Divide and Congruence: From Decomposition of Modalities to Preservation of Branching Bisimulation

Wan Fokkink[1,2], Rob van Glabbeek[3,4], and Paulien de Wind[1]

[1] Vrije Universiteit Amsterdam, Section Theoretical Computer Science, Amsterdam
[2] CWI, Department of Software Engineering, Amsterdam
[3] National ICT Australia, Sydney
[4] University of New South Wales, School of Computer Science and Engineering, Sydney
`{wanf,pdwind}@cs.vu.nl, rvg@cs.stanford.edu`

Abstract. We present a method for decomposing modal formulas for processes with the internal action τ. To decide whether a process algebra term satisfies a modal formula, one can check whether its subterms satisfy formulas that are obtained by decomposing the original formula. The decomposition uses the structural operational semantics that underlies the process algebra. We use this decomposition method to derive congruence formats for branching and rooted branching bisimulation equivalence.

1 Introduction

Structural operational semantics [20] provides process algebras and specification languages with an interpretation. It generates a labelled transition system, in which states are the closed terms over a (single-sorted, first-order) signature, and transitions between states may be supplied with labels. The transitions between states are obtained from a transition system specification, which consists of a set of proof rules called transition rules.

Labelled transition systems can be distinguished from each other by a wide range of behavioural equivalences, based on e.g. branching structure or decorated versions of execution sequences. Van Glabbeek [11] classified equivalences for processes that take into account the internal action τ. Here we focus on one such equivalence, called branching bisimulation [14].

In general a behavioural equivalence induced by a transition system specification is not a congruence, i.e. the equivalence class of a term $f(p_1, \ldots, p_n)$ need not be determined by the equivalence classes of its arguments $p_1, \ldots, p_n$. Being a congruence is an important property, for instance in order to fit the equivalence into an axiomatic framework. Syntactic formats for transition rules have been developed with respect to several behavioural equivalences, to ensure that such an equivalence is a congruence. These formats help to avoid repetitive congruence proofs. Several congruence formats were introduced for bisimulation, such as the De Simone format [21], the GSOS format [4], the tyft/tyxt format

[16], and the ntyft/ntyxt format [15]. Bloom [2] introduced congruence formats for weak and branching bisimulation and for rooted weak and branching bisimulation. These formats include so-called patience rules for arguments i of function symbols f, which imply that a term $f(p_1, \ldots, p_n)$ inherits the τ-transitions of its argument p_i. Furthermore, arguments of function symbols that contain running processes are marked, and this marking is used to restrict occurrences of variables in transition rules.

Behavioural equivalences can be characterised in terms of the observations that an experimenter could make during a session with a process. Modal logic captures such observations. A modal characterisation of an equivalence consists of a class C of modal formulas such that two processes are equivalent if and only if they make true the same formulas in C. For instance, Hennessy-Milner logic [17] is the modal characterisation of bisimulation.

Larsen and Liu [19] introduced a method for decomposing formulas from Hennessy-Milner logic for concrete processes, with respect to terms from a process algebra with a structural operational semantics in De Simone format. To decide whether a process algebra term satisfies a modal formula, one can check whether its subterms satisfy certain other formulas, obtained by decomposing the original formula. This method was extended by Bloom, Fokkink & van Glabbeek [3] to ntyft/ntyxt format without lookahead, and by Fokkink, van Glabbeek & de Wind to tyft/tyxt format in the full version of [9]. In [3], the decomposition method was applied to obtain congruence formats for a range of behavioural equivalences. The idea is that given an equivalence and its modal characterisation C, the congruence format for this equivalence must ensure that decomposing a formula in C always produces formulas in C.

Here we extend the work of [3] to processes with τ-transitions. We present a method for decomposing formulas from modal logic for processes with τ-transitions. In order minimise the complexity inherent in the combination of modal decomposition and the internal action τ, we apply the decomposition method to so-called abstraction-free TSSs, where only the patience rules contain the label τ in the conclusion. Furthermore, we use this decomposition method to obtain congruence formats for branching a rooted branching bisimulation. These formats include TSSs that are not abstraction-free, owing to the compositionality of the abstraction operator, which renames certain actions into τ. Our formats use two predicates on arguments of function symbols, to mark both running processes and processes that may have started running. Our congruence formats are more liberal than the simply BB and RBB cool formats from [2] and the RBB safe format from [7]. In Sect. 7 we will present a more in-depth comparison with congruence formats from [2, 7, 13].

In a companion paper [10], we derive congruence formats for η- and rooted η-bisimulation, with a reference to the current paper for the decomposition method. Thus we drive home the point that, in contrast to the ad hoc construction of congruence formats from the past, we can now systematically derive expressive congruence formats from the modal characterisations of behavioural equivalences.

2 Preliminaries

2.1 Equivalences on labelled transition systems

A *labelled transition system (LTS)* is a pair $(\mathbb{P}, \rightarrow)$ with $\mathbb{P}$ a set of *processes* and $\rightarrow\, \subseteq \mathbb{P} \times (A \cup \{\tau\}) \times \mathbb{P}$ where τ is an *internal action* and A a set of *actions* not containing τ. We use α, β, γ for elements of $A \cup \{\tau\}$ and a, b for elements of A. We write $p \xrightarrow{\alpha} q$ for $(p, \alpha, q) \in\, \rightarrow$ and $p \stackrel{\alpha}{\not\rightarrow}$ for $\neg\exists q \in \mathbb{P} : p \xrightarrow{\alpha} q$. Furthermore, $\stackrel{\epsilon}{\Longrightarrow}$ denotes the transitive-reflexive closure of $\xrightarrow{\tau}$.

Definition 1 ([14]). *A symmetric relation $B \subseteq \mathbb{P} \times \mathbb{P}$ is a* branching bisimulation *if pBq and $p \xrightarrow{\alpha} p'$ implies that either $\alpha = \tau$ and $p'\, B\, q$, or $q \stackrel{\epsilon}{\Longrightarrow} q' \xrightarrow{\alpha} q''$ for some q' and q'' with pBq' and $p'Bq''$.*

Processes p, q are branching bisimilar*, denoted by $p \leftrightarrow_b q$, if there exists a branching bisimulation B with pBq.*

Branching bisimulation is not a congruence with respect to most process algebras from the literature, meaning that the equivalence class of a term $f(p_1, \ldots, p_n)$ is not always determined by the equivalence classes of its arguments $p_1, \ldots, p_n$. A rootedness condition remedies this imperfection.

Definition 2 ([14]). *A symmetric relation $R \subseteq \mathbb{P} \times \mathbb{P}$ is a* rooted branching bisimulation *if pRq and $p \xrightarrow{\alpha} q'$ implies that $q \xrightarrow{\alpha} q'$ for some q' with $p' \leftrightarrow_b q'$.*

Processes p, q are rooted branching bisimilar*, denoted by $p \leftrightarrow_{rb} q$, if there exists a rooted branching bisimulation R with pRq.*

2.2 Modal logic

Modal logic aims to formulate properties of processes in an LTS. Following [11], we extend Hennessy-Milner logic [17] with the modal connectives $\langle\epsilon\rangle\varphi$ and $\langle\hat{\tau}\rangle\varphi$.

Definition 3. *The class $\mathbb{O}$ of* modal formulas *is defined as follows, where I ranges over all index sets:*

$$\mathbb{O} \qquad \varphi ::= \textstyle\bigwedge_{i\in I}\varphi_i \mid \neg\varphi \mid \langle\alpha\rangle\varphi \mid \langle\epsilon\rangle\varphi \mid \langle\hat{\tau}\rangle\varphi$$

$p \models \varphi$ denotes that p satisfies φ. By definition, $p \models \langle\alpha\rangle\varphi$ if $p \xrightarrow{\alpha} p'$ with $p' \models \varphi$, $p \models \langle\epsilon\rangle\varphi$ if $p \stackrel{\epsilon}{\Longrightarrow} p'$ with $p' \models \varphi$, and $p \models \langle\hat{\tau}\rangle\varphi$ if either $p \models \varphi$ or $p \xrightarrow{\tau} p'$ with $p' \models \varphi$. We use abbreviations $\top$ for the empty conjunction, $\varphi_1 \wedge \varphi_2$ for $\bigwedge_{i\in\{1,2\}} \varphi_i$, $\varphi\langle\alpha\rangle\varphi'$ for $\varphi \wedge \langle\alpha\rangle\varphi'$, and $\varphi\langle\hat{\tau}\rangle\varphi'$ for $\varphi \wedge \langle\hat{\tau}\rangle\varphi'$. We write $\varphi \equiv \varphi'$ if $p \models \varphi \Leftrightarrow p \models \varphi'$ for any process p in any LTS.

