes, Jan Kretínský, Kim Guldstrand Larsen, and Jirí Srba. Checking thorough refinement on modal transition systems is exptime-complete. In Martin Leucker and Carroll Morgan, editors, *ICTAC*, volume 5684 of *Lecture Notes in Computer Science*, pages 112–126. Springer, 2009.
6. Nathalie Bertrand, Axel Legay, Sophie Pinchinat, and Jean-Baptiste Raclet. A compositional approach on modal specifications for timed systems. In Karin Breitman and Ana Cavalcanti, editors, *ICFEM*, volume 5885 of *Lecture Notes in Computer Science*, pages 679–697. Springer, 2009.
7. Gérard Boudol and Kim Guldstrand Larsen. Graphical versus logical specifications. In André Arnold, editor, *CAAP*, volume 431 of *Lecture Notes in Computer Science*, pages 57–71. Springer, 1990.
8. Benoît Caillaud, Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Compositional design methodology with constraint markov chains. In *QEST*, pages 123–132. IEEE Computer Society, 2010.
9. Benoît Caillaud, Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Constraint markov chains. *Theor. Comput. Sci.*, 412(34):4373–4404, 2011.

10. Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2005.
11. Karlis Cerans, Jens Chr. Godskesen, and Kim Guldstrand Larsen. Timed modal specification - theory and tools. In Costas Courcoubetis, editor, *CAV*, volume 697 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 1993.
12. Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Timed i/o automata: a complete specification theory for real-time systems. In Karl Henrik Johansson and Wang Yi, editors, *HSCC*, pages 91–100. ACM ACM, 2010.
13. Alexandre David, Kim Guldstrand Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Ecdar: An environment for compositional design and analysis of real time systems. In Ahmed Bouajjani and Wei-Ngan Chin, editors, *ATVA*, volume 6252 of *Lecture Notes in Computer Science*, pages 365–370. Springer, 2010.
14. Benoît Delahaye, Joost-Pieter Katoen, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, Falak Sher, and Andrzej Wasowski. Abstract probabilistic automata. In Ranjit Jhala and David A. Schmidt, editors, *VMCAI*, volume 6538 of *Lecture Notes in Computer Science*, pages 324–339. Springer, 2011.
15. Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Apac: A tool for reasoning about abstract probabilistic automata. In *QEST*, pages 151–152. IEEE Computer Society, 2011.
16. Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Decision problems for interval markov chains. In Adrian Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *LATA*, volume 6638 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 2011.
17. Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Consistency and refinement for interval markov chains. *J. Log. Algebr. Program.*, 81(3):209–226, 2012.
18. Gregor Goessler and Jean-Baptiste Raclet. Modal contracts for component-based design. In Dang Van Hung and Padmanabhan Krishnan, editors, *SEFM*, pages 295–303. IEEE Computer Society, 2009.
19. Hans Hüttel and Kim Guldstrand Larsen. The use of static constructs in a modal process logic. In Albert R. Meyer and Michael A. Taitslin, editors, *Logic at Botik*, volume 363 of *Lecture Notes in Computer Science*, pages 163–180. Springer, 1989.
20. Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *LICS*, pages 266–277, 1991.
21. Line Juhl, Kim G. Larsen, and Jirí Srba. Modal transition systems with weight intervals. *J. Log. Algebr. Program.*, 81(4):408–421, 2012.
22. Kim Guldstrand Larsen. Modal specifications. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 1989.
23. Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wasowski. On modal refinement and consistency. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR*, volume 4703 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2007.
24. Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.
25. Kim Guldstrand Larsen and Bent Thomsen. A modal process logic. In *LICS*, pages 203–210, 1988.

Systematic Design of Abstractions in $\mathbb{K}^\star$

Irina Măriuca Asăvoae

Faculty of Computer Science, Alexandru Ioan Cuza University, Romania
mariuca.asavoae@info.uaic.ro

The Rewriting Logic Semantics Project [10] unifies algebraic denotational semantics and structural operational semantics, two areas in programming languages that evolved in separation. There is already consistent research work proving this unification amenable and useful. Among others, the $\mathbb{K}$ framework [13] is one of the protruding applications which recently achieved a peak in the specification of C semantics [6]. The perspective about such semantic definitions is that they are directly executable as interpreters in a rewriting logic language such as Maude [15] and testable for correctness on representative test-suites. Moreover, Maude's generic formal tools, such as the LTL model checker [5], can be used directly on the semantic definition to obtain program analysis and verification capabilities. The current work studies the latter mentioned aspect w.r.t. overcoming its limitations related to the *effectiveness* of the program analysis and verification in the context of the Rewriting Logic Semantics.

The program analysis and verification methods are, many times, characterized by the infamous ExpTime complexity. This generates an effectiveness problem for the program analysis/verification, problem which is traditionally by-passed by the abstract interpretation [4, 3]. In more details, the abstract interpretation framework standardizes a method for reducing the complexity of the analysis/verification methods by a *coherent projection* of the program's state space. The projection is actually a mapping from the state space of the *concrete* system into the state space of an *abstract* system such that the mapping preserves the transition relations. By coherent projection we understand that the class of properties currently analyzed/verified is reflected from the abstract system into the concrete system. Furthermore, sharp abstractions may also require an associated projection of the concrete transition operators into the abstract ones [2]. The coherent projection is founded by a Galois connection between the concrete and the abstract systems which are, in turns, viewed as lattices. The abstract interpretation framework is developed on top of the operational semantics and there exists an abundance of literature on practical instantiations of it [11].

On the other hand, the rewriting logic already has established its own notion of abstraction, namely the algebraic simulation [9, 8]. In more details, the standard notion of simulation is incrementally mirrored into the rewriting logic under the hypothesis that the systems being related are described as rewriting systems. The presentation in [9, 8] is eloquently sustained at each step by examples instantiating the concrete and the abstract rewrite systems and by

⋆ This work has been supported by Project POSDRU/88/1.5/S/47646 and by Contract ANCS POS-CCE, O2.1.2, ID nr 602/12516, ctr.nr 161/15.06.2010 (DAK).

providing the definition of the algebraic simulation also as a rewrite system. The beauty of this approach lies in the unified rewriting logic representation of both the systems and the simulation. Moreover, the approach is automatically executable using the tools implementing the rewriting logic [1]. Consequently, the rewriting logic and the algebraic simulations provide an appealing framework for specifying abstractions using the algebraic denotational semantics.

The view of the current work is drawn from the above stated facts. One fact is that Rewriting Logic Semantics unifies the operational semantics and the algebraic denotational semantics. Another fact is that the abstractions already proved to be a sine qua non in the field of analysis and verification. These two facts induce the natural idea of a systematic transportation of the results from the abstract interpretation into the Rewriting Logic Semantics. As such, the current work studies the transformation of Galois connections into theoroidal maps with a direct application to a systematic design of abstractions in $\mathbb{K}$. By systematic design we understand a methodology of relating two programming language $\mathbb{K}$-specifications. Namely, the concrete programming language is syntactically mapped into the abstract programming language, followed by a projection of this mapping at the level of the semantic rules. As such, using off-the-shelf abstractions available in the abstract interpretation is tantamount with relating $\mathbb{K}$-specifications in a compositional fashion. We adjust this transformation of the $\mathbb{K}$-specifications to the stages of abstraction described in [14], and exemplify using the notorious predicate abstraction. The choice of predicate abstraction is justified by the existence of consistent documentations in both worlds, namely in abstract interpretation [7] and in rewriting logic [12]. Hence, the bridge that the current work is building between these two worlds finds in the predicate abstraction solid grounds at both ends.

In conclusion, we acknowledge that this work is answering the challenge posed by the $\mathbb{K}$-framework of eagerly transporting the achievements from the operational semantics into the algebraic denotational semantics. Particularly, we would like to thank to Grigore Roşu for opening this view at the level of programming languages, and claiming it as well at the level of the abstractions a la abstract interpretation.

References

1. Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.
2. Patrick Cousot. Design of syntactic program transformations by abstract interpretation of semantic transformations (invited talk). In Ph. Codognet, editor, *Proceedings of the Seventeenth International Conference, ICLP 2001*, pages 4–5, Paphos, Cyprus, November/December 2001. LNCS 2237, Springer, Berlin.
3. Patrick Cousot. Formal verification by abstract interpretation. In Alwyn Goodloe and Suzette Person, editors, *4th NASA Formal Methods Symposium (NFM 2012)*,

volume 7226 of *Lecture Notes in Computer Science*, pages 3–7, Heidelberg, 2012. Springer-Verlag.

4. Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Symposium on Principles of Programming Languages*, pages 238–252. ACM Press, 1977.
5. Steven Eker, José Meseguer, and Ambarish Sridharanarayanan. The Maude LTL model checker. In Fabio Gaducci and Ugo Montanari, editors, *Workshop on Rewriting Logic and Its Applications*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier, September 2002.
6. Chucky Ellison and Grigore Rosu. An executable formal semantics of c with applications. In John Field and Michael Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 533–544. ACM, 2012.
7. Susanne Graf and Hassen Saidi. Construction of abstract state graphs with PVS. In *Proceedings of the 9th Conference on Computer-Aided Verification*, pages 72–83. Springer-Verlag, 1997.
8. José Meseguer, Miguel Palomino, and Narciso Martí-Oliet. Equational abstractions. *Theor. Comput. Sci.*, 403(2-3):239–264, 2008.
9. José Meseguer, Miguel Palomino, and Narciso Martí-Oliet. Algebraic simulations. *J. Log. Algebr. Program.*, 79(2):103–143, 2010.
10. José Meseguer and Grigore Rosu. The rewriting logic semantics project. *Theor. Comput. Sci.*, 373(3):213–237, 2007.
11. Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., 1999.
12. Miguel Palomino. A predicate abstraction tool for Maude. Documentation and tool available at `http://maude.sip.ucm.es/~miguelpt/bibliography.html`.
13. Grigore Rosu and Traian-Florin Serbanuta. An overview of the K semantic framework. *J. Log. Algebr. Program.*, 79(6):397–434, 2010.
14. David A. Schmidt and Bernhard Steffen. Program analysis *as* model checking of abstract interpretations. In Giorgio Levi, editor, *Static Analysis, 5th International Symposium, SAS '98, Pisa, Italy, September 14-16, 1998, Proceedings*, volume 1503 of *Lecture Notes in Computer Science*, pages 351–380. Springer, 1998.
15. Traian-Florin Serbanuta and Grigore Rosu. K-Maude: A rewriting based tool for semantics of programming languages. In Peter Csaba Ölveczky, editor, *Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Held as a Satellite Event of ETAPS 2010, Paphos, Cyprus, March 20-21, 2010, Revised Selected Papers*, volume 6381 of *Lecture Notes in Computer Science*, pages 104–122. Springer, 2010.

Bounded Model Checking of Recursive Programs with Pointers in $\mathbb{K}^\star$

Irina Măriuca Asăvoae[1], Frank de Boer[2,3], Marcello M. Bonsangue[3,2], Dorel Lucanu[1], Jurriaan Rot[3]

[1] Faculty of Computer Science - Alexandru Ioan Cuza University, Romania
{mariuca.asavoae, dlucanu}@info.uaic.ro
[2] Centrum voor Wiskunde en Informatica, The Netherlands
frb@cwi.nl
[3] LIACS - Leiden University, The Netherlands
{marcello, jrot}@liacs.nl

The current work is an approach to the automatic verification of programs with pointers and procedures, e.g., C or Java programs. The verification method of choice is abstract model checking and we focus on developing the algebraic theories of the abstract model and its verification method. More to the point, we follow the abstract interpretation perspective [2] which advocates applying analysis or verification methods over an abstract, simplified model. The advantage of the abstract model consists in the enhanced effectiveness for verification, or even existence of decidability results.

As the abstract modeling language we introduce *Shylock*, an imperative programming language with pointers and recursive procedures which use local and global variables with fields to store references in the heap. The simplicity of Shylock's syntax targets the control flow graph abstraction – an already standard view in abstract interpretation – and is specifically tailored for heap manipulation. Hence, from a concrete program, we abstract away any statements besides pointer manipulation, maintaining however the control flow graph structure of the concrete program. Consequently, we can focus on the problem of interest namely the heap structure and its evolution along the execution of a concrete program.

The definition of the abstract operators in abstract interpretation is, in our settings, equivalent to giving formal semantics for Shylock. The consistent design of the abstract operators is crucial in abstract interpretation because of two reasons: (i) it ensures the soundness of the abstract model w.r.t. the properties to be reflected in the concrete, and (ii) it has a direct influence on the effectiveness of the verification, which can be roughly measured by the size of the abstract state space. We refer [13] for a detailed description of the consistency of Shylock semantics, and we concentrate the current work on Shylock's algebraic representation and its verification method which is, in our choice, LTL model checking for safety properties.

We proceed to give a short description of the specifics of the abstract operators in our proposed abstraction. The semantics with a *standard* fresh object

⋆ This work has been supported by Project POSDRU/88/1.5/S/47646 and by Contract ANCS POS-CCE, O2.1.2, ID nr 602/12516, ctr.nr 161/15.06.2010 (DAK).

creation, which allocates an object identity *never used before*, is prone to generating an infinite object identity space during recursive calls. We propose an *improved* semantics which introduces a novel mechanism for managing the object identities in the heap. This mechanism resides in a combination of *memory reuse* upon generation of fresh object identities combined with a *renaming scheme* to resolve name clashes possibly resulting from the execution of a procedure return statement. Our renaming scheme defined for resolving name clashes in the context of memory reuse is based on the concept of *cut points* as introduced in [7]. Hence, under the assumption of a bounded visible heap, Shylock programs can be represented by finitary structures, namely *finite pushdown systems*. Moreover, in the presence of matching, the comparison of the abstract states is further simplified.

Shylock's standard and improved formal semantics are bisimilar, as we prove in [13]. For the current work, we use $\mathbb{K}$ [8] to give a bisimilar algebraic representation to each of these semantics. Moreover, we show that there is an algebraic bisimulation [6] between these two $\mathbb{K}$-specifications. We implement this algebraic bisimulation by defining a $\mathbb{K}$-specification which contains both semantics as they are, and protects them via labeling each of the two semantics. Note that, given the existence of a $\mathbb{K}$-specification for C's semantics [4], there is the opportunity of proving the algebraic simulation between the concrete (i.e., C) and abstract (i.e., Shylock) semantics. However, this concerns a more extensive future work.

We employ the $\mathbb{K}$-specification of Shylock to derive verification capabilities using LTL model checking. For this purpose we settle two aspects: the properties of interest and the model checking procedure. We present and discuss these in the followings.

The atomic properties are defined as regular expressions over the heap structure. The regular expressions are basically a Kleene algebra with tests [5], where the global and local variables are used as tests while the fields constitute the set of basic actions. The Maude LTL model checker [3] provides a substantial expressivity w.r.t. the atomic properties in comparison with other model checking tools. This feature justifies the appropriateness of using $\mathbb{K}$-Maude [9] for Shylock in order to consequently model check Shylock programs for safety properties. However, due to the prerequisites imposed by Maude's model checker, we can successfully verify only a restricted class of Shylock programs, namely the non-recursive ones. Next, we address this issue in more details and present the approached solutions.

Obviously, in the presence of recursive procedures, the stack in the pushdown system grows unboundedly and, even if the abstraction ensures a finitary representation by a pushdown system, the generated transition system is infinite. This makes the use of Maude's LTL model checker practical only for Shylock programs without recursion. To overcome this drawback we apply a further abstraction on the stack contents which enforces the finiteness of the equivalent transition system. However, the stack abstraction induces incompleteness in the model checking for Shylock.

On the other hand, the value of Shylock improved semantics concerns exactly its pushdown system characterization for which there are standardized model checking results [1, 14]. Hence, as a second solution for the Shylock safety model checking, we propose embedding the $\mathbb{K}$-specification for Shylock abstract semantics into a $\mathbb{K}$-specification for Shylock collecting semantics which produces the reachability automaton. A further coupling with the LTL property produces the full power LTL model checking capabilities. Consequently, this solution preserves the completeness of the model checking procedure for Shylock programs, solely under the standard assumption of a bounded visible heap.

Two final considerations on the work: (i) Shylock represents the syntactic extension of the language proposed in [12]. Namely, Shylock adds fields for variables so, even if it maintains the original program instructions, the abstract semantics has to be redesigned due to the change in the state structure. (ii) The proposed $\mathbb{K}$ specification for model checking pushdown systems is generic and complements the leading program verification perspective in $\mathbb{K}$, namely matching logic [11, 10].

References

1. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model checking. In Proc. CONCUR 97, vol. 1243 of LNCS, Springer, 1997.
2. P. Cousot, and R. Cousot. Abstract Interpretation and Application to Logic Programs, Journal of Logic Programming, 13(2–3), pp. 103–179, 1992.
3. S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL Model Checker. In WRLA 2002, vol. 71 of ENTCS, pp. 162–187, 2002.
4. C. Ellison, and G. Roşu. An Executable Formal Semantics of C with Applications. In POPL 2012, pp. 533–544.
5. D. Kozen. Kleene algebra with tests. ACM Trans. Program. Lang. Syst., 19:427–443, 1997.
6. J. Meseguer, M. Palomino, and N. Martí-Oliet. Equational abstractions. In Theoretical Computer Science, vol. 403(2-3), pp. 239–264, 2008.
7. N. Rinetzky, J. Bauer, T. W. Reps, S. Sagiv, and R. Wilhelm. A semantics for procedure local heaps and its abstractions. In POPL 2005, pp. 296–309, 2005.
8. G. Roşu, and T. F. Şerbănuţă. An overview of the K semantic framework. Journal of Logic and Algebraic Programming, 79(6):397–434, 2010.
9. G. Roşu, and T. F. Şerbănuţă. K-Maude: A Rewriting Based Tool for Semantics of Programming Languages. WRLA 2010, vol. 6381of LNCS, pp.104–122, Springer 2010.
10. G. Roşu, and A. Ştefănescu. Matching logic rewriting: Unifying operational and axiomatic semantics in a practical and generic framework. Technical Report `http://hdl.handle.net/2142/28357`, University of Illinois, 2011.
11. G. Roşu, and A. Ştefănescu. Towards a Unified Theory of Operational and Axiomatic Semantics. ICALP 2012, to appear.
12. J. Rot. A Pushdown Representation for Unbounded Object Creation. BsC thesis, Universiteit Leiden Opleiding Informatica, 2010.
13. J. Rot, I. M. Asavoae, F. de Boer, M. Bonsangue, and D. Lucanu. Interacting via the Heap in the Presence of Recursion. ICE 2012, Submitted.

14. S. Schwoon. Model-Checking Pushdown Systems. PhD thesis, Technische Universität München, 2002.

A $\mathbb{K}$-Based Methodology for Modular Design of Embedded Systems *

Mihail Asăvoae

Faculty of Computer Science
Alexandru Ioan Cuza University, Iaşi, Romania

1 Introduction

The increased complexity of current software systems requires, from a programming language definitional point of view, the manipulation of large program configurations. In the context of embedded software systems, it is important, as well, to capture the underlying architecture behavior. In this paper, we use the $\mathbb{K}$ framework [7] to study how to modularly define program executions in complex environments and then, how to apply abstractions on them.

Originated from the rewriting logic semantics project [6], $\mathbb{K}$ is specialized in defining programming languages and their afferent analysis tools. A $\mathbb{K}$ specification consists of *configurations* - multisets of labeled, nested cells that represent the structure of the system/program states, *rewrite rules* - either transformations of structural information or transitions in the program execution and *computations* - the executions of the specified system. The $\mathbb{K}$ framework proposes a concise notation to manipulate the configurations, which allows the $\mathbb{K}$ rewrite rules to use only the necessary semantic entities. These entities are represented as $\mathbb{K}$-cells. This mechanism works because of an underlying configuration abstraction, called context transformation [5], which automatically completes the configuration with the omitted cells. Next, we present an overview of our modeling methodology, which in turn is based on this context transformation and the inherited modularity of the $\mathbb{K}$ framework.

We take the general configuration C, which comprises of all semantic entities of the specified system and split it into the sub-configurations $C_1, C_2, \ldots, C_n$, each C_i, included into a separate $\mathbb{K}$ module. $\mathbb{K}$ provides a special cell, called k, which solely handles the actual computation. The k cell is included into each of the C_i sub-configurations and it is used, apart from the computational purpose, to pass messages among modules and to facilitate, in this way, the interaction between modules. These messages play the role of meta-assertions between modules. Therefore, a concrete execution in this system is an interleaving between computations and messages that are interchanged between modules. Next, we define abstractions in the following way: first wrap the concrete sub-configurations of interest into a configuration $wrap(C_i)$, and then define the abstract configuration A, which has as sub-configuration $wrap(C_i)$. The initial split of C into

* This work has been supported by Project POSDRU/88/1.5/S/47646 and by Contract ANCS POS-CCE, O2.1.2, ID nr 602/12516, ctr.nr 161/15.06.2010 (DAK).

C_is allows, when the abstraction is applied, to execute only the system's functionality that is of interest. Moreover, the abstraction controls the "concrete" execution and enables functionality reuse. A different consequence of this modeling methodology is that it permits to implant certain characteristics of the concrete semantic entities. We present next an instance of this approach.

While our methodology is general, we exemplify it on assembly programs executed in the presence of instruction and data caches and a main memory system. Our initial configuration C consists of the semantic entities to define the assembly language as well as the micro-architecture components. A natural split produces sub-configurations (and modules) for the language, the instruction cache, the data cache and the main memory system. Now, the language module plays the role of the processor, as it requests information from the memories. This allow us to apply using the wrapping, a timing behavior for the set of instructions in the language, or an power-consumption model. Also, wrapping the instruction cache and the main memory module [1] allows us to implement well-known abstract interpretation [3] based analyses for caches [8], for the purpose of timing analysis [9]. A prototype of this mechanism is implemented in $\mathbb{K}$-Maude [4], the $\mathbb{K}$ framework implementation on top of Maude system [2] and experimental results are presented wrt a reusability metric.

References

1. Mihail Asavoae, Irina Mariuca Asavoae, and Dorel Lucanu. On abstractions for timing analysis in the k framework. In *FOPARA*, LNCS, 2011. to appear.
2. Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *LNCS*. Springer, 2007.
3. Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252. ACM Press, 1977.
4. Traian Florin Şerbanuţă and Grigore Roşu. K-Maude: A rewriting based tool for semantics of programming languages. In *WRLA 2010*, volume 6381 of *LNCS*, pages 104–122, 2010.
5. Mark Hills and Grigore Rosu. Towards a module system for k. In *WADT*, pages 187–205, 2008.
6. José Meseguer and Grigore Roşu. The rewriting logic semantics project. *Theoretical Computer Science*, 373(3):213–237, 2007.
7. Grigore Roşu and Traian Florin Şerbănuţă. An overview of the K semantic framework. *Journal of Logic and Algebraic Programming*, 79(6):397–434, 2010.
8. Henrik Theiling, Christian Ferdinand, and Reinhard Wilhelm. Fast and precise wcet prediction by separated cache and path analyses. *Real-Time Systems*, 18(2/3):157–179, 2000.
9. Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):1–53, 2008.

An Object-Z Institution for Specifying Dynamic Object Behavior

Hubert Baumeister[1], Mohamed Bettaz[2], Mourad Maouche[3], and Mhamed Mosteghanemi[2]

[1] DTU Informatics, Technical University of Denmark,
hub@imm.dtu.dk
[2] Laboratoire Méthodes de Conception de Systèmes, ESI, Algeria,
m.bettaz@mesrs.dz, m.mosteghanemi@mesrs.dz
[3] Philadelphia University, Jordan,
mmaouch@philadelphia.edu.jo

1 Motivation

Institutions provide a general framework for describing logical systems; (co-) morphisms (between institutions) are used to connect such systems in the framework of multi-modeling languages allowing to deal with heterogeneous specifications in a consistent way [2].

In this work we define an institution OZS for Object-Z specifications, where the operation schemata are defined on an explicit model on the state of the system under specification.

The ultimate objective is to contribute to the building of heterogeneous specifications 'enclosing' Object-Z notation.

We assume familiarity with the most basic notions of institutions [4], [3] and Object-Z [5].

2 Signatures

A **signature** Σ in OZS is a pair $\Sigma = (S, F)$ where,

- $S = C \cup T$ is the disjoint union of class sorts C and primitive types T .
- $F = B \cup R \cup O$ declares a family B of operation symbols $B_{c \to t}$, ($c \in C$, $t \in T$) representing basic attributes, R a family of operation symbols $R_{c \to c'}$, ($c, c' \in C$) representing reference attributes, and O a family of operation symbols $O_{c,w}$ and $O_{c,w,s}$, ($c \in C$, $w \in S^*$, $s \in S$) representing operation schemata; $O_{c,w}$ is used for operation schemata returning no value, while $O_{c,w,s}$ is used for operation schemata returning a value.

A signature **morphism** $\varphi : \Sigma \to \Sigma'$, where $\Sigma = (C \cup T,\ B \cup R \cup O)$, and $\Sigma' = (C' \cup T',\ B' \cup R' \cup O')$ is a map $\varphi = (\varphi_S : S \to S', \varphi_F : F \to F')$, with:

- $\varphi_S(c) \in C'$ for $c \in C$, $\varphi_S(t) \in T'$ for $t \in T$, and

– $\varphi_F(b) \in B'_{\varphi_S(c)\to\varphi_S(t)}$ for $b \in B_{c\to t}, \varphi_F(r) \in R'_{\varphi_S(c)\to\varphi_S(c')}$ for $r \in R_{c\to c'}$, and $\varphi_F(o) \in O'_{\varphi_S(c),\varphi_S(w)}$ for $o \in O_{c,w}$ and $\varphi_F(o) \in O'_{\varphi_S(c),\varphi_S(w),\varphi_S(s)}$ for $o \in O_{c,w,s}$, where $\varphi_S(w) = \varphi_S(s_1)...\varphi_S(s_n)$, $(w = s_1...s_n)$.

OZS signatures and OZS signature morphisms define a category denoted *OZS-Sig*. The composition of morphisms is the composition of the corresponding functions, and the identity morphisms are the identity functions.

Fact 1. The category *OZS-Sig* is (finitely) co-complete.

Proof. Let G be a functor from J to $OZS - Sig$, $G(i) = \Sigma_i = (S^i, F^i)$, and $G(g) = (G(g)_S , G(g)_F)$, for $g : i \to j$ in J. Let G_S be a functor from J to $\mathcal{S}et$ defined by $G_S(i) = S^i$ and $G_S(g) = G(g)_S$, and G_F be a functor from J to $\mathcal{S}et$ defined $G_F(i) = F^i$, $G_F(g) = G(g)_F$. The colimit of the functor G is then defined by $\coprod G = (S^{co}, F^{co})$, where S^{co} is the colimit of the functor G_S, and F^{co} the functor colimit of the functor G_F . The co-cone morphisms $\iota_i^G = ((\iota_S^G)_i, (\iota_F^G)_i)$, from Σ_i to $\coprod G$ are given by the co-cone morphisms $\iota_i^{G_S}$ from S^i to S^{co} and $\iota_i^{G_F}$ from F^i to F^{co} . □

3 Models

Given a signature Σ, a Σ–**model** M defines:

– A set M_s for each sort $s \in S$, and
– An (explicit) state, $state_M = \Sigma_{c\in C}(M_c \to \prod_{a\in B_{c\to t}\cup R_{c\to c'}} M_{type(a)})$, where $type(a) = t$ if $a \in B_{c\to t}$ and $type(a) = pow(c')$ if $a \in R_{c\to c'}$ and $M_{pow(c')} = pow(M_{c'})$. (An element σ of $state_M$ is $\sigma_c(oid)(a)$ in M_t or $pow(M_{c'})$, with $c \in C$, $oid \in M_c$, and $a \in B_{c\to t} \cup R_{c\to c'}$).
– A function $o_M : state_M \times M_c \times M_{s_1} \times ... \times M_{s_n} \to state_M$ for each operation $o \in O_{c,w}$, and a function $o_M : state_M \times M_c \times M_{s_1} \times ... \times M_{s_n} \to M_s \times state_M$ for each operation $o \in O_{c,w,s}$, $(w = s_1...s_n)$.

A Σ–**homomorphism** $h : M \to M'$**,** where M and M' are Σ models is a family h of functions $(h_s : M_s \to M'_s)$ for $s \in S$ together with a function $h_{state} : state_M \to state_{M'}$ such that:

– $h_{state}(o_M(\sigma_c, m_1, ..., m_n)) = o_{M'}(h_{state}(\sigma_c), h_{s_1}(m_1), ..., h_{s_n}(m_n))$ for $o \in O_{c,w}$, $c \in C, w = s_1...s_n, s_i \in S_i$, and $\sigma_c \in state_M$. Analogous for $o \in O_{c,w,s}$.
– $h_t(\sigma_c(oid)(a)) = h_{state}(\sigma)_c(h_c(oid))(a)$ for all $\sigma \in state_M, oid \in M_c, c \in C, a \in B_{c\to t}$, and
– $h_{pow(c')}(\sigma_c(oid)(a)) = h_{state}(\sigma)_c(h_c(oid))(a)$ for all $\sigma \in state_M$, $oid \in M_c$, $c \in C$, $a \in R_{c\to c'}$, where $h_{pow(s)} : pow(M_s) \to pow(M'_s)$ with $h_{pow(s)}(N) = \{h_s(m) \mid m \in N\}$ for all $N \subseteq M_s$.

The identity homomorphism is the family of identities for the sets M_s and the identity on the set $state_M$.

The composition of homomorphisms is the composition of the family of the functions for the sets M_s and the composition of the functions for the state.

Given a signature morphism $\varphi : \Sigma \to \Sigma'$ in OZS-Sig, and a Σ'- model M, the $\varphi-$**reduct** of M is defined by:

- $(M\mid_\varphi)_s = M_{\varphi_S(s)}$, and
- $o_{M\mid_\varphi} = \varphi_F(o)_M$ for all $o \in O_{c,w} \cup O_{c,w,s}$.

For a Σ'–homomorphism $h : M \to M'$, the φ–**reduct** $h|_\varphi : M|_\varphi \to M'|_\varphi$ is defined by:

- $(h|_\varphi)_s = h_{\varphi_S(s)}$,
- Define the φ–reduct $\sigma|_\varphi \in state_{M|_\varphi}$ of a state $\sigma \in state_M$ by
  - $\sigma|_{\varphi_c}(oid)(a) = \sigma_{\varphi(c)}(oid)(\varphi_F(a))$ for $oid \in M_c$, $c \in C$, and $a \in F$.
- Then $(h|_\varphi)_{state}(\sigma|_\varphi)_c(h(oid))(a) = h_{state}(\sigma)_{\varphi_S(c)}(h(oid)(\varphi_F(a))$.

