A Rule Format for Rooted Branching Bisimulation

Valentina Castiglioni, Ruggero Lanotte, and Simone Tini

Dipartimento di Scienza e Alta Tecnologia, Università dell'Insubria, Como, Italy

Abstract. SOS rule formats are sets of syntactical constraints over SOS transition rules ensuring semantical properties of the derived LTS. Given a rule format, our proposal is to try to relax the constraints imposed by the format on each single transition rule at the price of introducing some reasonable constraint on the form of the whole set of rules, obtaining a new format ensuring the same semantical property and being less demanding than the original one. We demonstrate that this can be done by applying such an idea to a well established rule format ensuring the property of congruence for the rooted branching bisimulation equivalence.

1 Introduction

Structural operational semantics [1] (SOS) is a standard framework to provide process description languages with a semantics. The abstract syntax of a language is given through a *signature*, namely a set of operators together with their arity. The semantics is given through a *labeled transition system* (LTS), namely a set of states that represent processes and that are terms over the signature, together with a set of transitions between states describing computational steps. An LTS is defined by means of a *transition system specification* (TSS), namely a set of *transition rules* of the form $\frac{\text{premises}}{\text{conclusion}}$, which, intuitively, permit to derive transitions between processes from transitions between other processes.

To abstract away from information carried by an LTS that may be considered irrelevant in a given application context, several notions of *behavioral equivalence* were defined (see [2, 3]). Some of these equivalences, like *weak bisimulation* [4] and *branching bisimulation* [5], equate LTS states that incorporate so called *silent steps* representing internal moves by processes that are not observable by the external environment, and are referred to as *weak equivalences*. To ensure compositional modelling and verification, it is crucial that a given behavioral equivalence be a *congruence* w.r.t. all operators of the signature.

Since [6] a *transition rule format* is a set of syntactical constraints on the rules of the TSS, aiming to ensure a given property of the LTS. Several rule formats were developed to ensure the property of congruence for a given behavioral equivalence (see [7] for a survey). The original formats for weak equivalences were proposed in [8]. At present, the standard formats are those in [9–11].

Sometimes the syntactical constraints imposed by the formats on the premises of the transition rules are quite strict. For instance, the rule formats in [9–11]

prohibit the features of *lookahead*, namely the ability to test for two consecutive moves by a process, and *double testing* for running processes, namely the ability to test for two different moves by a process. Our idea is that in some cases it is worth to relax the constraints on the single rules, at the price of introducing some constraints on the form of the whole set of transition rules in the TSS, provided these are not too heavy. To demonstrate how this can be done, we consider the format of [9] for rooted branching bisimulation, we relax the constraints of this format by admitting both lookahead and double testing, and we add the reasonable constraint that lookahead and double testing come together with the ability of testing for an arbitrary number of silent steps, which means introducing a constraint on the set of rules of the TSS since a single rule is required for each number of silent steps. A natural extension of our work is to apply the same strategy to the formats of [10].

The paper is organized as follows: in Section 2, we recall some base notions on SOS, in Section 3 we recall the format of [9], in Section 4 we present our congruence format for rooted branching bisimulation and we end with some conclusions and discussion of related work in Section 5.

2 Preliminaries

In this section we recall some definitions that are standard in the SOS framework.

As usual, we assume that the abstract syntax of a process description language is given by a *signature*, namely a structure $\Sigma = (F, \mathsf{r})$, where (i) F is a set of *function names*, also called *language operators*, and (ii) $\mathsf{r} : F \to \mathbb{N}$ is a *rank function*, which gives the arity of a function name. An operator $f \in F$ is called a *constant* if $\mathsf{r}(f) = 0$. We also assume a set of (process) variables $\mathcal{V}$ disjoint from F, and let x, y, z range over $\mathcal{V}$. Let $W \subseteq \mathcal{V}$ be a set of variables. The set of Σ-terms over W, notation $T(\Sigma, W)$, is the least set satisfying: (i) $W \subseteq T(\Sigma, W)$, and (ii) if $f \in F$ and $t_1, \cdots, t_{\mathsf{r}(f)} \in T(\Sigma, W)$, then $f(t_1, \cdots, t_{\mathsf{r}(f)}) \in T(\Sigma, W)$. $T(\Sigma, \emptyset)$ is the set of all *closed terms*, also called *processes*, and abbreviated as $\mathcal{T}(\Sigma)$. $T(\Sigma, \mathcal{V})$ is the set of all *open terms* and abbreviated as $\mathbb{T}(\Sigma)$. By $\equiv$ we denote the syntactical equality relation between terms. Finally, $Var(t) \subseteq \mathcal{V}$ denotes the set of variables in term t, namely $Var(x) = \{x\}$ and $Var(f(t_1, \ldots, t_{\mathsf{r}(f)})) = \bigcup_{i=1}^{\mathsf{r}(f)} Var(t_i)$.

In SOS framework, the semantic model is that of LTSs.

Definition 1 (Labeled Transition System). *A* Labeled Transition System (LTS) *is a triple $(\mathcal{T}(\Sigma), \mathcal{A}, \rightarrow)$, where i) Σ is a signature; ii) $\mathcal{A}$ is a countable set of actions; and iii) $\rightarrow \subseteq \mathcal{T}(\Sigma) \times \mathcal{A} \times \mathcal{T}(\Sigma)$ is a transition relation.*

Following standard notation, we write $t \xrightarrow{a} t'$ for $(t, a, t') \in \rightarrow$. This represents a computation step of kind a taking process t to process t'.