Definition 4. *The subclasses $\mathbb{O}_b$ and $\mathbb{O}_{rb}$ of $\mathbb{O}$ are defined as follows:*

$$\begin{array}{ll} \mathbb{O}_b & \varphi ::= \bigwedge_{i\in I}\varphi_i \mid \neg\varphi \mid \langle\epsilon\rangle(\varphi\langle\hat{\tau}\rangle\varphi') \mid \langle\epsilon\rangle(\varphi\langle a\rangle\varphi') \\ \mathbb{O}_{rb} & \varphi ::= \bigwedge_{i\in I}\varphi_i \mid \neg\varphi \mid \langle\alpha\rangle\hat{\varphi} \mid \hat{\varphi}\ (\hat{\varphi} \in \mathbb{O}_b) \end{array}$$

The classes $\mathbb{O}_b^{\equiv}$ and $\mathbb{O}_{rb}^{\equiv}$ are the closures of $\mathbb{O}_b$, respectively $\mathbb{O}_{rb}$, under $\equiv$.

The last clause in the definition of $\mathbb{O}_{rb}$ guarantees that $\mathbb{O}_b \subseteq \mathbb{O}_{rb}$, which will be needed in the proof of Prop. 4. If this clause were omitted, it would still follow that $\mathbb{O}_b^{\equiv} \subseteq \mathbb{O}_{rb}^{\equiv}$, using structural induction together with $\langle\epsilon\rangle\varphi \equiv \varphi \vee \langle\tau\rangle\langle\epsilon\rangle\varphi$ and $\langle\hat{\tau}\rangle\varphi \equiv \varphi \vee \langle\tau\rangle\varphi$. Note that if $\varphi \in \mathbb{O}_b^{\equiv}$, then $\langle\epsilon\rangle\varphi \equiv \langle\epsilon\rangle(\varphi\langle\hat{\tau}\rangle\varphi) \in \mathbb{O}_b^{\equiv}$.

For $L \subseteq \mathbb{O}$, we write $p \sim_L q$ if p and q satisfy the same formulas in L. Note that, trivially, $p \sim_{\mathbb{O}_b} q \Leftrightarrow p \sim_{\mathbb{O}_b^{\equiv}} q$ and $p \sim_{\mathbb{O}_{rb}} q \Leftrightarrow p \sim_{\mathbb{O}_{rb}^{\equiv}} q$.

Theorem 1. *$p \leftrightarroweq_b q \Leftrightarrow p \sim_{\mathbb{O}_b} q$ and $p \leftrightarroweq_{rb} q \Leftrightarrow p \sim_{\mathbb{O}_{rb}} q$, for all $p, q \in \mathbb{P}$.*

A proof of this theorem is presented in the appendix.

2.3 Structural operational semantics

Let V be an infinite set of variables, with typical elements x, y, z. A syntactic object is *closed* if it does not contain any variables. A *signature* is a set Σ of function symbols f with arity $ar(f)$. We always take $|\Sigma|, |A| \leq |V|$. The set $\mathbb{T}(\Sigma)$ of terms over Σ and V is defined as usual. t, u denote terms and p, q closed terms. $var(t)$ is the set of variables that occur in t. A substitution is a partial function from V to $\mathbb{T}(\Sigma)$. A closed substitution σ is a total function from V to closed terms.

Definition 5. *A (*positive *or* negative*)* literal *is an expression $t \xrightarrow{\alpha} t'$ or $t \overset{\alpha}{\not\longrightarrow}$. A* (transition) rule *is of the form $\frac{H}{t \xrightarrow{\alpha} t'}$ with H a set of literals called the* premises. *$t \xrightarrow{\alpha} t'$ is the* conclusion *and t the* source *of the rule. A rule $\frac{\emptyset}{t \xrightarrow{\alpha} t'}$ is also written $t \xrightarrow{\alpha} t'$. A* transition system specification (TSS) *is a set of transition rules.*

Definition 6. *Let $P = (\Sigma, R)$ be a TSS. An* irredundant proof *from P of a rule $\frac{H}{t \xrightarrow{\alpha} t'}$ is a well-founded tree with the nodes labelled by literals and some of the leaves marked "hypothesis", such that the root has label $t \xrightarrow{\alpha} t'$, H is the set of labels of the hypotheses, and if μ is the label of a node that is not a hypothesis and K is the set of labels of the children of this node, then μ is positive and $\frac{K}{\mu}$ is a substitution instance of a rule in R.*

The proof of $\frac{H}{t \xrightarrow{\alpha} t'}$ is called irredundant because H must equal (instead of include) the set of labels of the hypotheses. This irredundancy will be crucial for the preservation of our congruence formats in Sect. 4.1 (see Prop. 2).

A TSS is meant to specify an LTS in which the transitions are closed positive literals. A TSS with only positive premises specifies an LTS in a straightforward way, but it is not so easy to associate an LTS to a TSS with negative premises. From [12] we adopt the notion of a well-supported proof of a closed literal. Literals $t \xrightarrow{\alpha} t'$ and $t \overset{\alpha}{\not\longrightarrow}$ are said to *deny* each other.

Definition 7. *Let $P = (\Sigma, R)$ be a TSS. A* well-supported proof *from P of a closed literal μ is a well-founded tree with the nodes labelled by closed literals, such that the root is labelled by μ, and if ν is the label of a node and K is the set of labels of the children of this node, then:*

1. *either ν is positive and $\frac{K}{\nu}$ is a closed substitution instance of a rule in R;*
2. *or ν is negative and for each set N of closed negative literals with $\frac{N}{\kappa}$ irredundantly provable from P and κ a closed positive literal denying ν, a literal in K denies one in N.*

$P \vdash_{ws} \mu$ denotes that a well-supported proof from P of μ exists. P is complete *if for each p and α, either $P \vdash_{ws} p \overset{\alpha}{\not\longrightarrow}$ or $P \vdash_{ws} p \xrightarrow{\alpha} p'$ for some p'.*

A complete TSS specifies an LTS, consisting of the *ws*-provable closed positive literals.

2.4 Notions regarding transition rules

In this section we present terminology for syntactic restrictions on rules, originating from [3, 15, 16].

Definition 8. *An* ntytt rule *is a rule in which the right-hand sides of positive premises are variables that are all distinct, and that do not occur in the source. An ntytt rule is an* ntyxt rule *if its source is a variable, an* ntyft rule *if its source contains exactly one function symbol and no multiple occurrences of variables, and an* nxytt rule *if the left-hand sides of its premises are variables.*

Definition 9. *A variable in a rule is* free *if it occurs neither in the source nor in right-hand sides of premises. A rule has* lookahead *if some variable occurs in the right-hand side of a premise and in the left-hand side of a premise. A rule is* decent *if it has no lookahead and does not contain free variables.*

The ntyft/ntyxt and ready simulation formats [15, 3] were originally introduced to guarantee congruence for bisimulation and ready simulation.

Definition 10. *A TSS is in* ntyft/ntyxt format *if it consists of ntyft and ntyxt rules, and in* ready simulation format *if moreover its rules do not have lookahead.*

A predicate $\aleph$ marks arguments of function symbols that contain running processes (cf. [3]). Typically, in process algebra, $\aleph$ holds for the arguments of the merge $\|$, but not for the arguments of alternative composition $+$.

Definition 11. *Let $\aleph$ be a unary predicate on $\{(f,i) \mid 1 \leq i \leq ar(f),\ f \in \Sigma\}$. If $\aleph(f,i)$, then argument i of f is* liquid*; otherwise it is* frozen*. An occurrence of x in t is* at an $\aleph$-liquid position *(or* $\aleph$-liquid *for short), if either $t = x$, or $t = f(t_1, \ldots, t_{ar(f)})$ and the occurrence is at an $\aleph$-liquid position in t_i for a liquid argument i of f; otherwise the occurrence is at an* $\aleph$-frozen *position.*

A patience rule for an argument i of a function symbol f expresses that term $f(p_1, \ldots, p_n)$ inherits the τ-transitions of argument p_i (cf. [2, 7]). We will require the presence of patience rules for $\aleph$-liquid arguments.

Definition 12. *An ntyft rule is a* patience rule *for f if it is of the form*

$$\frac{x_i \xrightarrow{\tau} y}{f(x_1,\ldots,x_i,\ldots,x_{ar(f)}) \xrightarrow{\tau} f(x_1,\ldots,x_{i-1},y,x_{i+1}\ldots,x_{ar(f)})}$$

It is an $\aleph$-patience rule *if* $\aleph(f,i)$.