The functor **OZS-Mod** from *OZS-Sig* to $\mathcal{C}at$ takes each signature Σ to the category having as objects Σ-models and as morphisms Σ-homomorphisms, and each *OZS-Sig*-morphism φ from Σ to Σ' to a functor from the category $OZS\text{-}Mod(\Sigma')$ to the category $OZS\text{-}Mod(\Sigma)$, sending each Σ'-model M to its φ-reduct $M\mid_\varphi$ and each Σ'-homomorphism h to its φ-reduct $h\mid_\varphi$.

Fact 2. The institution OZS has amalgamation.

Proof. Omitted for lack of space. □

4 Conclusion

In this paper we introduced some basic 'ingredients' leading to the definition of an institution. Proofs of facts, definition of sentences, satisfaction relations and satisfaction condition were omitted for lack of space. These will be given in the final version of this contribution, and with a running example. The difference with similar works is that our contibution is not limited to static structures, but deals also with the dynamic behavior of the system under design. As a future work, we plan to consider defining operation schemata on an implicit model [1] of the system state.

References

1. Baumeister H, Relations between Abstract Datatypes modeled as Abstract Datatypes, PhD thesis, Universitat des Saarlandes, Saarbrucken, May 1999.
2. Boronat A, Knapp A, Meseguer J, Wirsing M, What is a Multi-Modeling Language?, Lecture Notes in Computer Science, Vol. 5486, pp.266-289, 2009.
3. Diaconescu R, Three Decades of Institution Theory, In: Beziau, J.-Y. (ed.), Universal Logic: an Anthology, Birkhauser, 2011.
4. Goguen J and Burstall RM, Institutions: Abstract Model Theory for Specification and Programming, Journal of ACM, 39(1), 95-146, 1992.
5. Kim SK and Carrington D, A Formal Mapping between UML Models and Object-Z Specifications, Lecture Notes in Computer Science, Springer-Verlag, Vol.1878, pp.2-21, 2000.

A History-Dependent Probabilistic Strategy Language for Probabilistic Rewrite Theories

Lucian Bentea[1] and Peter Csaba Ölveczky[1,2]

[1] Department of Informatics, University of Oslo
[2] University of Illinois at Urbana-Champaign

Probabilistic rewrite theories [7, 1] have been introduced to specify probabilistic systems in rewriting logic. Since such theories combine probabilistic and nondeterministic features, all nondeterminism in the specification must be quantified to allow probabilistic analysis—including statistical model checking and quantitative analysis using, e.g., VeStA [10] or PVeStA [2]. This is achieved in [7, 1] by means of *adversaries* that quantify such nondeterminism. However, until now, there has not been any way for the user to define different adversaries for probabilistic rewrite theories.

In [4] we define a strategy language for specifying *memoryless* adversaries for probabilistic rewrite theories, and implement a probabilistic simulator and statistical model checker for probabilistic rewrite theories with strategy expressions. In that version, the user must first define the probability of applying a certain rewrite rule, then define the probability of using a certain context as a function of the selected rule, and finally the user must define the probability of choosing a particular matching substitution as a function of the selected rule and rewrite context. In [4] we show the usefulness of being able to define different strategies for a given specification by a cloud computing example, where different probabilistic strategies specify different load balancing policies.

Many systems' probabilistic behavior *is* history-dependent, where the probability distribution over successor states may depend on the computation history. For example, in probabilistic density control algorithms for wireless sensor networks, nodes turn themselves on and off in rounds, to save energy while maintaining sensing/network coverage. Different strategies for selecting active nodes could be to, e.g., select with high probability nodes that were previously not often activated (to allow the most active nodes to save energy), or to give high probability to the previously frequently activated nodes (which should therefore be "optimal" nodes).

Another example is that of randomized leader election protocols [9] where the aim is to designate a node of a distributed system as the leader. Assuming a single channel, such protocols execute in rounds as follows. In each round, every node sends a message on the channel with probability p. At the end of the round, the channel can be in different states, depending on how many messages it contains. If it contains one message, the leader is chosen as the node that sent the message; otherwise, another round of the protocol is executed. For the new round, the transmission probability p may also be updated according to the type of leader election protocol [9]:

- *oblivious*, in which p depends on the number of rounds executed so far;
- *uniform*, where p is a function of the status history of the channel at the end of each round, including the current one;
- *nonuniform*, in which p is a function of the local history of decisions taken by each node in all the rounds, including the current one.

In either case, the probability distribution over the states at the next step of the protocol is a function of its computation history.

The work reported in this abstract extends our work in [4] as follows:

1. We propose a language for specifying *history-dependent* adversaries of probabilistic rewrite theories. In this way, we allow defining different probabilistic history-dependent execution policies for a given system specification.
2. We extend the language in [4] to any possible choice order of the rule, context, and matching substitution, and therefore to any possible probabilistic dependencies between these choices. We also show that the extended language can specify *any* computable adversary of a probabilistic rewrite theory.
3. We define the semantics of our language by mapping each strategy expression to an adversary, and use this mapping to define both the probability-theoretic semantics, and the probabilistic rewrite theory semantics of our language. (This also defines the semantics of the subset of the strategy language in [4], for which no formal semantics has been given.)

For example, in an uniform leader election protocol, a node's decision of whether or not to transmit a message on the channel can be formalized by the following two object-oriented rewrite rules, written in Full Maude syntax:

```
rl [transmit]: < O : Node | trial : false >  < O' : Channel | buffer : ML >
=> < O : Node | trial : true >  < O' : Channel | buffer : (ML msgFrom(S)) > .

rl [not-transmit]: < O : Node | trial : false >  < O' : Channel | buffer : ML >
=> < O : Node | trial : true >  < O' : Channel | buffer : ML > .
```

together with a probabilistic rule strategy expression in our language:

```
psdrule RlStrat := given state: (CF < O : Channel | trial : false >)  history: RH
                   is: transmit -> prob(RH) ; not-transmit -> (1 - prob(RH)) .
```

where `CF` is a variable of sort `Configuration` that matches the rest of the current state, `RH` contains the entire history of rule applications up to the current state, and the function `prob : RuleHistory -> Rat` is defined equationally by:

```
eq prob(none) = 1/2 . --- the transmission probability in the first round
eq prob(RH ; transmit) = prob(RH) .  eq prob(RH ; not-transmit) = prob(RH) .
eq prob(RH ; nextRoundNull) = min(1, 2 * prob(RH)) .
eq prob(RH ; nextRoundCollision) = prob(RH) / 2 .
```

where `nextRoundNull` is the label of a rule that is matched at the end of a round, when the channel contains no messages, and similarly `nextRoundCollision` is the label of a rule that is matched when the channel contains two or more messages.

It is worth mentioning that well known tools such as Uppaal-SMC [6] and PRISM [8] do not support user-defined probabilistic strategies, nor history-dependent models. Uppaal-SMC relies on a probabilistic semantics given to networks of priced timed automata [5], in which the time delays in each location may also follow an uniform—and therefore non-memoryless—probability distribution. However, the probability distribution that each action determines over successor locations only depends on the current location of the automaton, and not its entire computation history. Similarly, in all models supported by PRISM, the probability distribution over next states only depends on the current state.

We have not yet implemented the above extensions of the probabilistic strategy language in Full Maude, but aim at having a prototype ready by the time of WADT. We refer to our technical report [3] for details about our strategy language and its semantics.

Acknowledgments. We gratefully acknowledge partial support for this work by AFOSR Grant FA8750-11-2-0084.

References

1. Gul Agha, José Meseguer, and Koushik Sen. PMaude: Rewrite-based specification language for probabilistic object systems. *Electronic Notes in Theoretical Computer Science*, 153(2), 2006.
2. Musab AlTurki and José Meseguer. PVeStA: A parallel statistical model checking and quantitative analysis tool. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *CALCO'11*, volume 6859 of *LNCS*, pages 386–392. Springer, 2011.
3. Lucian Bentea and Peter Csaba Ölveczky. A language for defining adversaries of probabilistic rewrite theories and its semantics. Manuscript available from: `http://heim.ifi.uio.no/~lucianb/publications/2012/prob-strat.pdf`, 2012.
4. Lucian Bentea and Peter Csaba Ölveczky. A probabilistic strategy language for probabilistic rewrite theories and its application to load balancing policies in cloud computing. Paper in preparation., 2012.
5. Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, Jonas van Vliet, and Zheng Wang. Statistical model checking for networks of priced timed automata. In Uli Fahrenberg and Stavros Tripakis, editors, *FORMATS*, volume 6919 of *LNCS*, pages 80–96. Springer, 2011.
6. Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Zheng Wang. Time for statistical model checking of real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV*, volume 6806 of *LNCS*, pages 349–355. Springer, 2011.
7. Nirman Kumar, Koushik Sen, José Meseguer, and Gul Agha. Probabilistic rewrite theories: Unifying models, logics and tools. Technical report UIUCDCS-R-2003-2347, Department of Computer Science, University of Illinois at Urbana-Champaign, 2003.
8. Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
9. Koji Nakano and Stephan Olariu. Uniform leader election protocols for radio networks. *IEEE Trans. Parallel Distrib. Syst.*, 13(5):516–526, 2002.

10. Koushik Sen, Mahesh Viswanathan, and Gul A. Agha. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *QEST*, pages 251–252. IEEE Computer Society, 2005.

Adaptable Transition Systems*

Roberto Bruni[1], Andrea Corradini[1], Fabio Gadducci[1],
Alberto Lluch Lafuente[2], and Andrea Vandin[2]

[1] Department of Informatics, University of Pisa, Italy
[2] IMT Institute for Advanced Studies Lucca, Italy

Introduction Self-adaptive systems have been advocated as a convenient solution to the problem of mastering the complexity of modern software systems and the continuous and rapid evolution of the environment where they operate. Should the analysis favour a *black-box* perspective, a software system is called "self-adaptive" if it can modify its behaviour in response to a change in its context.

On the contrary, *white-box* adaptation focuses on how adaptation is realised in terms of architectural and linguistic mechanisms and usually promotes a clear separation of adaptation and application logics. Our own contribution [1] characterizes adaptivity on the basis of a precisely identified collection of *control data*, deemed to be interpreted as those data whose manipulation triggers an adaptation. This view is agnostic with respect to the form of interaction with the environment, the level of context-awareness, the use of reflection for self-awareness. In fact, our definition applies equally well to most of the existing approaches for designing adaptive systems. Overall, it provides a satisfactory answer to the question "what is adaptation *conceptually*?".

But "what is adaptation *formally*?" and "which is the right way to reason about adaptation, *formally*?". Unfortunately, only few works (e.g. [6]) address the foundational aspects of adaptive systems, including their semantics and the use of formal reasoning methods, and very often only generic analysis techniques are applied. An example of the possibilities of such technique is our approach [2] to adaptive self-assembly strategies using Maude (and following precisely both [6] and [1]), where we applied standard simulation and statistical model checking.

Adaptable Transition Systems Building on the intuitions briefly discussed above and on some foundational models of component based systems (like *I/O automata* [5] and *interface automata* [3]), we aim at distilling a core, essential model of adaptive systems. We propose a simple formal model based on a new class of transition systems, and we sketch how this definition can be used to specify properties related to the adaptive behaviour of a system. A central role is again played by control data, as well as by the interaction among components and with the enviroment (not addressed explicitly in [1]).

* Reserach partially supported by the EU through the FP7-ICT Integrated Project 257414 ASCENS (Autonomic Service-Component Ensembles).

Let us recall that I/O and interface automata are possibly infinite-state automata where steps are labeled over three disjoint sets of actions, namely *input*, *output* and *internal* actions. The composition of two automata is defined only if certain disjointness constraints over the sets of actions are satisfied, and is obtained conceptually as a synchronous composition on shared actions and asynchronous on the others, the differences between the two models not being relevant at this level of abstraction.

Adaptable Transition Systems (ATSs) combine these features on actions within an extended Kripke frame presentation, in order to capture the essence of adaptativity. An ATS is a tuple $\mathcal{A} = \langle S, A, T, \Phi, l, \Phi^c \rangle$ where S are the states, $A = \langle I, O, H \rangle$ is a triple of three disjoint sets of input, output and internal actions, and $T \subseteq S \times A \times S$ is a transition relation, where by A here we improperly denote the union $I \uplus O \uplus H$. Furthermore, Φ is a set of atomic propositions, and $l : S \to 2^{\Phi}$ is a labeling function mapping states to sets of propositions. Finally, $\Phi^c \subseteq \Phi$ is a distinguished subset of *control propositions*, which play the role of the control data of [1].

A transition $s \xrightarrow{a} s' \in T$ is called an *adaptation* if it changes the control data, i.e., if there exists a $\varphi \in \Phi^c$ such that $\varphi \in l(s) \iff \varphi \notin l(s')$. Otherwise, it is called a *basic* transition. An action $a \in A$ is called a *control action* if it labels at least one adaptation, and the set of all control actions is denoted by C.

The relationship between the action set C and the alphabets I, O and H is arbitrary in general, but it could satisfy some pretty obvious constraints for specific classes of systems. For example, an ATS $\mathcal{A}$ is *self-adaptive* if $C \cap I = \emptyset$, i.e., if all adaptations are under the control of the system. If instead $C \subseteq I$ the system is *adaptable*; intuitively, adaptations cannot be executed locally but should be triggered by an external manager. Hybrid situations are possible as well, when a system has both input and local control actions.

A preliminary investigation suggests that the composition operations on I/O automata can be extended seamlessly to ATSs. It will be exploited, for example, to model the composition of an adaptable basic component $\mathcal{A}_B$ and an adaptation manager $\mathcal{A}_M$ that realizes the adaptation logics, for example a control loop in the style of the MAPE-K architecture [4]. In this case, natural well-formedness constraints could be expressed as suitable relations among sets of actions. For example, the manager controls *completely* the adaptivity features of the basic component if $C_B \subseteq O_M$; and if the manager itself is at least partly adaptable (i.e., $C_M \cap I_M \neq \emptyset$), a natural requirement to avoid circularities would be that $O_B \cap C_M = \emptyset$, i.e. that the basic component cannot govern the adaptivity of the manager. Composition of ATSs will also be used to model different kinds of aggregation of adaptive systems, like *ensembles* and *swarms*.

A computation of an ATS $\mathcal{A}$ is a finite or infinite sequence of consecutive transitions $\{s_i \xrightarrow{l_i} s_{i+1}\}_{i<n}$ from T (thus n can be ω). A computation is *basic* if it is made of basic transitions only, otherwise it is *adaptive*.

By the very nature of adaptive systems, properties that one could be interested to verify on them could be classified according to the kind of computations that are concerned with. For example we can consider: *global properties*, i.e. prop-

erties about the global behaviour of the system, including for example invariants, safety and liveness properties, or QoS levels that the system as a black-box must satisfy/guarantee; *basic properties*, i.e. properties that must be satisfied by basic computations of the system only; and *adaptation properties*, i.e. properties that may fail for basic computations, and that therefore need the adapting capability of the system to be satisfied.

In the talk we report some initial results on the use of ATSs for the specification of adaptive systems as well as on the epressiveness of composition operators, and we discuss on the definition of suitable logics and verification techniques (e.g. along the lines of the *Mode-extended Linear Temporal Logic* [7]) for expressing and analyzing system properties as those mentioned above .

References

1. Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch-Lafuente, and Andrea Vandin. A conceptual framework for adaptation. In Juan de Lara and Andrea Zisman, editors, *FASE*, volume 7212 of *LNCS*, pages 240–254. Springer, 2012.
2. Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch-Lafuente, and Andrea Vandin. Modelling and analyzing adaptive self-assembling strategies with Maude. In Francisco Durán, editor, *Preliminary Proceedings of WRLA*, 2012.
3. Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *ESEC/SIGSOFT FSE 2001*, volume 26(5) of *ACM SIGSOFT Software Engineering Notes*, pages 109–120. ACM, 2001.
4. Paul Horn. Autonomic Computing: IBM's perspective on the State of Information Technology, 2001.
5. Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *PODC 1987*, pages 137–151. ACM, 1987.
6. José Meseguer and Carolyn L. Talcott. Semantic models for distributed object reflection. In Boris Magnusson, editor, *ECOOP*, volume 2374 of *LNCS*, pages 1–36. Springer, 2002.
7. Yongwang Zhao, Dianfu Ma, Jing Li, and Zhuqing Li. Model checking of adaptive programs with mode-extended linear temporal logic. In *Engineering of Autonomic and Autonomous Systems (EASe), 2011 8th IEEE International Conference and Workshops on*, pages 40–48. IEEE Computer Society, 2011.

Entailment Systems for Default Reasoning

Valentín Cassano[1], Carlos G. Lopez Pombo[2,3], and Thomas S.E. Maibaum[1]

[1] Department of Computing and Software, McMaster University
[2] Departmento de Computación, Universidad de Buenos Aires
[3] Consejo Nacional de Investigaciones Científicas y Tecnológicas (CONICET)
cassanv@mcmaster.ca clpombo@dc.uba.ar tom@maibaum.org

Software certification is a widespread concern in Software Engineering for, in many cases, the compliance of software with certain requirements becomes indispensable because of the critical tasks it is supposed to perform. In that respect, formal methods have provided a promising toolset for coping with this issue. However, they are mostly based on monotonic logical frameworks. This imposes some considerations on the manipulation of the corpus of evidence justifying how a piece of software fulfills some critical properties. To provide an example, additional evidence obtained about a software artifact, e.g., data gathered from further testing, possibly invalidates previous claims made about it. Within a monotonic reasoning framework, this vitiates further analysis from logical interest[4]. Thus, the need to create formal frameworks capable of dealing with such scenarios then becomes indispensable.

In view of the above, nonmonotonic reasoning appears as an adequate framework for coping with the class of problems just presented. In brief, nonmonotonic reasoning emerged from the field of Artificial Intelligence, and it was originally motivated by a prevailing discontent in the area with the limits of first order classical logic as a framework for reasoning about action and causation[5]. Notwithstanding, this initial motivation was rapidly outgrown and nonmonotonic reasoning became a subject of study in its own right. The underlying philosophy of this type of reasoning is a cogent reflection of the fact that reasoning is not in general reducible to the consequences obtainable from a set of definite claims. Instead, it is also being carried out under some judiciously chosen hypotheses. These are understood as tentative claims and, in view of their nature, may be abandoned in the light of new evidence.

In turn, there exist several alternative formalizations of a nonmonotonic consequence relation, but in this work we will concentrate on *default reasoning*. This formalism was proposed originally by Reiter in [4] as a way of reasoning with incomplete information[6]. Intuitively, in such situations some surmises are posited to carry out further reasoning. As a result, in default reasoning, the premises of its consequence relation consist of a set of *facts*, conceived of as representing unalterable judgments, and a set of *default rules*, understood as conditionally supporting some tentative claims. In virtue of this conditional conception, de-

[4] Since in the presence of monotonicity, once a claim is established, it is regarded as being absolute, hence its rebuttal necessarily leads to an inconsistent situation.
[5] q.v. *Nonmonotonic Reasoning* in [1] for an historical account of the field.
[6] q.v. [4], p. 87.

fault rules become the source of non-monotonicity, for the sustainability of the claims they establish may be undermined by the addition of further claims. In addition, the nature of default rules forbids two conflicting tentative claims to be considered in the same situation, as, in such a circumstance, the justifications of both claims are contradicted. Thence, for a given set of facts, default rules determine a class of alternative situations that may arise because of conflicting tentative claims. These alternative situations are known as *extensions*. The consequences of some given premises are obtained either skeptically, or credulously, based on what can be demonstrated in every, or some, extension, respectively.

Perhaps the most illustrative example of a default reasoning argument is explained in reference to the legal principle of *Presumption of Innocence*. This principle states that in the absence of further information, it should be sustained that anyone accused of committing a crime is innocent. Arguably, if obtained in this way, the innocence of the accused must be regarded as a tentative claim. That is, if during the course of a trial sufficient convicting evidence is presented, the claim denoting the innocence of the accused can no longer be sustained. This general form of reasoning is not only typical of default reasoning, but also of software certification. For, as previously mentioned, in the latter situation, additional evidence can possibly invalidate otherwise established claims.

Now, since its introduction, several distinct formalizations of Reiter's original idea have been proposed, leading to what nowadays forms a class of default reasoning formalisms. We regard these formalizations as problematic, for they have been described from an operational point of view. We consider that this underlying approach mitigates against the study of general properties of default reasoning systems and that this is particularly reflected in the study of the formal properties of consequence relations for them. In view of these considerations, we propose a categorical formalization of the concept of an *entailment system for default reasoning*. Our formalization follows the methodology introduced by Meseguer in [3]; where a categorical definition of entailment systems is introduced as a syntactic counterpart of institutions [2] providing an extremely general and powerful tool for the study of the general properties of monotonic entailment relations. We elaborate on the way in which our approach tackles the previously mentioned problems and illustrate how most of the standard systems of default reasoning are captured by this new, abstract formalization. In addition, we explain how the approach advocated here elucidates some considerations with respect to general properties of consequence relations for default reasoning, thus underpinning the study of these properties.

To provide an overview of our formulation of an entailment system for default reasoning, we consider the following steps:

i. We fix an underlying entailment system over which default reasoning will be carried out.
ii. Then, resorting to an adequate abstraction of default rules, analogous to the case of sentences of an entailment system, we construct a category of premises for an entailment relation for default reasoning. The objects of this

category are seen as tuples consisting of a set of facts and a set of default rules.

iii. In turn, the concept of the class of extensions associated with some given premises is defined by a functor mapping premises to classes of sets of sentences. These sets of sentences are regarded as augmentations of the set of facts with a number of tentative claims being supported by certain default rules.

iv. An entailment relation for default reasoning is then a relation between premises and sentences, and it is referred to as skeptical, or credulous, depending on whether the relation between some given premises and a sentence holds whenever that sentence belongs to every, or some, extension of these premises, respectively.

v. Lastly, the desired properties of an entailment relation for default reasoning are expressed in relation to the extensions of some given premises.

The contribution of our approach, besides its generality, is twofold. Firstly, it provides a formal framework, within the theory of general logical systems, for the study of different systems of default reasoning, with an emphasis on the departure from the traditional operational view. Namely, our categorical definition of entailment systems for default reasoning, which resorts to general principles entrenched in the theory of entailment systems, identifies the basic concepts and structures involved in various default reasoning formalisms, and it establishes, based on these constructions, some minimal requirements these formalisms are considered to satisfy. We argue that this methodological standpoint aids in the formulation, and posterior analysis, of general properties of entailment relations for default reasoning. Secondly, it internalizes a notion of structure preserving mapping between premises which allows for their structuring in a nonmonotonic setting. This favors the study of concepts analogous to conservative extensions in classical entailment systems, but now for default reasoning.

References

1. D. M. Gabbay and J. Woods. *Handbook of the History of Logic: The Many Valued and Nonmonotonic Turn in Logic*, volume 8. North-Holland, Amsterdam, 2007.
2. J. A. Goguen and R. M. Burstall. Introducing institutions. In *Logics of Programs*, volume 164 of *LNCS*, pages 221–256. 1984.
3. J. Meseguer. General logics. In *Logic Colloquium 1987*, pages 275–329, 1989.
4. R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980.

Representing CASL in a Proof-Theoretical Logical Framework

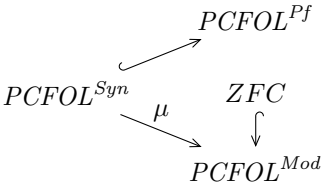
Mihai Codescu[1], Fulya Horozal[2], Iulia Ignatov[2], and Florian Rabe[2]

[1] DFKI GmbH Bremen, Germany
[2] Computer Science, Jacobs University Bremen, Germany

The Common Algebraic Specification Language CASL [9] provides a standard language for algebraic specification, expressive enough to subsume and thus unify many of the existing languages. At the basic level, the CASL logic is multi-sorted partial first-order logic with subsorting and induction. Moreover, sublogics can be obtained by restricting one or many of the features of the logics (e.g. partiality, subsorting) and the design of the CASL logic allows for language extensions (e.g. higher-order or coalgebraic extensions) which can be integrated easily. An extension of CASL to *heterogeneous* multi-logic specifications is supported by the Heterogeneous Tools Set Hets [8].

The Logic Atlas and Integrator (LATIN) project [1] develops a foundationally unconstrained framework for the representation of logics and translations between them [11]. It integrates proof-theoretical frameworks (such as LF [4]) and model-theoretic frameworks (such as institutions [3]) by giving a general definition of a logical framework, called the LATIN meta-framework [2].

The LATIN meta-framework follows a "logics-as-theories and translations-as-morphisms" approach, providing a generic construction of a logic from a theory graph in a logical framework. In particular, this has the advantage that we can use declarative logic representations in the LATIN meta-framework to automatically give rise to implementations of these logics. Thus, Hets can be extended conveniently not only on the developer's side but also directly by the user, in a significantly simplified manner.



In this work, we represent CASL as a logic in the instantiation of the LATIN meta-framework with LF [4] as a concrete representation language. Our representation follows closely the definition of the CASL logic in [9]. Firstly, we represent the syntax, proof theory and model theory of the multi-sorted version of CASL (*PCFOL*) in LF, which results in the diagram of LF theories and theory morphisms on the right. $PCFOL^{Syn}$ encodes the syntax by declaring one LF symbol for every class of expressions in CASL (e.g., sorts, functions, predicates, terms, etc.). This representation includes declaration patterns for CASL signatures, a new notion we add to LF that represents the different kind of symbol declarations allowed in signatures of a logic or a related declarative language [5]. $PCFOL^{Pf}$ encodes the proof-theory by adding one LF constant for every proof rule. $PCFOL^{Mod}$ encodes CASL models as theories of set theory ZFC, and μ interprets the CASL syntax by mapping each CASL concept in $PCFOL^{Syn}$ to a CASL model.

Secondly, we represent only the syntax of subsorting explicitly in a signature $CASL^{Syn}$ and obtain its semantics and proof theory via a translation of subsorting in *PCFOL*. We then use a logic program in the Twelf [10] implementation of LF combined with a pattern-based functor to implement the signature and expression translation from $CASL^{Syn}$ to $PCFOL^{Syn}$. The Twelf meta-theory guarantees that this translation preserves typing. The Twelf sources of our representation are available in the LATIN Logic Atlas [6].

This representation of CASL has two major benefits. Firstly, CASL extensions or variants can now be easily defined in LF and added to Hets automatically. Secondly, we can apply knowledge management services [7] (such as search, change management, querying, presentation, etc.) via the LATIN framework to CASL specifications.

References

1. M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, and F. Rabe. Project Abstract: Logic Atlas and Integrator (LATIN). In J. Davenport, W. Farmer, F. Rabe, and J. Urban, editors, *Intelligent Computer Mathematics*, volume 6824 of *Lecture Notes in Computer Science*, pages 287–289. Springer, 2011.
2. M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, F. Rabe, and K. Sojakova. Towards Logical Frameworks in the Heterogeneous Tool Set Hets. In T. Mossakowski and H. Kreowski, editors, *Recent Trends in Algebraic Development Techniques 2010*, volume 7137 of *Lecture Notes in Computer Science*, pages 139–159. Springer, 2012.
3. J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.
4. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.
5. F. Horozal and F. Rabe. Representing categories of theories in a proof-theoretical logical framework. `https://svn.kwarc.info/repos/fhorozal/pubs/theory-categories_abst.pdf`. Submitted to WADT 2012.
6. M. Kohlhase, T. Mossakowski, and F. Rabe. The LATIN Project, 2009. see `https://trac.omdoc.org/LATIN/`.
7. M. Kohlhase, F. Rabe, and V. Zholudev. Towards MKM in the Large: Modular Representation and Scalable Software Architecture. In S. Autexier, J. Calmet, D. Delahaye, P. Ion, L. Rideau, R. Rioboo, and A. Sexton, editors, *Intelligent Computer Mathematics*, volume 6167 of *Lecture Notes in Computer Science*, pages 370–384. Springer, 2010.
8. T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. In O. Grumberg and M. Huth, editor, *TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522, 2007.
9. Peter D. Mosses, editor. *CASL Reference Manual*. Number 2960 in LNCS. Springer Verlag, 2004.
10. F. Pfenning and C. Schürmann. System description: Twelf - a meta-logical framework for deductive systems. *Lecture Notes in Computer Science*, 1632:202–206, 1999.

11. F. Rabe. A Logical Framework Combining Model and Proof Theory. *Mathematical Structures in Computer Science*, 2012. to appear; see http://kwarc.info/frabe/Research/rabe_combining_10.pdf.

Compiling Logics

Mihai Codescu[1], Fulya Horozal[2], Till Mossakowski[1], and Florian Rabe[2]

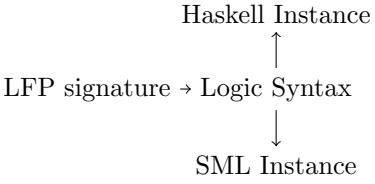
[1] DFKI GmbH Bremen, Germany
[2] Computer Science, Jacobs University Bremen, Germany

In [2], we presented an extension of the Heterogeneous Tool Set HETS [5] with a framework for representing logics independently of their foundational assumptions. The key idea [6] is that a graph of theories in a type theoretical logical framework like LF [3] can fully represent a model theoretic logic. Our integration used this construction to make the process of extending HETS with a new logic more declarative, on one side, and fully formal, on the other side.

However, the new logic in HETS inherits the syntax of the underlying logical framework. This is undesirable for multiple reasons. Firstly, a logical framework unifies many concepts that are distinguished in individual logics. Examples are binding and application (unified by higher-order abstract syntax), declarations and axioms (unified by the Curry-Howard correspondence), and different kinds of declarations (unified by LFP – LF with declaration patterns – as introduced in [4]). Therefore, users of a particular logic may find it unintuitive to use the concrete syntax (and the associated error messages) of the logical framework.

Secondly, only a small fragment of the syntax of the logical framework is used in a particular logic. For example, first-order logic only requires two base types *term* for terms and *form* for formulas and not the whole dependent type theory of LF. Therefore, it is unnecessarily complicated if implementers of additional services for a particular logic have to work with the whole abstract syntax of the logical framework. Such services include in particular logic translations from logics defined in LF to logics implemented by theorem provers.

Therefore, we introduce an architecture that permits compiling logics defined in LF into custom definitions in arbitrary programming languages. This is similar to parser generators, which provide implementations of parsers based on a language definition in a context-free grammar. Our work provides implementations of a parser and a type-checker based on a context-sensitive language definition in the recent extension of LF with *declaration patterns* (LFP) [4]. Here declaration patterns give a formal specification of the syntactic shape of the declarations in the theories of a logic.