LTSs are built by means of transition systems specifications, namely sets of transition rules of the form $\frac{\text{premises}}{\text{conclusion}}$. Here we assume that these rules are in the *ntyft*-format [12]. This choice is reasonable since ntyft-format is very general and for transition system specifications that are complete (see Def. 5 below)

it guarantees that bisimilarity equivalence relation is a congruence w.r.t. all operators in F.

Definition 2 (*ntyft*-rule, [12]). *A* ntyft-rule *is of the form*

$$\frac{\{t_j \xrightarrow{a_j} y_j \mid j \in J\} \qquad \{t_k \xrightarrow{b_k} \not\rightarrow \mid k \in K\}}{f(x_1, \ldots, x_{\mathsf{r}(f)}) \xrightarrow{a} t}$$

with J, K at most countable sets of indexes, $t_j, t_k, t \in \mathbb{T}(\Sigma)$, $a_j, b_k, a \in \mathcal{A}$, $y_j \in \mathcal{V}$, $f \in F$, $x_1, \ldots, x_{\mathsf{r}(f)} \in \mathcal{V}$, such that:

- *the $x_1, \ldots, x_{r(f)}$ and the y_j for $j \in J$ are all distinct variables.*

The expressions $t_j \xrightarrow{a_j} y_j$ (resp. $t_k \xrightarrow{b_k} \not\rightarrow$) above the line are called *positive* (resp. *negative*) *premises*. Given a rule ρ, we denote the set of positive (resp. negative) premises by $\mathrm{pprem}(\rho)$ (resp. $\mathrm{nprem}(\rho)$), and the set of all premises by $\mathrm{prem}(\rho) = \mathrm{pprem}(\rho) \cup \mathrm{nprem}(\rho)$. The expression $f(x_1, \ldots, x_{\mathsf{r}(f)}) \xrightarrow{a} t$ below the line is called *conclusion*, notation $\mathrm{conc}(\rho)$, where $f(x_1, \ldots, x_{\mathsf{r}(f)})$ is called the *source* of ρ, notation $\mathrm{src}(\rho)$, the x_i are the *source variables* denoted by $x_i \in \mathrm{src}(\rho)$, and term t is the *target* of ρ, notation $\mathrm{trgt}(\rho)$. We denote the set of variables in ρ by $\mathit{Var}(\rho)$, *free variables* by $\mathrm{free}(\rho) = \mathit{Var}(\rho) \setminus (\{x_1, \ldots, x_{\mathsf{r}(f)}\} \cup \{y_j \mid j \in J\})$, and *bound variables* by $\mathrm{bound}(\rho) = \mathit{Var}(\rho) \setminus \mathrm{free}(\rho)$.

Definition 3 (*ntyft*-TSS, [12]). *A* ntyft-transition system specification*, ntyft-TSS for short, is a set of ntyft-rules.*

Assigning an LTS to a TSS having rules with negative premises is not trivial. See [7] for a deep discussion. Let us describe the approach we adopt here, namely that of *least three-valued stable model*, introduced in [13] in logic programming.

An expression $t \xrightarrow{a} t'$ (resp. $t \xrightarrow{a} \not\rightarrow$) is called a *positive* (resp. *negative*) *literal* where $t, t' \in \mathbb{T}(\Sigma)$ and $a \in \mathcal{A}$. So, premises and conclusions in rules are literals.

A *substitution* is a mapping $\sigma_{\mathcal{V}} : \mathcal{V} \to \mathbb{T}(\Sigma)$. A substitution is *closed* if it maps each variable to a closed term in $\mathcal{T}(\Sigma)$. A substitution extends to terms by $\sigma_{\mathcal{V}}(f(t_1, \ldots, t_{\mathsf{r}(f)})) = f(\sigma_{\mathcal{V}}(t_1), \ldots, \sigma_{\mathcal{V}}(t_{\mathsf{r}(f)}))$, to literals by $\sigma_{\mathcal{V}}(t \xrightarrow{a} t') = \sigma_{\mathcal{V}}(t) \xrightarrow{a} \sigma_{\mathcal{V}}(t')$ and $\sigma_{\mathcal{V}}(t \xrightarrow{b} \not\rightarrow) = \sigma_{\mathcal{V}}(t) \xrightarrow{b} \not\rightarrow$, and to *ntyft*-rules by $\sigma_{\mathcal{V}}(\rho) = \frac{\bigcup_{\pi \in \mathrm{prem}(\rho)} \sigma_{\mathcal{V}}(\pi)}{\sigma_{\mathcal{V}}(\mathrm{conc}(\rho))}$. A closed substitution instance of a *ntyft*-rule is called a *closed ntyft-rule*. We denote with $\frac{H}{\pi}$ a closed *ntyft*-rule, and with $\frac{N}{\pi}$ a closed *ntyft*-rule having only negative premises (i.e. all elements in N are negative literals).

Given a set of closed positive literals P, a collection of closed negative literals N *holds* for P, denoted $P \models N$, iff for each $t \xrightarrow{b} \not\rightarrow \in N$ we have that $t \xrightarrow{b} t' \notin P$ for any $t' \in \mathcal{T}(\Sigma)$.