An ntytt rule is $\aleph$-*patient* if it is irredundantly provable from the $\aleph$-patience rules. Such rules have the form $\frac{t\xrightarrow{\tau}y}{C[t]\xrightarrow{\tau}C[y]}$ with $C[]$ an $\aleph$-liquid context, meaning that the context symbol $[]$ occurs at an $\aleph$-liquid position.

Definition 13. *A TSS is* abstraction-free *with respect to $\aleph$ if only $\aleph$-patience rules have a conclusion of the form $t \xrightarrow{\tau} u$.*

2.5 Ruloids

To decompose modal formulas, we use a result from [3], where for any TSS P in ready simulation format a collection of decent nxytt rules, called P-*ruloids*, is constructed. We explain this construction on a rather superficial level; the precise transformation can be found in [3].

First P is converted to a TSS in decent ntyft format. In this conversion from [16], free variables in a rule are replaced by closed terms, and if the source is of the form x then this variable is replaced by a term $f(x_1,\ldots,x_n)$ for each $f \in \Sigma$. Next, using a construction from [8], left-hand sides of positive premises are reduced to variables. Roughly the idea is, given a premise $f(t_1,\ldots,t_n) \xrightarrow{\alpha} y$ in a rule r, and a rule $\frac{H}{f(x_1,\ldots,x_n)\xrightarrow{\alpha}t}$, to transform r by replacing the aforementioned premise by H, y by t, and the x_i by the t_i; this is repeated (transfinitely) until all positive premises with a non-variable left-hand side have disappeared. In the final transformation step, rules with a *negative* conclusion $t \stackrel{\alpha}{\not\longrightarrow}$ are introduced. The motivation is that instead of the notion of well-founded provability in Def. 7, we want a more constructive notion like Def. 6, by making it possible that a negative premise is matched with a negative conclusion. A rule r with a conclusion $f(x_1,\ldots,x_n) \stackrel{\alpha}{\not\longrightarrow}$ is obtained by picking one premise from each rule with a conclusion $f(x_1,\ldots,x_n) \xrightarrow{\alpha} t$, and including the denial of each of the selected premises as a premise of r. For this last transformation it is essential that rules do not have lookahead.

The resulting TSS, which is in decent ntyft format, is denoted by P^+. The notion of irredundant provability is adapted in a straightforward fashion to accommodate rules with a negative conclusion. In [3] it is established that $P \vdash_{ws} \mu$ if and only if μ is irredundantly provable from P^+, for all closed literals μ. P-ruloids are those decent nxytt rules that are irredundantly provable from P^+. The following correspondence result from [3] between a TSS and its ruloids plays a crucial role in the decomposition method employed here. It says that there is a well-supported proof from P of a transition $p \xrightarrow{a} q$, with p a closed substitution instance of a term t, if and only if there is a proof of this transition that uses at the root a P-ruloid with source t.

Proposition 1 ([3]). *Let P be a TSS in ready simulation format. Then $P \vdash_{ws} \sigma(t) \xrightarrow{\alpha} p$ if and only if there are a P-ruloid $\frac{H}{t \xrightarrow{\alpha} u}$ and a σ' with $P \vdash_{ws} \sigma'(\mu)$ for $\mu \in H$, $\sigma'(t) = \sigma(t)$ and $\sigma'(u) = p$.*

It is not hard to see that the notion of abstraction-freeness is preserved by the transformation to P-ruloids.

Lemma 1. *If a TSS P is abstraction-free with respect to some $\aleph$, then all P-ruloids with a conclusion of the form $t \xrightarrow{\tau} u$ are $\aleph$-patient.*

3 Decomposition of Modal Formulas

In this section we show how one can decompose formulas from $\mathbb{O}$. To each term t and formula φ we assign a set $t^{-1}(\varphi)$ of *decomposition mappings* $\psi : V \to \mathbb{O}$. Each of these mappings $\psi \in t^{-1}(\varphi)$ guarantees that $\sigma(t) \models \varphi$ if $\sigma(x) \models \psi(x)$ for $x \in \mathit{var}(t)$. Vice versa, whenever $\sigma(t) \models \varphi$, there is a decomposition mapping $\psi \in t^{-1}(\varphi)$ with $\sigma(x) \models \psi(x)$ for $x \in \mathit{var}(t)$. This is formalised in Thm. 2.

In order minimise the complexity inherent in the combination of modal decomposition and the internal action τ, we apply the decomposition method to abstraction-free TSSs. In Sect. 4, where we will develop congruence formats on the basis of this decomposition method, we will be able to circumvent the restriction to abstraction-free TSSs, owing to the compositionality of the abstraction operator.

Definition 14. *Let P be a TSS in ready simulation format, which contains the $\aleph$-patience rules and is abstraction-free with respect to $\aleph$. We define $\cdot^{-1} : \mathbb{T}(\Sigma) \times \mathbb{O} \to \mathcal{P}(V \to \mathbb{O})$ as follows. Let t denote a univariate term, i.e. without multiple occurrences of the same variable.*

1. *$\psi \in t^{-1}(\bigwedge_{i\in I} \varphi_i)$ iff for $x \in V$*
$$\psi(x) = \bigwedge_{i \in I} \psi_i(x)$$
where $\psi_i \in t^{-1}(\varphi_i)$ for $i \in I$.
2. *$\psi \in t^{-1}(\neg\varphi)$ iff there is a function $h : t^{-1}(\varphi) \to \mathit{var}(t)$ with*
$$\psi(x) = \bigwedge_{\chi \in h^{-1}(x)} \neg\chi(x) \qquad \text{for } x \in V$$
3. *$\psi \in t^{-1}(\langle\alpha\rangle\varphi)$ iff there is a P-ruloid $\frac{H}{t \xrightarrow{\alpha} u}$ and a $\chi \in u^{-1}(\varphi)$ with*
$$\psi(x) = \begin{cases} \chi(x) \;\wedge \displaystyle\bigwedge_{x \xrightarrow{\beta} y \in H} \langle\beta\rangle\chi(y) \;\wedge \displaystyle\bigwedge_{x \stackrel{\gamma}{\nrightarrow} \in H} \neg\langle\gamma\rangle\top & \text{if } x \in \mathit{var}(t) \\ \top & \text{if } x \notin \mathit{var}(t) \end{cases}$$

4. *$\psi \in t^{-1}(\langle\epsilon\rangle\varphi)$ iff there is a $\chi \in t^{-1}(\varphi)$ with*

$$\psi(x) = \begin{cases} \langle\epsilon\rangle\chi(x) & \textit{if } x \textit{ occurs } \aleph\textit{-liquid in } t \\ \chi(x) & \textit{otherwise} \end{cases}$$

5. *$\psi \in t^{-1}(\langle\hat{\tau}\rangle\varphi)$ iff one of the following holds:*
   (a) $\psi \in t^{-1}(\varphi)$, or
   (b) there is an $x_0 \in V$ that occurs $\aleph$-liquid in t and a $\chi \in t^{-1}(\varphi)$ such that

$$\psi(x) = \begin{cases} \langle\hat{\tau}\rangle\chi(x) & \textit{if } x = x_0 \\ \chi(x) & \textit{if } x \neq x_0 \end{cases}$$

6. *$\psi \in \rho(t)^{-1}(\varphi)$ for $\rho : var(t) \to V$ not injective iff there is a $\chi \in t^{-1}(\varphi)$ with*

$$\psi(x) = \bigwedge_{y\in\rho^{-1}(x)} \chi(y) \qquad \textit{for } x \in V$$

It is not hard to see that if $\psi \in t^{-1}(\varphi)$, then $\psi(x) \equiv \top$ for $x \notin var(t)$.

To explain the idea behind Def. 14, we expand on two of its cases. Consider $t^{-1}(\neg\varphi)$, and let σ be any closed substitution. We have $\sigma(t) \not\models \varphi$ if and only if there is no $\chi \in t^{-1}(\varphi)$ such that $\sigma(x) \models \chi(x)$ for all $x \in var(t)$. In other words, for each $\chi \in t^{-1}(\varphi)$, $\psi(x)$ must contain a conjunct $\neg\chi(x)$, for some $x \in var(t)$.