An overview of the architecture is given on the right. The centerpiece is the compiler, which takes an LFP signature and produces an abstract representation of the syntax of the defined logic, including parsing and type-checking algorithms in a generic functional programming language. In a second step, the logic syntax can be easily serialized in, e.g., Haskell or SML.

A logic syntax corresponds to the straightforward implementation one would choose in a functional programming language. Given an LFP signature Σ, the compiler generates one abstract data type for every type family declaration in Σ. Here all type dependencies are erased, and only those LFP symbols with higher-order arguments are supported that can be readily interpreted as binders. Moreover, for every declaration pattern in Σ, one data type of declarations is generated, and lists of such declarations form the type of signatures. Finally, one type family must be distinguished as the type of formulas, which permits the definition of a type of theories.

For example, for propositional logic, the LF type $\mathit{form} : \mathit{type}$ generates one data type *form* whose constructors are the connectives. The declaration pattern `pattern PropVar` $= \{F : \mathit{form}\}$ for propositional variables gives rise to a data type `PropVar` with one constructor taking a string argument, i.e., the name of the declared symbol.

In addition, the compiler produces two families of recursive functions. Firstly, parsing functions map from a generic representation of the concrete syntax to the generated data types. This generic representation uses one keyword for each LFP declaration pattern to introduce declarations. For expressions, it uses a simplified variant of OpenMath objects [1]. Parsing of actual user input into this representation is straightforward and does not depend on Σ (except for possibly customizing the parser with the fixities and precedences declared in Σ). Secondly, typing functions map from the generated data types to a generic representation of the abstract syntax. This representation is based on LFP, and type checking the latter is again straightforward and parametric in Σ.

We are currently integrating this framework into HETS, using a re-implementation of first-order logic as an easy test case. This will provide insight about how much functionality is still missing in order to make the implementation of new logics in HETS fully declarative. Important such functionality includes signature colimits, amalgamability checks, and theorem prover interfaces. The plan for the future is to extend the framework with these features, and successively replace HETS' Haskell-coded logics with compiled LF-defined logics.

References

1. S. Buswell, O. Caprotti, D. Carlisle, M. Dewar, M. Gaetano, and M. Kohlhase. The Open Math Standard, Version 2.0. Technical report, The Open Math Society, 2004. See http://www.openmath.org/standard/om20.
2. M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, F. Rabe, and K. Sojakova. Towards Logical Frameworks in the Heterogeneous Tool Set Hets. In T. Mossakowski and H. Kreowski, editors, *Recent Trends in Algebraic Development Techniques 2010*, volume 7137 of *Lecture Notes in Computer Science*, pages 139–159. Springer, 2012.
3. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.
4. F. Horozal and F. Rabe. Representing categories of theories in a proof-theoretical logical framework. https://svn.kwarc.info/repos/fhorozal/pubs/theory-categories_abst.pdf. Submitted to WADT 2012.

5. T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. In O. Grumberg and M. Huth, editor, *TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522, 2007.
6. F. Rabe. A Logical Framework Combining Model and Proof Theory. *Mathematical Structures in Computer Science*, 2012. to appear; see http://kwarc.info/frabe/Research/rabe_combining_10.pdf.

On the Concurrent Semantics of Transformation Systems with Negative Application Conditions

A. Corradini[1], R. Heckel[2], F. Hermann[3], S. Gottmann[3], and N. Nachtigall[3]

[1] andrea(at)di.unipi.it, Dipartimento di Informatica, Università di Pisa, Italy
[2] reiko(at)mcs.le.ac.uk, University of Leicester, UK
[3] {frank.hermann,susann.gottmann,nico.nachtigall}(at)uni.lu, Interdisciplinary Center for Security, Reliability and Trust, Université du Luxembourg, Luxembourg

Extended Abstract

Graph Transformation Systems (GTSs) are an integrated formal specification framework for modelling and analysing structural and behavioural aspects of systems. The evolution of a system is modelled by the application of rules to the graphs representing its states and, since typically such rules have local effects, GTSs are particularly suitable for modelling concurrent and distributed systems where several rules can be applied in parallel. Thus, it is no surprise that a large body of literature is dedicated to the study of the concurrent semantics of graph transformation systems.

The classical results include – among others – the definitions of shift-equivalence and parallel production, exploited in the Church-Rosser and Parallelism theorems [5]: briefly, derivations that differ only in the order in which independent steps are applied are considered to be equivalent. Several years later, taking inspiration from the theory of Petri nets, deterministic processes were introduced [4], which are a special kind of GTSs, endowed with a partial order, and can be considered as canonical representatives of shift-equivalence classes of derivations. Next, the unfolding of a GTS was defined as a typically infinite non-deterministic process which summarizes all the possible derivations of a GTS [2]. Recently, all these concepts have been generalized to transformation systems based on ($\mathcal{M}$-)adhesive categories [6, 3, 1].

In this paper, we consider the concurrent semantics of GTSs that use the concept of Negative Application Conditions (NACs) for rules [7], which is widely used in applied scenarios. A NAC allows one to describe a sort of "forbidden context", whose presence around a match of a rule inhibits the application of the rule. Existing contributions that generalize the concurrent semantics of GTSs to the case with NACs are not always satisfactory.

For example, the definition of shift-equivalence was generalized to adhesive transformation systems with NACs in [11]. However, as shown in [8], unlike the case without NACs, the notion of *sequential independence* among derivation steps is not stable under switching. More precisely, it is possible to find a derivation made of three direct derivations $s = (s_1; s_2; s_3)$ where $(s_1; s_2)$ are sequentially independent, but the corresponding steps $(s'_1; s'_2)$ are not sequentially independent in the shift-equivalent derivation $s' = (s'_3; s'_1; s'_2)$ (obtained

with the switchings $(2 \leftrightarrow 3; 1 \leftrightarrow 3)$). This fact implies that it is not possible to generalize in a direct way to the case with NACs the results concerning deterministic and non-deterministic processes that are known to hold for plain GTSs. For example, it is not possible to establish a one-to-one correspondence between shift-equivalent classes of derivations and isomorphism classes of processes, as done for example in [4] for DPO rewriting without NACs, because in the above situation s_1 and s'_1 would be represented by the same event in the process, and similarly for s_2 and s'_2, but then these two events should be both causally related and independent, resulting in a contradiction. Clearly, for the same reason all other process based semantics (including non-deterministic processes and unfoldings) cannot be applied straightforwardly to GTSs with NACs.

Inspired by a solution proposed in [8], we first show that the stability of sequential independence with respect to switching is still guaranteed if all the NACs $L \to N$ are *incremental*, i.e., if there are two decompositions $L \to N_1 \to N = L \to N = L \to N_2 \to N$, then there exist either an $f : N_1 \to N_2$ or a $g : N_2 \to N_1$ such that the resulting triangles commute. Next we present a construction that, given a rule r with arbitrary NACs generates a set of rules $Inc(r)$ with incremental NACs only preserving the rewriting relation, i.e., such that $G \Rightarrow_r H$ if and only if there is a rule $r' \in Inc(r)$ such that $G \Rightarrow_{r'} H$. The construction can be extended to a function from derivations over a set of rules $\mathcal{G}$ to derivations over the set $Inc(\mathcal{G})$. By exploiting this function we define a new notion of independence: $d_1; d_2$ are independent in $\mathcal{G}$ iff $Inc(d_1; d_2)$ are independent in $Inc(\mathcal{G})$. This stricter notion of independence should enjoy the desired properties needed to lift the process and unfolding semantics to DPO rewriting with NACs.

In the presentation, we shall also discuss other problematic aspects of the currently available concurrent semantics of DPO rewriting with NACs. For example, one can observe that in the case without NACs the concurrent execution of sequentially independent steps is "safe", i.e., independent from the internal interleaving of the addition and deletion of structural elements within the modelled system. Instead, we will show that this is not generally the case for rules with NACs, because the forbidden pattern of one rule could show up temporarily during the application of the other rule. In this framework, we will discuss a stricter condition of independence that ensures safety.

Finally, we will discuss the relationship between the above topics and our previous work where we argued that shift-equivalence with NACs is too strict to relate all equivalent transformation sequences, and we introduced the notion of permutation equivalence that captures exactly all equivalent linearisations of a process [9, 10] obtained from a transformation sequence with NACs. Since, permutation equivalence is more general than switch equivalence with NACs, the problem that consecutive independent steps are not preserved from one linearization to an equivalent one is again present. As we will show, this problem does not occur for systems with incremental NACs.

References

1. Paolo Baldan, Andrea Corradini, Tobias Heindel, Barbara König, and Pawel Sobocinski. Unfolding grammars in adhesive categories. In *CALCO*, volume 5728 of *LNCS*, pages 350–366. Springer, 2009.
2. Paolo Baldan, Andrea Corradini, Ugo Montanari, and Leila Ribeiro. Unfolding semantics of graph transformation. *Inf. Comput.*, 205(5):733–782, 2007.
3. Andrea Corradini, Frank Hermann, and PawełSobociński. Subobject Transformation Systems. *Applied Categorical Structures*, 16(3):389–419, 2008.
4. Andrea Corradini, Ugo Montanari, and Francesca Rossi. Graph processes. *Funda. Info.*, 26:241–265, 1996.
5. H. Ehrig. Introduction to the Algebraic Theory of Graph Grammars (A Survey). In *Graph Grammars and their Application to Computer Science and Biology*, volume 73 of *LNCS*, pages 1–69. Springer, 1979.
6. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theor. Comp. Science. Springer, 2006.
7. A. Habel, R. Heckel, and G. Taentzer. Graph Grammars with Negative Application Conditions. *Special issue of Fundamenta Informaticae*, 26(3,4):287–313, 1996.
8. Reiko Heckel. DPO Transformation with Open Maps. Submitted., 2012.
9. Frank Hermann. Permutation Equivalence of DPO Derivations with Negative Application Conditions based on Subobject Transformation Systems. *EC-EASST*, 16, 2009.
10. Frank Hermann, Andrea Corradini, and Hartmut Ehrig. Analysis of Permutation Equivalence in M-adhesive Transformation Systems with Negative Application Conditions. *MSCS*, 2012. To appear.
11. L. Lambers, H. Ehrig, F. Orejas, and U. Prange. Parallelism and Concurrency in Adhesive High-Level Replacement Systems with Negative Application Conditions. In H. Ehrig, J. Pfalzgraf, and U. Prange, editors, *Proceedings of the ACCAT workshop at ETAPS 2007*, volume 203 / 6 of *ENTCS*, pages 43–66. Elsevier, 2008.

Decision Algebra: Parameterized Specification of Decision Models

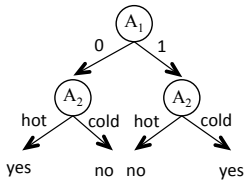
Antonina Danylenko[1], Wolf Zimmermann[2] and Welf Löwe[1]

[1] Linnaeus University, Software Technology Group,
351 95 Växjö, Sweden
{antonina.danylenko,welf.lowe}@lnu.se
[2] Martin-Luther-Universität Halle Wittenberg, Institut für Informatik,
061 20 Halle(Saale), Germany
{wolf.zimmermann}@informatik.uni-halle.de

Introduction. Processing decision information to adjust and alter applications' behaviour is a constitutive part in different fields of Computer Science, such as Data Mining, Software Engineering, Artificial Intelligence, etc. In general, decision information is an information that is used to deduce the relationship between a certain context and a certain decision. It is usually represented by a decision model (or classifier), which is a set of rules that determines a target decision based on the current context. Frequently used examples of decision models are decision tables, decision trees, support vector machines, etc.

Problem. In general, capturing decision information in decision models can have the problems like data replication (redundancy in the stored information) and model overfitting possibly leading to data fragmentation (the amount of information is too small to make a statistically significant decision) [5]. Thus, an important choice to be made is to select and implement a certain decision model for a particular problem domain. Issues such as model accuracy, capturing time, robustness, and scalability must be considered and can involve tradeoff between (1) the expressiveness of the model, i.e., the representation should be as close as possible to the initial captured data, (2) the memory required to keep this high-detailed decision model and (3) the time and initial data required in order to choose among the alternative decisions it can represent [3]. Moreover, developing or adjusting algorithms for processing this information might require adding new operations which, in general, have a negative impact on the problem domain memory requirements and performance.

Decision Algebra. Because of this variety of application domains with decision problems (each coming with different notations and tailored implementations) we consider it worthwhile to introduce a general algebraic specification, referred to as Decision Algebra, for describing an abstract data type that corresponds to a general decision model representation. Speaking of decision models we may have in mind decision models originated in Machine Learning filed, which keep different data (e.g. distributions, coefficients, probabilities) required for a correct decision making. Other examples may include decision models in Software Engineering, such as models representing program analysis results, software complexity models or service quality models.



spec *BinaryC* = *C+*
sorts *binaryC*
opns
⊔*: binaryC binaryC → binaryC*
yes: → binaryC
no: → binaryC

Fig. 1. Example of the Decision Tree *DT*

Every model has in general its own specification based on which different types of implementation are allowed. Therefore, it could be more effective to give one general specification for decision model, which can be parameterized, such that concrete instantiations may be obtained by suitable actualization of the parameter [2]. Such general algebraic specification for decision models can play an important conceptual role in software specifications where a decision process is involved, and development focused on functional programming.

Decision Algebra Specification. In our previous work [1] we introduced the Decision Algebra that was limited to decision trees and decision tables, where the specification was not parameterized and was particularly dedicated to certain decision model representation. The purpose of the current work is to present the Decision Algebra $DA(T, C, DF(T, C))$ as a parameterized specification that provides the representation of decision information as decision function DF along with a set of operations and equations. By replacing a formal parameter DF by an actual parameter specification we can obtain a new specification which defines a concrete decision model implementation.

spec $DA(T, C, DF(T, C)) = DF(T, C)+$
opns *evert: df int → df; approx: df int → df; apply*$^{(n)}$*: fun df*$^{(n)}$ *→ df*
vars d_1, d_2 *: df; d: df*$^{(n)}$*; i : int; f : fun*
axioms *arity(approx*$(d_1, i)) \leq$ *arity*$(d_1) =$ *true*
evert$(d_1, i) \equiv d_1 =$ *true* *arity(evert*$(d_1, i)) =^{int}$ *arity*$(d_1) =$ *true*
apply$^{(n)}$*(f, d) = eval*$^{(n)}$ *(f, d) approx*$(d_1, i) \sqsubseteq d_1 =$ *true*

A decision function $df^n : T \to C$, where $T = A_1 \times \ldots \times A_n$ is a tuple of contexts (attributes) that lead to a decision C, serves as a formal parameter specification which in turn is also parameterized by the specific decision information that is kept in a certain decision model, i.e. by concrete representation of tuple T and decision C.

spec $DF(T, C) = T+, C+, INT$
sorts *df, fun*
opns *decide: df t → c;* $\sqsubseteq$*: df df → bool;* $\equiv$*: df df → bool;*
arity: df → int[1]*; eval*$^{(n)}$*: fun df*^{n} *→ df*
vars d_1, d_2 *: df*
axioms *(*$\forall$ *x: t, decide*$(d_1, x) \sqsubseteq$ *decide*$(d_2, x) =$ *true)* $\Rightarrow \sqsubseteq (d_1, d_2) =$ *true*
($\forall$ *x :t, decide*$(d_1, x) =^c$ *decide*$(d_2, x) =$ *true)* $\Rightarrow \equiv (d_1, d_2) =$ *true*

In this case the only requirement for parameter C is definition of partial order $\sqsubseteq$ over decisions $c_1, c_2 \in C$.

To specify parameter passing mechanism in this work we define a specification morphism that replaces a formal by an actual parameter specification and

[1] Note that *arity* operation specifies the number of attributes that influences the final decision

therefore results in a concrete decision model representation. For example, the decision tree model depicted in Figure 1 corresponds to the specific implementation $MyDT(T)$ of the decision function $DF(T, BinaryC)$ that is parameterized with the $BinaryC$ values representing the decision C.

spec $MyDT(T) = T+,\ BinaryC+,\ INT`+$
sorts $A_1,\ A_2,\ A_3,\ V_1,\ V_2,\ fun$
sub-sorts $A_1 \sqsubseteq df,\ A_2 \sqsubseteq df,\ A_3 \sqsubseteq df$
opns
$tuple\colon V_1 V_2 \to t$; $merge\colon \to fun$; $eval^{(2)}\colon fun\ df\ df \to df$; $decide\colon t\ df \to binaryC$;
$a_1\colon A_2\ A_2 \to A_1$; $a_2\colon A_3\ A_3 \to A_2$; $\sqcup\colon A_1\ A_1 \to A_1$; $\sqcup\colon A_1\ A_2 \to A_1$;
$leaf\colon c \to A_3$; $0\colon \to V_1$; $1\colon \to V_1$; $hot\colon \to V_2$;
$\Pi_1\colon T \to V_1$; $\Pi_2\colon T \to V_2$ $cold\colon \to V_2$;
vars $x_1,\ x_2, u_1, u_2\colon df;\ t_1, t_2\colon t,\ c_1, c_2\colon binaryC$
axioms $(decide(t,\ leaf(c_1)) = c_1;\ eval^{(2)}(merge, x_1, x_2) = x_1 \sqcup x_2$
$(\Pi_1(t_1) = 0) \Rightarrow decide(t_1,\ a_1(x_1, x_2)) = decide(t_1,\ x_1)$
$(\Pi_1(t_1) = 1) \Rightarrow decide(t_1,\ a_1(x_1, x_2)) = decide(t_1,\ x_2)$
$(\Pi_2(t_1) = hot) \Rightarrow decide(t_1,\ a_1(x_1, x_2)) = decide(t_1,\ x_1)$
$(\Pi_2(t_1) = cold) \Rightarrow decide(t_1,\ a_1(x_1, x_2)) = decide(t_1,\ x_2)$
$a_1(x_1, x_2) \sqcup a_2(u_1, u_2) = a_1((x_1 \sqcup a_2(u_1, u_2)), (x_2 \sqcup a_2(u_1, u_2)))$
$a_2(x_1, x_2) \sqcup a_2(u_1, u_2) = a_2((x_1 \sqcup u_2), (x_2 \sqcup u_2))$
$leaf(c_1) \sqcup leaf(c_2) = leaf(c_1 \sqcup c_2);\ approx(a_1(x_1, x_2),\ 1) = x_1 \sqcup x_2$
$approx(a_1(a_2(x_1, x_2), a_2(u_1, u_2)),\ 2) = a_1((x_1 \sqcup u_1), (x_2 \sqcup u_2))$

Outcome. Central idea of the Decision Algebra is the idea of a function, which computes a result that depends on the values of its inputs. This very much correlates with the ideas of functional programming that gives the clearest possible view of abstraction (in a function) and data abstraction (in an abstract data type) [4, 6]. Therefore, using our Decision Algebra, decision models can be directly implemented as a data type in functional languages, whereas in other languages one is forced to describe them by different specific data structures with a number of additional operations. Another strengths of having a such general algebraic specification for decision models is that we can define general functions which can be used in different applications and, therefore, construct the pattern of computation: transform every decision model in a same way and combine the decision models using same operator. Due to this generalization, insights can be gained at an abstract level or reused between application domains, paving the way for a deeper problem understanding and, possibly, for novel and more efficient algorithms for combining different decision models.

References

1. Antonina Danylenko, Jonas Lundberg, and Welf Löwe. Decisions: Algebra and Implementation. In *7th International Conference on Machine Learning and Data Mining (MLDM 2011)*, New-York/USA, August-September 2011.
2. Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer-Verlag, 1985.
3. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
4. Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala: A Comprehensive Step-by-step Guide*. Artima Incorporation, USA, 1st edition, 2008.
5. M. Steinbach P.-N. Tan and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 2005.

6. Simon Thompson. *The Haskell: The Craft of Functional Programming.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1999.

A module algebra for behavioural specifications

Răzvan Diaconescu

Simion Stoilow Institute of Mathematics of the Romanian Academy

The structuring of specifications, or specification in-the-large, is a crucial paradigm for building complex specification. Moreover it is known to have superior specification power than unstructured or flat specification paradigm, in the sense that one is able to specify classes of models that cannot be specified by flat specifications (e.g. see [6] for some examples). The modern specification languages, including CASL [1] and CafeOBJ [7] among others, implement rather powerful structuring or modularization techniques based upon the rather vast institution theory literature. An important part of the study of modularization systems is the development of appropriate module algebras [2, 9, 6, 15] which includes both the definition of appropriate structuring operators and the investigation of relevant equations between module expressions.

On the other hand behavioral specification [13, 10, 11, 8, 12, 14, 4] constitutes one of the most promising algebraic specifications paradigm, which in my opinion is still insufficiently developed both at the mathematical and methodological levels. One aspect that needs further serious consideration may be a comprehensive theory of structuring behavioral specifications which may serve as a basis for a proper realization of behavioral specification paradigm within concrete specification languages. An important part of such investigation would consist of a development of an algebra for behavioral modules.

One may raise the following (rather legitimate) point. Given the current well developed sophisticated modularization theories available within the algebraic specification literature, would it not be enough just to instantiate some of those to some appropriate behavioral logic institution, such as some version of hidden algebra? I think this is only partly possible due to some of the peculiar specificities of the behavioral or hidden algebra institutions. For example the union of signatures displays a very specific type of partiality induced by two factors: (1) the requirement to keep the hidden and the visible sorts separate, and (2) the so-called 'encapsulation' condition on the signature morphisms that is crucial for the Satisfaction Condition to hold.

Here we report a very recent development of a module algebra for behavioral specifications meant to be part of a comprehensive theory of structuring behavioral specifications. The main features of this theory are as follows.

1. While it is an institution dependent study, on the one hand it is based upon well established institution theoretic techniques (e.g. pushouts, inclusion systems, etc.) and, on the other hand it relies upon most recent advances in institution-independent modularization theory (e.g. [6, 5]).
2. The behavioral specification logic considered is a version [8] of Goguen's hidden algebra [10].

3. The module algebra is a partial one (in the sense of [3]) and satisfies rather interesting versions of important rules such as associativity, distributivity, etc.

References

1. Edigio Astesiano, Michel Bidoit, Hélène Kirchner, Berndt Krieg-Brückner, Peter Mosses, Don Sannella, and Andrzej Tarlecki. CASL: The common algebraic specification language. *Theoretical Computer Science*, 286(2):153–196, 2002.
2. Jan Bergstra, Jan Heering, and Paul Klint. Module algebra. *Journal of the Association for Computing Machinery*, 37(2):335–372, 1990.
3. Peter Burmeister. *A Model Theoretic Oriented Approach to Partial Algebras*. Akademie-Verlag, Berlin, 1986.
4. Răzvan Diaconescu. Coinduction for preordered algebras. *Information and Computation*, 209(2):108–117, 2011.
5. Răzvan Diaconescu. An axiomatic approach to structuring specifications. *Theoretical Computer Science*, 433:20–42, 2012.
6. Răzvan Diaconescu and Ionuţ Ţuţu. On the algebra of structured specifications. *Theoretical Computer Science*, 412(28):3145–3174, 2011.
7. Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998.
8. Răzvan Diaconescu and Kokichi Futatsugi. Behavioural coherence in object-oriented algebraic specification. *Universal Computer Science*, 6(1):74–96, 2000. First version appeared as JAIST Technical Report IS-RR-98-0017F, June 1998.
9. Răzvan Diaconescu, Joseph Goguen, and Petros Stefaneas. Logical support for modularisation. In Gerard Huet and Gordon Plotkin, editors, *Logical Environments*, pages 83–130. Cambridge, 1993. Proceedings of a Workshop held in Edinburgh, Scotland, May 1991.
10. Joseph Goguen and Răzvan Diaconescu. Towards an algebraic semantics for the object paradigm. In Hartmut Ehrig and Fernando Orejas, editors, *Recent Trends in Data Type Specification*, volume 785 of *Lecture Notes in Computer Science*, pages 1–34. Springer, 1994.
11. Joseph Goguen and Grant Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, 2000.
12. Rolf Hennicker and Michel Bidoit. Observational logic. In A. M. Haeberer, editor, *Algebraic Methodology and Software Technology*, number 1584 in LNCS, pages 263–277. Springer, 1999. Proc. AMAST'99.
13. Horst Reichel. Behavioural equivalence – a unifying concept for initial and final specifications. In *Proceedings, Third Hungarian Computer Science Conference*. Akademiai Kiado, 1981. Budapest.
14. Grigore Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000.
15. Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specifications and Formal Software Development*. Springer, 2012.

Query Languages are Cartesian Monads (Extended abstract)

Zinovy Diskin[1,2] and Tom Maibaum[1]

[1] McMaster University, Canada
[2] University of Waterloo, Canada

zdiskin@gsd.uwaterloo.ca,tom@maibaum.org

Query languages (QLs) are an important ingredient of information technologies. A number of QLs are well-known and work well in their respective domains: SQL in the relational world, XQuery and XPath for semi-structured data/XML, and OCL in the model-driven engineering (MDE) world. However, modern software is extremely heterogeneous, and combines components from different technological worlds. In addition, new applications often require querying facilities, even if the latter are not recognized as queries. For example, such a situation is typical for the model transformation area in the MDE world, where queries are encoded as transformation rules and are hidden in the transformation code. Thus, we often need to translate from one query language to another, or to identify queries in a particular piece of code. Hence, we need a generic notion of a query language independent of a particular data definition language. More precisely, given a data definition language DDL, what is a mathematical model for a query language QL over DDL? Surprisingly, it seems that such a generic model of QL was never mathematically formulated, and this is where our contribution goes.

Our interest in the problem is mainly motivated by MDE applications [2]. We call data schemas *metamodels*, and data instances *models*. This fits with the jargon of institutions, where metamodels are theories, and their models are our models. Our solution to the generic QL-over-DDL problem consists of the following steps.

1 DDL and its semantics. We model a DDL together with its semantics as a split fibration $\boldsymbol{p}$: $\boldsymbol{Mod} \to \boldsymbol{MMod}$.

Objects of category $\boldsymbol{MMod}$ are metamodels, and arrows are mappings between them. The reader may think of metamodels as graphs with diagram predicates (Makkai's generalized sketches), that is, pairs $M = (G_M, C_M)$ with G_M the carrier graph of M and C_M the constraints declared in M. Metamodel mappings are then graph morphisms compatible with constraints (very much like theory moprhisms in institutions are signature morphisms compatible with sentences); details can be found in [1]. Given a metamodel M as above, its *instance* A can be thought of as a data graph G_A typed over G_M, i.e., supplied with a typing mapping $t_A\colon G_A \to G_M$, such that the constraints are satisfied, $t_A \models C_M$; we again refer to [1] for details.

2 QL and its semantics. We model a QL over a DDL $\boldsymbol{p}$ by a pair of monads $(\mathsf{Q}_{\mathrm{def}}, \mathsf{Q})$ over categories $\boldsymbol{MMod}$ and $\boldsymbol{Mod}$, resp. The first monad describes the

syntax (query definitions), and the second one provides the semantics (query execution). In contrast to ordinary algebraic operations, whose results stay within the same carrier universe, the result of a query is recognized as new information (of a new type), and hence goes beyond the original dataset. In other words, a query Q to a model $t_A\colon G_A \to G_M$ adds to the latter new elements, and thus can be seen as an augmented typing mapping $t_A^+\colon G_A^+ \to G_M^+$ from the augmented data graph to the augmented type graph (there are inclusions $X \hookrightarrow X^+ = Q(X)$ for all three components $X = t_A, G_A, G_M$). Simple sanity checks show that we should require functor $\boldsymbol{p}$ to be a monad morphism.

Moreover, a fundamental property of queries is that the original data are not affected: queries compute new data but do not change the original. Mathematical modeling of this property results in a number of equations, which can be summarized by saying that monad Q is $\boldsymbol{p}$-Cartesian, i.e., the Cartesian and the monad structure work in sync. We come to a compact algebraic model of QL as a Cartesian monad over a fibration modeling DDL. Unfortunately, this nice algebraic definition only works well under an additional assumption that all queries are *monotonic*: an injection of data instances $i\colon G_A \to G_B$ gives rise to an injection of query results $i^+\colon G_A^+ \to G_B^+$. Thus, our observations result in the following main definition.

Definition 1 () An *abstract DDL* is a split fibration $\boldsymbol{p}\colon \boldsymbol{Mod} \to \boldsymbol{MMod}$. A *monotonic query language* over $\boldsymbol{p}$ is a pair of monads $(\mathsf{Q}_{\mathrm{def}}, \mathsf{Q})$ such that $\boldsymbol{p}$ is a monad morphism, and monad Q is $\boldsymbol{p}$-Cartesian.

3 View mechanism.

Theorem 1. *Let* $(\mathsf{Q}, \mathsf{Q}_{\mathrm{def}})$ *be a monotonic query language over an abstract DDL* $\boldsymbol{p}\colon \boldsymbol{Mod} \to \boldsymbol{MMod}$. *This gives rise to a split fibration* $\boldsymbol{p}_{\mathsf{Q}}\colon \boldsymbol{Mod}_{\mathsf{Q}} \to \boldsymbol{MMod}_{\mathsf{Q}_{\mathrm{def}}}$ *between the corresponding Kleisli categories.*

The Kleisli categories above have immediate practical interpretations. Morphisms in $\boldsymbol{MMod}_{\mathsf{Q}_{\mathrm{def}}}$ are nothing but view definitions: they map elements of the source metamodel to queries against the target one. Correspondingly, morphisms in $\boldsymbol{Mod}_{\mathsf{Q}}$ are view executions composed from query execution followed by retyping. The theorem says that for monotonic query languages, the view execution mechanism is compositional: execution of a composed view (i.e., execution of the composed query followed by retyping) equals the composition of executions (execute the first query, retype, then execute the second query and retype).

References

1. Z. Diskin and U. Wolter. A diagrammatic logic for object-oriented visual modeling. *Electron. Notes Theor. Comput. Sci.*, 203(6):19–41, 2008.
2. Zinovy Diskin, Tom Maibaum, and Krzysztof Czarnecki. Intermodeling, queries, and Kleisli categories. In Juan de Lara and Andrea Zisman, editors, *FASE*, volume 7212 of *Lecture Notes in Computer Science*, pages 163–177. Springer, 2012.