Definition 4 (Proof of a closed transition rule). *A* proof *from a TSS T of a closed transition rule $\frac{H}{\pi}$ is an upwardly branching tree in which all upwardly paths are finite, and the nodes are labeled by closed literals such that:*

- *the root is labeled by* π*;*
- *if* K *is the set of the labels of the nodes directly above a node labeled* l*, then:*
  - *either* $K = \emptyset$ *and* $l \in H$*;*
  - *or* $\frac{K}{l}$ *is a closed substitution instance of a transition rule in* T*.*

Given a TSS T, we consider a partitioning of the collection of positive literals to three disjoint sets: i) the set C of positive literals that are *certainly true*; ii) the set U of positive literals for which it is *unknown* whether or not they are true; and iii) the set of remaining literals that are *false*. Such a partitioning, determined by C and U, constitutes a three-valued stable model, denoted $\langle C, U\rangle$, for T if:

- a positive transition π is in C if and only if T proves a closed transition rule $\frac{N}{\pi}$, where N contains only negative literals and $C \cup U \models N$;
- a positive transition π is in $C \cup U$ if and only if T proves a closed transition rule $\frac{N}{\pi}$, where N contains only negative literals and $C \models N$.

Each TSS T allows an (information-)least three-valued stable model $\langle C, U\rangle$, in the sense that the set U is maximal. In [14] *two-valued stable models* were studied, which are three-valued stable models for which the set of unknown positive literals is empty.

Definition 5 (Complete TSS, [15]). *A TSS is* complete *if its least three-valued stable model is a two-valued stable model.*

If a TSS is complete, then it allows only one three-valued stable model, which is taken as the LTS built from the TSS. Only complete TSSs are considered to be meaningful. Notice that a TSS that does not contain transition rules with negative premises is complete for sure.

3 Rooted Branching Bisimulation as a Congruence

Behavioral equivalence relations over processes are usually defined to abstract away information provided by an LTS which is not considered to be relevant for a given application context. Here we consider branching bisimulation, one of those that identify LTS states that incorporate so called silent steps.

In the following we assume that $\mathcal{A}$ contains the special *silent action* τ. The reflexive and transitive closure of relation $\xrightarrow{\tau}$ is denoted with $\xrightarrow{\varepsilon}$. Finally, let us introduce notation $t \xrightarrow{\varepsilon}_n t'$ for $n \in \mathbb{N}$: we have $t \xrightarrow{\varepsilon}_0 t'$ if $t \equiv t'$ and $t \xrightarrow{\varepsilon}_{n+1} t'$ if $t \xrightarrow{\tau} t''$ and $t'' \xrightarrow{\varepsilon}_n t'$ for some $t'' \in \mathcal{T}(\Sigma)$. Hence, $\xrightarrow{\varepsilon} = \bigcup_{n\in\mathbb{N}} \xrightarrow{\varepsilon}_n$.

Definition 6 (Branching bisimulation, [5]). *Take a three-valued stable model* $\langle C, U\rangle$*. A symmetric relation* $\mathcal{B}$ *over* $\mathcal{T}(\Sigma)$ *is a* branching bisimulation *with respect to* C *if whenever* $s\,\mathcal{B}\,t$ *and* $s \xrightarrow{a} s' \in C$ *we have:*

- *either* $a = \tau$ *and* $s'\,\mathcal{B}\,t$*;*
- *or* $t \xrightarrow{\varepsilon} t'', t'' \xrightarrow{a} t' \in C$ *for* $t', t'' \in \mathcal{T}(\Sigma)$ *such that* $s\,\mathcal{B}\,t''$ *and* $s'\,\mathcal{B}\,t'$*.*

We call s and t *branching bisimilar* if there exists a branching bisimulation relation $\mathcal{B}$ such that $s\,\mathcal{B}\,t$. The union of all branching bisimulations over $\mathcal{T}(\Sigma)$ is the greatest branching bisimulation over $\mathcal{T}(\Sigma)$, it is called *branching bisimilarity* and it is denoted with $\leftrightarrows_{bb}$. Branching bisimilarity is an equivalence relation [16].

A crucial property of process description languages to ensure compositional modelling and verification is the compatibility of process operators with the behavioral relation chosen for the application context. In algebraic terms the compatibility of a behavioral equivalence R with operator $f \in F$ is a congruence.

Definition 7 (Congruence). *An equivalence relation R over $\mathcal{T}(\Sigma)$ is a* congruence *if for all $f \in F$, $f(s_1, \ldots, s_{\mathsf{r}(f)})\,R\,f(t_1, \ldots, t_{\mathsf{r}(f)})$ whenever $s_i\,R\,t_i$ for $i = 1, \ldots, \mathsf{r}(f)$.*

Branching bisimulation is not a congruence for the nondeterministic choice operator + defined by rules $\frac{x_1 \xrightarrow{a} y_1}{x_1+x_2 \xrightarrow{a} y_1}$ and $\frac{x_2 \xrightarrow{a} y_2}{x_1+x_2 \xrightarrow{a} y_2}$, which is offered by most of process description languages in the literature. To remedy to this problem the rootedness condition is usually assumed.