Consider $t^{-1}(\langle\alpha\rangle\varphi)$, and let σ be any closed substitution. The question is under which conditions $\psi(x) \in \mathbb{O}$ on $\sigma(x)$, for $x \in var(t)$, there is a transition $\sigma(t) \xrightarrow{\alpha} q$ with $q \models \varphi$. According to Prop. 1, there is such a transition if and only if there is a closed substitution σ' with $\sigma'(t) = \sigma(t)$ and a P-ruloid $\frac{H}{t\xrightarrow{\alpha}u}$ such that (1) the premises in $\sigma'(H)$ are satisfied and (2) $\sigma'(u) \models \varphi$. The first condition is covered if for $x \in var(t)$, $\psi(x)$ contains conjuncts $\langle\beta\rangle\top$ for $x \xrightarrow{\beta} y \in H$ and conjuncts $\neg\langle\gamma\rangle\top$ for $x \overset{\gamma}{\not\rightarrow} \in H$. By adding a conjunct $\chi(x)$, and replacing each conjunct $\langle\beta\rangle\top$ by $\langle\beta\rangle\chi(y)$, for some $\chi \in u^{-1}(\varphi)$, the second condition is covered as well.

The following theorem will be the key to the forthcoming congruence results.

Theorem 2. *Given a complete TSS P in ready simulation format, which contains the $\aleph$-patience rules and is abstraction-free with respect to $\aleph$. For any term t, closed substitution σ and $\varphi \in \mathbb{O}$:*

$$\sigma(t) \models \varphi \;\Leftrightarrow\; \exists\psi \in t^{-1}(\varphi)\ \forall x \in var(t) : \sigma(x) \models \psi(x)$$

Proof. By structural induction on φ. First we treat the case where t is univariate.

– $\varphi = \bigwedge_{i\in I} \varphi_i$
  $\sigma(t) \models \bigwedge_{i\in I} \varphi_i \;\Leftrightarrow\; \forall i \in I : \sigma(t) \models \varphi_i \Leftrightarrow\; \forall i \in I\ \exists\psi_i \in t^{-1}(\varphi_i)\ \forall x \in var(t) : \sigma(x) \models \psi_i(x) \Leftrightarrow\; \exists\psi \in t^{-1}(\bigwedge_{i\in I} \varphi_i)\ \forall x \in var(t) : \sigma(x) \models \psi(x)$.
– $\varphi = \neg\varphi'$
  $\sigma(t) \models \neg\varphi' \Leftrightarrow \sigma(t) \not\models \varphi' \Leftrightarrow \exists h : t^{-1}(\varphi') \to var(t)\ \forall\chi \in t^{-1}(\varphi') : \sigma(h(\chi)) \not\models \chi(h(\chi)) \Leftrightarrow \exists h : t^{-1}(\varphi') \to var(t)\ \forall x \in var(t) : \sigma(x) \models \bigwedge_{\chi\in h^{-1}(x)} \neg\chi(x) \Leftrightarrow \exists\psi \in t^{-1}(\neg\varphi')\ \forall x \in var(t) : \sigma(x) \models \psi(x)$.

– $\varphi = \langle\alpha\rangle\varphi'$
($\Rightarrow$) Let $\sigma(t) \models \langle\alpha\rangle\varphi'$. Then $P \vdash_{ws} \sigma(t) \xrightarrow{\alpha} p$ with $p \models \varphi'$. By Prop. 1 there is a P-ruloid $\frac{H}{t \xrightarrow{\alpha} u}$ and a σ' with $P \vdash_{ws} \sigma'(\mu)$ for $\mu \in H$, $\sigma'(t) = \sigma(t)$, i.e. $\sigma'(x) = \sigma(x)$ for $x \in \mathit{var}(t)$, and $\sigma'(u) = p$. Since $\sigma'(u) \models \varphi'$, by induction there is a $\chi \in u^{-1}(\varphi')$ with $\sigma'(z) \models \chi(z)$ for $z \in \mathit{var}(u)$. Furthermore, $\sigma'(z) \models \chi(z) \equiv \top$ for $z \notin \mathit{var}(u)$. Define $\psi \in t^{-1}(\langle\alpha\rangle\varphi')$ as in Def. 14.3, using $\frac{H}{t \xrightarrow{\alpha} u}$ and χ. Let $x \in \mathit{var}(t)$. For $x \xrightarrow{\beta} y \in H$, $P \vdash_{ws} \sigma'(x) \xrightarrow{\beta} \sigma'(y) \models \chi(y)$, so $\sigma'(x) \models \langle\beta\rangle\chi(y)$. Moreover, for $x \overset{\gamma}{\not\longrightarrow} \in H$, $P \vdash_{ws} \sigma'(x) \overset{\gamma}{\not\longrightarrow}$, so the consistency of $\vdash_{ws}$ (see [12]) yields $P \not\vdash_{ws} \sigma'(x) \xrightarrow{\gamma} q$ for all closed terms q, and thus $\sigma'(x) \models \neg\langle\gamma\rangle\top$. Hence $\sigma(x) = \sigma'(x) \models \psi(x)$.

($\Leftarrow$) Let $\psi \in t^{-1}(\langle\alpha\rangle\varphi')$ with $\sigma(x) \models \psi(x)$ for $x \in \mathit{var}(t)$. There is a P-ruloid

$$\frac{\{x \xrightarrow{\beta_i} y_i \mid i \in I_x,\ x \in \mathit{var}(t)\} \cup \{x \overset{\gamma_j}{\not\longrightarrow} \mid j \in J_x,\ x \in \mathit{var}(t)\}}{t \xrightarrow{\alpha} u}$$

and a $\chi \in u^{-1}(\varphi')$ with $\psi(x) = \chi(x) \wedge \bigwedge_{i \in I_x} \langle\beta_i\rangle\chi(y_i) \wedge \bigwedge_{j \in J_x} \neg\langle\gamma_j\rangle\top$ for $x \in \mathit{var}(t)$. For $x \in \mathit{var}(t)$, $\sigma(x) \models \psi(x)$ yields, for $i \in I_x$, $P \vdash_{ws} \sigma(x) \xrightarrow{\beta_i} p_i$ with $p_i \models \chi(y_i)$ for some closed term p_i; moreover, for $j \in J_x$, $P \not\vdash_{ws} \sigma(x) \xrightarrow{\gamma_j} q$ for all closed terms q, so by the completeness of P, $P \vdash_{ws} \sigma(x) \overset{\gamma_j}{\not\longrightarrow}$. Define $\sigma'(x) = \sigma(x)$ and $\sigma'(y_i) = p_i$ for $x \in \mathit{var}(t)$ and $i \in I_x$. Here we use that the y_i are all different and do not occur in t. Then $\sigma'(z) \models \chi(z)$ for $z \in \mathit{var}(u)$, since $\mathit{var}(u) \subseteq \{x, y_i \mid x \in \mathit{var}(t), i \in I_x\}$. So by induction, $\sigma'(u) \models \varphi'$. Moreover, for $x \in \mathit{var}(t)$, $P \vdash_{ws} \sigma'(x) \xrightarrow{\beta_i} \sigma'(y_i)$ for $i \in I_x$, and $P \vdash_{ws} \sigma'(x) \overset{\gamma_j}{\not\longrightarrow}$ for $j \in J_x$, so by Prop. 1, $P \vdash_{ws} \sigma'(t) \xrightarrow{\alpha} \sigma'(u)$. Hence $\sigma(t) = \sigma'(t) \models \langle\alpha\rangle\varphi'$.

– $\varphi = \langle\epsilon\rangle\varphi'$
($\Rightarrow$) We prove by induction on n: if $P \vdash_{ws} p_i \xrightarrow{\tau} p_{i+1}$ for $i \in \{0, \ldots, n-1\}$ with $\sigma(t) = p_0$ and $p_n \models \varphi'$, then there is a $\psi \in t^{-1}(\langle\epsilon\rangle\varphi')$ with $\sigma(x) \models \psi(x)$ for $x \in \mathit{var}(t)$.

$n = 0$ Since $\sigma(t) = p_0 \models \varphi'$, by induction on formula size, there is a $\chi \in t^{-1}(\varphi')$ with $\sigma(x) \models \chi(x)$ for $x \in \mathit{var}(t)$. Define $\psi \in t^{-1}(\langle\epsilon\rangle\varphi')$ as in Def. 14.4, using χ. Then $\sigma(x) \models \psi(x)$ for $x \in \mathit{var}(t)$.