Statistical Model-Checking for Composite Actor Systems*

Jonas Eckhardt[1], Tobias Mühlbauer[1], José Meseguer[2], and Martin Wirsing[3]

[1] Technical University of Munich
[2] University of Illinois at Urbana-Champaign
[3] Ludwig Maximilian University of Munich

A wide class of real-world entities can be expressed as a composition of communicating sub-entities. The Internet is a prominent example of such an entity which is composed of various participants and systems and is hierarchically structured in different layers and networks. Such composed entities are often safety- and security-critical, have strong qualitative and quantitative formal requirements, equally important time-critical properties, and need to dynamically interact with a potentially probabilistic environment.

In order to formally specify and analyze such composed entities, we use rewriting logic as the semantic framework and Maude, a language and system based on rewriting logic that offers the possibility of executing and formally analyzing specifications. Our specifications are based on the actor model of computation [1], a mathematical model of concurrent computation in distributed systems. The actor model allows the natural description of entities which communicate through message passing. Temporal logic properties of such actor-based models are model checked either by exact model checking algorithms or, in an approximate but more scalable way, by statistical model checking. The idea of statistical model checking is to verify the satisfaction of a temporal logic property by statistical methods up to a user-specified level of statistical confidence. For this, a large enough number of Monte-Carlo simulations of the system are performed, and the formula is evaluated on each of the simulations. PVeStA [3] is an extension and parallelized version of the VeStA model checking tool. It supports the statistical model checking analysis of probabilistic rewrite theories in Maude. Properties are thereby expressed as QuaTEx formulas.

Current statistical model checking methods require that the system is purely probabilistic, i.e., that there is no non-determinism in the choice of transitions. This is nontrivial to achieve in a way that faithfully models system behavior for a distributed system, where many different components may perform local transitions concurrently. In PMaude [2], Agha et al. present an approach to avoid un-quantified non-determinism by associating continuous probability distributions with message delays and computation time and by relying on the fact, that for continuous distributions the probability of sampling the same real number twice is zero. In [4], AlTurki et al. present a different approach to achieve the same goal by introducing a scheduler that provides a deterministic ordering of

* This work has been partially sponsored by the EU-funded project FP7-257414 ASCENS and AFOSR Grant FA8750-11-2-0084

messages. These approaches allow the model checking of standard probabilistic temporal logic properties as well as quantiative temporal logic properties.

Both approaches rely on a "flat soup of actors" in the model and neither can handle composite models. In this work, we (i) contribute a coordination mechanism that faithfully models the distributed behavior of composite actor systems and reflects the so-called "Russian Dolls" model [7] to support an arbitrary hierarchical composition of entities; and (ii) show that this mechanism naturally supports a scheduling approach for composite actor models that guarantees the absence of non-determinism which is an important prerequisite for statistical model checking. Our method is very general and can be applied to a wide range of composite actor systems. To the best of our knowledge, our solution is the first one making it possible to analyze such systems in a faithful way by statistical model checking.

In particular, we extend the actor model of computation to support actor composition, i.e., actors may contain a soup of actors, which again may contain soups of actors, and so on. This hierarchical composition is analogous to the "Russian Dolls" model [7]. In Maude, we specify a composite actor as a term, e.g., < 0.1.2 | config: C>, which contains an address (here 0.1.2) and an inner soup of objects and messages (C)[4]. Furthermore, we introduce a hierarchical naming scheme, which allows the automatic generation of fresh names. Our scheduling approach that prevents un-quantified non-determinism works as follows: Messages are emitted as inactive messages at any level of a composed actor hierarchy. Inactive messages are inserted into the scheduler by equational simplification which stores the messages in a strict order. Whenever the actor system cannot perform any transition, the scheduler inserts the message that is scheduled to become active next into the subconfiguration which emitted it. Our approach only requires local transition rules and boundary crossing rules to cross a single composition boundary to be specified. This means that no rules need to be specified that take the whole system composition into account.

Our methodology to verify composite distributed systems follows the following stages:

1. Specification of the real-world entity as a composite actor system in PMaude.
2. Definition of appropriate standard probabilistic temporal logic properties and quantiative temporal logic properties for describing the required quality of service properties.
3. Specification of an initial state which contains the top-level scheduler.
4. Formal analysis of the defined properties over the initial state using statistical model checking in PVeStA [3].

A simple case study illustrates our method for designing and validating composite distributed systems; we model an abstraction of a distributed system as

[4] For the sake of brevity, we ommited additional properties of a composite actor. Composite actors are constructed by op <_:_|config:_, _> : Address ActorType Config AttributeSet −> Actor . .

a binary tree of composite actors[5]. Messages are randomly sent between the leaf actors in the tree while the intermediate composite actors only forward the messages up or down in the composition hierarchy.

References

1. Gul Agha. *Actors: a model of concurrent computation in distributed systems*. MIT Press, 1986.
2. Gul Agha, José Meseguer, and Koushik Sen. PMaude: Rewrite-based Specification Language for Probabilistic Object Systems. *ENTCS*, 153:213–239, 2006.
3. M. AlTurki and J. Meseguer. PVeStA: A parallel statistical model checking and quantitative analysis tool. In *CALCO*, volume 6859 of *LNCS*, pages 386–392, 2011.
4. Musab AlTurki, José Meseguer, and Carl A. Gunter. Probabilistic Modeling and Analysis of DoS Protection for the ASV Protocol. *ENTCS*, 234:3–18, 2009.
5. Jonas Eckhardt. A Formal Analysis of Security Properties in Cloud Computing. Master's thesis, LMU Munich, TU Munich, 2011.
6. Jonas Eckhardt, Tobias Mühlbauer, Musab AlTurki, José Meseguer, and Martin Wirsing. Stable availability under denial of service attacks through formal patterns. In *FASE*, pages 78–93, 2012.
7. José Meseguer and Carolyn L. Talcott. Semantic Models for Distributed Object Reflection. In *Proceedings of the 16th European Conference on Object-Oriented Programming*, ECOOP, pages 1–36. Springer, 2002.
8. Tobias Mühlbauer. Formal Specification and Analysis of Cloud Computing Management. Master's thesis, LMU Munich, TU Munich, 2011.

[5] The abstraction is deliberately simple as we only want to illustrate the usefullness of our methodology for highly hierarchical systems. Other applications of our approach are shown in [6, 8, 5].

On Linear Contravariant Semantics*

Ignacio Fábregas, David de Frutos Escrig, and Miguel Palomino

Departamento de Sistemas Informáticos y Computación, UCM
fabregas@fdi.ucm.es {defrutos, miguelpt}@sip.ucm.es

Classical semantics for processes usually treat all the actions in *Act* uniformly. Certainly, there have been several works where some modality features have been taken into account, in particular the input-output character [4], but this has only been applied to some very specific semantics. Recently, we have studied [3, 1, 2] the covariant-contravariant simulation which introduces, nicely and in a quite general way, modalities in the simulation framework.

Our goal now is to extend the dual nature of covariant and contravariant actions to the linear-time semantics, such as traces, failures, or readiness, targeting the interesting cases. Except for the trace semantics, these are not related with plain simulation but with ready simulation and this is why we have started our study by considering ready cc-simulation. Classical ready simulations impose that pairs of related processes have the same offerings: we keep this same constraint for the ready cc-simulation since equality is dual to itself.

Definition 1. *Given $P = (P, A, \rightarrow_P)$ and $Q = (Q, A, \rightarrow_Q)$, two labeled transition systems for the alphabet A, and a partition $\{A^r, A^l\}$ of this alphabet, a* ready covariant-contravariant simulation *is a relation $R \subseteq P \times Q$ such that for every pRq we have:*

- *$I(p) = I(q)$.*
- *For all $a \in A^r$ and all $p \xrightarrow{a} p'$ there exists $q \xrightarrow{a} q'$ with $p'Rq'$.*
- *For all $b \in A^l$ and all $q \xrightarrow{b} q'$ there exists $p \xrightarrow{b} p'$ with $p'Rq'$.*

We write $p \lesssim_{ccR} q$ if there exists a ready covariant-contravariant simulation R such that pRq.

The same transformation which provides the axiom governing ready simulation from that defining simulation can be applied to the axioms for cc-simulation, producing:

(RS^r_p) $a_r x \sqsubseteq a_r x + a_r y.$ (RS^l_p) $a_l x + a_l y \sqsubseteq a_l x.$

In a similar way, we can obtain the axioms for the induced equivalence:

$(\mathsf{RS}^{r,r}_{\equiv})$ $a_r(x + b_r y + b_r z) = a_r(x + b_r y + b_r z) + a_r(x + b_r y).$
$(\mathsf{RS}^{r,l}_{\equiv})$ $a_r(x + a_l y) = a_r(x + a_l y) + a_r(x + a_l y + a_l z).$
$(\mathsf{RS}^{l,r}_{\equiv})$ $a_l(x + a_r y) = a_l(x + a_r y) + a_l(x + a_r y + a_r z).$
$(\mathsf{RS}^{l,l}_{\equiv})$ $a_l(x + b_l y + b_l z) = a_l(x + b_l y + b_l z) + a_l(x + b_l y).$

* Research supported by the Spanish projects DESAFIOS10 TIN2009-14599-C03-01, TESIS TIN2009-14321-C02-01 and PROMETIDOS S2009/TIC-1465.

After taking care of all the necessary details, the completeness proof for the axioms for cc-simulation in [3] can be transferred to ready cc-simulation.

Plain traces (covariant traces in the following) associate to each process p a set of traces $\mathrm{Tr}(p)$ in such a way that the trace preorder $\lesssim_T$ is defined as: $p \lesssim_T q$ iff $\mathrm{Tr}(p)) \subseteq \mathrm{Tr}(q)$. Instead, whenever all the actions are contravariant, contravariant traces should allow us to define just the "opposite" order $p \lesssim_{cT} q$ iff $q \lesssim_T p$, by means of set inclusion between the corresponding sets of contravariant traces $\mathrm{Tr}_C(p)$; that is, $p \lesssim_{cT} q$ iff $\mathrm{Tr}_C(p) \subseteq \mathrm{Tr}_C(q)$. Since $T_1 \subseteq T_2$ iff $\overline{T_2} \subseteq \overline{T_1}$, in this simple case it would be enough to take as contravariant traces the set of *no-traces* of each process, that is, the complement of the set of traces: $\mathrm{Tr}_C(p) = \overline{\mathrm{Tr}(p)}$. In the presence of both covariant and contravariant action we combine the two definitions above to obtain the following recursive definition of the set of cc-traces of a process:

Definition 2. *Given a partition $\{A^r, A^l\}$ of a set of actions, we say that α is a covariant-contravariant trace of the process p (denoted by $\alpha \in \mathrm{Tr}_{cc}(p)$) if and only if:*

- $\alpha = \langle\rangle$, *or*
- $\alpha = \langle b\rangle$ *with* $b \in A^l$, *and* $p \stackrel{b}{\not\longrightarrow}$, *or*
- $\alpha = \langle a\rangle \cdot \alpha'$, $a \in A^r$, *and* $p \stackrel{a}{\longrightarrow} p'$ *with* $\alpha' \in \mathrm{Tr}_{cc}(p')$, *or*
- $\alpha = \langle b\rangle \cdot \alpha'$ *with* $b \in A^l$, $\alpha' \neq \langle\rangle$ *and* $\alpha' \in \mathrm{Tr}_{cc}(p')$ *for all* $p \stackrel{b}{\longrightarrow} p'$.

The covariant-contravariant trace preorder $p \lesssim_{ccT} q$ is defined by: $p \lesssim_{ccT} q$ iff $\mathrm{Tr}_{cc}(p) \subseteq \mathrm{Tr}_{cc}(q)$.

Unfortunately, neither the cc-trace preorder nor the induced equivalence are finitely axiomatizable. This is caused by the impossibility to traverse an arbitrarily long sequence of contravariant actions to prove a valid equality such as $a_1\bar{b}(a_1 + a_2) =_{ccT} a_1\bar{b}a_1 + a_1\bar{b}a_2$.

*****Failures

Classical (covariant) failure semantics is defined by means of refusals: $R \subseteq Act$ is refused by p, denoted by $R \in \mathrm{Ref}(p)$, iff $I(p) \cap R = \emptyset$. These are connected with the traces of processes to obtain the failures: $\langle t, R\rangle \in \mathrm{F}(p)$ iff there exists $p \stackrel{t}{\longrightarrow} p'$ with $I(p') \cap R = \emptyset$.

Duality provides again the natural definition of cc-refusal: for $R = R^r \cup R^l$ with $R^r \subseteq A^r$ and $R^l \subseteq A^l$, R is cc-refused by p iff $I(p) \cap R^r = \emptyset$ or $I(p) \cap \overline{R^l} \neq \emptyset$, denoted by $R \in \mathrm{Ref}_{cc}(p)$. Then cc-failures are obtained by taking $\langle t, R\rangle \in \mathrm{F}_{cc}(p)$ if and only if $t = c_1 \ldots c_k$ and $p \models Q_1 c_1 Q_2 c_2 \ldots Q_k c_k$ $(p \stackrel{t}{\longrightarrow} p' \Rightarrow (I(p') \cap R^r = \emptyset$ or $I(p') \cap \overline{R^l} \neq \emptyset))$, where $Q_i \equiv \exists$ iff $c_i \in A^r$ and $Q_i \equiv \forall$ iff $c_i \in A^l$. The meaning of an expression of the form

$$p \models Q_1 c_1 Q_2 c_2 \ldots Q_k c_k \ (p \stackrel{t}{\longrightarrow} p' \Rightarrow \varphi),$$

is the following:

- If $k = 1$ and $c_1 = a \in A^r$, we have: $p \models \exists a\ (p \xrightarrow{a} p' \Rightarrow \varphi)$ iff there exists $p \xrightarrow{a} p'$ and $p' \models \varphi$.
- If $k = 1$ and $c_1 = b \in A^l$, we have: $p \models \forall b\ (p \xrightarrow{b} p' \Rightarrow \varphi)$ iff $(p \xrightarrow{b} p' \Rightarrow p' \not\models \varphi)$.
- If $k > 1$, then $p \models Q_1 c_1 Q_2 c_2 \dots Q_k c_k (p \xrightarrow{t} p' \Rightarrow \varphi)$ iff $Q_1 p \xrightarrow{c_1} p''$ and $p'' \models Q_2 c_2 \dots Q_k c_k (p \xrightarrow{t'} p' \Rightarrow \varphi)$, where $t' = c_2 \dots c_k$.

It is interesting to observe that the intuitive interpretation of cc-failures is nice and simple: a refusal tells us that the process cannot accept any of the offered covariant actions in R^r, or is ready to execute some contravariant action that is not allowed by the offered contravariant actions in R^l. Both of these properties have a negative nature: it is bad to accumulate many failures. However, as in the ordinary case, we also have embedded the positive information that is provided by the traces of processes.

*****Readiness, Failure traces and Ready traces

As done in the definition of ready cc-simulation, ready sets should treat in a uniform way covariant and contravariant actions, and therefore they are easily defined from our cc-traces. For $R \in A$, we say that $\langle t, R\rangle \in \mathrm{Re}_{cc}(p)$ if and only if $t = c_1 \dots c_k$ and $p \models Q_1 c_1 Q_2 c_2 \dots Q_k c_k\ (p \xrightarrow{t} p' \Rightarrow I(p') = R)$.

Once we have our definitions for cc-failures semantics and cc-readiness semantics, those for cc-failure traces and cc-ready traces are easily obtained. For the case of cc-failures traces we have, taking $R_i = R_i^r \cup R_i^l$ with $R_i^r \subseteq A^r$ and $R_i^l \subseteq A^l$:

$\langle R_0 c_1 R_1 \dots c_k R_k\rangle \in \mathrm{FT}_{cc}(p)$ if and only if
$p \models R_0 Q_1 c_1 R_1 Q_2 c_2 R_2 \dots Q_k c_k\ (p \xrightarrow{c_1 \dots c_k} p' \Rightarrow (I(p') \cap R_k^r = \emptyset \text{ or } I(p') \cap \overline{R_k^l} \neq \emptyset))$,

where the general meaning of such an expression is defined in a similar way as for the case of failures (requiring $I(p'') \cap R_k^r = \emptyset$ or $I(p'') \cap \overline{R_k^l} \neq \emptyset$ when defining the meaning of any quantification $Q_i c_i R_i$ for each transition $p \xrightarrow{c_i} p''$).

References

1. L. Aceto, I. Fábregas, D. de Frutos-Escrig, A. Ingólfsdóttir, and M. Palomino. Graphical representation of covariant-contravariant modal formulae. In B. Luttik and F. Valencia, editors, *EXPRESS*, volume 64 of *EPTCS*, pages 1–15, 2011.
2. L. Aceto, I. Fábregas, D. de Frutos Escrig, A. Ingólfsdóttir, and M. Palomino. Relating modal refinements, covariant-contravariant simulations and partial bisimulations. In F. Arbab and M. Sirjani, editors, *Fundamentals of Software Engineering, FSEN 2011*, LNCS, pages 268–283. Springer, To appear.
3. I. Fábregas, D. de Frutos-Escrig, and M. Palomino. Equational characterization of covariant-contravariant simulation and conformance simulation semantics. In L. Aceto and P. Sobociński, editors, *Proceedings Seventh Workshop on Structural Operational Semantics, Paris, France, 30 August 2010*, volume 32 of *EPTCS*, pages 1–14, 2010.
4. N. Lynch. I/O automata: A model for discrete event systems. *In 22nd Annual Conference on Information Sciences and Systems*, pages 29–38, 1988.

Soft Constraints with Lexicographic Ordering*

Fabio Gadducci and Giacoma Valentina Monreale

Department of Informatics, University of Pisa, Italy

1 Introduction

Classical Constraint Satisfaction Problems (CSPs) search for the assignment of a set of variables that may satisfy a family of requirements. Constraint propagation (as e.g. represented by local consistency algorithms) embeds any reasoning which consists of explicitly forbidding values or their combinations for some variables of a problem because a given subset of its constraints cannot be satisfied otherwise.

The soft framework extends the classical constraint notion in order to model preferences: the aim is to provide a single environment where suitable properties (e.g. on constraint propagation) could be proven and inherited by all the instances. Technically, this is done by adding to the classical notion of CSP a representation of the levels of satisfiability of each constraint. Albeit appearing with alternative presentations in the literature, the additional component consists of a poset (stating the desiderability among levels) equipped with a binary operation (defining how two levels can be combined). Besides their flexibility, the use of these formalisms has been advocated for two main reasons. First of all, for their flexibility: their abstract presentation allows for recasting many concrete cases previously investigated in the literature; moreover, for their modularity: suitable operators can be defined, in order to manipulate such structures and build new ones.

Definition 1. *A* general valuation structure *(GVS) is 4-tuple* $G = \langle A, \leq, \otimes, \top \rangle$ *such that* $G^{\leq} = \langle A, \leq, \top \rangle$ *is a join semilattice (JSL) with top,* $G^{\otimes} = \langle A, \otimes, \top \rangle$ *is a commutative monoid, and distributivity* $a \otimes (b \vee c) = (a \otimes b) \vee (a \otimes c)$ *holds.*

For $G^{\leq}$ being a JSL means that there exists a join $a \vee b$ for every pair $a, b \in A$ (hence, for every finite, non-empty subset of A). From distributivity it follows that the tensor operator preserves the order, i.e., $a \leq b$ implies $a \otimes c \leq b \otimes c$.

Our GVSs are a generalization of valuation structures [5], replacing their total order with a JSL, as well as of c-semirings [2] (also known as absorptive semirings [1]), removing the requirement of a bottom element $\bot$ (which is by construction also an annihilator, i.e., such that $a \otimes \bot = \bot$). The lack of such a (necessarily unique) element is going to be be pivotal in our proposal for modelling constraints whose degree of satisfaction is based on lexicographic orders.

Indeed, it proved impossible to recast in the soft CSP fold the case of lexicographic orders, i.e., sets whose elements are pairs and the position plays a role. Assuming two partial orders $\leq_0$ and $\leq_1$, the associated lexicographic order is

* Partially supported by the EU FP7-ICT IP ASCEns and by the MIUR PRIN SisteR.

$$\langle a_0, a_1\rangle \leq_l \langle b_0, b_1\rangle \text{ if } \begin{cases} a_0 <_0 b_0 & \text{or} \\ a_0 =_0 b_0 \ \&\ a_1 \leq_1 b_1 \end{cases}$$

with $a < b$ meaning that $a \leq b$ and $a \neq b$. It is easy to see that $\leq_l$ is a partial order. Indeed, it is a JSL if both $\leq_0$ and $\leq_1$ are so, and the latter also is bounded, i.e., it has a bottom $\bot_1$. Explicitly, the definition is given below as

$$\langle a_0, a_1\rangle \vee_l \langle b_0, b_1\rangle = \begin{cases} \langle b_0, a_1 \vee_1 b_1\rangle & \text{if } a_0 =_0 b_0 \\ \langle b_0, b_1\rangle & \text{if } a_0 <_0 b_0 \text{ (and similarly for } b_0 <_0 a_0) \\ \langle a_0 \vee_0 b_0, \bot_1\rangle & \text{otherwise} \end{cases}$$

The condition on the existence of the bottom $\bot_1$ is not restrictive. Indeed, for any GVS G a new one is obtained by adding an annihilator: by construction it is also the bottom, yet it bears no change in the definition of the tensor operator.

Assuming two GVSs G_0 and G_1, a monoidal tensor $\otimes_l$ is also easily defined pointwise. However, the distributivity law usually fails. Such a failure motivated in [4] the introduction of a novel formalism, alternative to the standard soft constraint technology. In our work we follow a more traditional path, proving that under suitable conditions on G_0, also $Lex(G_0, G_1) = \langle A_0 \times A_1, \leq_l, \otimes_l, \langle \top_0, \top_1\rangle\rangle$ falls back to the standard soft CSP fold. Our main result is summed up below.

Definition 2. *Let G be a GVS. We say that $\otimes$ strictly preserves the order if $a < b$ implies $a \otimes c < b \otimes c$ for all elements $a, b, c \in A$.*

As usual, $a < b$ means that $a \leq b$ and $a \neq b$. Note that the formulation above forbids the presence in A of the annihilator (or, equivalently, of the bottom).

Proposition 1. *Let G_0 and G_1 be two GVSs. If $\leq_0$ is total and it is strictly preserved by $\otimes_0$, then also $Lex(G_0, G_1)$ is a GVS.*

Since $\leq_0$ is a total order, the third item of the characterization of the join operator $\vee_l$ is never verified, hence it is not necessary to require the bottom for G_1. The lack of the third item is the main reason for the theorem to hold.

Note that the conditon of the theorem implies that G_0 is what is called a stricly monotonic valuation structure [3]. An alternative, possibly simpler to verify characterization for such structures can be find below (see also [1]).

Lemma 1. *Let G be a GVS such that $\leq$ is total. Then, $\otimes$ strictly preserves the order iff it is cancellative, i.e., if $a \otimes c = b \otimes c$ implies $a = b$ for all $a, b, c \in A$.*

References

1. S. Bistarelli and F. Gadducci. Enhancing constraints manipulation in semiring-based formalisms. In G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors, *ECAI 2006*, volume 141 of *FAIA*, pages 63–67. IOS, 2006.

2. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of ACM*, 44(2):201–236, 1997.
3. M.C. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3):311–342, 2003.
4. M. Hoelzl, M. Meier, and M. Wirsing. Which soft constraints do you prefer. In G. Rosu, editor, *WRLA 2008*, volume 283(2) of *ENTCS*, pages 189–205. Elsevier, 2009.
5. T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *IJCAI 1995*, pages 631–637. Morgan Kaufman, 1995.

On Open Semantics for Reactive Systems*

Fabio Gadducci, Giacoma Valentina Monreale and Ugo Montanari

Department of Informatics, University of Pisa, Italy

Reactive systems (RSs) [5] represent an abstract formalism for specifying the dynamics of a computational model. The usual specification technique is based on a reduction semantics: a set representing the possible states of the system, plus an unlabeled relation among these states, denoting the possible evolutions of the system. Despite the advantage of conveying the semantics with relatively few rules, the main drawback of reduction semantics is that the dynamics of a system is described in a monolithic way, and so it can be interpreted only by inserting the system in appropriate contexts, where a reduction may take place. To make the analysis of systems simpler, it is often necessary to consider descriptions allowing the analysis of the behavior of each single subcomponent, thus increasing modularity and enhancing the opportunities for verification.

The framework of RSs thus offers techniques allowing to distill labelled transition systems (LTSs), hence, behavioral equivalences, for formalisms specified by a reduction system. It is centered on the concepts of *term*, *context* and *reduction rules*: contexts are arrows of a category, terms are arrows having as domain 0, an object denoting groundness, and reduction rules are pairs of (ground) terms.

Definition 1 (Reactive System). *A* reactive system $\mathbb{C}$ *consists of*

1. *a category* $\mathbf{C}$ *with a distinguished object* 0*;*
2. *a composition-reflecting subcategory* $\mathbf{D}$ *of* reactive contexts*;*
3. *a set of pairs* $\mathfrak{R} \subseteq \bigcup_{I \in |\mathbf{C}|} \mathbf{C}(0, I) \times \mathbf{C}(0, I)$ *of* reduction rules*.*

Intuitively, reactive contexts are those in which a reduction can occur. By composition-reflecting we mean that $d' \circ d \in \mathbf{D}$ implies $d, d' \in \mathbf{D}$. Note that the rules have to be ground, i.e., left-hand and right-hand sides have to be terms without holes and, moreover, with the same codomain.

The reduction relation is generated from the reduction rules by closing them under all reactive contexts. The key idea behind the LTS derivation is instead the following: a system p has a labelled transition $p \xrightarrow{c} p'$ if the system obtained by inserting p inside the (unary) minimal context c may reduce to p'. The notion of "minimal" context is expressed in terms of the categorical notion of relative pushout (RPO), which ensures that bisimilarity is a congruence when enough RPOs exist. Should all the possible contexts allowing a reduction be admitted, the resulting equivalence, denoted as saturated bisimilarity, would also result in a congruence. However, it is usually intractable, since it has to tackle a potentially infinite set of contexts. The problem has been addressed in [2], by introducing an "efficient" characterization (so-called semi-saturation) of these semantics.

* Partially supported by the EU FP7-ICT IP ASCEns and by the MIUR PRIN SisteR.

However, in some interesting cases it turns out that bisimilarity via minimal contexts is too fine-grained, while the saturated one is too coarse. As for process calculi, the standard way out of the empasse it is to consider barbs [6] (predicates on the states of a system) and barbed equivalences (adding the check of such predicates in the bisimulation game). The flexibility of the definition allows for recasting a variety of observational, bisimulation-based equivalences. So, in [1], a suitable notion of barbed saturated semantics for RSs is introduced.

Despite its applicability, the main limit suffered by RSs is the restriction to the use of ground rules for describing the dynamics of a system: this is often a strong requirement, in the modeling of open-ended systems operating in an ever changing environment. So, in [4], this problem is addressed, by developing a theory for open RSs considering also open terms and parametric rewriting rules. This means that, differently from Definition 1, no distinguished object is required and the domain of the left-hand and right-hand sides of a reduction rule can be any object besides 0. So, now the synthesized transitions are labelled not only with the minimal context but also with the most general instantiation allowing a reduction: $p \xrightarrow[x]{c} p'$ if p inserted into the context c and instantiated with the (possibly open) term x may evolve into a state p'. The notions of minimal context and most general instantiation are captured at once by the notion of *lux*. However, open RSs have suffered so far from two main drawbacks. First of all, no finitary presentation of the derived LTS has been found, possibly via a set of inference rules according to the SOS-style. Most importantly, the bisimilarity induced by the synthesized LTS for open terms results to be a congruence only under very restrictive conditions, hindering the applicability of the framework.

The solution to the first issue can be obtained by following the one for closed RSs [3], i.e., by illustrating how the synthesized LTS can be equipped with a SOS-like presentation via an encoding into tile systems. Instead, as far as the second issue, the solution we propose is to study a suitable notion of barbed saturated bisimilarity for open RSs (which by definition is guaranteed to be a congruence), proposing a characterization via the lux LTS and the semi-saturated game.

References

1. F. Bonchi, F. Gadducci, and G. V. Monreale. Reactive systems, barbed semantics, and the mobile ambients. In *FoSSaCS 2009*, volume 5504 of *LNCS*. Springer, 2009.
2. F. Bonchi, B. König, and U. Montanari. Saturated semantics for reactive systems. In *LICS 2006*, pages 69–80. IEEE Computer Society, 2006.
3. B. Bruni, F. Gadducci, U. Montanari, and P. Sobocinski. Deriving weak bisimulation congruences from reduction systems. In *CONCUR 2005*, volume 3653 of *LNCS*, pages 293–307. Springer, 2005.
4. B. Klin, V. Sassone, and P. Sobocinski. Labels from reductions: Towards a general theory. In *CALCO 2005*, volume 3629 of *LNCS*, pages 30–50. Springer, 2005.
5. J.J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In *CONCUR 2000*, volume 1877 of *LNCS*, pages 243–258. Springer, 2000.
6. R. Milner and D. Sangiorgi. Barbed bisimulation. In *ICALP 1992*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

From Interface Theories to Assembly Theories*
Extended Abstract

Rolf Hennicker[1] and Alexander Knapp[2]

[1] Ludwig-Maximilians-Universität München
hennicke@pst.ifi.lmu.de
[2] Universität Augsburg
knapp@informatik.uni-augsburg.de

Reactive software components are commonly understood as encapsulated units which communicate with their environment via well-defined interfaces. Interface specifications provide a means to describe the visible behaviour of interacting components. They serve, on the one hand, to express what is expected from the environment for a correct functioning of a component, and, on the other hand, to specify what is offered by a component. For the development of component systems on the basis of interfaces some rudimentary requirements should be respected which we summarise by the abstract concept of an interface theory (inspired by the notion of an interface language in [1]). An *interface theory* $\mathscr{F} = (\mathcal{F}, \multimap\!\circ, \otimes, \preccurlyeq, \rightarrow)$ comprises

- a class $\mathcal{F}$ of *interface specifications*,
- a symmetric *connectability relation* $\multimap\!\circ \subseteq \mathcal{F} \times \mathcal{F}$ to express when two interfaces are statically connectable,
- a partial, commutative and associative *composition operator* $\otimes : \mathcal{F} \times \mathcal{F} \rightharpoonup \mathcal{F}$ which is defined for connectable interfaces,
- a reflexive and transitive *refinement relation* $\preccurlyeq \subseteq \mathcal{F} \times \mathcal{F}$, and
- an *accession* relation $\rightarrow \subseteq \multimap\!\circ$ to express, by $F \rightarrow G$, that the communication assumptions of F are satisfied by G.