Definition 8 (Rooted branching bisimulation). *Take a three-valued stable model $\langle C, U\rangle$. A symmetric relation $\mathcal{R}$ over $\mathcal{T}(\Sigma)$ is a* rooted branching bisimulation *with respect to C if whenever $s\,\mathcal{R}\,t$ and $s \xrightarrow{a} s' \in C$ we have $t \xrightarrow{a} t' \in C$ for t' such that $s' \leftrightarrows_{bb} t'$.*

We call s and t *rooted branching bisimilar* if there exists a rooted branching bisimulation relation $\mathcal{R}$ such that $s\,\mathcal{R}\,t$. The union of all rooted branching bisimulations over $\mathcal{T}(\Sigma)$ is the greatest rooted branching bisimulation over $\mathcal{T}(\Sigma)$, it is called *rooted branching bisimilarity* and it is denoted with $\leftrightarrows_{rb}$. Rooted branching bisimilarity is clearly an equivalence relation.

In the following we recall the rule format RBB-safe [9], which ensures that rooted branching bisimulation is a congruence for all operators in the TSS.

A *patience rule* for the i-th argument of a function symbol $f \in F$ is a *ntyft*-rule of the form

$$\frac{x_i \xrightarrow{\tau} y_i}{f(x_1, \ldots, x_{\mathsf{r}(f)}) \xrightarrow{\tau} f(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots, x_{\mathsf{r}(f)})}$$

Following [9], we assume that each argument of each function symbol $f \in F$ is labeled either *tame* or *wild*. A *context*, denoted with $C[\,]$, is an open term in $\mathbb{T}(\Sigma)$ with one occurrence of the *context symbol* $[\,]$.

Definition 9 (w-nested context, [9]). *The collection of* w-nested contexts *is defined inductively by:*

- $[\,]$ *is w-nested;*
- *if $C[\,]$ is w-nested, and argument i of function symbol f is wild, then also $f(t_1, \ldots, t_{i-1}, C[\,], t_{i+1}, \ldots, t_{\mathsf{r}(f)})$ is w-nested.*

Definition 10 (RBB safe TSS, [9]). *A TSS T is called* RBB safe*, with respect to a tame/wild labeling of arguments of function symbols in F, if each of its transition rules is*

1. *either a patience rule for a wild argument of a function symbol,*
2. *or a ntyft-rule ρ with source $f(x_1, \ldots, x_{r(f)})$ and right-hand sides of positive premises $\{y_j \mid j \in J\}$, such that the following requirements are fulfilled:*
   (a) *Variables y_j for $j \in J$ do not occur in left-hand sides of premises of ρ;*
   (b) *If argument i of f is wild and does not have a patience rule in T, then x_i does not occur in left-hand sides of premises of ρ;*
   (c) *If argument i of f is wild and has a patience rule in T, then x_i occurs in the left-hand side of no more than one premise of ρ, where this premise*
      i. is positive,
      ii. does not contain the relation $\xrightarrow{\tau}$, and
      iii. has left-hand side x_i;
   (d) *Variables y_j for $j \in J$ and variables x_i for i a wild argument of f may only occur at w-nested positions in the target of ρ.*

Theorem 1 (Rooted branching bisimulation as a congruence, [9]). *If a complete TSS is RBB safe, then the rooted branching bisimulation equivalence that it induces is a congruence.*

In [9] several counter-examples are given to show that the syntactic constraints of Def. 10 cannot be relaxed in any trivial way. Our aim is to show that the constraints that prohibit *lookahead* (constraint 2a) and *double testing* for wild arguments of operators (constraint 2c) on the single rules of the TSS can be relaxed, provided that suitable and reasonable (i.e non too-demanding) constraints on the whole set of rules of the TSS are introduced. In the following we assume a TSS T containing the CCS-like sequencing operator $\cdot$ defined by the rules $\frac{}{a \cdot x \xrightarrow{a} x}$ for $a \in \mathcal{A}$, the CCS-like nondeterministic choice operator $+$ recalled above, the idle process 0 and the unary operators f_1, f_2 defined below.

Constraint 2a in Def. 10 prohibits *lookahead*, namely the ability of testing for two (or more) subsequent moves by a source argument. An example of rule violating this constraint is the following rule ρ_{f_1} for operator f_1:

$$\frac{x_1 \xrightarrow{a} y_1 \quad y_1 \xrightarrow{b} y_2}{f_1(x_1) \xrightarrow{a} 0}$$

Let us consider processes $s \equiv a \cdot b \cdot 0$ and $t \equiv a \cdot \tau \cdot b \cdot 0$. We have $s \leftrightarrows_{rb} t$. However, we have $f_1(s) \xrightarrow{a} 0$ while $f_1(t) \not\xrightarrow{a}$. Thus $f_1(s) \not\leftrightarrows_{rb} f_1(t)$. In this example the role of the silent action τ in the definition of t is crucial. On one side, the capability of performing τ is not discriminating in the evaluation of branching bisimulation equivalence of processes (see Def. 6). Hence, processes $b \cdot 0$ and $\tau \cdot b \cdot 0$, which are reached through action a by s and t, respectively, are branching bisimilar, thus implying that $s \leftrightarrows_{rb} t$. On the other hand, action τ becomes relevant when we focus on exact process evolution sequences. While process s can immediately perform b after a, process t cannot, namely after performing a it has to do the

action τ to reach a state in which it is able to perform b, and these two different evolutions are discriminated by the premises of ρ_{f_1}. Our proposal is to permit testing for an a move followed by a b move, provided that this comes together with the testing for an arbitrary number of τ-moves between these two moves labeled a and b. This means admitting the following set of rules in the TSS, provided we introduce the constraint that the TSS contains all of them:

$$\left\{ \frac{x_1 \xrightarrow{a} y_1 \qquad y_1 \xrightarrow{\epsilon}_n y_2 \qquad y_2 \xrightarrow{b} y_3}{f_1(x_1) \xrightarrow{a} y_3} \mid n \in \mathbb{N} \right\}$$