$n > 0$ Since $P \vdash_{ws} \sigma(t) \xrightarrow{\tau} p_1$, by Prop. 1 there is a P-ruloid $\frac{H}{t \xrightarrow{\tau} u}$ and a σ' with $P \vdash_{ws} \sigma'(\mu)$ for $\mu \in H$, $\sigma'(t) = \sigma(t)$, i.e. $\sigma'(x) = \sigma(x)$ for $x \in \mathit{var}(t)$, and $\sigma'(u) = p_1$. Since $\sigma'(u) = p_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} p_n \models \langle\epsilon\rangle\varphi'$, by induction on n, there is a $\chi \in u^{-1}(\langle\epsilon\rangle\varphi')$ with $\sigma'(y) \models \chi(y)$ for $y \in \mathit{var}(u)$. Furthermore, $\sigma'(y) \models \chi(y) \equiv \top$ for $y \notin \mathit{var}(u)$. Since P is abstraction-free, by Lem. 1, the P-ruloid $\frac{H}{t \xrightarrow{\tau} u}$ must be $\aleph$-patient. Thus $H = \{x_0 \xrightarrow{\tau} y_0\}$, where x_0 occurs $\aleph$-liquid in t and, since t is univariate, $u = t[y_0/x_0]$. Moreover, $y_0 \notin \mathit{var}(t)$, so u is also univariate. The occurrence of y_0 in u is $\aleph$-liquid, so according to Def. 14.4, $\chi(y_0)$ is of the form $\langle\epsilon\rangle\varphi''$. Let $\psi(x_0) = \chi(y_0)$, $\psi(y_0) = \chi(x_0) \equiv \top$, and $\psi(z) = \chi(z)$ otherwise. By alpha-conversion, $\chi \in u^{-1}(\langle\epsilon\rangle\varphi')$ implies $\psi \in t^{-1}(\langle\epsilon\rangle\varphi')$. For $x \in \mathit{var}(t) \backslash \{x_0\}$, $\sigma(x) = \sigma'(x) \models \chi(x) = \psi(x)$. Furthermore, $P \vdash_{ws} \sigma'(x_0) \xrightarrow{\tau} \sigma'(y_0)$ and $\sigma'(y_0) \models \chi(y_0) = \psi(x_0)$; so since $\psi(x_0)$ is of the form $\langle\epsilon\rangle\varphi''$, $\sigma(x_0) = \sigma'(x_0) \models \psi(x_0)$.

($\Leftarrow$) Let $\psi \in t^{-1}(\langle\epsilon\rangle\varphi')$ with $\sigma(x) \models \psi(x)$ for $x \in \mathit{var}(t)$. Then there is a $\chi \in t^{-1}(\varphi')$ with $\psi(x) = \langle\epsilon\rangle\chi(x)$ if x occurs $\aleph$-liquid in t and $\psi(x) = \chi(x)$ otherwise. For each x that occurs $\aleph$-liquid in t, $\sigma(x) \models \psi(x) = \langle\epsilon\rangle\chi(x)$, i.e. $\sigma(x) \overset{\epsilon}{\Longrightarrow} p_x$ with $p_x \models \chi(x)$. Define $\sigma'(x) = p_x$ if x occurs $\aleph$-liquid in t and $\sigma'(x) = \sigma(x)$ otherwise. Due to the presence of the $\aleph$-patience rules and the fact that t is univariate, $\sigma(t) \overset{\epsilon}{\Longrightarrow} \sigma'(t)$. Furthermore, $\sigma'(x) \models \chi(x)$ for $x \in \mathit{var}(t)$, so by induction on formula size, $\sigma'(t) \models \varphi'$. Hence $\sigma(t) \models \langle\epsilon\rangle\varphi'$.

– $\varphi = \langle\hat{\tau}\rangle\varphi'$

($\Rightarrow$) Suppose $\sigma(t) \models \langle\hat{\tau}\rangle\varphi'$. Then either $\sigma(t) \models \varphi'$ or $P \vdash_{ws} \sigma(t) \xrightarrow{\tau} p \models \varphi'$ for some closed term p. In the first case, by induction there is a $\psi \in t^{-1}(\varphi')$ such that $\sigma(x) \models \psi(x)$ for $x \in \mathit{var}(t)$; by Def. 14.5(a), $\psi \in t^{-1}(\langle\hat{\tau}\rangle\varphi')$, and we are done. In the second case, by Prop. 1 there is a P-ruloid $\frac{H}{t \xrightarrow{\tau} u}$ and a closed substitution σ' with $P \vdash_{ws} \sigma'(\mu)$ for $\mu \in H$, $\sigma'(t) = \sigma(t)$, i.e. $\sigma'(x) = \sigma(x)$ for $x \in \mathit{var}(t)$, and $\sigma'(u) = p$. Since $\sigma'(u) \models \varphi'$, by induction there is a $\chi \in u^{-1}(\varphi')$ such that $\sigma'(y) \models \chi(y)$ for $y \in \mathit{var}(u)$. Furthermore, $\sigma'(y) \models \chi(y) \equiv \top$ for $y \notin \mathit{var}(u)$. Since P is abstraction-free, by Lem. 1, the P-ruloid $\frac{H}{t \xrightarrow{\tau} u}$ must be $\aleph$-patient. Thus $H = \{x_0 \xrightarrow{\tau} y_0\}$, where x_0 occurs $\aleph$-liquid in t and, since t is univariate, $u = t[y_0/x_0]$. Moreover, $y_0 \notin \mathit{var}(t)$, so u is also univariate. Let $\psi(x_0) = \langle\hat{\tau}\rangle\chi(y_0)$, $\psi(y_0) = \chi(x_0) \equiv \top$, and $\psi(x) = \chi(x)$ otherwise. By Def. 14.5(b) together with alpha-conversion, $\psi \in t^{-1}(\langle\hat{\tau}\rangle\varphi')$. For $x \in \mathit{var}(t)\backslash\{x_0\}$, $\sigma(x) = \sigma'(x) \models \chi(x) = \psi(x)$. Moreover, since $P \vdash_{ws} \sigma'(x_0) \xrightarrow{\tau} \sigma'(y_0)$ and $\sigma'(y_0) \models \chi(y_0)$, it follows that $\sigma(x_0) = \sigma'(x_0) \models \langle\hat{\tau}\rangle\chi(y_0) = \psi(x_0)$.

($\Leftarrow$) Suppose $\psi \in t^{-1}(\langle\hat{\tau}\rangle\varphi')$ with $\sigma(x) \models \psi(x)$ for all $x \in \mathit{var}(t)$. If $\psi \in t^{-1}(\varphi')$, then by induction $\sigma(t) \models \varphi'$, so $\sigma(t) \models \langle\hat{\tau}\rangle\varphi'$, and we are done. Suppose that for some $\chi \in t^{-1}(\varphi')$ and some x_0 that occurs $\aleph$-liquid in t, $\psi(x) = \chi(x)$ for $x \neq x_0$ and $\psi(x_0) = \langle\hat{\tau}\rangle\chi(x_0)$. Then $\sigma(x) \models \chi(x)$ for $x \in \mathit{var}(t)\backslash\{x_0\}$. Furthermore, $\sigma(x_0) \models \psi(x_0) = \langle\hat{\tau}\rangle\chi(x_0)$, so either $\sigma(x_0) \models \chi(x_0)$ or $\sigma(x_0) \xrightarrow{\tau} p \models \chi(x_0)$ for some closed term p. In the first case, by induction $\sigma(t) \models \varphi'$, so $\sigma(t) \models \langle\hat{\tau}\rangle\varphi'$, and we are done. In the second case, define $\sigma'(x) = \sigma(x)$ for $x \in \mathit{var}(t)$ and $\sigma'(y_0) = p$. Since $P \vdash_{ws} \sigma'(x_0) \xrightarrow{\tau} \sigma'(y_0)$, and $\frac{x_0 \xrightarrow{\tau} y_0}{t \xrightarrow{\tau} t[y_0/x_0]}$ is an $\aleph$-patient P-ruloid, by Prop. 1, $P \vdash_{ws} \sigma'(t) \xrightarrow{\tau} \sigma'(t[y_0/x_0])$. Furthermore, by induction $\sigma'(t[y_0/x_0]) \models \varphi'$. Hence $\sigma(t) = \sigma'(t) \models \langle\hat{\tau}\rangle\varphi'$.

Finally, suppose t is not univariate. Let $t = \rho(u)$ for some univariate u and $\rho : \mathit{var}(u) \rightarrow V$ not injective. $\sigma(\rho(u)) \models \varphi \Leftrightarrow \exists\chi \in u^{-1}(\varphi)\ \forall y \in \mathit{var}(u) : \sigma(\rho(y)) \models \chi(y) \Leftrightarrow \exists\chi \in u^{-1}(\varphi)\ \forall x \in \mathit{var}(t) : \sigma(x) \models \bigwedge_{y\in\rho^{-1}(x)} \chi(y) \Leftrightarrow \exists\psi \in t^{-1}(\varphi)\ \forall x \in \mathit{var}(t) : \sigma(x) \models \psi(x)$. $\square$

The part of Thm. 2 that deals with the modalities $\bigwedge_{i\in I}$, $\neg$ and $\langle\alpha\rangle$ only has been established in [9]. There, a few examples are given showing how Def. 14 can be used to decompose a modal formula, as well as a counterexample showing that the completeness requirement in Thm. 2 cannot simply be skipped. The inclusion of the modalities $\langle\epsilon\rangle$ and $\langle\hat{\tau}\rangle$ is new. The following example illustrates the use of the decomposition method on a formula with the modality $\langle\epsilon\rangle$.