For all interfaces $F_1, F_2, G_1, G_2 \in \mathcal{F}$ the following properties are required:

1. *Compositional refinement:* If $G_1 \multimap\!\circ G_2$, $F_1 \preccurlyeq G_1$ and $F_2 \preccurlyeq G_2$, then $F_1 \multimap\!\circ F_2$ and $(F_1 \otimes F_2) \preccurlyeq (G_1 \otimes G_2)$.
2. *Preservation of accession by refinement:* If $G_1 \rightarrow G_2$ and $F_1 \preccurlyeq G_1$, $F_2 \preccurlyeq G_2$, then $F_1 \rightarrow F_2$.

The result of interface composition yields again an interface describing the visible (blackbox) behaviour of a composite component. What is still missing is the specification of architectural information and therefore also the possibility to observe communications between components. To tackle this issue we show how we can move from interface theories to assembly theories. An *assembly theory* $(\mathcal{A}, \mathit{intfs}, \sqsubseteq, \mathit{cs}, \mathit{pack})$ over an interface theory $\mathscr{F} = (\mathcal{F}, \multimap\!\circ, \otimes, \preccurlyeq, \rightarrow)$ is given by

* This work has been partially sponsored by the European Union under the FP7-project ASCENS, 257414. A full paper is actually submitted for journal publication.

- a class $\mathcal{A}$ of *assemblies*,
- an *interface selector* function $intfs : \mathcal{A} \to \wp_{\text{idx}}(\mathcal{F})$, providing the finitely indexed set $(F_i)_{1 \leq i \leq n}$ of interfaces over which an assembly is built,
- a reflexive and transitive *assembly refinement relation* $\sqsubseteq \subseteq \mathcal{A} \times \mathcal{A}$,
- a *communication"=safety predicate* $cs \subseteq \mathcal{A}$ to express when the interfaces of an assembly work properly together, and
- an *encapsulation operation* $pack : \mathcal{A} \to \mathcal{F}$ to encapsulate an assembly into an interface,

such that for all assemblies $A, B \in \mathcal{A}$ the following conditions are satisfied:

1. *Interface"=wise assembly refinement:* Let $intfs(A) = (F_i)_{1 \leq i \leq n}$ and $intfs(B) = (G_i)_{1 \leq i \leq n}$. If $cs(B)$ and $F_i \preccurlyeq G_i$ for $i = 1, \ldots, n$, then $A \sqsubseteq B$.
2. *Preservation of communication"=safety:* If $cs(B)$ and $A \sqsubseteq B$, then $cs(A)$.
3. *Refinement encapsulation:* If $cs(B)$ and $A \sqsubseteq B$, then $pack(A) \preccurlyeq pack(B)$.

In general, there is left some freedom to construct an assembly theory over a given interface theory. We can, however, define a canonical construction of an assembly theory over any arbitrary interface theory in the following way.

Theorem 1. *Let $\mathscr{F} = (\mathcal{F}, \multimap, \otimes, \preccurlyeq, \to)$ be an interface theory. Then $(\mathcal{A}, intfs, \sqsubseteq, cs, pack)$ with*

$$\begin{gathered}
\mathcal{A} = \{(F_i)_{1 \leq i \leq n} \in \wp_{\text{idx}}(\mathcal{F}) \mid (F_i)_{1 \leq i \leq n} \text{ connectable}\} \ , \\
intfs((F_i)_{1 \leq i \leq n}) = (F_i)_{1 \leq i \leq n} \ , \\
(F_i)_{1 \leq i \leq m} \sqsubseteq (G_i)_{1 \leq i \leq n} \iff m = n \wedge \forall 1 \leq i \leq n \,.\, F_i \preccurlyeq G_i \ , \\
cs((F_i)_{1 \leq i \leq n}) \iff \forall 1 \leq j \leq n \,.\, F_j \to \bigotimes (F_i)_{1 \leq i \neq j \leq n} \ , \\
pack(A) = \bigotimes (F_i)_{1 \leq i \leq n}
\end{gathered}$$

is an assembly theory over $\mathscr{F}$.

We can also show that certain properties, like incremental design, propagate from the interface level to the assembly level.

As a concrete instantiation we consider a modal interface theory such that interfaces are given by modal I/O-transition systems with distinguished may and must-transitions and with weak modal refinement [2] and we construct a modal assembly theory with a flexible communication"=safety predicate that ensures that any output of a component will eventually be taken by its environment.

References

1. Luca de Alfaro and Thomas A. Henzinger. Interface-based design. In Manfred Broy, Johannes Grünbauer, David Harel, and C. A. R. Hoare, editors, *Engineering Theories of Software-intensive Systems*, volume 195 of *NATO Science Series: Mathematics, Physics, and Chemistry*, pages 83–104. Springer, 2005.
2. Hans Hüttel and Kim Guldstrand Larsen. The use of static constructs in a modal process logic. In Albert R. Meyer and Michael A. Taitslin, editors, *Logic at Botik*, volume 363 of *Lect. Notes Comp. Sci.*, pages 163–180, 1989.

Streamlining Policy Creation in Policy Frameworks

Mark Hills[1]

Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

Abstract. *Policy frameworks* provide a technique for improving reuse in program analysis: the same language frontend, and a core analysis semantics, can be shared among multiple analysis policies for the same language, while analysis domains (such as units of measurement) can be shared among frameworks for different languages. One limitation of policy frameworks is that, in practice, adding a new policy can still require a significant level of knowledge about the internals of the semantics definition. This abstract describes work on extending policy frameworks to solve this limitation, making policies reflective over their requirements and generating the policy semantics from a higher-level policy description language.

Using executable language definitions, such as rewriting logic semantics [9] (RLS) or K [11] definitions running in Maude [4], program analysis can be treated as a form of non-standard program evaluation over appropriate domains of abstract values. An example of this approach is the unit safety analysis developed first for BC [3] (a small calculator language) and then for a small subset of C [10]. In this analysis, the abstract values were units of measurement [1] (e.g., meters, seconds, lumens). The analysis semantics modeled the operation of language constructs over these units, detecting errors in cases where units were used incorrectly, e.g., when two different units were added.

However, in this work, rules specific to the analysis were tangled with analysis-agnostic rules, making it challenging to reuse parts of the existing semantics in a new analysis. Solving this problem was the goal of the C Policy Framework [5], or CPF, an extensible analysis framework for C defined in Maude. The core of CPF includes an analysis-generic frontend, allowing annotations to be added (in comments) as function contracts or within function bodies, and an abstract C semantics. To define a specific analysis *policy*, this core is extended with an analysis-specific definition of abstract values, a specific annotation language, and equational definitions for a number of *hooks*, representing points in the semantics that differ between analysis policies. Later work extended this to the SILF language [7], which provides a simpler, more modular environment for experimenting with analysis policies. This work also extended the annotation mechanism to type-like annotations, a feature subsequently added to CPF.

Unfortunately, even though the frameworks for C and SILF were structured modularly [8, 2], they were sometimes not modular *in practice*, a point raised in

conversations with the author. To actually implement a new policy, the implementer requires a detailed knowledge of both Maude and of the entire provided core semantics. This includes knowing which hooks must be defined to provide the policy-specific semantics and which modules provide base variants of functionality that can be directly reused or extended. In this abstract we describe ongoing work on two features being added to the SILF policy framework (and later to CPF) to help solve this problem: the reflective extraction of extension point information, and a DSL for describing analysis policies.

Reflective Extraction of Extension Point Information Each hook operation defined in the core framework semantics is identified using a Maude `metadata` attribute. Additional operations are defined equationally to identify modules that can be used directly or extended to provide specific features, e.g. a base annotation language. Using a combination of standard rewriting and Maude's reflective capabilities, the policy description language, described below, can then "ask" a framework about defined extension points.

A Policy Description Language Using the extracted policy information, a policy description language is used to describe the three standard parts of an analysis policy. First, the domain of policy values is defined algebraically, with pretty-printing rules defined to display policy values appropriately in messages. Second, the analysis-specific behavior for each hook is defined, making use of the policy values and of predefined reporting operations for errors and warnings (potentially with source locations [6]). Third, the annotation language used for the policy is defined by extending a provided base annotation language with policy-specific annotations, e.g., `@unit(E)` to calculate the unit of an expression in a units policy. These three items are then used to generate a parser for the annotation language, used in conjunction with the language parser, and a Maude specification of the value domain and the analysis semantics. This generation process also creates many of the repetitive cases needed to properly handle the propagation of error information from children to parents (ensuring errors in child expressions do not trigger new, spurious errors in parents), something done manually now.

References

1. NIST Website, International System of Units (SI). http://physics.nist.gov/cuu/Reference/unitconversions.html.
2. C. Braga and J. Meseguer. Modular Rewriting Semantics in Practice. In *Proceedings of WRLA'04*, volume 117 of *ENTCS*, pages 393–416. Elsevier, 2005.
3. F. Chen, G. Roşu, and R. P. Venkatesan. Rule-Based Analysis of Dimensional Safety. In *Proceedings of RTA'03*, volume 2706 of *LNCS*, pages 197–207. Springer, 2003.
4. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *LNCS*. Springer, 2007.

5. M. Hills, F. Chen, and G. Roşu. A Rewriting Logic Approach to Static Checking of Units of Measurement in C. In *Proceedings of RULE'08*. Elsevier, 2008. To Appear.
6. M. Hills, P. Klint, and J. Vinju. RLSRunner: Linking Rascal with K for Program Analysis. In *Proceedings of SLE'11*, LNCS. Springer-Verlag, 2011. To Appear.
7. M. Hills and G. Roşu. A Rewriting Logic Semantics Approach To Modular Program Analysis. In *Proceedings of RTA'10*, volume 6 of *Leibniz International Proceedings in Informatics*, pages 151 – 160. Schloss Dagstuhl - Leibniz Center of Informatics, 2010.
8. J. Meseguer and C. Braga. Modular Rewriting Semantics of Programming Languages. In *Proceedings of AMAST'04*, volume 3116 of *LNCS*, pages 364–378. Springer, 2004.
9. J. Meseguer and G. Rosu. The rewriting logic semantics project. *Theoretical Computer Science*, 373(3):213–237, 2007.
10. G. Roşu and F. Chen. Certifying Measurement Unit Safety Policy. In *Proceedings of ASE'03*, pages 304 – 309. IEEE, 2003.
11. G. Roşu and T. F. Şerbănuţă. An Overview of the K Semantic Framework. *Journal of Logic and Algebraic Programming*, 79(6):397–434, 2010.

Representing Categories of Theories in a Proof-Theoretical Logical Framework

Fulya Horozal and Florian Rabe

Computer Science, Jacobs University Bremen, Germany

Logical frameworks have been introduced as an abstract formalism for representing logics and studying their properties. We are interested in how these frameworks support *(i)* the representation of the categories of signatures or theories usually associated with logics and related declarative languages and *(ii)* the reasoning about and the formulation of algorithms on signatures and theories such as type-checking or functor application.

Model theoretical logical frameworks, such as institutions [3], work with fully abstract categories. This has the advantage that the framework is independent of the particular logic; thus theory-related concepts such as model classes can be formulated in full generality. However, this makes it difficult to provide generic tool support for reasoning and algorithms about individual theories.

Proof theoretical logical frameworks, such as LF [4], on the other hand, work with concrete theories and represent each individual theory of a logic. They provide generic tool support for individual theories. However, they do not have an abstract concept to represent the category of theories as a whole, and therefore, they cannot reason about or operate on arbitrary theories.

The Logic Atlas and Integrator (LATIN) project [1] developed a foundationally unconstrained logical framework [2], which unifies aspects of both model theoretical and proof theoretical frameworks. In LATIN, logics are represented as diagrams of theories, and translations between logics as theory morphisms between these diagrams. Below, we focus on the LF instance of the LATIN framework, but our work can be generalized to other frameworks (e.g., Isabelle [5]).

In LATIN-LF, the syntax of a logic is represented as an LF signature L and theories of the logic as inclusion morphisms $L \hookrightarrow \Sigma$, where Σ adds the declarations of the non-logical symbols to L. Thus, the category of theories of L is a subcategory of the coslice category $LF\backslash L$ containing only inclusions.

However, this category usually contains a lot more objects than needed. For instance, if $L = FOL$ represents first-order logic, then Σ should only contain declarations of function or predicate symbols. More precisely, if *term* and *form* are the types of first-order terms and formulas, then Σ should contain only declarations of the form $f : \mathit{term} \to \ldots \to \mathit{term} \to \mathit{term}$ (n-ary function symbols) and $p : \mathit{term} \to \ldots \to \mathit{term} \to \mathit{form}$ (n-ary predicate symbols).

In LATIN-LF, functors between categories of theories are represented as LF signature morphisms. More specifically, a functor from the category of theories of L to that of L' is obtained by an LF signature morphism $l : L \to L'$, and functor application is obtained by pushout along l.

However, this is not expressive enough to cover all interesting functors. For example, it cannot represent the translation from FOL to ZFC, where the first-

order declaration $f : term \to term$ is translated to a set F together with an axiom that F is a unary ZFC function on the universe.

In this work, we introduce LFP, a major extension of the LF type theory with what we call *declaration patterns*. LFP permits elegant and concise representation of categories of theories in LF. Our extension is based on the central observation that theories of virtually all logics L consist of a list of declarations each of which must conform to one of a few patterns that are fixed by L. By adding a new primitive to LF that defines such patterns, we are able to represent these categories.

More specifically, declaration patterns use parameterized signatures with a simple type theory built on top. Interestingly, we have to introduce one additional primitive concept in LFP that is technically independent but practically necessary for declaration patterns: We usually need sequences of LF objects to define the declaration patterns of a logic L even if the formulas of L do not use sequences. For example, in FOL, we would introduce the declaration patterns $func = \lambda n : Nat.\, \{f : term^n \to term\}$ and $pred = \lambda n : Nat.\, \{p : term^n \to form\}$ for n-ary function and predicate symbols respectively, where Nat is the type of natural numbers in LFP and $term^n$ denotes the sequence $term, \ldots, term$ of length n.

Declaration patterns also prove crucial for logic translations because they permit concise representations of functors between theory categories. We introduce a notion of pattern-based functors between categories of theories. These are defined by induction on the list of declarations in a theory using one case for every declaration pattern. Moreover, functors that arise from pushouts can be recovered as a special case.

LFP is being implemented as part of the MMT system [6]. This will permit matching arbitrary LF signatures against the declaration patterns defined in an LFP signature, and translating matching signatures along the functors described in LFP.

References

1. M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, and F. Rabe. Project Abstract: Logic Atlas and Integrator (LATIN). In J. Davenport, W. Farmer, F. Rabe, and J. Urban, editors, *Intelligent Computer Mathematics*, volume 6824 of *Lecture Notes in Computer Science*, pages 287–289. Springer, 2011.
2. M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, F. Rabe, and K. Sojakova. Towards Logical Frameworks in the Heterogeneous Tool Set Hets. In T. Mossakowski and H. Kreowski, editors, *Recent Trends in Algebraic Development Techniques 2010*, volume 7137 of *Lecture Notes in Computer Science*, pages 139–159. Springer, 2012.
3. J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.
4. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.
5. L. Paulson. *Isabelle: A Generic Theorem Prover*. Springer, 1994.
6. F. Rabe and M. Kohlhase. A Scalable Module System. see http://arxiv.org/abs/1105.0548, 2011.

Designing DSLs – A Craftsman's Approach for the Railway Domain using CASL

Phillip James[1], Alexander Knapp[2], Till Mossakowski[3], and Markus Roggenbach[1]

[1] Swansea University, UK
[2] Augsburg University, Germany
[3] DFKI GmbH Bremen, Germany

In this work, we present ongoing research concerning the formalization of Domain Specific Languages (DSLs) [12, 6] within CASL. This work forms part of a general programme to make systematic use of Domain Specific Languages to ease system verification in the railway domain [9] using algebraic specification.

It is common industrial practice to describe domain specific vocabulary using DSLs, for example Invensys Rail have developed a DSL for the railway domain [8]. Often, such DSLs aim to capture domain concepts and relations between these concepts. Typically, such endeavours use a mixture of UML class diagrams (as "formal elements") with natural language descriptions complementing these diagrams. We take such DSLs based on UML class diagrams and natural language as a starting point for developing a formal DSL for system verification in CASL [13].

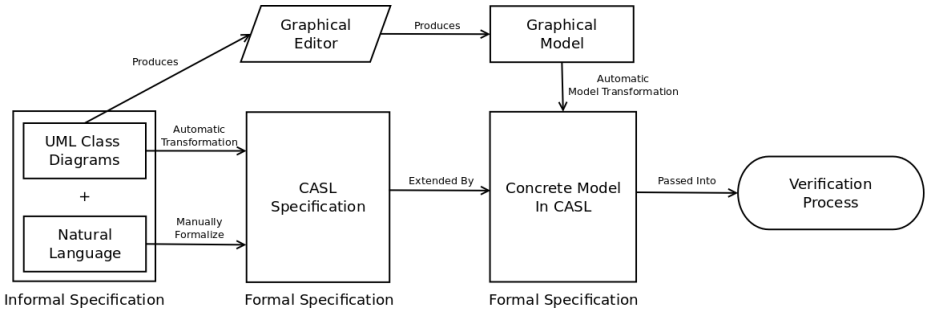


Fig. 1. Our Approach for developing DSLs based using CASL.

Concretely, we suggest a two step procedure – see Figure 1: Firstly, UML class diagrams from the domain description are automatically translated into a CASL specifications. Secondly, the complementary natural language descriptions are manually formalized in CASL. Such a semi-automatic procedure has the advantage that the domain concepts and the static relations between them can be captured by the domain expert in the graphical format offered by UML class diagrams. Also, the DSL design methodology outlined by James and Roggenbach [9] shows that once captured in CASL, such a DSL can be used as a backbone for an

automatic verification framework encapsulating formal methods. That is, EMF/GMF [5] can be used to generate a graphical editor for DSL models. These graphical models can then be translated via the Epsilon [10] framework to CASL specifications. To verify such models in CASL further logical concepts are often required in the language. Here, the CASL structuring constructs support definitional extensions of the language, which – if used in a systematic way – can provide fully automatic verification in the domain under consideration [9].

To support this approach, we develop:

- a new institution for UML class diagrams extending the institution given by Cengarle and Knapp [4].
- a new comorphism from this UML institution to CASL, extending a comorphism from OWL to CASL outlined by Luttich et al. in [11].

To illustrate the above translation is suitable, we consider and translate a number of example DSLs [8, 2] for the railway domain. Finally, we discuss the benefits of the approach.

With respect to domain specific languages and verification, a similar approach has been taken by Andova et al. [1], who use ASF+SDF for prototyping a DSL and then verify the produced transition system. Also, much work also been completed by Dines Bjørner into creating DSLs, in an academic setting, for transport networks [2, 3] using both natural language and the RSL specification language. To show the usefulness of our approach, we not only translate industrial examples of DSLs, but also the DSL for the railway domain as developed by Bjørner [2].

Finally, our translation from UML class diagrams to CASL differs from the approach proposed by Hussmann et al. [7]. Firstly, we concentrate on translating an independently given semantics for UML whereas Hussmann et al. use CASL in order to give a semantics to UML class diagrams. Secondly, we concentrate our translation around static aspects only, whereas Hussmann et al. provide special CASL sorts to deal with any dynamic aspects in order to cater for integration with dynamic UML diagrams [14].

References

1. S. Andova, M. van den Brand, and L. Engelen. Prototyping the semantics of a dsl using asf+sdf: Link to formal verification of dsl models. In Francisco Durán and Vlad Rusu, editors, Proceedings Second International Workshop on *Algebraic Methods in Model-based Software Engineering,* Zurich, Switzerland, 30th June 2011, volume 56 of *Electronic Proceedings in Theoretical Computer Science*, pages 65–79. Open Publishing Association, 2011.
2. D Bjørner. Formal Software Techniques for Railway Systems. *CTS2000: 9th IFAC Symposium on Control in Transportation Systems*, pages 1–12, 2000.
3. D Bjørner. *DOMAIN ENGINEERING Technology Management, Research and Engineering.* Japan Advanced Institute of Science and Technology, 2009.
4. Mara Victoria Cengarle and Alexander Knapp. An institution for UML 2.0 static structures. Technical Report TUM-I0807, Technische Universitt München, 2008.

5. Eclipse Consortium. Eclipse Graphical Modeling Framework (GMF), 2005. Available at http://www.eclipse.org/gmf.
6. M. Fowler. *Domain Specific Languages*. Addison-Wesley, 2010.
7. H. Hussmann, M. Cerioli, and H. Baumeister. From uml to casl (static part). Technical Report DISI-TR-00-06, DISI-Universit di Genova, 2000.
8. Invensys Rail. Data Model – Version 1A, 2010.
9. P James and M Roggenbach. Designing domain specific languages for verification: First steps. In Peter Hofner, Annabelle McIver, and Georg Struth, editors, *ATE-2011 – Proceedings of the First Workshop on Automated Theory Engineering*, volume 760 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.
10. D.S. Kolovos, L.M. Rose, and R.F. Paige. *The Epsilon Book*. 2010. Available at http://www.eclipse.org/epsilon.
11. Klaus Lüttich, Till Mossakowski, and Bernd Krieg-Brückner. Ontologies for the semantic web in casl. In José Fiadeiro, Peter Mosses, and Fernando Orejas, editors, *Recent Trends in Algebraic Development Techniques*, volume 3423 of *Lecture Notes in Computer Science*, pages 106–125. Springer Berlin / Heidelberg, 2005.
12. M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Computing Survey*, 37(4), 2005.
13. Peter D. Mosses. *CASL Reference Manual*. Springer, 2004.
14. Gianna Reggio, Maura Cerioli, and Egidio Astesiano. Towards a rigorous semantics of uml supporting its multiview approach. In Heinrich Hussmann, editor, *Fundamental Approaches to Software Engineering*, volume 2029 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2001.

Satisfiability calculi: the semantic counterpart of proof calculi in general logics

Carlos G. Lopez Pombo[1,3], Pablo F. Castro[2,3], Nazareno M. Aguirre[2,3], and Tomas S.E. Maibaum[4]

[1] Department of Computing, FCEyN, Universidad de Buenos Aires
[2] Department de Computing, FCEFQyN, Universidad Nacional de R Cuarto
[3] Consejo Nacional de Investigaciones Científicas y Tecnológicas (CONICET)
[4] Department of Computing and Software, McMaster University
clpombo@dc.uba.ar {pcastro,naguirre}@dc.exa.unrc.edu.ar tom@maibaum.org

The theory of institutions, presented by Goguen and Burstall in [1], provides a formal and generic definition of what a logical system is from a model theoretical point of view. Institutions were proposed as a general version of abstract model theory [2]. As was shown in [3, 4], they offer a suitable formal framework for addressing heterogeneity in specifications, and have been used to provide formal semantics to UML [5] and other languages related to computer science and software engineering. In [6], Meseguer extended the framework of institutions by providing a proof theoric point of view. To do so, he took the notion of entailment system and described a categorical construction capable of capturing the notion of proof. In Meseguer's words:

> A reasonable objection to the above definition of logic[5] is that *it abstracts away* the structure of proofs, since we know only that a set Γ of sentences entails another sentence φ, but no information is given about the internal structure of such a $\Gamma \vdash \varphi$ entailment. This observation, while entirely correct, may be a virtue rather than a defect, because the entailment relation is precisely what remains *invariant* under many equivalent proof calculi that can be used for a logic.

The previous remarks indicate that what is hidden behind the notion of proof calculus is essentially the implementation of the entailment relation of a logic. Before Meseguer's work, there was an imbalance in the definition of a logic (in the scope of institution theory) by not taking into account its deductive aspects. Meseguer concentrates exclusively on the proof theoretical aspects of a logic, providing not only the definition of entailment system, but also complementing it with the notion of proof calculus in order to obtain what he calls a logical system. We believe that this has tilted the imbalance in favor of the syntactic aspect, ignoring the fact that the same lack of an operational view that can be observed in the definition of entailment system now appears with respect to the notion of satisfiability (i.e., the satisfaction relation of an institution). This observation was motivated by the fact that several mathematical frameworks

[5] **Authors note**: He refers to the definition of logic as a structure that is constituted by an entailment system plus an institution (see [6, Def. 6]).

in computer science rely on model construction, either for proving properties, e.g., with model-checkers, or for finding counterexamples, as with tableaux techniques. These techniques constitute an important stream of research in logic; in particular, these methods play an important role in automatic software validation and verification. Our goal is to provide an abstract characterization of this class of semantics based tools for logical systems. This is accomplished by introducing a categorical characterization of the notion of satisfiability calculus which embraces logical tools such as tableaux, resolution, sequent calculi, etc. Thus, it can be thought of as a formalization of a semantic counterpart of Meseguer's proof calculus. The origins of these logical tools can be traced back to the works of Beth [7], Herbrand [8] and Gentzen [9]; Beth's ideas were used by Smullyan to formulate the tableau method for first-order predicate logic [10]. Herbrand's and Gentzen's work inspired the formulation of resolution systems as presented by Robinson [11]. These methods are strongly related to the semantics of a logic; and, therefore, they are often used for obtaining or finding models; notice that this is not possible in *pure* deductive methods, such as natural deduction or Hilbert systems, as formalized by Meseguer.

Definition: ***[Satisfiability calculus]*** A *satisfiability calculus* is a structure of the form $\langle \mathsf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \{\models^{\Sigma}\}_{\Sigma \in |\mathsf{Sign}|}, \mathbf{M}, \mathbf{Mods}, \mu \rangle$ satisfying the following conditions: – $\langle \mathsf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \{\models^{\Sigma}\}_{\Sigma \in |\mathsf{Sign}|} \rangle$ is an institution, – $\mathbf{M} : \mathsf{Th}_0 \to \mathsf{Struct}_{SC}$ is a functor. Let $T \in |\mathsf{Th}_0|$, then $\mathbf{M}(T) \in |\mathsf{Struct}_{SC}|$ is the canonical model structure of T, – $\mathbf{Mods} : \mathsf{Struct}_{SC} \to \mathsf{Cat}$ is a functor. Let $T \in |\mathsf{Th}_0|$, then $\mathbf{Mods}(\mathbf{M}(T))$ is the set of canonical models of T; the composite functor $\mathbf{Mods} \circ \mathbf{M} : \mathsf{Th}_0 \to \mathsf{Cat}$ will be denoted by $\mathbf{models}$, and – $\mu : \mathbf{models}^{\mathsf{op}} \dot{\to} \mathbf{Mod}$ is a natural transformation such that for each $T = \langle \Sigma, \Gamma \rangle \in |\mathsf{Th}_0|$ the image of $\mu_T : \mathbf{models}^{\mathsf{op}}(T) \to \mathbf{Mod}(T)$ is the category of models $\mathbf{Mod}(T)$. The map μ_T is called the *projection of the category of models* of the theory T. [6]

The intuition behind this definition is that functor $\mathbf{M}$ assigns, to every theory, a model structure from the category Struct_{SC}. The functor $\mathbf{Mods}$ projects the ones that represent canonical models. Finally, for any theory T, the functor μ_T relates each of these sets of conditions to the corresponding canonical model.

Tableau method for first-order predicate logic Consider the tableau method for first-order logic defined in [10]. Let $s_0 \xrightarrow{\tau_0} s_1 \xrightarrow{\tau_1} s_2 \xrightarrow{\tau_2} \dots$ be a branch in a tableau. Every s_i with $i < \omega$, is a set of formulae. In each step, the application of a rule decomposes one formula in this set into its constituent parts, preserving satisfiability. Thus, the limit set of the branch is a set of formulae containing all the constituent parts of the original set of formulae for which the tableau was built. Following this, every open branch expresses, by means of a set of formulae, the class of models satisfying it. Now, in order to define the tableau method as a satisfiability calculus, we must provide formal definitions for $\mathbf{M}$, $\mathbf{Mods}$ and μ. Struct_{SC} must be defined as the category of legal tableaux. In

[6] As usual, Th^I denotes the category theories of institution I and Th_0^I its complete subcategory induced by the axiom preserving morphisms.

this context, a tableau will be a relation between two sets of formulae; the set of formulae for which the models are constructed, and the set of formulae from which these models are constructed. Then, given $\Sigma \in |\mathsf{Sign}|$ and $\Gamma \subseteq \mathbf{Sen}(\Sigma)$, $Str^{\Sigma,\Gamma}$ is the category of tableaux assuming Γ. In $Str^{\Sigma,\Gamma}$ objects are sets of formulae over signature Σ, and morphisms represent tableaux. Struct_{SC} is then defined to be the category in which objects are all possible structures $Str^{\Sigma,\Gamma}$, and morphisms are the homomorphic extensions of the morphisms in $||\mathsf{Th}_0||$ to the structure of the tableaux presented above. The functor $\mathbf{M}$ must be understood as the relation between a theory in $|\mathsf{Th}_0|$ and its category of legal structures representing tableaux, so to every theory T, $\mathbf{M}$ associates the strict monoidal category $\langle Str^{\Sigma,\Gamma}, \cup, \emptyset \rangle$, and for every theory morphism $\sigma : \langle \Sigma, \Gamma \rangle \to \langle \Sigma', \Gamma' \rangle$, $\mathbf{M}$ associates a morphism $\widehat{\sigma} : Str^{\Sigma,\Gamma} \to Str^{\Sigma',\Gamma'}$ which is the homomorphic extension of σ to the structure of the tableaux. The functor **Mods** provide the means for obtaining the category containing the closure of those structures in $Str^{\Sigma,\Gamma}$ that represent the closure of the branches in saturated tableaux and, finally, the natural transformation μ relates the structures representing saturated tableaux with the canonical model satisfying the set of formulae denoted by the source of the corresponding morphism.