Let τ^n denote the sequence $\tau \cdot \ldots \cdot \tau$ of n actions τ, and $s_n \equiv a \cdot \tau^n \cdot b \cdot 0$ (notice $s \equiv s_0$ and $t \equiv s_1$). We have that $f_1(s_n) \xrightarrow{a} 0$ for all $n \in \mathbb{N}$, thus implying that $f_1(s_m) \leftrightarrows_{rb} f_1(s_n)$ for all $m, n \in \mathbb{N}$.

Constraint 2c in Def. 10 prohibits *double testing*, namely the ability of testing for two (or more) moves by a source argument, for arguments labeled as wild. An example of rule violating this constraint is the second of the rules below:

$$\frac{x_1 \xrightarrow{a} y_1}{f_2(x_1) \xrightarrow{a} f_2(y_1)}\ a \in \mathcal{A} \qquad \frac{x_1 \xrightarrow{a} y_1 \quad x_1 \xrightarrow{b} y_2}{f_2(x_1) \xrightarrow{c} 0}$$

where the argument of f_2 has to be wild due to constraint 2d of Def. 10 applied to the first rule. Let us take processes $s \equiv a \cdot (a \cdot 0 + b \cdot 0)$ and $t \equiv a \cdot t'$, with t' defined with the classical recursive construct as $t' \equiv a \cdot 0 + \tau \cdot (b \cdot 0 + \tau \cdot t')$. We have $s \leftrightarrows_{rb} t$. However, we have $f_2(s) \xrightarrow{a} f_2(a \cdot 0 + b \cdot 0) \xrightarrow{c} 0$ while $f_2(t) \xrightarrow{a} f_2(t')$ and neither $f_2(t')$ nor any process reachable from $f_2(t')$ through any sequence of τ-moves can make any c move. Thus $f_2(s) \not\leftrightarrows_{rb} f_2(t)$. Here the processes $a \cdot 0 + b \cdot 0$ and t', which are reached through action a by s and t, respectively, are branching bisimilar. Their difference, sensed by the second rule for f_2, is that $a \cdot 0 + b \cdot 0$ can perform both a and b, whereas t' is not able to reach (through any sequence of τ moves) any state in which both a and b are enabled, despite it can reach through τ actions a state where a is enabled and another state where b is enabled. Our proposal is to permit double testing for moves a and b, provided that these moves may follow an arbitrary number of τ steps. This means admitting the following set of rules in the TSS, provided we add the constraint that the TSS contains all of them:

$$\left\{ \frac{x_1 \xrightarrow{\epsilon}_m y_1 \quad y_1 \xrightarrow{a} y_2 \quad x_1 \xrightarrow{\epsilon}_n y_3 \quad y_3 \xrightarrow{b} y_4}{f_2(x_1) \xrightarrow{c} 0} \mid m, n \in \mathbb{N} \right\}$$

Notice that with these rules we get both $f_2(a \cdot 0 + b \cdot 0) \xrightarrow{c} 0$ and $f_2(t') \xrightarrow{c} 0$, thus implying $f_2(s) \leftrightarrows_{rb} f_2(t)$.

4 Congruence Format for Rooted Branching Bisimulation

As discussed in the previous section, lookahead and double testing can be admitted in the RBB safe format of [9], provided that sets of rules testing for sequences

of τ moves of different length are all introduced in the TSS. Below we introduce the notion of meta transition rule, which denotes a set of transition rules that test for the ability of performing sequences of τ moves of all possible lengths.

Definition 11 (Positive meta premise). *A* positive meta premise *is an expression of the form*

$$t \Longrightarrow \xrightarrow{a} y$$

The meta premise $t \Longrightarrow \xrightarrow{a} y$ represents the set

$$[\![t \Longrightarrow \xrightarrow{a} y]\!] := \left\{ \{ t \xrightarrow{\varepsilon}_n y' \quad y' \xrightarrow{a} y \} \mid n \in \mathbb{N} \right\}$$

of countable many sets of premises. Intuitively, $t \Longrightarrow \xrightarrow{a} y$ holds if there exists an $n \in \mathbb{N}$ and a substitution $\sigma_{\mathcal{V}}$ such that $\sigma_{\mathcal{V}}(t)$ can reach a state able to perform the action a through a sequence of n τ-actions.