Example 1. Let $A = \{a\}$ and $P = (\Sigma, R)$, where Σ consists of a binary function symbol $\|$ with liquid arguments, and R contains the rules $\frac{x \xrightarrow{\alpha} x'}{x\|y \xrightarrow{\alpha} x'\|y}$ and $\frac{y \xrightarrow{\alpha} y'}{x\|y \xrightarrow{\alpha} x\|y'}$ for $\alpha \in \{a, \tau\}$. The TSS P is complete and in ready simulation format. Furthermore, it contains the two patience rules and is abstraction-free.

We compute $(x\|y)^{-1}(\langle\epsilon\rangle\langle a\rangle\top)$. By Def. 14.4, for each $\psi \in (x\|y)^{-1}(\langle\epsilon\rangle\langle a\rangle\top)$ we have $\psi(x) = \langle\epsilon\rangle\chi(x)$ and $\psi(y) = \langle\epsilon\rangle\chi(y)$ for some $\chi \in (x\|y)^{-1}(\langle a\rangle\top)$. According to Def. 14.3, $(x\|y)^{-1}(\langle a\rangle\top) = \{\chi_1, \chi_2\}$, where χ_1 and χ_2 are constructed from the only P-ruloids with a conclusion $x\|y \xrightarrow{a} \_$, namely $\frac{x \xrightarrow{a} x'}{x\|y \xrightarrow{a} x'\|y}$ and $\frac{y \xrightarrow{a} y'}{x\|y \xrightarrow{a} x\|y'}$, together with $\xi_1 \in (x'\|y)^{-1}(\top)$ resp. $\xi_2 \in (x\|y')^{-1}(\top)$:

$$\begin{array}{ll} \chi_1(x) = \xi_1(x) \wedge \langle a\rangle\xi_1(x') \equiv \langle a\rangle\top & \chi_2(x) = \top \\ \chi_1(y) = \top & \chi_2(y) = \xi_2(y) \wedge \langle a\rangle\xi_2(y') \equiv \langle a\rangle\top \end{array}$$

Hence $(x\|y)^{-1}(\langle\epsilon\rangle\langle a\rangle\top) = \{\psi_1, \psi_2\}$ with ψ_1 and ψ_2 defined as follows:

$$\begin{array}{ll} \psi_1(x) = \langle\epsilon\rangle\chi_1(x) \equiv \langle\epsilon\rangle\langle a\rangle\top & \psi_2(x) = \langle\epsilon\rangle\chi_2(x) = \langle\epsilon\rangle\top \equiv \top \\ \psi_1(y) = \langle\epsilon\rangle\chi_1(y) = \langle\epsilon\rangle\top \equiv \top & \psi_2(y) = \langle\epsilon\rangle\chi_2(y) \equiv \langle\epsilon\rangle\langle a\rangle\top \end{array}$$

4 Branching Bisimulation as a Congruence

We proceed to apply the decomposition method from the previous section to derive congruence formats for branching and rooted branching bisimulation equivalence. The idea is that the branching bisimulation format must guarantee that a formula from $\mathbb{O}_b$ is always decomposed into formulas from $\mathbb{O}_b^{\equiv}$ (see Prop. 3). Likewise, the rooted branching bisimulation format must guarantee that a formula from $\mathbb{O}_{rb}$ is always decomposed into formulas from $\mathbb{O}_{rb}^{\equiv}$ (see Prop. 4). This implies the desired congruence results (see Thm. 3 and Thm. 4). In the derivation of the congruence formats, we will circumvent the restriction in the decomposition method to abstraction-free TSSs, using compositionality of the abstraction operator.

4.1 Congruence formats

We assume a second predicate Λ on arguments of function symbols, to denote that the processes they contain may have started running, but might currently be resting, in which case no patience rules are needed for these arguments. Always $\aleph \subseteq \Lambda$.

Definition 15. *Let $\aleph \subseteq \Lambda$. An ntytt rule $\frac{H}{t \xrightarrow{\alpha} u}$ is* rooted branching bisimulation safe *with respect to $\aleph$ and Λ if:*

1. *it has no lookahead,*
2. *right-hand sides of premises occur only Λ-liquid in u, and*
3. *if x occurs exactly once*[5] *in t, at a Λ-liquid position, then:*
   (a) *all occurrences of x in the rule are Λ-liquid,*
   (b) *x has no $\aleph$-liquid occurrences in left-hand sides of negative premises,*
   (c) *x has at most one $\aleph$-liquid occurrence in the left-hand side of one positive premise, and this premise has a label from A, and*
   (d) *if x occurs $\aleph$-frozen in t, then x does not occur $\aleph$-liquid in left-hand sides of premises.*

In case Λ is the universal predicate, we say that the rule is branching bisimulation safe *with respect to $\aleph$.*

Definition 16. *A TSS in ready simulation format is in* rooted branching bisimulation format *if, for some $\aleph \subseteq \Lambda$, it consists of the $\aleph$-patience rules and rules that are rooted branching bisimulation safe with respect to $\aleph$ and Λ.*

A TSS in ready simulation format is in branching bisimulation format *if, for some $\aleph$, it consists of the $\aleph$-patience rules and rules that are branching bisimulation safe with respect to $\aleph$.*

If a TSS P is in rooted branching bisimulation format then there are smallest predicates $\aleph_0$ and Λ_0 such that P consists of the $\aleph_0$-patience rules and rules that are rooted branching bisimulation safe with respect to $\aleph_0$ and Λ_0. Namely the Λ_0-liquid arguments are *generated* by requirements 2 and 3(a) of Def. 15; they are the smallest collection of arguments such that these two requirements are satisfied. Given Λ_0, $\aleph_0$ is the unique collection of arguments within Λ_0 for which patience rules exists. For any TSS P, $\aleph_0$ and Λ_0 can be calculated in this way, and whether P is in rooted branching bisimulation format then depends solely on whether requirements 1 and 3(b–d) of Def. 15 are satisfied.

When restricting to TSSs consisting of nxytt rules only, it becomes easier to reformulate the definition of the rooted branching bisimulation format without mentioning $\aleph$. Namely, requirements 3(b–d) of Def. 15, together with the existence of the patience rules required in Def. 16 then amount to

3. (b) x does not occur as the left-hand side of a negative premise,
   (c) x occurs at most once as the left-hand side of a positive premise, and this premise has a label from A, and
   (d) if within t, x occurs in an argument of an operator f for which there is no patience rule, then x does not occur in the left-hand side of premises.

[5] For the rooted branching bisimulation format in Def. 16, only the requirements for rules in which t is univariate matter. The formulation of Def. 15 for general terms t paves the way for Prop. 2.

A TSS is now in rooted branching bisimulation format if it consists of patience rules and rules that are rooted branching bisimulation safe with respect to Λ and that collection of patience rules.

Using this, it is not hard to see that the rooted branching bisimulation format strengthens the RBB safe format from [7].

In the definition of modal decomposition, we did not use the rules from the original TSS P, but the P-ruloids. Therefore we must verify that if P is in (rooted) branching bisimulation format, then so are the P-ruloids.

Proposition 2. *If a TSS P is in (rooted) branching bisimulation format with respect to some $\aleph$ (and Λ), then each P-ruloid is either $\aleph$-patient or (rooted) branching bisimulation safe with respect to $\aleph$ (and Λ).*

The proof of Prop. 2 is omitted here. The key part of the proof is to show that the decent (rooted) branching bisimulation format is preserved under irredundant provability. The adjective irredundant is essential here; this preservation result would fail if "junk" could be added to the premises of derived transition rules.

4.2 Preservation of modal characterisations

In this section we prove that given a TSS in rooted branching bisimulation format, if $\psi \in t^{-1}(\varphi)$ with $\varphi \in \mathbb{O}_b$, then $\psi(x) \in \mathbb{O}_b^{\equiv}$ if x occurs only Λ-liquid in t. (That is why in the branching bisimulation format, Λ must be universal.) If $\varphi \in \mathbb{O}_{rb}$, then $\psi(x) \in \mathbb{O}_{rb}^{\equiv}$ for all variables x.