References

1. Goguen, J.A., Burstall, R.M.: Introducing institutions. In Clarke, E.M., Kozen, D., eds.: Procs. of the Carnegie Mellon Workshop on Logic of Programs. Volume 184 of LNCS, Springer-Verlag (1984) 221–256
2. Diaconescu, R., ed.: Institution-independent Model Theory. Volume 2 of Studies in Universal Logic. Birkhäuser (2008)
3. Mossakowski, T., Maeder, C., Luttich, K.: The heterogeneous tool set, Hets. In Grumberg, O., Huth, M., eds.: Procs. of the 13th. Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2007). Volume 4424 of LNCS, Springer-Verlag (April 2007) 519–522
4. Tarlecki, A.: Towards heterogeneous specifications. In Gabbay, D., de Rijke, M., eds.: Frontiers of Combining Systems. Volume 2 of Studies in Logic and Computation. Research Studies Press (2000) 337–360
5. Cengarle, M.V., Knapp, A., Tarlecki, A., Wirsing, M.: A heterogeneous approach to UML semantics. In Degano, P., DeNicola, R., Meseguer, J., eds.: Procs. of Concurrency, graphs and models (Essays dedicated to Ugo Montanari on the occasion of his 65th. birthday). Number 5065 in LNCS, Springer-Verlag (2008), 383–402
6. Meseguer, J.: General logics. In Ebbinghaus, H.D., Fernandez-Prida, J., Garrido, M., Lascar, D., Artalejo, M.R., eds.: Procs. of the Logic Colloquium '87. Volume 129, North Holland (1989) 275–329
7. Beth, E.W.: The Foundations of Mathematics. North Holland (1959)
8. Herbrand, J.: Investigation in proof theory. In Goldfarb, W.D., ed.: Logical Writings. Harvard University Press (1969) 44–202.
9. Gentzen, G.: Investigation into logical deduction. In Szabo, M.E., ed.: The Collected Papers of Gerhard Gentzen. North Holland (1969) 68–131.
10. Smullyan, R.M.: First-order Logic. Dover Publishing (1995)
11. Robinson, J.A.: A machine-oriented logic based on the resolution principle. Journal of the ACM **12**(1) (1965) 23–41

Categorical Characterization of Structure Building Operations

Carlos G. Lopez Pombo[1,2] and Marcelo F. Frias[3,2]

[1] Department of Computing, School of Science, Universidad de Buenos Aires
[2] Consejo Nacional de Investigaciones Científicas y Tecnológicas (CONICET)
[3] Department of Software Engineering, Buenos Aires Institute of Technology
clpombo@dc.uba.ar mfrias@itba.edu.ar

Modularity is a key concept when aiming at good designs. In [1, 2], Parnas extensively discusses how modular designs of software artifacts result in higher quality software by enabling reuse, separation of concerns and better maintainability. Since Parnas's foundational work, practitioners build software artifacts (and particularly software specifications), modularly.

In [3], Goguen and Burstall present a categorical formalization of the abstract model theory of a logic using the formalism of *institutions*. Institutions provide an abstract view of a logic that enables the study of properties of a formalism independently of notational issues. For instance, [4] surveys interesting results about well-known properties such as interpolation, within the framework of institutions. In [5], Sanella and Tarlecki provide a set of structure-building operations that enable the modular construction of specifications from theories taken from a given institution. In [6], Borzyszkowski presented a logical system for the structure building operations (SBOs) introduced by Sanella and Tarlecki, as well as an extensive discussion on the conditions under which the proposed calculus is complete. Intuitively we would like to consider structured specifications as syntactically sweetened ways of describing a system whose behavior is equivalent to a monolithic non-structured specification. This is not the case in the SBOs presented in [5]. Let us recall their definition: *1*) a flat specification is assumed to be a legal structured specification, *2*) the union of two structured specifications over the same signature is considered as a structured specification, *3*) the translation over a morphism between signatures is a structured specification, and *4*) the derivation (understood as the hiding of symbols) along a signature morphism is a structured specification. In Borzyszkowski's work two operations **Sig** and **Mod** are defined for the retrieval of the signature and the class of models of a structured specification, respectively. Both operations, defining the semantics of the structuring mechanism, are defined inductively on the structure of the specification. Therefore, in order to preserve the idea of structure as denotation, a question arises:

> *Is it possible to define a function assigning a flat specification to every structured one?*

While the problem is easily solved (positively) for specifications built from unions and translations, the class of models resulting from hiding symbols is explained as the reducts of the models of a set of axioms described in terms of symbols

that no longer belong to the language. Thus, the language is many times unable to syntactically describe the conditions needed to impose the required structure on models.

Assume the institution under observation is a first-order language with conjunction, and that derivations only take place on constant symbols. Thus, a derived structured specification can be described by means of a flat specification in which those axioms stating properties of a constant symbol c are replaced by a single one where a variable x_c is existentially quantified, as follows:

$$\{Ax_1(c), \ldots, Ax_k(c)\} \quad \mapsto \quad \exists x_c(Ax_1(x_c) \wedge \cdots \wedge Ax_k(x_c)) \ . \tag{1}$$

This solution can be formalized by defining a family of endo institution representations [7] indexed by an appropriate subset of signature morphisms.

Hiding symbols is the formal resource for stating that a certain symbol (usually a function or predicate) is to be considered private to a software module. Thus, for a technique as the one hinted in (1) to be useful, we should be able to quantify over function symbols. Higher-order logic gives us a solution, but at the expense of missing an effective, sound and complete calculus.

In this work, given an institution I, we solve the problem of properly defining the semantics for structured specifications in terms of flat ones, by representing the underlying institution in a more expressive one (namely, U) in which first-order objects provide semantics for functions in I.

First-order relational logics provide a solution. In these institutions, first-order citizens are relations. An institution representation targeting such relational institution interprets functions and predicates as algebraic terms, thus associating to them a relational semantics. This is done by defining the signature functor in a way that function and predicate extra logical symbols in the source logical language are mapped to constant extra logical symbols in the target relational algebraic language. In this way, functions become first-order citizens of the relational institution, and the transformation presented in Eq. 1 can thus be applied.

In [6], Borzyszkowski also showed that whenever an institution representation $\rho = \langle \rho^{Sign}, \rho^{Sen}, \rho^{Mod} \rangle$ satisfies the weak-amalgamation property, ρ-expansion of models and ρ^{Sign} preserves pushouts, then it can be extended to $\widehat{\rho}$, a mapping from structured specifications over the logical language to structured specifications over the algebraic language.

Let $I = \langle \mathsf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \{\models_\Sigma\}_{\Sigma \in |\mathsf{Sign}|} \rangle$ be the institution of a first-order logic, and $R = \langle \mathsf{Sign}^R, \mathbf{Sen}^R, \mathbf{Mod}^R, \{\models^R_\Sigma\}_{\Sigma \in |\mathsf{Sign}^R|} \rangle$ the institution of an algebraic relational language such that $\rho = \langle \rho^{Sign}, \rho^{Sen}, \rho^{Mod} \rangle : I \to R$ is an institution representation satisfying the weak-amalgamation property, ρ-expansion of models and such that ρ^{Sign} preserves pushouts. Let $||\mathsf{Sign}^R||\,|_C$ be the restriction of $||\mathsf{Sign}^R||$ to those morphisms that only act over constant symbols, while preserving the rest of the signature. Lemma 1 then follows.

Lemma 1. *For all* $\sigma \in ||\mathsf{Sign}||$, $\rho^{Sign}(\sigma) \in ||\mathsf{Sign}^R||\,|_C$.

Let $\{T^R_\sigma\}_{\sigma \in ||\mathsf{Sign}^R|||_C}$ the family of endo institution representations performing the translation presented in Eq. (1).

Definition 1. *The family of functions* $\{T_\sigma\}_{\sigma \in ||\mathsf{Sign}||}$ *is defined as* $T_\sigma(SP) = T^R_{\rho^{Sign}(\sigma)}(\widehat{\rho}(SP))$.

Theorem 1. $\mathbf{Mod}[SP] = \rho^{Mod}(\mathbf{Mod}(T_\sigma(SP)))$

References

1. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Communications of the ACM **15**(12) (1972) 1053–1058.
2. Parnas, D.L.: Designing software for ease of extension and contraction. IEEE Transactions on Software Engineering **5**(2) (1979) 128–138.
3. Goguen, J.A., Burstall, R.M.: Introducing institutions. In Clarke, E.M., Kozen, D., eds.: Proceedings of the Carnegie Mellon Workshop on Logic of Programs. Volume 184 of Lecture Notes in Computer Science., Springer-Verlag (1984) 221–256
4. Tarlecki, A.: Bits and pieces of the theory of institutions. In Pitt, D.H., Abramsky, S., Poigné, A., Rydeheard, D.E., eds.: Proceedings of the Category Theory and Computer Programming, tutorial and workshop. Volume 240 of Lecture Notes in Computer Science., Springer-Verlag (1986) 334–363
5. Sannella, D., Tarlecki, A.: Specifications in an arbitrary institution. Information and computation **76**(2–3) (1988) 165–210
6. Borzyszkowski, T.: Logical systems for structured specifications. Theoretical Computer Science **286** (2002) 197–245
7. Tarlecki, A.: Moving between logical systems. In Haveraaen, M., Owe, O., Dahl, O.J., eds.: Selected papers from the 11th Workshop on Specification of Abstract Data Types Joint with the 8th COMPASS Workshop on Recent Trends in Data Type Specification. Volume 1130 of Lecture Notes in Computer Science., Springer-Verlag (1996) 478–502
8. Tarski, A.: On the calculus of relations. Journal of Symbolic Logic **6**(3) (September 1941) 73–89
9. Haeberer, A.M., Veloso, P.A.: Partial relations for program derivation: adequacy, inevitability and expressiveness. In: Proceedings of IFIP TC2 working conference on constructing programs from specifications, IFIP TC2: Software: Theory and Practice, North Holland (1991) 310–352
10. Frias, M.F.: Fork algebras in algebra, logic and computer science. Volume 2 of Advances in logic. World Scientific Publishing Co., Singapore (2002)
11. Frias, M.F., Baum, G.A., Maibaum, T.S.E.: Interpretability of first-order dynamic logic in a relational calculus. In de Swart, H., ed.: Proceedings of the 6th. Conference on Relational Methods in Computer Science (RelMiCS) - TARSKI. Volume 2561 of Lecture Notes in Computer Science., Oisterwijk, The Netherlands, Springer-Verlag (October 2002) 66–80
12. Frias, M.F., Lopez Pombo, C.G.: Interpretability of first-order linear temporal logics in fork algebras. Journal of Logic and Algebraic Programming **66**(2) (2006) 161–184

Execution modes as local states — towards a formal semantics for reconfigurable systems*

Alexandre Madeira[1,2,3], Manuel A. Martins[2], Luís S. Barbosa[1]

[1] HASLab - INESC TEC and DI, Universidade do Minho, Portugal
[2] CIDMA - Department of Mathematics, Universidade de Aveiro, Portugal
[3] Critical Software S.A., Portugal

This talks aims at contributing to the on-going quest for formal logics and semantics to specify *reconfigurable systems*. The qualifier *reconfigurable* refers to systems whose execution modes, and not only the values stored in their internal memory, change in response to the continuous interaction with an external environment. There are two basic approaches to formally capture requirements for this sort of software: one of them emphasizes *behaviour* and its evolution; the other is focused on *data* and their transformations. Within the first paradigm, reactive systems are typically specified through (some variant of) *state-machines*. Such models capture evolution in terms of event occurrences and their impact in the system's internal state configuration (e.g. [1]). In the dual, data-oriented approach the system's functionality is specified in terms of input-output relations modelling operations on *data*. A specification presents a theory in a suitable logic, expressed over a signature which captures its syntactic interface. Its semantics is a class of concrete algebras acting as models for the specified theory [6, 12]. The starting point for our approach, is that both dimensions (data and behaviour) are interconnected: the functionality offered by a reconfigurable system, at each moment, may depend on the stage of its evolution. We model the reconfiguration dynamics as a transition structure, whose nodes are interpreted as different execution *modes*. Therefore, each of such nodes is endowed with a first-order structure modelling its (mode) local functionality (cf. [10]). Properties of these local models are expressed over the signature underlying the actual systems' interface. Thus, models are *structured* state-machines whose states denote *first-order-structures*, rather than *sets*.

Once chosen the semantics, the next issue concerns the definition of a suitable specification logic. Modal languages are the natural choice to talk about transition systems. Modal logic, however, is not expressive enough to deal with properties holding in specific states, a limitation which is overcome in hybrid

* On-going collaboration with Răzvan Diaconescu, namely on the hybridization method, is greatly acknowledged.This work is funded by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT - Foundation for Science and Technology, under contract `PTDC/EIA-CCO/108302/2008`, *Centro de Investigação e Desenvolvimento em Matemática e Aplicações* of Universidade de Aveiro, and doctoral grant `SFRH/BDE/33650/2009` supported by FCT and *Critical Software S.A., Portugal.*

logics [3, 5] by considering a special set of symbols for naming states. Additionally, we need to specify the local configurations, at each state, as first-order structures. This entails the combination of hybrid, modal features with first-order logic. Therefore, we resort to a variant of the *hybrid first order modal logic* (e.g. [5]). Note, however, that our semantic perspective, *(execution) modes-as-(local) states*, differs from the more common *states-as-algebras* approach (see, e.g. [9, 4, 2]). In our own approach, each node does not correspond to a configuration of variables over a unique first-order structure, but to a first-order structure modelling its behavior and functionality. Technically, we resort to rigid variables for non rigid operations, in opposition to the usual approach where rigid operations act on non rigid variables. Also hybrid versions of first-order modal logics may be found in the literature as specification formalisms. For instance, in [13] a logic is defined (for cyber-physical systems) based on a dynamical version of first-order logic (over the reals) with nominals. Also there, as usual, nodes are just a collection of values of system variables at a given point of execution, evaluated in an unique first-order structure.

The ability to relate models is a key point in developing programs from specifications. On the one hand, design by stepwise refinement entails the need for relating models at different levels of abstraction, up to some notion of behavioural equivalence or satisfaction of "the same properties". In axiomatic specification, refinement corresponds to inclusion of admissible model classes. In the model oriented paradigm, on the other hand, simulations, i.e., behaviour preserving relations on the state spaces, are used to establish refinement. Since, our models include both a state transition system and a family of first-order-structures, both aspects have to be considered when relating them. Therefore, we discuss a structured notion of simulation/bisimulation and how consequence is preserved along it.

The approach just sketched can be put into a broader perspective resorting to the authors recent work on hybridisation of institutions. Reference [11] introduces a systematic method to extend arbitrary institutions [8, 7] with hybrid logic features. Concretely, they are extended with Kripke semantics, for multi-modalities with arbitrary arities, as well as nominals and local satisfaction operators. This paves the way for the definition of an *institution independent methodology* to specify reconfigurable systems. The relevance of this generalisation step is in line with a basic engineering concern which recommends that the choice of a specification framework should depend on the nature of the requirements one has to deal with. For example, it may happen that, in a specific context, one would prefer to equip each local state with a partial algebra, a hidden algebra, a propositional valuation or even a hybrid logic model (since the method recurs), rather than with first-order structures as above.

On this talk we will revisit hybrid models at this level of generality, characterizing abstract notions of simulation and bisimulation appropriated for the job at hands.

References

1. Luca Aceto, Anna Ingólfsdóttir, Kim Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification.* Cambridge University Press, 2007.
2. Michel Bidoit and Rolf Hennicker. An algebraic semantics for contract-based software components. In José Meseguer and Grigore Rosu, editors, *AMAST*, volume 5140 of *Lecture Notes in Computer Science*, pages 216–231. Springer, 2008.
3. Patrick Blackburn. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of IGPL*, 8(3):339–365, 2000.
4. E. Börger and R. Stärk. *Abstract state machines: A method for high-level system design and analysis.* Springer-Verlag, 2003.
5. Torben Brauner. *Hybrid Logic and its Proof-Theory.* Applied Logic Series. Springer, 2010.
6. R. Diaconescu and K. Futatsugi. Logical foundations of CafeOBJ. *Theor. Comput. Sci.*, 285(2):289–318, 2002.
7. Razvan Diaconescu. *Institution-independent Model Theory.* Birkhäuser Basel, 2008.
8. Joseph A. Goguen and Rod M. Burstall. Institutions: abstract model theory for specification and programming. *J. ACM*, 39:95–146, January 1992.
9. Yuri Gurevich. Evolving algebras. In *IFIP Congress (1)*, pages 423–427, 1994.
10. A. Madeira, J. M. Faria, M. A. Martins, and L. S. Barbosa. Hybrid specification of reactive systems: An institutional approach. In *SEFM 2011, LNCS vol. 7041*, pages 269–285, 2011.
11. Manuel A. Martins, Alexandre Madeira, Razvan Diaconescu, and Luís Soares Barbosa. Hybridization of institutions. In *CALCO 2011, LNCS vol. 6859*, pages 283–297, 2011.
12. Till Mossakowski, Anne Haxthausen, Donald Sannella, and Andrzej Tarlecki. CASL: The common algebraic specification language: Semantics and proof theory. *Computing and Informatics*, 22:285–321, 2003.
13. André Platzer. Towards a hybrid dynamic logic for hybrid dynamic systems. *Electron. Notes Theor. Comput. Sci.*, 174:63–77, June 2007.

Constructions – Models of Signatures with Dependency Structure

Grzegorz Marczyński

Institute of Informatics, Warsaw University
gmarc@mimuw.edu.pl

1 Introduction

This paper is the continuation of the research presented in [2]. We define models and specifications that fully exercise the potential of signatures with dependency structure.

Our aim is to have categorical primitives suitable to specify software system architectures in the simplest diagrammatic way. As the basic architectural primitive we choose parametric software modules represented by a notion of a construction. Our constructions are similar to constructions of [3] or generic units of architectural specifications in CASL [1].

We define construction signatures, construction models and construction specifications. Typically (e.g. in [3]) symbols in signatures of parametric modules are a divided in two – the required and provided parts. In such case construction models are maps that, given the model of the required part, provide the model of the result part.

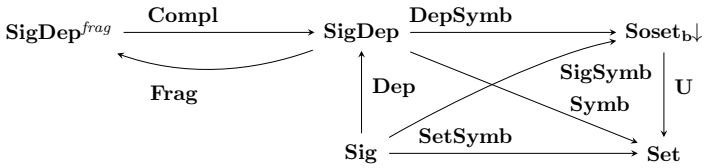
Our approach is different. The construction signatures are fragments of signatures with dependency structure from [2]. Fragments consist of the complete signature (result part) and the set of symbols that are defined by the construction. Our construction models are simply classes of models, subject to certain conditions formulated upon the symbol dependency from the construction signature. We define the category of construction signatures and their morphisms that are p-morphisms, as defined in [2]. Such representation make the pushout operation applicable to construction signatures. Therefore, constructions may be connected via fitting spans and the sum operation on construction models is defined.

Two constructions connected by a fitting span may be joined by a sum operation that corresponds to what is usually called an application of a generic unit (e.g. in CASL [1]). However, in contrast to the classical application, the sum, as we define it, is a symmetric operation. Moreover, the sum operation may be used not only to join a parametric module and a its actual parameter module, but also to join two parametric modules. It gives rise to the recursive parametric module application, where two sides are mutually actual parameters for each other. In our approach this operation is defined as the simple amalgamation of the construction models, i.e. the classes of models.

We also present the results of the ongoing research concerning the notion of construction signatures refinement morphisms and refinement of construction specifications.

2 Preliminaries

Let $Eq^{=} = \langle \mathbf{Sig}, Mod, Sen, \models \rangle$ be the institution of equational logic with many sorted algebraic signatures. We have the following functors, as defined in [2]



where

- **SigDep** is the enrichment of **Sig** by strict bounded dependency structure, with **SigDep**-objects being pairs $\Sigma_< = \langle \Sigma, <_\Sigma \rangle$ of $\Sigma \in \mathbf{Sig}$ and $<_\Sigma \subseteq \mathbf{SetSymb}(\Sigma) \times \mathbf{SetSymb}(\Sigma)$ – dependency relation that $<_{\mathbf{SigSymb}(\Sigma)} \subseteq <_\Sigma$ and $\langle \mathbf{SetSymb}(\Sigma), <_\Sigma \rangle \in \mathbf{Soset_b}\downarrow$; **SigDep**-morphisms are such algebraic signature morphisms $\sigma \colon \Sigma \to \Sigma'$, for which a function $\mathbf{SetSymb}(\sigma)$ is a $\mathbf{Soset_b}\downarrow$-mo rphism (a p-morphism),
- $\mathbf{SigSymb} \colon \mathbf{Sig} \to \mathbf{Soset_b}\downarrow$ transforms algebraic signatures to bounded strict orders of signatures' symbols (basic dependency $<_{\mathbf{SigSymb}(\Sigma)}$),
- $\mathbf{Dep} \colon \mathbf{Sig} \to \mathbf{SigDep}$ is the basic dependency functor (dependency of operation symbols on sort symbols),
- **DepSymb** and **U** are projections,
- $\mathbf{Symb} = \mathbf{DepSymb}; \mathbf{U}$ and $\mathbf{SetSymb} = \mathbf{SigSymb}; \mathbf{U}$,
- $\mathbf{SigDep}^{frag} = (\mathbf{Set} \downarrow \mathbf{Symb})$ is the *category of signature fragments*, with $\mathbf{SigDep}^{frag}$-objects being pairs $\langle f \colon A \to \mathbf{Symb}(\Sigma_<), \Sigma_< \rangle$ and $\mathbf{SigDep}^{frag}$-morphisms being pairs $\langle f \colon A \to A', \sigma \colon \Sigma_< \to \Sigma'_< \rangle$,
- $\mathbf{Frag} \colon \mathbf{SigDep} \to \mathbf{SigDep}^{frag}$ takes $\Sigma_<$ to $\langle id_{\mathbf{Symb}(\Sigma_<)}, \Sigma_< \rangle$,
- $\mathbf{Compl} \colon \mathbf{SigDep}^{frag} \to \mathbf{SigDep}$ is the *completion functor* that is a projection that takes $\langle f \colon A \to \mathbf{Symb}(\Sigma_<), \Sigma_< \rangle$ to $\Sigma_<$.

Given a signature fragment $\underline{\mathcal{B}} = \langle f \colon A \to \mathbf{Symb}(\Sigma_<), \Sigma_< \rangle \in \mathbf{SigDep}^{frag}$ and an element $b \in \mathbf{Symb}(\Sigma_<)$, let $S_b = \{a \in \mathbf{Symb}(\Sigma_<) \mid a < b\}$, the *dependency structure of* b is $\underline{\mathcal{B}}^b_<\Downarrow = \langle f|_{A \cap S_b} \colon A \cap S_b \to \mathbf{Symb}(\Sigma_< \cap S_b), \Sigma_< \cap S_b \rangle$; the *closed-down subsignature fragment induced by an element* b is $\underline{\mathcal{B}}^b_<\downarrow = \underline{\mathcal{B}}^b_<\Downarrow \cup \{b\}$.

3 Constructions and Fittings

A *construction signature* is a signature fragment $\underline{\mathcal{B}} \in \mathbf{SigDep}^{frag}$ with injective internal mapping, i.e. for $\underline{\mathcal{B}} = \langle f \colon A \to \mathbf{Symb}(\Sigma_<), \Sigma_< \rangle$, f is injective.

A *construction signature morphism* $\underline{\omega}\colon \underline{\mathcal{B}}_1 \to \underline{\mathcal{B}}_2$ is a $\mathbf{SigDep}^{frag}$-morphism.

A construction model of a construction signature $\underline{\mathcal{B}}$ is $Con \subseteq [\![\mathbf{Compl}(\underline{\mathcal{B}})]\!]$ such that, for any $a \in \underline{\mathcal{B}}$ and any two models $M, M' \in Con$, if $M|_{\underline{\mathcal{B}}^{a}\Downarrow} = M'|_{\underline{\mathcal{B}}^{a}\Downarrow}$ then $M|_{\underline{\mathcal{B}}^{a}\downarrow} = M'|_{\underline{\mathcal{B}}^{a}\downarrow}$.

The *reduct of* Con_2 *w.r.t.* $\underline{\omega}$ is a $\underline{\mathcal{B}}_1$-construction model defined as $Con_2|_{\underline{\omega}} = \{M|_{\mathbf{Compl}(\underline{\omega})} \mid M \in Con_2\}$. By $[\![\underline{\mathcal{B}}]\!]^c$ we denote a class of all construction models of a construction signature $\underline{\mathcal{B}}$.

A *construction specification* over a construction signature $\underline{\mathcal{B}}$ is a specification $SPB \in \mathbf{Spec}(\mathbf{Compl}(\underline{\mathcal{B}}))$.

A construction model $Con \in [\![\underline{\mathcal{B}}]\!]^c$ is a model of a construction specification SPB over $\underline{\mathcal{B}}$, denoted by $Con \models^c SPB$, iff for all $a \in \mathbf{Compl}(\underline{\mathcal{B}})$, for all $A \subseteq \mathbf{Compl}(\underline{\mathcal{B}})$ the following conditions hold

1. if $a \in \underline{\mathcal{B}}$, for all $M \in Con$, if $M|_{\underline{\mathcal{B}}^{a}\Downarrow} \models SPB|_{\underline{\mathcal{B}}^{a}\Downarrow}$ then $M|_{\underline{\mathcal{B}}^{a}\downarrow} \models SPB|_{\underline{\mathcal{B}}^{a}\downarrow}$
2. if $a \notin \underline{\mathcal{B}}$, for all $M \models SPB$, if $M|_{\underline{\mathcal{B}}^{a}\Downarrow} \in Con|_{\underline{\mathcal{B}}^{a}\Downarrow}$ then $M|_{\underline{\mathcal{B}}^{a}\downarrow} \in Con|_{\underline{\mathcal{B}}^{a}\downarrow}$
3. for all $M \models SPB$, if for all $b \in A$, $M|_{\underline{\mathcal{B}}^{b}\downarrow} \in Con|_{\underline{\mathcal{B}}^{b}\downarrow}$ then $M|_{\underline{\mathcal{B}}^{A}\downarrow} \in Con|_{\underline{\mathcal{B}}^{A}\downarrow}$.

A *construction fitting* $ft\colon \underline{\mathcal{B}}_1 \nwarrow\!\nearrow \underline{\mathcal{B}}_2 = \langle \underline{\varphi_1}\colon \underline{\mathcal{F}} \to \underline{\mathcal{B}}_1, \underline{\varphi_2}\colon \underline{\mathcal{F}} \to \underline{\mathcal{B}}_2\rangle$, between two construction signatures $\underline{\mathcal{B}}_1$ and $\underline{\mathcal{B}}_2$, is a span in $\mathbf{SigDep}^{frag}$, where $\underline{\mathcal{F}}$ is an empty signature fragment (i.e. for $\underline{\mathcal{F}} = \langle f\colon A \to \mathbf{Symb}(\Sigma_<), \Sigma_<\rangle$, $A = \emptyset$) and the pushout of $\underline{\varphi_1}$ and $\underline{\varphi_2}$, $\underline{\beta_1}$ and $\underline{\beta_2}$, yields the construction signature $\underline{\mathcal{B}}$, i.e. its internal mapping is injective.

A *sum of construction models* Con_1 *and* Con_2 *w.r.t. the fitting* ft is defined as $Con_1 \oplus_{ft} Con_2 = \{M \in [\![\mathbf{Compl}(\underline{\mathcal{B}})]\!] \mid M|_{\underline{\beta_1}} \in Con_1, M|_{\underline{\beta_2}} \in Con_2\}$.

Theorem 1 *The sum of construction models is a construction model.*

Construction specifications SPB_1 *and* SPB_2 *are compatible w.r.t. a construction fitting* $ft = \langle \varphi_1, \varphi_2\rangle$ iff the following conditions hold

1. for all $a \in \underline{\mathcal{B}}$ and all $a_i \in \underline{\mathcal{B}}_i$ such that $a = \beta_i(a_i)$, for $i \in \{1, 2\}$, for all $M \in [\![\underline{\mathcal{B}}]\!]$ such that $M|_{\underline{\mathcal{B}}^{a}\Downarrow} \models SPB|_{\underline{\mathcal{B}}^{a}\Downarrow}$ if $M|_{\underline{\mathcal{B}}^{a}\downarrow} \models \underline{\beta_i}_{a_i}(SPB_i|_{\underline{\mathcal{B}}_1^{a_1}\downarrow})$ then $M|_{\underline{\mathcal{B}}^{a}\downarrow} \models SPB|_{\underline{\mathcal{B}}^{a}\downarrow}$
2. for all $Con_1 \models^c SPB_1$, for all $Con_2 \models^c SPB_2$, for any $A \subseteq \mathbf{Compl}(\underline{\mathcal{B}})$ let $A_1 = \underline{\beta_1}^{-1}(\underline{\mathcal{B}}^{A}\downarrow) \subseteq \mathbf{Compl}(\underline{\mathcal{B}}_1)$ and $A_2 = \underline{\beta_2}^{-1}(\underline{\mathcal{B}}^{A}\downarrow) \subseteq \mathbf{Compl}(\underline{\mathcal{B}}_2)$, for all $M \models SPB$, if for $i \in \{1, 2\}$, $(M|_{\underline{\beta_i}})|_{\underline{\mathcal{B}}_i^{A_i}\downarrow} \in Con_i|_{\underline{\mathcal{B}}_i^{A_i}\downarrow}$ then $M|_{\underline{\mathcal{B}}^{A}\downarrow} \in (Con_1 \oplus_{ft} Con_2)|_{\underline{\mathcal{B}}^{A}\downarrow}$

Theorem 2 *If* SPB_1 *and* SPB_2 *are compatible w.r.t. the fitting* $\langle \varphi_1, \varphi_2\rangle$ *and* $Con_1 \models^c SPB_1$, $Con_2 \models^c SPB_2$, *then* $Con_1 \oplus_{ft} Con_2 \models^c \underline{\beta_1}(SPB_1)$ **and** $\underline{\beta_2}(SPB_2)$.

References

1. CoFI. CASL *Reference Manual*. 2960 (IFIP Series). 2004.
2. Grzegorz Marczyński. Algebraic signatures enriched by dependency structure. In *WADT*, pages 226–250, 2010.
3. D. Sannella and A. Tarlecki. Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Informatica*, 25(3):233–281, 1988.

Semantics of the distributed ontology language: Institutes and Institutions

Till Mossakowski[1,3], Oliver Kutz[1], and Christoph Lange[1,2]

[1] Research Center on Spatial Cognition, University of Bremen
[2] Computer Science, University of Birmingham
[3] DFKI GmbH Bremen

OWL is a popular language for ontologies. Yet, the restriction to a decidable description logic often hinders ontology designers from expressing knowledge that cannot (or can only in quite complicated ways) be expressed in a description logic. A practice to deal with this problem is to intersperse OWL ontologies with first-order axioms, e.g. in the case of bio-ontologies where mereological relations such as parthood are of great importance, though not definable in OWL. However, these remain informal annotations to inform the human designer, rather than first-class citizens of the ontology with formal semantics and impact on reasoning.

A variety of languages is used for formalising ontologies.[4] Some of these, as RDF, OBO and UML, can be seen more or less as fragments and notational variants of OWL, while others, like F-logic and Common Logic (CL), clearly go beyond the expressiveness of OWL.