Definition 12 (Meta transition rule). *A* meta transition rule*, notation $\tilde{\rho}$, is of the form*

$$\frac{\{t_j \xrightarrow{a_j} y_j \mid j \in J\} \quad \{t_k \xrightarrow{b_k} \not\rightarrow \mid k \in K\} \quad \{z_l \Longrightarrow \xrightarrow{a_l} y_l \mid l \in L\}}{f(x_1, \ldots, x_{\mathrm{r}(f)}) \xrightarrow{a} t}$$

with J, K, L at most countable sets of indexes, $t_j, t_k, t \in \mathbb{T}(\Sigma)$, $a_j, b_k, a_l, a \in \mathcal{A}$, $y_j, z_l, y_l \in \mathcal{V}$, $f \in F$, $x_1, \ldots, x_{\mathrm{r}(f)} \in \mathcal{V}$, such that:

- *the $x_1, \ldots, x_{\mathrm{r}(f)}$, the y_j for $j \in J$ and the y_l for $l \in L$ are all distinct variables.*

A meta transition rule $\tilde{\rho}$ like in Def. 12 represents the set $[\![\tilde{\rho}]\!]$ of all the transition rules of the form

$$\frac{\{t_j \xrightarrow{a_j} y_j \mid j \in J\} \quad \{t_k \xrightarrow{b_k} \not\rightarrow \mid k \in K\} \quad \{\mu_l \mid l \in L\}}{f(x_1, \ldots, x_{\mathrm{r}(f)}) \xrightarrow{a} t}$$

such that $\mu_l \in [\![z_l \Longrightarrow \xrightarrow{a_l} y_l]\!]$.

Definition 13 (Meta TSS). *A* meta TSS *is a set of meta transition rules.*

The meta TSS T represents the TSS $[\![T]\!] = \cup_{\tilde{\rho} \in T} [\![\tilde{\rho}]\!]$. Clearly, $[\![T]\!]$ is a *ntyft*-TSS. So all definitions of Section 2 directly lift to meta TSSs.

Now, we are able to extend the RBB safe format of [9] with lookahead and double testing for running processes.

Definition 14 (Meta RBB safe TSS). *A meta TSS T is called* meta RBB safe*, with respect to a tame/wild labeling of arguments of function symbols in F, if each of its transition rules is*

1. *either a patience rule for a wild argument of a function symbol;*

2. or a meta transition rule $\tilde{\rho}$ of the form

$$\frac{\{t_j \xrightarrow{a_j} y_j \mid j \in J\} \quad \{t_k \xrightarrow{b_k} \not\rightarrow \mid k \in K\} \quad \{z_l \Longrightarrow \xrightarrow{a_l} y_l \mid l \in L\}}{f(x_1, \ldots, x_{\mathsf{r}(f)}) \xrightarrow{a} t}$$

with constraints:

(a) actions a_l, for $l \in L$, are in $\mathcal{A} \setminus \{\tau\}$, namely they can not be action τ;
(b) if $x_i \in [\bigcup_{j\in J} var(t_j) \cup \bigcup_{k\in K} var(t_k)]$ for $i = 1, \ldots, \mathsf{r}(f)$, then i is a tame argument for f;
(c) no y_j for $j \in J$ and y_l for $l \in L$ occurs in $[\bigcup_{j\in J} var(t_j) \cup \bigcup_{k\in K} var(t_k)]$;
(d) variables x_i for i wild argument of f, y_j for $j \in J$ and y_l for $l \in L$ may occur only at w-nested positions in the target t.

Notice that Def. 14 admits lookahead, since for $l \in L$ we may have that $z_l \equiv y_j$ for some $j \in J$ or $z_l \equiv y_{l'}$ for some $l' \in L$. Double testing for a wild argument i of an operation $f \in F$ is admitted since we may have $z_l \equiv z_{l'} \equiv x_i$ for $l, l' \in L$.

Let us remark that meta rules have been already used in [10], called GSOS rules with lookahead, with the purpose of observing a partial form of lookahead, namely a sequence of τ-moves followed by a non silent move.

Notice that in Def. 14 we do not need the constraint 2b of Def. 10, which imposes that testing for a move by a wild argument for an operator f requires that there is a patience rule for it. To explain the reason, let us take the operators

$$\frac{x_1 \xrightarrow{a} y_1}{f(x_1) \xrightarrow{a} g(y_1)} \qquad \frac{x_1 \xrightarrow{b} y_1}{g(x_1) \xrightarrow{b} g(y_1)}$$

that do not respect Def. 10 since the patience rule for the argument of g is missing, and processes $a \cdot b \cdot 0$ and $a \cdot \tau \cdot b \cdot 0$. We have $a \cdot b \cdot 0 \leftrightarrows_{rb} a \cdot \tau \cdot b \cdot 0$ but $f(a \cdot b \cdot 0) \not\leftrightarrows_{rb} f(a \cdot \tau \cdot b \cdot 0)$ since $f(a \cdot b \cdot 0) \xrightarrow{a} g(b \cdot 0) \xrightarrow{b} 0$ whereby $f(a \cdot \tau \cdot b \cdot 0) \xrightarrow{a} g(\tau \cdot b \cdot 0) \xrightarrow{b} \not\rightarrow$. Definition 10 requires the patience rule for the argument of g, so $g(\tau \cdot b \cdot 0) \xrightarrow{\tau} g(b \cdot 0) \xrightarrow{b} 0$ and, therefore, $f(a \cdot b \cdot 0) \leftrightarrows_{rb} f(a \cdot \tau \cdot b \cdot 0)$. By adopting the meta rules as in Def. 14, we can write

$$\frac{x_1 \xrightarrow{a} y_1}{f(x_1) \xrightarrow{a} g(y_1)} \qquad \frac{x_1 \Longrightarrow \xrightarrow{b} y_1}{g(x_1) \xrightarrow{b} g(y_1)}$$

and the patience rule for the argument of g is no more needed, since we have $f(a \cdot \tau \cdot b \cdot 0) \xrightarrow{a} g(\tau \cdot b \cdot 0) \xrightarrow{b} 0$ and, thus, $f(a \cdot b \cdot 0) \leftrightarrows_{rb} f(a \cdot \tau \cdot b \cdot 0)$.