Proposition 3. *Let P be an abstraction-free TSS in rooted branching bisimulation format, with respect to some $\aleph$ and Λ. For any term t and variable x that occurs only Λ-liquid in t:*

$$\varphi \in \mathbb{O}_b \;\Rightarrow\; \forall \psi \in t^{-1}(\varphi) : \psi(x) \in \mathbb{O}_b^{\equiv}$$

Proof. We apply structural induction on $\varphi \in \mathbb{O}_b$. Let $t \in \mathbb{T}(\Sigma)$ and $\psi \in t^{-1}(\varphi)$, and let x occur only Λ-liquid in t. First we treat the case where t is univariate. If $x \notin var(t)$, then $\psi(x) \equiv \top \in \mathbb{O}_b^{\equiv}$. Suppose x occurs once in t.

- $\varphi = \bigwedge_{i\in I} \varphi_i$ with $\varphi_i \in \mathbb{O}_b$ for $i \in I$. By Def. 14.1, $\psi(x) = \bigwedge_{i\in I} \psi_i(x)$ with $\psi_i \in t^{-1}(\varphi_i)$ for $i \in I$. By induction, $\psi_i(x) \in \mathbb{O}_b^{\equiv}$ for $i \in I$, so $\psi(x) \in \mathbb{O}_b^{\equiv}$.
- $\varphi = \neg\varphi'$ with $\varphi' \in \mathbb{O}_b$. By Def. 14.2, there is a function $h : t^{-1}(\varphi') \to var(t)$ such that $\psi(x) = \bigwedge_{\chi \in h^{-1}(x)} \neg\chi(x)$. By induction, $\chi(x) \in \mathbb{O}_b^{\equiv}$ for $\chi \in h^{-1}(x)$, so $\psi(x) \in \mathbb{O}_b^{\equiv}$.
- $\varphi = \langle\epsilon\rangle(\varphi_1\langle\hat{\tau}\rangle\varphi_2)$ with $\varphi_1, \varphi_2 \in \mathbb{O}_b$. By Def. 14.4, either $\psi(x) = \langle\epsilon\rangle\chi(x)$ if x occurs $\aleph$-liquid in t, or $\psi(x) = \chi(x)$ if x occurs $\aleph$-frozen in t, for some $\chi \in t^{-1}(\varphi_1\langle\hat{\tau}\rangle\varphi_2)$. By Def. 14.1, $\chi(x) = \chi_1(x) \wedge \chi_2(x)$ with $\chi_1 \in t^{-1}(\varphi_1)$ and $\chi_2 \in t^{-1}(\langle\hat{\tau}\rangle\varphi_2)$. By Def. 14.5, either $\chi_2(x) = \langle\hat{\tau}\rangle\xi(x)$ and x occurs $\aleph$-liquid in t, or $\chi_2(x) = \xi(x)$, for some $\xi \in t^{-1}(\varphi_2)$. So $\psi(x)$ is of the form $\langle\epsilon\rangle(\chi_1(x)\langle\hat{\tau}\rangle\xi(x))$, $\langle\epsilon\rangle(\chi_1(x) \wedge \xi(x))$ or $\chi_1(x) \wedge \xi(x)$. By induction, $\chi_1(x), \xi(x) \in \mathbb{O}_b^{\equiv}$. Hence $\psi(x) \in \mathbb{O}_b^{\equiv}$.

- $\varphi = \langle\epsilon\rangle(\varphi_1\langle a\rangle\varphi_2)$ with $\varphi_1, \varphi_2 \in \mathbb{O}_b$. By Def. 14.4, either $\psi(x) = \langle\epsilon\rangle\chi(x)$ if x occurs $\aleph$-liquid in t, or $\psi(x) = \chi(x)$ if x occurs $\aleph$-frozen in t, for some $\chi \in t^{-1}(\varphi_1\langle a\rangle\varphi_2)$. By Def. 14.1, $\chi(x) = \chi_1(x) \wedge \chi_2(x)$ with $\chi_1 \in t^{-1}(\varphi_1)$ and $\chi_2 \in t^{-1}(\langle a\rangle\varphi_2)$. By induction, $\chi_1(x) \in \mathbb{O}_b^{\equiv}$. By Def. 14.3,

$$\chi_2(x) = \xi(x) \wedge \bigwedge_{x \xrightarrow{\beta} y \in H} \langle\beta\rangle\xi(y) \wedge \bigwedge_{x \overset{\gamma}{\nrightarrow} \in H} \neg\langle\gamma\rangle\top$$

for some $\xi \in u^{-1}(\varphi_2)$ and P-ruloid $\frac{H}{t \xrightarrow{a} u}$. Since $a \neq \tau$, by Prop. 2, $\frac{H}{t \xrightarrow{a} u}$ is rooted branching bisimulation safe with respect to $\aleph$ and Λ. Since the occurrence of x in t is Λ-liquid, x occurs only Λ-liquid in u. Moreover, variables in right-hand sides of premises in H occur only Λ-liquid in u. So by induction, $\xi(x) \in \mathbb{O}_b^{\equiv}$ and $\xi(y) \in \mathbb{O}_b^{\equiv}$ for $x \xrightarrow{\beta} y \in H$. We distinguish two cases.

Case 1: The occurrence of x in t is $\aleph$-liquid. Then $\psi(x) = \langle\epsilon\rangle\chi(x)$. Since $\frac{H}{t \xrightarrow{a} u}$ is rooted branching bisimulation safe with respect to $\aleph$ and Λ and an nxytt rule, x does not occur in left-hand sides of negative premises in H, and at most once in the left-hand side of one positive premise in H, which is of the form $x \xrightarrow{b} y$ with $b \in A$. Hence either $\chi_2(x) = \xi(x)$ or $\chi_2(x) = \xi(x)\langle b\rangle\xi(y)$. Since $\psi(x) = \langle\epsilon\rangle(\chi_1(x) \wedge \chi_2(x))$, either $\psi(x) = \langle\epsilon\rangle(\chi_1(x) \wedge \xi(x)) \in \mathbb{O}_b^{\equiv}$ or $\psi(x) \equiv \langle\epsilon\rangle(\chi_1(x) \wedge \xi(x)\langle b\rangle\xi(y)) \in \mathbb{O}_b^{\equiv}$.

Case 2: The occurrence of x in t is $\aleph$-frozen. Then $\psi(x) = \chi(x)$. Since $\frac{H}{t \xrightarrow{a} u}$ is rooted branching bisimulation safe with respect to $\aleph$ and Λ and an nxytt rule, x does not occur in left-hand sides of premises in H. So $\chi_2(x) = \xi(x)$, and thus $\psi(x) = \chi_1(x) \wedge \chi_2(x) = \chi_1(x) \wedge \xi(x) \in \mathbb{O}_b^{\equiv}$.

Finally, suppose t is not univariate. Then $t = \rho(u)$ for some univariate term u and $\rho : \mathit{var}(u) \to V$ not injective. By Def. 14.6, $\psi(x) = \bigwedge_{y \in \rho^{-1}(x)} \chi(y)$ for some $\chi \in u^{-1}(\varphi)$. Since u is univariate, and for each $y \in \rho^{-1}(x)$ the occurrence in u is Λ-liquid, $\chi(y) \in \mathbb{O}_b^{\equiv}$ for $y \in \rho^{-1}(x)$. Hence $\psi(x) \in \mathbb{O}_b^{\equiv}$. □

Proposition 4. *Let P be an abstraction-free TSS in rooted branching bisimulation format, with respect to some $\aleph$ and Λ. For any term t and variable x:*

$$\varphi \in \mathbb{O}_{rb} \;\Rightarrow\; \forall\psi \in t^{-1}(\varphi) : \psi(x) \in \mathbb{O}_{rb}^{\equiv}$$

Proof. We apply structural induction on $\varphi \in \mathbb{O}_{rb}$. Let $t \in \mathbb{T}(\Sigma)$ and $\psi \in t^{-1}(\varphi)$. We restrict attention to the case where t is univariate; the general case then follows just as at the end of the proof of Prop. 3. If $x \notin \mathit{var}(t)$, then $\psi(x) \equiv \top \in \mathbb{O}_{rb}^{\equiv}$. So suppose x occurs once in t.