This situation has motivated the distributed ontology language (DOL), a language currently under active development within the ISO standard 17347 Ontology Integration and Interoperability (OntoIOp). In DOL, heterogeneous and distributed ontologies can be expressed. At the heart of this approach is a graph of ontology languages and translations [9]. This graph will enable users to

- relate ontologies that are written in different formalisms (e.g. prove that the OWL version of the foundational ontology DOLCE is logically entailed by the first-order version);
- re-use ontology modules even if they have been formulated in a different formalism;
- re-use ontology tools like theorem provers and module extractors along translations between formalisms.

In this contribution, we will present the syntax and semantics of DOL. DOL shares many features with the language HetCASL [8] which underlies the Heterogeneous Tool Set Hets [10]. However, it also adds a number of new features:

- ontology module extraction: give me a subtheory that contains all relevant logical information w.r.t. some subsignature;
- projections of theories to a sublogic;
- ontology alignments, which involve partial or even relational variants of signature morphisms;

[4] For the purpose of this paper, "ontology" can be equated with "logical theory".

– combination of theories via colimits;
– referencing of all items by URLs, or, more general, IRIs.

What is the semantics of DOL? Previous presentations of the semantics of heterogeneous logical theories [13, 4, 11, 7, 9] relied heavily on the theory of institutions [6]. The central insight of the theory of institutions is that logical notions such as model, sentence, satisfaction and derivability should be indexed over signatures (vocabularies). In order to abstract from any specific form of signature, category theory is used.

However, the use of category theory diminishes the set of potential readers. Moreover, there is a line of *signature-free* thinking in logic and ontology research; for example, Common Logic [3] names its signature-free approach a chief novel feature. Likewise, many abstract studies of consequence and satisfaction systems [5, 12, 1, 2] disregard signatures. Hence, we base our semantics on the newly introduced notion of *institutes*. These start with the signature-free approach, and then introduce signatures a posteriori, assuming that they form a preorder. While this approach covers only signature inclusions, not renamings, it is much simpler than the category-based approach of institutions. Of course, for features like colimits, full institution theory is needed. We therefore show that institutes and institutions can be integrated smoothly.

References

1. A. Avron. Simple consequence relations. *Inf. Comput.*, 92(1):105–140, 1991.
2. W.A. Carnielli, M. Coniglio, D.M. Gabbay, P. Gouveia, and C. Sernadas. *Analysis and synthesis of logics: how to cut and paste reasoning systems*. Applied logic series. Springer, 2008.
3. Common Logic Working Group. Common Logic: Abstract syntax and semantics. Technical report, 2003.
4. R. Diaconescu. Grothendieck institutions. *Applied categorical structures*, 10:383–402, 2002.
5. G. Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–213. North-Holland, Amsterdam, 1969.
6. J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39:95–146, 1992. Predecessor in: LNCS 164, 221–256, 1984.
7. O. Kutz, T. Mossakowski, and D. Lücke. Carnap, Goguen, and the Hyperontologies: Logical Pluralism and Heterogeneous Structuring in Ontology Design. *Logica Universalis*, 4(2):255–333, 2010. Special Issue on 'Is Logic Universal?'.
8. T. Mossakowski. Hetcasl - heterogeneous specification. language summary, 2004.
9. Till Mossakowski and Oliver Kutz. The Onto-Logical Translation Graph. In *Modular Ontologies—Proceedings of the Fifth International Workshop (WoMO 2011)*, volume 230 of *Frontiers in Artificial Intelligence and Applications*, pages 94–109. IOS Press, 2011.
10. Till Mossakowski, Christian Maeder, and Klaus Lüttich. The Heterogeneous Tool Set. In Orna Grumberg and Michael Huth, editors, *TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522. Springer-Verlag Heidelberg, 2007.

11. Till Mossakowski and Andrzej Tarlecki. Heterogeneous logical environments for distributed specifications. In Andrea Corradini and Ugo Montanari, editors, *WADT 2008*, number 5486 in Lecture Notes in Computer Science, pages 266–289. Springer, 2009.
12. D. Scott. Rules and derived rules. In S. Stenlund, editor, *Logical Theory and Semantic Analysis*, pages 147–161. Reidel, 1974.
13. A. Tarlecki. Towards heterogeneous specifications. In D. Gabbay and M. de Rijke, editors, *Frontiers of Combining Systems 2, 1998*, Studies in Logic and Computation, pages 337–360. Research Studies Press, 2000.

Dualities for modal algebras

Pedro Miguel Teixeira Olhero Pessoa Nora

Department of Mathematics, Universidade de Aveiro, Portugal

Motivated by questions in semantics of modal (propositional) logics, over the past years several duality results which extend the classical Priestley and Stone dualities to categories of algebras with operators were established (see [3, 1], for instance). In this talk we follow an idea of Halmos [2] and consider larger categories "of algebras with hemimorphisms" where these operators appear as morphisms. More specifically:

1. We show how functoriality of the Kleisli construction (see [5]) can be used to deduce in a uniform manner that these categories are dually equivalent to full subcategories $\mathbf{K}$ of the Kleisli category of (variants of) the Vietoris monad $\mathbb{V}$ on the category of ordered compact Hausdorff spaces [4]. That is, we conclude that the canonical functor

$$\hom(-,1) : \mathbf{K}^{\mathrm{op}} \to \mathbf{A}$$

   (where $\mathbf{A}$ denotes the category of Boolean algebras/Heyting algebras/ distributive lattice/... with hemimorphisms) is an equivalence by simply observing that the induced morphism of monads is an isomorphism. This way we obtain Halmos duality for Stone spaces and Boolean relations (see [2, 7]) as well as a similar result for the category of Priestley spaces and certain relations.
2. We investigate the monoidal (closed) structures on the Kleisli category of $\mathbb{V}$ induced by the Cartesian product and show that in some cases the corresponding operation on the algebraic side represents bimorphisms.
3. We also investigate further properties of these categories, in particular idempotent split completeness. In this case, using general results of [6], we conclude that these categories "of algebras with hemimorphisms" are also dually equivalent to categories of certain algebras of $\mathbb{V}$, and we describe these algebras intrinsically.

References

1. M. M. Bonsangue, A. Kurz, and I. M. Rewitzky, *Coalgebraic representations of distributive lattices with operators*, Topology Appl., 154 (2007), pp. 778–791.
2. P. R. Halmos, *Algebraic logic*, Chelsea Publishing Co., New York, 1962.
3. C. Kupke, A. Kurz, and Y. Venema, *Stone coalgebras*, Theoret. Comput. Sci., 327 (2004), pp. 109–134.
4. L. Nachbin, *Topologia e Ordem*, Univ. of Chicago Press, 1950. In Portuguese, English translation: Topology and Order, Van Nostrand, Princeton (1965).
5. D. Pumplün, *Eine Bemerkung über Monaden und adjungierte Funktoren*, Math. Ann., 185 (1970), pp. 329–337.

6. R. Rosebrugh and R. J. Wood, *Split structures*, Theory Appl. Categ., 13 (2004), pp. No. 12, 172–183.
7. G. Sambin and V. Vaccaro, *Topology and duality in modal logic*, Ann. Pure Appl. Logic, 37 (1988), pp. 249–296.

Formal Specification of the Kademlia and the Kad Routing Tables in Maude

Isabel Pita[1] and María Inés Fernández Camacho[1]

Dpto. Sistemas Informticos y Computación.
Universidad Complutense de Madrid, Spain
{ipandreu,minesfc}@sip.ucm.es

1 Abstract

Distributed hash tables (DHTs) are designed to locate objects in distributed environments, like P2P systems, without the need for a centralized server. There are a large number of existing DHTs proposals; the best known are: Chord [11], Pastry [10], CAN [9], and Kademlia [4]. However, only Kademlia has been implemented in real networks through the eMule[1] and aMule[2] clients. Kad is the name given to the implementation of Kademlia incorporated to eMule and aMule, and it shows important differences from the original. There are also two BitTorrent overlays that use Kademlia: one by Azureus clients[3] and one by many other clients including Mainline[4] and BitComet[5].

Most DHTs identify both nodes and files with n-bit quantities, and keep the information of shared files in the nodes with an ID *close* to the file ID.The notion of *close* difers from DHT to DHT. Each node has a routing table to store contact information about others to access the information shared by them. Kademlia organizes its routing table as a binary tree. Contacts are stored in the tree leaves, in a special sort of FIFO queue called k-buckets. Each bucket is identified by the common prefix of the IDs it contains, which is given by the path from the root. Thus, each bucket covers some range of the ID space, and together the buckets cover the entire ID space with no overlap. The policy to add contacts to buckets in Kademlia is as follows: if the contact already exists, it is moved to the tail of the queue; otherwise, if the bucket has free space it is inserted at its tail; but if the bucket has not free space, the first contact in the bucket is contacted and if it is still online it is moved to the tail of the bucket and the new contact discarded; if the first bucket contact is not online, it is discarded and the new contact inserted at the tail. This process is different in the Kad routing tables. It does not require to contact the first bucket node, instead it has a process that removes all contacts that are not alive from time to time. In both cases routing tables are dynamic in the sense that they should be able to send messages to

[1] http://www.emule-project.net
[2] http://www.amule.org
[3] http://azureus.sourceforge.net
[4] http://www.bittorrent.com
[5] http://www.bitcomet.com

other peers and they should have a notion of time for raising events and detect no answered messages.

Although the different DHTs have been extensively studied through theoretical simulations and analysis, there is a lack of formal specifications for all of them. Bakhshi and Gurov give in [1] a formal verification of Chord's stabilization algorithm using the π-calculus. Lately Lu, Merz, and Weidenbach [3] have modeled Pastry's core routing algorithms in the specification language TLA$^+$ and have proved properties of correctness and consistency using its model checker. There is a preliminary study of the Kademlia searching process protocol by the first author in [7], and a distributed specification of the protocol in [8]. Based on these preliminary studies we have realized a detailed specification of the Kademlia and the eMule/aMule Kad routing tables in the Maude formal specification language which include features not previously specified as sending messages to update the Kademlia routing table or raising events for populate and remove offline nodes in the Kad routing table .

The Maude algebraic specification language we use, is based on rewriting logic [2, 6] and it supports both equational and rewriting logic computations while it offers simple an elegant time simulation resources. Since the specifications are directly executable, Maude can be used to prototype the systems as well as to prove properties of them.In particular the Real-Time Maude tool provides facilities to analyze the effect of time on the system.

As far as we know there is no other formal specification of the Kademlia and Kad routing tables. Right now the best sources to understand both protocol details are the original paper on the Kademlia DHT and the source code of the Kad implementation. Thus our first contribution are the benefits of having a formal specification of a system that is being consulted by many developers. In addition, the formalization of the routing tables allows us to compare the original version of Kademlia with the real implementation made in Kad. It also allows us to detect some open issues on the original description, mainly related with message passing.

However, our main contribution is the integration of the dynamic aspects of the routing table with the full protocol specification. The specification includes the ability of the routing tables to send and receive messages autonomously from the node by using different levels of configurations. Detection of non answered messages is done by assigning a time out when sending the message and triggering the appropriate action when the time expires. The actions that are performed automatically in Kad from time to time, like populate almost empty buckets, or remove offline contacts from the buckets are triggered when their time expire. These actions require to have a notion of time defined in different parts of the routing table and allow us to study the interleaving of actions that take place at different points in time.

References

1. R. Bakhshi, and D. Gurov, Verification of Peer-to-peer Algoritms: A Case Study. ENTCS 181, pages 35–47. Elsevier, 2007.

2. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Mart-Oliet, J. Meseguer, and C. Talcott, All About Maude - A High-Performance Logical Framework. LNCS 4350. Springer, 2007.
3. T. Lu, S. Merz, and C. Weidenbach. Model Checking the Pastry Routing Protocol. In J. Bendisposto, M. Leuschel, and M. Roggenbach, editors, *In Proceedings of the 10th International Workshop on Automatic Verification of Critical Systems, AVOCS 2010*, pages 19–21. Universität Düsseldorf, 2010.
4. P. Maymounkov, and D. Mazieres, Kademlia: A Peer-to-peer Information System Based on the XOR Metric. *In Proceedings of the 1st International Workshop on Peer-to Peer Systems (IPTPS02)*, 2002.
5. Mysicka, D., Reverse Engineering of eMule. An analysis of the implementation of Kademlia in eMule. Semester thesis, Dept. of Computer Science, Distributed Computing group, ETH Zurich, 2006.
6. Ölveczky, P., and Meseguer, J., Semantics and pragmatics of Real-Time Maude, *Higher Order Symbol. Comput.*, volume 20, number 1-2, pages 161–196. Kluwer Academic Publishers. 2007,
7. Pita, I., A formal specification of the Kademlia distributed hash table. In V. M. Gulías, J. Silva and A. Villanueva, editors, 10th Spanish Workshop on Programming Languages, PROLE 2010, pages 223–234. Ibergarceta Publicaciones.2010.
8. Pita, I. and Riesco, A., Specifying and Analyzing the Kademlia Protocol in Maude. 9th International Workshop on Rewriting Logic and its Applications, WRLA 2012.
9. Ratnasamy S, Francis P, Handley M, Karp R, and Shenker S. A Scalable Content-Addressable Network. *In Proceedings of SIGCOMM*, 2001.
10. Rowstron A, and Druschel P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Middleware 2001 : IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001. *Proceedings In Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg (2001)*, pp. 329-350. 2001.
11. Stoica I, Morris R, Karger D, Kaashoek M, and Balakrishnan H. Chord: A scalable peer-to-peer lookup service for Internet applications. IEEE/ACM Trans. Netw., volume 11, number 1, pages 17–32. 2003.

Mechanically Verifying Logic Translations

Florian Rabe[1] and Kristina Sojakova[2]

[1] Jacobs University, Bremen, Germany
[2] Carnegie Mellon University, Pittsburgh, USA

Institutions and institution comorphisms [GB92, GR02] are a key tool in the model theoretical meta-analysis of logics. Institutions form an abstract categorical definition of logic based signatures, sentences, models, and satisfaction. And institution comorphisms translate one institution into another.

While being its greatest strength, the abstractness of the institution framework also leads to a major drawback: institutions cannot be represented in a purely formalist setting, using a declarative language with simple and decidable validity criteria. Indeed, the Hets system [MML07], the biggest implementation of institution theory, implements the syntactical aspects of institutions in a general purpose programming language.

Therefore, in [Rab12, CHK+12], we showed how institutions and institution comorphisms can be represented in the proof/type theoretical logical framework LF [HHP93]. More precisely, we gave a general construction that defines institutions and comorphisms from certain diagrams of LF signatures.

In this work, we improve upon the above results by giving a substantially stronger construction of an institution comorphism. In the above constructions, the proof of the satisfaction condition for comorphisms hinges on the commutativity of a certain diagram of LF signatures. Specifically two morphism have to be equal: one of these morphisms represents the interpretation of objects in the reduced model, the other one represents the interpretation of translated objects in the original model.

While perfectly reasonable, this condition is very strong: It implies that all objects are interpreted equally according to the very weak definitional equality of the LF type theory. But to obtain the satisfaction condition of a comorphism, it is in fact sufficient to show that the two morphisms interpret all formulas as provably equivalent statements.

However, it has previously been very difficult to even state that condition formally. We can now give a solution to this problem by applying our recent work on logical relations. In [RS12], we developed a theory of logical relations for LF and showed how they can be expressed as formal objects in the LF type theory and thus type-checked mechanically.

Our central consists of two theorems: *(i)* logical relations can be used to express the observational equivalence of two morphisms *(ii)* observational equivalence of two morphisms can be used to show the satisfaction condition of a comorphism.

Moreover, we show that the model expansion property can be expressed in terms of the validity of certain LF morphisms as well. Together, these results

provide a simple and elegant way to formalize and mechanically institution comorphisms.

This permits the mechanic verification of semantic logic translations that are commonly used to borrow theorem provers [CM97]. This is in contrast to proof theoretic borrowing such as [MP08], which explicitly translates the found proofs back into the main logic for verification. Our approach permits the semantic verification of the soundness of the borrowing operation once and for all so that no proof translation is necessary.

References

CHK+12. M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, F. Rabe, and K. Sojakova. Towards Logical Frameworks in the Heterogeneous Tool Set Hets. In T. Mossakowski and H. Kreowski, editors, *Recent Trends in Algebraic Development Techniques 2010*, volume 7137 of *Lecture Notes in Computer Science*, pages 139–159. Springer, 2012.

CM97. M. Cerioli and J. Meseguer. May I Borrow Your Logic? (Transporting Logical Structures along Maps). *Theoretical Computer Science*, 173:311–347, 1997.

GB92. J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.

GR02. J. Goguen and G. Rosu. Institution morphisms. *Formal Aspects of Computing*, 13:274–307, 2002.

HHP93. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

MML07. T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. In O. Grumberg and M. Huth, editor, *TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522, 2007.

MP08. J. Meng and L. Paulson. Translating Higher-Order Clauses to First-Order Clauses. *Journal of Automated Reasoning*, 40(1):35–60, 2008.

Rab12. F. Rabe. A Logical Framework Combining Model and Proof Theory. *Mathematical Structures in Computer Science*, 2012. to appear; see `http://kwarc.info/frabe/Research/rabe_combining_10.pdf`.

RS12. F. Rabe and K. Sojakova. Logical Relations in Twelf. see `http://kwarc.info/frabe/Research/RS_logrels_12.pdf`, 2012.

A Generic Program Slicing Technique based on Language Definitions

Adrián Riesco[1], Irina Măriuca Asăvoae[2], Mihail Asăvoae[2]

[1] Universidad Complutense de Madrid, Spain
ariesco@fdi.ucm.es
[2] Alexandru Ioan Cuza University, Romania
{mariuca.asavoae, mihail.asavoae}@info.uiac.ro

1 Introduction

In this paper we propose a static technique for slicing programs based on a meta-level analysis of the programming language definition. This would allow a certain degree of parameterization of the slicing algorithm as modification on the language semantics could automatically be carried to it. Our approach builds on the formal executable semantics of the language of interest, given as a rewriting logic [6] theory and on the source program.

Our slicing methodology combines two steps: (1) a generic analysis of the formal executable semantics followed by (2) a data dependency analysis of the program. Step (1) is a fixpoint computation of the set of the smallest language constructs that have side-effects which, in turn step (2) uses to extract safe program slices based on a set of variables of interest. We exemplify our approach on the classical WHILE language augmented with a side-effect assignment and read/write statements, specified in Maude [3].

We make the standard assumption that equations are oriented from left to right, hence we consider them as rewrite rules. Then the algorithm computes the set of basic language constructs which produce side-effects, by inspecting the conditions and the right-hand side of each rewrite rule in the definition. For this inspection, we employ unification and an adaptation of the backward chaining technique [7]. The results are used as contexts in step (2) to infer a safe program slicing as described in the followings. We start with the program, P and a set of variables of interest V. First we identify and label the contexts containing variables of interest and increment the set V with the other variables appearing in the currently identified context. We run this step until V stabilizes. At the end, the sliced program is represented by the skeleton term containing all the labeled contexts.

Program slicing is a general and well-founded technique that has a wide range of applications in program debugging, testing and analysis. [8] contains a comprehensive survey on this topic, with an emphasis on the distinction between static and dynamic slicing methods. The approach in [5] is used to apply slicing to languages specified as unconditional term rewriting systems. It relates the dynamic dependences tracking with reduction sequences, and, applying successive transformations on the original language semantics, it gathers the necessary

dependency relations via rewriting. The recent work in [1] proposes a first slicing technique of rewriting logic computations. It takes as input an execution trace and computes dependency relations using a backward tracing mechanism. Both this work and its sequent extension to conditional term rewriting systems, in [2], perform dynamic slicing by executing the semantics for an initial given state. In comparison, we propose a static approach that is centered around the rewriting logic theory of the language definition.

Our two steps slicing algorithm resembles the approach in [4], where an algorithm mechanically extracts slices from an intermediate representation of the language semantics definition. The algorithm relies on a well-defined transformation between a programming language semantics and this common representation. It also generalizes the notions of static and dynamic slices to that of constrained slices. What we propose is to eliminate the translation step to the intermediate representation and to work directly on the language semantics.

Rewriting logic based definitions of programming languages support program executability and, at the same time, provide all the necessary information to build analysis tools. We propose a generic algorithm based on a meta-level analysis of the language semantics to extract useful information for program slicing.

References

1. Alpuente, M., Ballis, D., Espert, J., Romero, D.: Backward trace slicing for rewriting logic theories. In: CADE. pp. 34–48 (2011)
2. Alpuente, M., Ballis, D., Frechina, F., Romero, D.: Backward trace slicing for conditional rewrite theories. In: LPAR. pp. 62–76 (2012)
3. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L. (eds.): All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic, LNCS, vol. 4350. Springer (2007)
4. Field, J., Ramalingam, G., Tip, F.: Parametric program slicing. In: POPL. pp. 379–392 (1995)
5. Field, J., Tip, F.: Dynamic dependence in term rewriting systems and its application to program slicing. Information & Software Technology 40(11-12), 609–636 (1998)
6. Martí-Oliet, N., Meseguer, J.: Rewriting logic: roadmap and bibliography. Theor. Comput. Sci. 285(2), 121–154 (2002)
7. Sterling, L., Shapiro, E.Y.: The Art of Prolog - Advanced Programming Techniques. MIT Press (1986)
8. Tip, F.: A survey of program slicing techniques. J. Prog. Lang. 3(3) (1995)

Distances Between Processes: a Pure Algebraic Approach *

David Romero Hernández, David de Frutos Escrig

Dpto. Sistemas Informáticos y Computación
Facultad CC. Matemáticas, Universidad Complutense de Madrid, Spain
dromeroh@pdi.ucm.es, defrutos@sip.ucm.es

In order to define an (abstract) semantics for processes, we need just to define an adequate equivalence relation, $\equiv$, relating the processes in some universe, *Proc*. Then the values of this semantics are just the corresponding equivalence classes, and two processes have the same semantics if and only if they are equivalent. Once we have fixed such a semantics we can compare two processes, but the output of this comparison is just a Boolean value. In particular, when two processes are not equivalent we do not have a general way to measure "how far away" they are from being equivalent.

It is well-known the use of these equivalences to formalize the notion of implementation: a process implements some specification (given by another process) when they are equivalent w.r.t. the adequate semantics. But, if we follow this approach, we have no flexibility at all: our process has to satisfy in a precise way all of the constraints imposed by the specification, or it will not be a correct implementation. Instead, in the real world we often use other, more flexible quality requirements, where the specification establishes the ideal behavior of the system but some (limited) deviations from it are allowed without invalidating the correctness of the implementation.

We need a notion of distance between processes to make precise how far away two processes are from being equivalent w.r.t. some given semantics. It is true that metrics have been used for a long time to formalize the semantics of infinite processes by means of (the limit of) those of their finite approximations. But these metrics were just a very particular case that only cared for "the first" disagreement between the compared processes. Instead, now we look for more general distances which would be applicable to any syntactic process algebra (i.e. to any signature Σ) and any semantics (by means, for instance, of the axiomatization of the desired semantics).

We have already introduced the basic ideas of our approach in [4], comparing our distances with other proposals based on the simulation game that has been presented in papers such as [1]. Here, we will discuss how to define these distances in a general way by introducing metric algebras and their algebraic specification by means of distance equations.

* Partially supported by the Spanish projects TESIS (TIN2009-14312-C02-01), DESAFIOS10 (TIN2009-14599-C03-01) and PROMETIDOS S2009/TIC-1465.

Definition 1. *Let $\mathcal{D}$ an adequate domain for distance values (e.g. $\mathbb{N}$, $\mathbb{R}^+$, $\mathbb{Q}^+$) and Σ a (classic) signature. A $(\mathcal{D},\Sigma)$-algebra is a Σ-algebra $\langle \mathcal{A}, \Sigma_{\mathcal{A}}\rangle$ and a collection of relations $\langle \equiv_d, d \in \mathcal{D}\rangle$, $\equiv_d \subseteq \mathcal{D}\times\mathcal{D}$, such that:*

1. $a \equiv_0 a$ *for all* $a \in \mathcal{A}$.
2. $a \equiv_d b \Leftrightarrow b \equiv_d a$ *for all* $a, b \in \mathcal{A}$ *for all* $d \in \mathcal{D}$.
3. $d \leq d'$, $a \equiv_d b \Rightarrow a \equiv_{d'} b$ *for all* $a, b \in \mathcal{A}$ *for all* $d, d' \in \mathcal{D}$.
4. $(a \equiv_d b \wedge b \equiv_{d'} c) \Rightarrow a \equiv_{d+d'} c$ *for all* $a, b, c \in \mathcal{A}$ *for all* $d, d' \in \mathcal{D}$.
5. $f \in \Sigma$, $ar(f) = k$, $a_i \equiv_{d_i} b_i$ *for all* $i \in 1..k \Rightarrow f(\overline{a}) \equiv_{\Sigma_i d_i} f(\overline{b})$.

Definition 2. *A deduction system for distances between terms in $\mathcal{T}_\Sigma(X)$ is a collection of rules including:*

1. *Partial reflexivity:* $t \equiv_0 t$.
2. *Symmetry:* $t \equiv_d t' \Rightarrow t' \equiv_d t$.
3. *Triangular transitivity:* $t \equiv_d t'$, $t' \equiv_{d'} t'' \Rightarrow t \equiv_{d+d'} t''$.
4. *Linear substitution: If t is linear in x and $t' \equiv_d t''$ then $t[t'/x] \equiv_d t[t''/x]$.*
5. *Instantiation:* $t \equiv_d t' \Rightarrow t[t''/x] \equiv_d t'[t''/x]$.
6. *A set of distance equations* $E_{\mathcal{D}} = \{t_i \equiv_{d_i} t'_i \mid i \in I\}$.

Remark 1. i) The intuitive meaning of $t \equiv_d t'$ is "t and t' are at most d-far away".
ii) The set $E_{\mathcal{D}}$ will include the adequate axioms to characterize the desired distance w.r.t. to some semantics for processes $\sim_{sem}$, so that we expect that $\equiv_0 = \sim_{sem}$.

Finite processes can be represented by terms in the free $(\mathbf{0}, Act, +)$-algebra which correspond to the classic domain of $BCCSP(Act)$ processes.

Definition 3. *Given a set of actions Act, the set BCCSP(Act) of processes is that defined by the BNF-grammar:* $p ::= \mathbf{0} \mid ap \mid p + q$.

Example 1. 1. Given a distance $\overline{d} : Act \times Act \to \mathcal{D}$ between the basic actions in Act, the basic bisimulation distance over $\Sigma = (\mathbf{0}, Act, +)$, defining $BCCSP(Act)$ as in Def. 3, will be defined by extending Def. 1, taking

$$\text{(PD) } at \equiv_{\overline{d}(a,b)} bt \text{ for all } a, b \in Act$$

and the set of bisimulation axioms transmuted into $\equiv_0$

$$(B1)\ x + y \equiv_0 y + x \qquad (B2)\ x + x \equiv_0 x$$

$$(B3)\ (x + y) + z \equiv_0 x + (y + z) \qquad (B4)\ z + \mathbf{0} \equiv_0 z$$

For instance, let us take $Act = \{a, b, c\}$ and define $\overline{d}(a, b) = 1$, $\overline{d}(a, c) = 2$ and $\overline{d}(b, c) = 1$. Now we can show that $ab\mathbf{0} + bb\mathbf{0} \equiv_3 ac\mathbf{0} + cc\mathbf{0}$, since by (PD)

$$b\mathbf{0} \equiv_1 c\mathbf{0} \Rightarrow \left\{ \begin{array}{l} ab\mathbf{0} \equiv_1 ac\mathbf{0} \ \ (Def.\ 1.5\ taking\ a\ as\ f) \\ \qquad\qquad \wedge \\ bb\mathbf{0} \equiv_1 bc\mathbf{0} \ \ (Def.\ 1.5\ taking\ b\ as\ f) \\ \qquad\qquad \wedge \\ bc\mathbf{0} \equiv_1 cc\mathbf{0} \ \ (by(PD)) \end{array} \right\} \overset{(Def.\ 1.4)}{\Rightarrow} bb\mathbf{0} \equiv_2 cc\mathbf{0}$$

and again applying Def. 1.5 taking $+$ as f we can conclude $ab\mathbf{0} + bb\mathbf{0} \equiv_3 ac\mathbf{0} + cc\mathbf{0}$

2. The basic distance associated to any semantics for processes which can be axiomatized is obtained by replacing the axiomatization for bisimulation in the previous example by the corresponding axiomatization of the corresponding semantics, using again the relation $\equiv_0$.
   We will compare those "global" algebraic distances with those based on games in [2]. As we have seen in [4], these game-based distances are "more local". They are defined taking into account only isolated computations, while we consider the set of computations as a whole. In fact, is the use of $\Sigma_i d_i$ in Def. 1.5 which generates this global treatment. Using $max_i d_i$ we would obtain the algebraic characterization of the game-based distances.

The basic distances above are based on the idea of adding the distances for the actions that do no match when comparing two processes, looking for the pairings which produce the minimal disagreements. Although the algebraic approach above only considers finite terms, following [3] we can extend it also to infinite processes, by considering continuous metric algebras, so that even two processes with infinitely many disagreements between them could be proved to be at some finite distance. Following the classic approach (see e.g. [1]) we will use weighted distances, which are introduced here by adapting the concept of guarded recursion.

Definition 4. *A $(\mathcal{D}, \Sigma)$-weighted algebra for the collection of reducing weights factors* $\mathrm{rwf} : \Sigma \to \mathcal{D}$ *is a $(\mathcal{D}, \Sigma)$-algebra which satisfies:*

6. $f \in \Sigma$, $ar(f) = k$, $a_i \equiv_{d_i} b_i$ *for all* $i \in 1..k \Rightarrow f(\overline{a}) \equiv_{\mathrm{rwf}(f) * \Sigma_i d_i} f(\overline{b})$.

We will change our deduction system in a similar way and also introduce the corresponding versions of continuous algebras. Following [3] we will consider the particular case of acceptance trees thus getting algebraic support for a notion of distance for the testing semantics, which is well-known to be very close to the failures semantics.

References

1. U. Fahrenberg, A. Legay, and C. R. Thrane. The quantitative linear-time–branching-time spectrum. In S. Chakraborty and A. Kumar, editors, *FSTTCS*, volume 13 of *LIPIcs*, pages 103–114. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.
2. Uli Fahrenberg, Claus R. Thrane, and Kim G. Larsen. Distances for weighted transition systems: Games and properties. In Mieke Massink and Gethin Norman, editors, *QAPL*, volume 57 of *EPTCS*, pages 134–147, 2011.
3. M. Hennessy. Algebraic theory of processes. MIT Press, 1988.
4. D. Romero-Hernández and D. Frutos-Escrig. Defining distances for all process semantics. In *FORTE 2012*, LNCS, to appear. Springer.