Let us argue that all constraints in Def. 14 cannot be relaxed in any trivial way. Firstly, let us show why, as in [9], some arguments of functions deserve a special treatment. These arguments are labeled as wild. The special treatment consists in constraints 2b and 2d in Def. 14.

Example 1. Let us consider the transition rules

$$\frac{x_1 \xrightarrow{a} y_1}{f(x_1) \xrightarrow{a} y_1} \qquad \frac{x_1 \xrightarrow{a} y_1}{g(x_1) \xrightarrow{a} f(y_1)}$$

and processes $a.\tau.a.0$ and $a.a.0$. We have $a.\tau.a.0 \leftrightarrows_{rb} a.a.0$. However, we have $g(a.\tau.a.0) \not\leftrightarrows_{rb} g(a.a.0)$. In fact, $g(a.\tau.a.0) \xrightarrow{a} f(\tau.a.0)$ and $g(a.a.0) \xrightarrow{a} f(a.0)$, where $f(\tau.a.0) \not\leftrightarrows_{bb} f(a.0)$ since $f(\tau.a.0) \not\xrightarrow{a}$ and $f(a.0) \xrightarrow{a} 0$. The rule for g has $f(y_1)$ as target, where y_1 occurs in the target of the premise $x_1 \xrightarrow{a} y_1$. This implies that it may happen that when the argument x_1 of g is instantiated by two processes p and p' with $p \leftrightarrows_{rb} p'$, we have that the argument y_1 of f is instantiated by two a-derivatives, q and q' respectively, such that $q \leftrightarrows_{bb} q'$ but $q \not\leftrightarrows_{rb} q'$. Arguments of operators that may be instantiated with processes related by $\leftrightarrows_{bb}$ but not by $\leftrightarrows_{rb}$ are labeled wild. This is exactly what is required by constraint 2d in Def. 14. As required by constraint 2b in Def. 14, they cannot be tested by premises of the form $x \xrightarrow{a} y$ since these premises are able to discriminate them. ■

By next example, we show why meta premises cannot test for τ moves (constraint 2a, Def. 14).

Example 2. Let us consider the transition rules

$$\frac{x_1 \xrightarrow{a} y_1}{g(x_1) \xrightarrow{a} f(y_1)} \qquad \frac{x_1 \Longrightarrow\!\!\xrightarrow{\tau} y_1}{f(x_1) \xrightarrow{\tau} y_1}$$

We have $a.\tau.a.0 \leftrightarrows_{rb} a.a.0$. However, $g(a.\tau.a.0) \not\leftrightarrows_{rb} g(a.a.0)$. In fact we have $g(a.\tau.a.0) \xrightarrow{a} f(\tau.a.0) \xrightarrow{\tau} a.0 \xrightarrow{a} 0$, whereby $g(a.a.0) \xrightarrow{a} f(a.0) \not\xrightarrow{\tau}$. ■

By next example, we show why in meta premises we cannot have an arbitrary term in the left side, and we only allow variable z_l.

Example 3. Let us consider the transition rules

$$\frac{x_1 \xrightarrow{a} y_1}{g(x_1) \xrightarrow{a} y_1} \qquad \frac{g(x_1) \Longrightarrow\!\!\xrightarrow{a} y_1}{f(x_1) \xrightarrow{a} y_1} \qquad \frac{x_1 \xrightarrow{a} y_1}{h(x_1) \xrightarrow{a} f(y_1)}$$

We have $a.a.0 \leftrightarrows_{rb} a.\tau.a.0$. However, $h(a.a.0) \not\leftrightarrows_{rb} h(a.\tau.a.0)$. In fact we have $h(a.a.0) \xrightarrow{a} f(a.0) \xrightarrow{a} 0$, whereby $h(a.\tau.a.0) \xrightarrow{a} f(\tau.a.0) \not\xrightarrow{a}$. ■

By next example, we show why we cannot allow variables that are targets of premises or meta premises to be source of classic premises (constraint 2c, Def. 14).

Example 4. Let us consider the transition rules

$$\frac{x_1 \xrightarrow{a} y_1 \quad y_1 \xrightarrow{a} y_2}{f(x_1) \xrightarrow{a} y_2} \qquad \frac{x_1 \Longrightarrow\!\!\xrightarrow{a} y_1 \quad y_1 \xrightarrow{a} y_2}{g(x_1) \xrightarrow{a} y_2}$$

We have $a.\tau.a.0 \leftrightarrows_{rb} a.a.0$. However, we have that $f(a.\tau.a.0) \not\leftrightarrows_{rb} f(a.a.0)$ since $f(a.\tau.a.0) \not\xrightarrow{a}$ while $f(a.a.0) \xrightarrow{a} 0$. Analogously, $g(a.\tau.a.0) \not\leftrightarrows_{rb} g(a.a.0)$ since $g(a.\tau.a.0) \not\xrightarrow{a}$ while $g(a.a.0) \xrightarrow{a} 0$. ■

To prove the congruence result, we have to deal with well-founded rules.

Definition 15 (Well-foundedness). *Let H be a set of premises and meta premises. The* variable dependency graph *of H is a directed graph $G_H = (V, E)$ given by:*

- $V = \bigcup_{h \in H} Var(h)$;
- $E = \{\langle x, y\rangle \mid t \xrightarrow{a} y \in H \text{ and } x \in Var(t) \text{ or } x \Longrightarrow\!\!\xrightarrow{a} y \in H\}$.

We say that H is well-founded *if any backward chain of edges in G_H is finite. A meta transition rule $\tilde{\rho}$ is called* well-founded *if the set of all its premises and meta premises is well-founded. A meta TSS is called* well-founded *if all its meta transition rules are well-founded.*

Theorem 2. *If a complete and well-founded meta TSS T is meta RBB safe, then the rooted branching bisimulation equivalence that it induces is a congruence.*

5 Conclusions

We considered the format of [9], which ensures the congruence property for rooted branching bisimulation, we relaxed the constraints on the single rules by allowing both double testing for wild arguments of operators and lookahead, at the price of constraining these features to come together the testing for an arbitrary number of silent steps. We argued that this means introducing a constraint on the form of the whole set of rules. Our idea can be naturally extended to the formats in [10, 11].

An example of operator that is captured by our format and that is outside the formats in [9–11] is the *copying* operator, originally proposed in [17] for languages that do not consider silent actions, and defined in [11] by the following rules:

$$\frac{x_1 \xrightarrow{a} y_1}{cp(x_1) \xrightarrow{a} cp(y_1)} \quad a \in \mathcal{A} \qquad\qquad \frac{x_1 \xrightarrow{l} y_1 \quad x_1 \xrightarrow{r} y_1}{cp(x_1) \xrightarrow{s} cp(y_1) \parallel cp(y_2)}$$

where $l, r \in \mathcal{A}$ are the *left* and *right forking*, respectively, s is the *split* action, and $\parallel$ is the parallel composition operator. In [11] this operator is admitted in the format thanks to the *two-tiered* approach to SOS proposed in [10, 11]. The idea is to divide function symbols in F into two classes: *principal operators* and *abbreviations*, where an abbreviation can be obtained by grouping together the arguments of a principal operator. Proofs are given that abbreviations are syntactic sugar and do not have to obey the syntactic restrictions of a congruence format, provided they abbreviate principal operators that do so. This is an advantage since if a given equivalence is a congruence w.r.t. an operator $f \in F$ that is outside from a congruence format, one can find an operator f^* that is considered to be principal and abbreviated by f and that obeys the constraints of the format. In [11] an operator for which cp is an abbreviation is provided that is captured by the format. Here we do not need to search for such an abbreviation since cp is already in the format.

Acknowledgements We are grateful to Wan Fokkink for feedback on a preliminary version of this paper.

References

1. Plotkin, G.: A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University (1981)
2. van Glabbeek, R.J.: The linear time - branching time spectrum. In: CONCUR '90. Volume 458 of LNCS., Springer (1990) 278–297
3. van Glabbeek, R.J.: The linear time - branching time spectrum ii. In: CONCUR '93. Volume 715 of LNCS., Springer (1993) 66–81
4. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. J. Assoc. Comput. Mach. **32** (1985) 137–161
5. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. J. Assoc. Comput. Mach. **43**(3) (1996) 555–600
6. de Simone, R.: Higher-level synchronising devices in meije-sccs. Theoret. Comput. Sci. **37** (1985) 245–267
7. Aceto, L., Fokkink, W.J., Verhoef, C.: Structural operational semantics. In: Handbook of Process Algebra. Elsevier (2001) 197–292
8. Bloom, B.: Structural operational semantics for weak bisimulations. Theoret. Comput. Sci. **146** (1995) 25–68
9. Fokkink, W.J.: Rooted branching bisimulation as a congruence. J. Comput. Syst. Sci. **60**(1) (2000) 13–37
10. van Glabbeek, R.J.: On cool congruence formats for weak bisimulations. Theoret. Comput. Sci. **412**(28) (2011) 3283–3302
11. Fokkink, W.J., van Glabbeek, R.J., de Wind, P.W.: Divide and congruence: From decomposition of modal formulas to preservation of branching and η-bisimilarity. Inf. Comput **214** (2012) 59–85
12. Groote, J.F.: Transition system specifications with negative premises. Theoret. Comput. Sci. **118**(2) (1993) 263–299
13. Przymusinski, T.C.: The well-founded semantics coincides with the three-valued stable semantics. Fundam. Inform. **13**(4) (1990) 445–463
14. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: 5th Conference on Logic Programming, MIT Press (1988) 1070–1080
15. van Glabbeek, R.J.: The meaning of negative premises in transition system specifications ii. In: ICALP'96. Volume 1099 of LNCS., Springer (1996) 502–513
16. Basten, T.: Branching bisimilarity is an equivalence indeed! Inform. Proc. Lett. **58**(3) (1996) 141–147
17. Bloom, B., Istrail, S., Meyer, A.: Bisimulation can't be traced. J. Assoc. Comput. Mach. **42**(1) (1995) 232–268