Multiset Rewriting for the Verification of Depth-Bounded Processes with Name Binding ⋆

Fernando Rosa-Velardo and María Martos-Salgado ⋆⋆

Sistemas Informáticos y Computación, Universidad Complutense de Madrid
E-mail: mrmartos@estumail.ucm.es, fernandorosa@sip.ucm.es

Multiset rewriting. *Multiset Rewriting Systems* (MSR) [3] is a specification language for security protocols, in which facts are first-order atomic formulae and transitions are given by means of rewriting rules with existential quantifications. For instance, the rule

$$A_0(k), Ann(k') \rightarrow \exists x.(A_1(k,x), N(enc(k', \langle x, k\rangle)), Ann(k'))$$

specifies the first rule of the Needham-Schroeder protocol, in which a principal A with key k ($A_0(k)$) decides to talk to another principal, with a key k' that has been announced ($Ann(k')$), for which it creates a nonce x and sends to the network the pair $\langle x, k\rangle$ ciphered under k'. Another formalism based on multiset rewriting is *Constraint Multiset Rewriting Systems* (CMRS) [4]. In CMRS facts are first-order atomic formulae, but the terms that can appear as part of such formulae must belong to a *constraint system*, and no existential quantification is considered. For instance, the rule $count(x), visit \rightarrow count(x+1), enter(x+1)$ could be used to count the number of visits to a web site.

ν-MSR. In [7] we combined the features of MSR and CMRS, obtaining ν-MSR. On the one hand, we maintain the existential quantifications in [3] to keep a compositional approach, closer to that followed in process algebra with name binding. On the other hand, we restrict terms in atomic formulae to be pure names, that can only be compared with equality or inequality, unlike the arbitrary terms over some syntax in MSR, or terms in a constraint system in CMRS. With this minimal set of primitives we can specify process algebra with name binding, which allows us to obtain new decidability results for those formalisms in a common framework.

A ν-MSR term can be of the form $\mathbf{0}$, $p(a_1, \ldots, a_n)$, $M_1 + M_2$ or $\nu a M$, where M, M_1 and M_2 are terms. We denote by $\mathcal{M}$ the set of ν-MSR terms. Terms are identified up to a congruence relation $\equiv$, the least congruence where α-conversion of bound names is allowed, such that $(\mathcal{M}, +, \mathbf{0})$ is a commutative monoid and the three following equations hold: $\nu a.\nu b.M \equiv \nu b.\nu a.M$, $\nu a.\mathbf{0} \equiv \mathbf{0}$, and $\nu a.(M_1 + M_2) \equiv \nu a.M_1 + M_2$ if $a \notin fn(M_2)$. A rule t is an expression of the form $t : X_1 + \ldots + X_n \rightarrow \nu\tilde{a}.(Y_1 + \ldots + Y_m)$. Then, a ν-MSR system is given by an initial term and a set of rules $\mathcal{R}$. The semantics are defined by three standard

⋆ This abstract describes the work in [7].

⋆⋆ Authors partially supported by the MEC Spanish project DESAFIOS10 TIN2009-14599-C03-01, and the CAM program PROMETIDOS S2009/TIC-1465.

rules, stating that rewritings can happen inside + and ν, and modulo $\equiv$, and the rule schema (t), that assumes $\sigma(\mathit{Var}(t)) \cap \tilde{a} = \emptyset$, as follows:

$$(t)\ \frac{t = l \to r \in \mathcal{R} \quad \sigma : \mathit{Var}(t) \to \mathit{Id}}{\sigma(l) \to \sigma(r)} \qquad \frac{M_1 \equiv M_1' \to M_2' \equiv M_2}{M_1 \to M_2}\ (\equiv)$$

$$(+)\ \frac{M_1 \to M_2}{M_1 + M \to M_2 + M} \qquad \frac{M_1 \to M_2}{\nu a.M_1 \to \nu a.M_2}\ (\nu)$$

For instance, let $t : p(x), q(x) \to \nu b.p(b)$ be a rule in $\mathcal{R}$. Then the rewriting $p(a), q(a) \to \nu b.p(b)$ can take place. Consider now the term $p(b), q(b)$. In order to apply the previous rule, one must necessarily consider the substitution σ given by $\sigma(x) = b$, that does not satisfy $\sigma(\mathit{Var}(t)) \cap \{b\} = \emptyset$. Therefore, we need to first rename b in the right handside of the rule, obtaining (e.g. if we replace b by c) $\nu c.p(c)$.

Depth-boundedness. Depth-boundedness is a semantic restriction recently introduced for the π-calculus [6]. Intuitively, a process is depth-bounded whenever the interdependence of names is bounded in any process reachable from it. We have introduced this notion for ν-MSR. For instance, if we can reach all the terms of the form $\nu a_1, \ldots, \nu a_n.(p(a_1, a_2), p(a_2, a_3), \ldots, p(a_{n-1}, a_n))$ then the ν-MSR is not depth-bounded. However, if all the reachable terms are of the form $\nu a.\nu a_1. \ldots .\nu a_n.(p(a, a_1), \ldots, p(a, a_n))$ then the ν-MSR is depth-bounded. Following [6], we have proved that depth-bounded ν-MSR belong to the class of (strict) Well Structured Transition Systems (WSTS) [5] which are basically transition systems endowed with a well quasi-order for which the transition relation is monotonic. For them, properties like termination, boundedness or coverability are decidable.

Application to other formalisms. Then we have applied the verification results obtained for depth-bounded ν-MSR for other process algebra like the π-calculus (already considered in [6]), MSR [3] and the Ambient Calculus (AC) [2]. We sketch here the results obtained for AC. AC is a process algebra for the specification of concurrent systems executing in a dynamical hierarchical topology. Its syntax is defined as follows:

Processes		**Actions**	
$P, Q ::=$	processes	$\pi ::=$	capabilities
$(\nu n)P$	restriction	$\mathit{in}\ n$	can enter in n
0	inactivity	$\mathit{out}\ n$	can exit n
$P \mid Q$	composition	$\mathit{open}\ n$	can open n
$!P$	replication		
$n[P]$	ambient		
$\pi.P$	capability action		

The main concept is that of an ambient, a place where computations happen. Thus, $n[P]$ describes an ambient named n, containing a process P. There are several operations that an ambient (actually, the processes it contains) may perform: entering another ambient, exiting from its parent ambient, and being opened by its parent ambient. The reduction rules managing these operations are the following:

$$\begin{array}{ll} n[in\ m.P \mid Q] \mid m[R] \to m[n[P \mid Q] \mid R] & \text{(In)} \\ m[n[out\ m.P \mid Q] \mid R] \to n[P \mid Q] \mid m[R] & \text{(Out)} \\ open\ n.P \mid n[Q] \to P \mid Q & \text{(Open)} \end{array}$$

In [7] we encode AC inside ν-MSR and see that, with this encoding, depth-boundedness corresponds to mobile ambients in which the interdependence of names is bounded, and in which the height in the hierarchy of ambients is also bounded.

We encode a process P inside an ambient called x by a multiset of ν-MSR predicates $[\![P]\!]^x$. We represent the ambient hierarchy using predicates of the form $amb(x, y, z)$, meaning that y is an ambient with name x, inside ambient z. Moreover, we represent sequential processes by predicates as, for example, $in\ x.P$ or $out\ x.P$. Then, we define ν-MSR rules that encode the transition relation of AC. For example, the rules managing entering or exiting an ambient are:

$$ok, amb(x, y, z), (in\ t.P)^y, amb(t, u, z) \to ok, amb(x, y, u), amb(t, u, z), [\![P]\!]^y$$

$$ok, amb(x, y, z), amb(t, u, y), (out\ x.P)^u \to ok, amb(x, y, z), amb(t, u, z), [\![P]\!]^u$$

Given an ambient name n and a process P, the *name convergence* problem is that of deciding whether $P \to^* n[Q] \mid R$ for some Q and R. For depth-bounded processes (even with name restriction and with the open capability) the name convergence problem is decidable (unlike in general, even without name restriction and without the open capability [1]). Indeed, it is enough to decide whether $\nu a.amb(n, a, \top)$ can be covered from $[\![P]\!]^\top$, where $\top$ is the top ambient in the hierarchy.

References

1. Boneva, I., Talbot, J.-M. When Ambients Cannot be Opened! TCS **333**(1-2):127-169, 2005.
2. Cardelli, L., Gordon, A.D.: Mobile ambients. TCS **240**(1) (2000) 177–213
3. Cervesato, I., Durgin, N.A., Lincoln, P., Mitchell, J.C., Scedrov, A.: A meta-notation for protocol analysis. In: CSFW. (1999) 55–69
4. Delzanno, G.: An overview of MSR(C): A CLP-based framework for the symbolic verification of parameterized concurrent systems. ENTCS **76** (2002)
5. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! TCS **256**(1-2) (2001) 63–92
6. Meyer, R.: On boundedness in Depth in the pi-calculus. In IFIP TCS. Volume 273 of IFIP, Springer (2008) 477–489
7. Rosa-Velardo, F., Martos-Salgado, M.: Multiset Rewriting for the Verification of Depth-Bounded Processes with Name Binding. Inf. Comput. **215** (2012) 68–87

Deriving architectural reconfigurations

Alejandro Sanchez[1,2], Luis S. Barbosa[2], and Daniel Riesco[1]

[1] Departamento de Informática, Universidad Nacional de San Luis,
Ejécito de los Andes 950, D5700HHW San Luis, Argentina
{asanchez,driesco}@unsl.edu.ar
[2] HASLab INESC TEC & Universidade do Minho,
Campus de Gualtar, 4710-057 Braga, Portugal
lsb@di.uminho.pt

Complex software systems are built by plugging together heterogenous computational entities in very large, often loosely coupled, highly dynamic configurations. Change being the norm, rather than the exception, dynamic reconfiguration emerged as a main theme in architectural design.

Archery [5, 4] is an architectural description language that aims at addressing reconfigurability as a first class concern. It is based on two kinds of entities: *components*, the typical *loci* of computation, and *connectors* representing interaction protocols between them. The language supports the specification of architectural types, called *patterns* in this context, and their instances which form the concrete building blocks of architectural designs. A specific language extension provides a set of operations to (re)configure pattern instances. Another extension is envisaged for the specification of logic constraints that pattern instances are supposed to fulfil.

Archery behavioural and structural semantics are given, respectively, by a translation to mCRL2 [2] and by an encoding into *bigraphical reactive systems* [3]. mCRL2 is a specification language for reactive systems which combines a process algebra [1], for describing system's behaviours, with higher-order abstract equational data types, for handling structured data domains. Behavioural properties can also be described in the μ-calculus. Translating the behavioural component of Archery descriptions into mCRL2, provides access to the associated toolset enabling simulation, visualisation, behavioural reduction and verification. Bigraphical reactive systems (BRS) and their theory, on the other hand, were developed to study systems in which locality and linking of computational agents varies independently. They provide a general unifying theory in which different calculi for concurrency and mobility can be represented. In particular, if some conditions are met, it is possible to automatically derive from a given BRS, a labelled transition system (LTS), in which behavioural equivalence is a congruence.

This fact is explored in the present paper to represent and further analyse architectural reconfigurations as transitions in a LTS, systematically derived from the bigraphical encoding of an Archery specification. States in the derived LTS stand for configurations which can be reached by applying a sequence of Archery reconfiguration operations. Each transition $s \to s'$ stands for a reconfiguration process leading from a configuration s to another s'. Such a process can be atomic, i.e. corresponding to the application of a single reconfiguration

operation, or structured, abstracting a whole sequence of such operations. Since, typically, the space of possible configurations is infinite, a quotient space, up to some notion of configuration equivalence, needs to be considered.

The possibility to derive a LTS from a given Archery specification requires a number of conditions to be met. First, the semantics presented in [4] needs to be refined by providing a sorting that rules out the representation of any invalid Archery configuration. Then, the encoding in a category $\textsc{Bg}(\mathcal{K}, \mathcal{R})$, for a particular signature $\mathcal{K}$ and set R of reaction rules, has to be restricted to a category $\textsc{Bg}(\Sigma, \mathcal{R})$, where $\Sigma = (\Theta, \mathcal{K}, \Phi)$ is a sorting over sorts Θ, signature $\mathcal{K}$, and formation rule Φ. We also need to ensure that the sorting defined does not prevent the necessary underlying structure required for the derivation. This is equivalent to proving that the forgetful functor $\mathcal{F} : \textsc{Bg}(\Sigma, \mathcal{R}) \to \textsc{Bg}(\mathcal{K}, \mathcal{R})$ is safe [3].

Once the LTS is obtained from a configuration, it can be used as any other LTS. Thus, reconfigurations can be animated, visualised, analysed and verified. One may even go a step further and define *reconfiguration constraints* as μ-calculus formulae over the derived LTS. After translation, constraints can be verified within the mCRL2 toolset.

For example, consider the Client-Server pattern. It prescribes configurations arranged by instances of two element types: client and server. The main design principle of the pattern is that clients can only connect to servers and vice-verse. Such a principle can be expressed as a modal property indicating that an operation connecting a client with another client cannot occur, and similarly for servers.

References

1. J. C. M. Baeten, T. Basten, and M. A. Reniers. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press, 2010.
2. J. F. Groote, A. Mathijssen, M. Reniers, Y. Usenko, and M. van Weerdenburg. The formal specification language mCRL2. In *Methods for Modelling Software Systems: Dagstuhl Seminar 06351*, 2007.
3. R. Milner. *The space and motion of communicating agents*, volume 54. Cambridge University Press, 2009.
4. Alejandro Sanchez, Luis S. Barbosa, and Daniel Riesco. Bigraphical modelling of architectural patterns (to appear). In *Post-proceedings of the Eighth International Symposium on Formal Aspects of Component Software*, FACS 2011. Springer, 2011.
5. Alejandro Sanchez, Luis S. Barbosa, and Daniel Riesco. A language for behavioural modelling of architectural patterns. In *Proceedings of the Third Workshop on Behavioural Modelling*, BM-FA '11, pages 17–24, New York, NY, USA, 2011. ACM.

Verifying Parallel Recursive Programs by Regular Approximations of Context-Free Languages

Pierre-Yves Schobbens

Precise Research Center, University of Namur

Abstract. Given a context-free grammar, we propose families of regular approximations of the language of this grammar based on the characteristic LR automata. We also present two applications: generating a lexer directly from the grammar to solve LR conflicts, and heuristically checking the intersection of context-free languages, which is used to verify parallel recursive programs.

1 Introduction

Context-free grammars, also called algebraic grammars, have widespread use in compute science, but we focus here on two main uses: the description and parsing of languages [3, 1], and the description and verification of the executions of recursive programs [14, 7, 2]. Unfortunately, many problems about context-free grammars (universality, ambiguity, language inclusion and intersection) are undecidable, in contrast to the corresponding problems on regular expressions. It is thus natural to look for regular approximations of context-free languages, that can heuristically solve these problems in some practically useful instances. Many fixed schemes are available [4, 5, 9, 8, 10, 11, 12, 13], but we need a family of approximations whose precision can be increased incrementally. To formalise this intuition, we introduce the notion of k-precise approximation, and show that existing schemes are not k-precise.

Our approximations are based on the LR(k) items. It is easy to build a non-deterministic pushdown automaton with items in its stack, that recognizes the language of the grammar. We show that the possible contents of this stack after reading a given word form a regular language (that we represent by an *inner automaton*), and similarly after reading a viable prefix.

2 Applications

Building a lexer

Our regular over-approximation can be used to build an automaton that will read the input ahead of the parser [1], and give it information to solve conflicts in deterministic parsers, such as LR, LALR, LL. This effect is classically obtained

by manually crafting an ad-hoc lexer. We express the lexer and the parser rules in a single grammar, and let our approximation scheme build a contextual lexer automatically. The context conditions are then provided by the grammar itself, and the cooperation between the parser and the lexer is automatic.

For each LR state, we need their predecessors on the stack. We can naturally associate predecessors to each LR state by following the structure of the LR automaton. The loops in the LR automaton give rise to loops inside the inner automaton, and the directed acyclic graph of loop-free part give rise to a similar dag in the inner automaton.

For each conflictual state S of the LR automaton, we group the items of S according to their action: either shift ($G_{\rightarrow}$), or reduce a given production r (G_r). The initial state of each group is then completed by adding its predecessors (as explained above), then reductions and expansions.

The automata are then constructed using abstraction to keep them finite. We take the product of these automata, in order to follow the possibilities in parallel. A state of the product is final if all the components but one (corresponding to the decision taken) are empty. The decision of the surviving automaton is then taken by the LR parser, that proceeds until next conflict. If no decision can be taken, the product automaton will comprise a loop on the end-of-input symbol. The technique then fails to generate a deterministic parser. It is then possible to either use a finer abstraction, in the hope to resolve the conflict, or to generate a non-deterministic parser (e.g., a GLR parser [15]).

Recursive parallel program verification

A programme with recursive procedure calls can be naturally modelled as a context-free language, presented in the form of a recursive automata, also called "railroad diagrams': each procedure corresponds to a non-terminal, and the flow graph of the body of the procedure forms corresponds to the automaton [14]. If several parallel threads P_i are synchronized (e.g. by rendez-vous or global variables), the problem is naturally modelled as an intersection of context-free languages.We can also check properties if their complement P_0 (the bad executions) is a context-free language. Each P_i can be over-approximated by a regular language R_i, and we can check whether their intersection $\bigcap_i R_i = I$ is empty. If yes, we are guaranteed to have no bad executions; else, the intersection may contain spurious executions due to the over-approximation. We thus focus on some P_i and compute $P_i \cap I$, that is context-free. We use the method of [6], that is based on on-the-fly construction of the GLR automaton. Our method similarly builds a determinized automaton on-the-fly, and they blend easily to compute the product. The method only generates reachable elements, and the intersection is not empty iff it generates a tuple of states, all of which are final. If $P_i \cap I$ is empty, we are done. Otherwise, the algorithm can provide a specific sentence w leading to this final tuple. We then check if w is accepted by the other P_j. If yes, we found a counterexample; else we found a R_j needing refinement, and we increase the precision to eliminate the spurious counterexample.

References

1. Manuel E. Bermudez and Karl M. Schimpf. A practical arbitrary look-ahead LR parsing technique. In *SIGPLAN Symposium on Compiler Construction*, pages 136–144. ACM, 1986.
2. O. Burkart and B. Steffen. Model checking for context-free processes. In *CONCUR'92*, pages 123–137. Springer, 1992.
3. K. Culik II and R. Cohen. Lr-regular grammars–an extension of lr (k) grammars*. *Journal of Computer and System Sciences*, 7(1):66–96, 1973.
4. Ömer Eğecioğlu. Strongly regular grammars and regular approximation of context-free languages. In *Proceedings of the 13th International Conference on Developments in Language Theory*, DLT '09, pages 207–220, Berlin, Heidelberg, 2009. Springer-Verlag.
5. Edmund Grimley Evans. Approximating context-free grammars with a finite-state calculus. In *ACL '98*, pages 452–459, Stroudsburg, PA, USA, 1997. Association for Computational Linguistics.
6. Thomas Hanneforth. A practical algorithm for intersecting weighted context-free grammars with finite-state automata. In *FSM-NLP*, pages 57–64, Blois, France, July 2011. Association for Computational Linguistics.
7. Zhenyue Long, Georgel Calin, Rupak Majumdar, and Roland Meyer. Language-theoretic abstraction refinement. In Juan de Lara and Andrea Zisman, editors, *FASE*, volume 7212 of *LNCS*, pages 362–376. Springer, 2012.
8. M. Mohri and M.J. Nederhof. Regular approximation of context-free grammars through transformation. *Robustness in language and speech technology*, 17:153–163, 2001.
9. M. Mohri and F.C.N. Pereira. Dynamic compilation of weighted context-free grammars. In *Proceedings of the 17th international conference on Computational linguistics-Volume 2*, pages 891–897. Association for Computational Linguistics, 1998.
10. M.J. Nederhof. Context-free parsing through regular approximation. In *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pages 13–24. Association for Computational Linguistics, 1998.
11. M.J. Nederhof. Practical experiments with regular approximation of context-free languages. *Computational Linguistics*, 26(1):17–44, 2000.
12. M.J. Nederhof. Regular approximation of cfls: a grammatical view. *Advances in Probabilistic and other Parsing Technologies*, 16:221–241, 2000.
13. F.C.N. Pereira and R.N. Wright. Finite-state approximation of phrase structure grammars. In *Proceedings of the 29th annual meeting on Association for Computational Linguistics*, pages 246–255. Association for Computational Linguistics, 1991.
14. G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, March 2000.
15. M. Tomita. *Generalized LR parsing*. Springer, 1991.

From Bialgebraic Semantics to Universal Simulators of Cellular Automata

Extended Abstract

Baltasar Trancón y Widemann

Computer Science and Ecological Modelling, University of Bayreuth, DE

The *structural operational semantics* (SOS) approach of Plotkin [4] is one of the most popular variants of operational semantics, and has been applied practically to a wide variety of programming calculi and languages. It has gained considerable theoretical interest after the categorization by Turi and Plotkin [6]. There, a certain well-behaved SOS rule format is shown to correspond to a distributive law of a syntactical functor Σ over a behavioral functor B. This view entails numerous nice mathematical properties: Both a syntactical and a denotational model arise automatically as the initial Σ-algebra and the final B-coalgebra, respectively. A unique *bialgebra* homomorphism connects the two, giving denotational and operational semantics simultaneously: Denotational semantics because a homomorphism from the initial Σ-algebra can be seen as a syntax-directed interpretation in a semantic domain; operational semantics because a homomorphism into the final B-coalgebra can be seen as a fully abstract specification of machine transitions. For well-behaved behaviour B, there is only one form of bisimulation, and that is a Σ-congruence, making the semantics fully abstract by construction.

The practical status of bialgebraic SOS is dialectic: On the one hand, it has been noted in [1] that the semantics are directly executable, and hence yield interpreters for embedded domain-specific languages, in a host language with support for recursion and corecursion, such as the lazy functional programming language Haskell. On the other hand, applied cases apparently remain largely confined to the field of process calculi; see [2] for an overview. The paradigmatic issue of existing applications is the compositional treatment of sequential, concurrent and/or nondeterministic process steps, in the style of classical process algebras such as CCS and CSP, or the more recent π-calculus.

We have recently reported a rather different application of bialgebraic SOS [5]: multi-dimensional cellular automata and their high-level application, agent-based models. They show some potential of the approach not fully realized in process calculus examples: The intricate spatio-temporal structure of cellular automata decomposes neatly, with spatial configurations of states and temporal evolution relegated orthogonally to Σ and B, respectively. The distributive law specifies the topology and neighbourhood shapes, in the form of SOS rules for all spatially composite cases, and the local transition rule of the particular cellular automaton, in the form of the single-cell base case. The approach affords a compositionality not usually found in accounts of cellular automata, having the same automata-theoretic view on individual states as on spatial composites, and en-

courages the definition of nontrivial topologies that do not have straightforward mappings to classical implementations, based on arrays and index manipulation.

The compound inductive–coinductive principle entailed by the bialgebraic approach has been used to prove the equivalence of the SOS-style semantics to classical array-based simulation algorithms that compute convolutions of the state matrix with a neighbourhood kernel. The bialgebra decomposition also has interesting implications for a philosophical analysis of the modelled systems, with the distributive law axiomatically stating an equivalence between ontic (algebraic) and epistemic (coalgebraic) states. In particular, from the fact mentioned above that bisimulation is a congruence, it follows directly that no proper emergent properties can exist in such models: the behaviour of spatial aggregates is always a direct functional consequence of the behaviour and arrangement of their parts. This finding is in remarkable contradiction to the prominent role of emergence in scientific literature discussing the models.

Here, we discuss real-world aspects of the derivation of a universal simulator for cellular automata, as an embedded domain-specific language in Haskell, from the bialgebraic semantics. We describe how to implement the underlying coupled recursion–corecursion scheme, how to abstract from concrete neighbourhood shapes, and how to separate the generic, topological part of the distributive law from the specific, local transition part. We demonstrate how laziness can be exploited in a theoretically sound way for dealing with spatially unbounded state. We indicate how relevant practical properties of the simulator, such as parallelism and optimization potential, can be read off the SOS rules.

We show how the simulator can be turned into a potentially massively parallel Haskell program by a minuscule and straightforward change. Some measurements from a first experiment on a multicore machine and the implied potential for larger applications are discussed. We describe how the key idea underlying the distributive law for cellular automata can be understood, and potentially generalized to other scenarios, by analogy to the design of divide-and-conquer-parallel algorithms. As such, the bialgebraic approach to system semantics could potentially be understood as a canonical representation of inherent parallelism. Finally, we comment the analogy of the parallelism as specified by bialgebraic interpreters to methods from parallel numerical functional programming, namely data distribution algebras [3], a formalized approach to parallel algorithmic skeletons which gives effectively executable specifications for divide-and-conquer solutions to data-parallel problems.

References

1. Jaskelioff, M., Ghani, N., Hutton, G.: Modularity and implementation of mathematical operational semantics. Electronic Notes in Theoretical Computer Science 229(5), 75–95 (2011)
2. Klin, B.: Bialgebras for structural operational semantics: an introduction. Theoretical Comp.uter Science 412(38), 5043–5069 (2011)

3. Pepper, P., Sdholt, M.: Deriving parallel numerical algorithms using data distribution algebras: Wang's algorithm. Tech. Rep. 96-2, Technische Universitt Berlin, Fachbereich Informatik (1996)
4. Plotkin, G.D.: A structural approach to operational semantics. Tech. Rep. DAIMI FN-19, Computer Science, Aarhus University, Denmark (1981)
5. Trancn y Widemann, B., Hauhs, M.: Distributive-law semantics for cellular automata and agent-based models. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) Proceedings 4th International Conference on Algebra and Coalgebra (CALCO 2011). pp. 344–358. No. 6859 in Lecture Notes in Computer Science, Springer (2011)
6. Turi, D., Plotkin, G.D.: Towards a mathematical operational semantics. In: Proceedings 12th International Conference on Logic in Computer Science (LICS 1997). pp. 280–291. IEEE (1997)

On the Instantiation of Parameterised Specifications

Ionuţ Ţuţu

[1] Department of Computer Science, University of Leicester, United Kingdom
[2] "Simion Stoilow" Institute of Mathematics of the Romanian Academy, Romania

We present an overview of the formalization of parameterised specifications in the abstract framework of institution theory [7]. The goal of our work is to identify a set of distinctive features of specifications languages that have a fundamental role in defining and instantiating parameterised specifications. The investigations follow the recent developments in the field of abstract structured specifications [2], and in this way are independent not only of the underlying logical system but also of the actual structuring operators. We consider both simultaneous and sequential instantiation of parameters, and allow sharing between the body of the parameterised specification and the instances of the parameters, as well as between the various parameters of a generic specification.

From the point of view of formal specifications, parameterisation plays an essential role in increasing the expressive power of the underlying specification formalisms that support this mechanism. When certain conditions are met by the base specification language, the systematic use of parameterisation allows the development of complex module expressions in which new parameterised specifications can be obtained by partially instantiating existing generic specifications, or instantiating their parameters with other generic specifications. As it was pointed out in [6], we can gain in this way both the specification power of richer languages and the desired properties for specification and verification of the simpler ones.

The first part of our study is dedicated to the examination of parameterised objects and their instantiation in categories endowed with an auxiliary structure that is rich enough for the study of parameterisation – a distributive quasi-inclusion system. The new concept arises as a slight generalization of the well known notion of inclusion system [5] that allows the existence of distinct quasi-inclusions opposite one to the other. Its main advantage is the possibility to lift quasi-inclusions from the signatures of the underlying logical system to structured specifications. Furthermore, we require that the considered category satisfies a number of properties that are essential to parameterisation such as the existence of free extensions [3] for a specific class of morphisms. We begin with the analysis of the simpler case of objects that have only one parameter, and then generalize the constructions for the more elaborated case of multiple parameterised objects. For the later we discuss two distinct instantiation procedures and identify a set of conditions that are sufficient for guaranteeing that the possible instantiation scenarios produce isomorphic results.

Since the aforementioned properties are too restrictive for numerous categories of signatures belonging to the foundations of various specification lan-

guages (for example the OBJ family of languages [8, 4]), in the second part of our study we concentrate on the basic properties of functors that would assist the analysis of parameterisation in the framework of abstract structured specifications. In this sense, we introduce the property of strongly lifting co-cones which, together with faithfulness, allows functors to lift both quasi-inclusion systems and co-limits. These results (applied to the structuring functor), combined with those developed for the signatures of the base institution, allow us to focus on the examination of multiple parameterised specifications at the level of structured institutions.

The present work extends the theory of pushout-style parameterisation [1] in two directions. First, it imposes minimum restrictions on the instantiation of parameters, thus allowing both sharing between the parameters, and between the body of the parameterised specification and the instances of the parameters. Second, it was developed within the high level context of abstract structured specifications and in this way it is independent of both the underlying logical system and the concrete structuring operators.

Our efforts concentrate on the study of two main instantiation procedures, distinct not only by their inner workings but also by the situations in which they can be employed. When both can be applied, the results of all possible instantiations of a parameterised specification prove to be isomorphic, assuming that a specified set of sufficient conditions hold. We investigate these conditions for a number of base logical systems and structuring formalisms and show they are smoothly satisfied.

References

1. Rod M. Burstall and Joseph A. Goguen. Putting theories together to make specifications. In *International Joint Conferences on Artificial Intelligence*, volume 2, pages 1045–1058, 1977.
2. Răzvan Diaconescu. An axiomatic approach to structuring specifications. *Theoretical Computer Science*, 433:20–42, 2012.
3. Răzvan Diaconescu and Ionuţ Ţuţu. On the algebra of structured specifications. *Theoretical Computer Science*, 412(28):3145–3174, 2011.
4. Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ report: the language, proof techniques, and methodologies for object-oriented algebraic specification.* AMAST series in computing. World Scientific, 1998.
5. Răzvan Diaconescu, Joseph A. Goguen, and Petros Stefaneas. Logical support for modularisation. In Gerard Huet and Gordon Plotkin, editors, *Logical Environments*, pages 83–130. Cambridge, 1993.
6. Joseph A. Goguen. *Higher-order functions considered unnecessary for higher-order programming*, pages 309–351. Addison-Wesley, 1990.
7. Joseph A. Goguen and Rod M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.
8. Joseph A. Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. In Joseph A. Goguen and Grant Malcolm, editors, *Software engineering with OBJ: algebraic specification in action*, Advances in formal methods. Kluwer Academic, 2000.

Author Index