- The cases $\varphi = \bigwedge_{i \in I} \varphi_i$ and $\varphi = \neg\varphi'$ proceed as in the proof of Prop. 3.
- $\varphi = \langle\alpha\rangle\varphi'$ with $\varphi' \in \mathbb{O}_b$. By Def. 14.3,

$$\psi(x) = \chi(x) \wedge \bigwedge_{x \xrightarrow{\beta} y \in H} \langle\beta\rangle\chi(y) \wedge \bigwedge_{x \overset{\gamma}{\nrightarrow} \in H} \neg\langle\gamma\rangle\top$$

for some $\chi \in u^{-1}(\varphi')$ and P-ruloid $\frac{H}{t \xrightarrow{\alpha} u}$. By induction, $\chi(x) \in \mathbb{O}_{rb}^{\equiv}$. (Induction may be applied because $\varphi' \in \mathbb{O}_b \subseteq \mathbb{O}_{rb}$.) By Prop. 2, $\frac{H}{t \xrightarrow{\alpha} u}$ is either rooted branching bisimulation safe with respect to $\aleph$ and Λ or $\aleph$-patient. In either case, variables in right-hand sides of premises in H occur only Λ-liquid in u. By Prop. 3, $\chi(y) \in \mathbb{O}_b^{\equiv}$ for $x \xrightarrow{\beta} y \in H$, so $\langle\beta\rangle\chi(y) \in \mathbb{O}_{rb}^{\equiv}$. Also $\neg\langle\gamma\rangle\top \in \mathbb{O}_{rb}^{\equiv}$. Hence $\psi(x) \in \mathbb{O}_{rb}^{\equiv}$.

– $\varphi \in \mathbb{O}_b$. If the occurrence of x in t is Λ-liquid, then $\psi(x) \in \mathbb{O}_{rb}^{\equiv}$ follows from Prop. 3. So we can assume that this occurrence is Λ-frozen, and hence $\aleph$-frozen. The cases $\varphi = \bigwedge_{i\in I} \varphi_i$ and $\varphi = \neg\varphi'$ proceed as before. We focus on the other two cases.

∗ $\varphi = \langle\epsilon\rangle(\varphi_1\langle\hat{\tau}\rangle\varphi_2)$ with $\varphi_1, \varphi_2 \in \mathbb{O}_b \subseteq \mathbb{O}_{rb}$. Since the occurrence of x in t is $\aleph$-frozen, by Def. 14.4, $\psi(x) = \chi(x)$ for some $\chi \in t^{-1}(\varphi_1\langle\hat{\tau}\rangle\varphi_2)$. By Def. 14.1, $\chi(x) = \chi_1(x) \wedge \chi_2(x)$ with $\chi_1 \in t^{-1}(\varphi_1)$ and $\chi_2 \in t^{-1}(\langle\hat{\tau}\rangle\varphi_2)$. Since the occurrence of x in t is $\aleph$-frozen, by Def. 14.5, $\chi_2(x) = \xi(x)$ for some $\xi \in t^{-1}(\varphi_2)$. By induction, $\chi_1(x), \xi(x) \in \mathbb{O}_{rb}^{\equiv}$. Hence $\psi(x) \in \mathbb{O}_{rb}^{\equiv}$.

∗ $\varphi = \langle\epsilon\rangle(\varphi_1\langle a\rangle\varphi_2)$ with $\varphi_1, \varphi_2 \in \mathbb{O}_b \subseteq \mathbb{O}_{rb}$. Since the occurrence of x in t is $\aleph$-frozen, by Def. 14.4, $\psi(x) = \chi(x)$ for some $\chi \in t^{-1}(\varphi_1\langle a\rangle\varphi_2)$. By Def. 14.1, $\chi(x) = \chi_1(x) \wedge \chi_2(x)$ with $\chi_1 \in t^{-1}(\varphi_1)$ and $\chi_2 \in t^{-1}(\langle a\rangle\varphi_2)$. By induction, $\chi_1(x), \chi_2(x) \in \mathbb{O}_{rb}^{\equiv}$. Hence $\psi(x) \in \mathbb{O}_{rb}^{\equiv}$. □

4.3 Congruence results

Finally we are in a position to prove the promised congruence results.

Lemma 2. *Given a complete, abstraction-free TSS in branching bisimulation format, with respect to some* $\aleph$. *If* $\sigma(x) \leftrightarroweq_b \sigma'(x)$ *for* $x \in var(t)$, *then* $\sigma(t) \leftrightarroweq_b \sigma'(t)$.

Proof. By Thm. 1, $\sigma(x) \leftrightarroweq_b \sigma'(x)$ implies $\sigma(x) \sim_{\mathbb{O}_b^{\equiv}} \sigma'(x)$ for $x \in var(t)$. Let $\sigma(t) \models \varphi \in \mathbb{O}_b$. By Thm. 2 there is a $\psi \in t^{-1}(\varphi)$ with $\sigma(x) \models \psi(x)$ for $x \in var(t)$. Since Λ is universal, by Prop. 3, $\psi(x) \in \mathbb{O}_b^{\equiv}$ for $x \in var(t)$. Since $\sigma(x) \sim_{\mathbb{O}_b^{\equiv}} \sigma'(x)$, $\sigma'(x) \models \psi(x)$ for $x \in var(t)$. By Thm. 2, $\sigma'(t) \models \varphi$. Likewise, $\sigma'(t) \models \varphi \in \mathbb{O}_b$ implies $\sigma(t) \models \varphi$. So $\sigma(t) \sim_{\mathbb{O}_b} \sigma'(t)$. Hence $\sigma(t) \leftrightarroweq_b \sigma'(t)$. □

Theorem 3. *Given a complete TSS* $P = (\Sigma, R)$ *in branching bisimulation format, with respect to some* $\aleph$. *If* $\sigma(x) \leftrightarroweq_b \sigma'(x)$ *for* $x \in var(t)$, *then* $\sigma(t) \leftrightarroweq_b \sigma'(t)$.

Proof. Let P' be obtained from P, by changing in all rules, expect the $\aleph$-patience rules, conclusions of the form $t \xrightarrow{\tau} u$ into $t \xrightarrow{i} u$, for a fresh action $i \notin A \cup \{\tau\}$. By construction, P' is abstraction-free and in branching bisimulation format with respect to $\aleph$. So by Lem. 2, $\leftrightarroweq_b$ is a congruence for all operators of P'.

Let P'' be obtained from P' by adding a new operator τ_i with rules

$$\frac{x \xrightarrow{\alpha} y}{\tau_i(x) \xrightarrow{\alpha} \tau_i(y)} \quad (\alpha \neq i) \qquad\qquad \frac{x \xrightarrow{i} y}{\tau_i(x) \xrightarrow{\tau} \tau_i(y)}$$

This operator turns all i-labels into τ-labels. It is well-known and trivial to check that $\leftrightarroweq_b$ is a congruence for τ_i as well.

If follows trivially that for any operator $f \in \Sigma$ the behaviour of $\tau_i \circ f$ in P'' is the same as the behaviour of f in P. So as $\leftrightarroweq_b$ is a congruence for $\tau_i \circ f$ in P'', it must be a congruence for f in P. $\square$

Lemma 3. *Given a complete, abstraction-free TSS in rooted branching bisimulation format, with respect to some* $\aleph$ *and* Λ*. If* $\sigma(x) \leftrightarroweq_{rb} \sigma'(x)$ *for* $x \in var(t)$*, then* $\sigma(t) \leftrightarroweq_{rb} \sigma'(t)$*.*

Theorem 4. *Given a complete TSS in rooted branching bisimulation format, with respect to some* $\aleph$ *and* Λ*. If* $\sigma(x) \leftrightarroweq_{rb} \sigma'(x)$ *for* $x \in var(t)$*, then* $\sigma(t) \leftrightarroweq_{rb} \sigma'(t)$*.*

The proof of Lem. 3 is similar to the one of Lem. 2, except that Prop. 4 is applied instead of Prop. 3. Likewise, the proof of Thm. 4 is similar to the one of Thm. 3.

5 Applications

In this section we present four applications of the rooted branching bisimulation format.

5.1 Basic Process Algebra

Basic process algebra BPA [1] assumes a collection *Act* of constants, called *atomic actions*, which upon execution terminate successfully. The signature of BPA moreover includes function symbols $\_+\_$ and $\_\cdot\_$ of arity two, called *alternative composition* and *sequential composition*, respectively. Intuitively, $t_1 + t_2$ executes either t_1 or t_2, while $t_1 \cdot t_2$ first executes t_1 and upon successful termination executes t_2. We assume a special atomic action *tick* $\in$ *Act*, indicating the activity of successful termination upon executing the internal action τ, and a special constant *deadlock* δ, outside *Act*, which does not display any behaviour. These intuitions are made precise by means of the transition rules for $\mathrm{BPA}_\delta^{tick}$ presented below. In these rules, ℓ ranges over *Act*, and α over $A = \{\ell, \ell_\surd \mid \ell \in Act\}$. The label $\ell_\surd$ denotes that upon execution of ℓ, the process terminates successfully.

$$\ell \xrightarrow{\ell_\surd} \delta \qquad \frac{x_1 \xrightarrow{\alpha} y}{x_1 + x_2 \xrightarrow{\alpha} y} \qquad \frac{x_2 \xrightarrow{\alpha} y}{x_1 + x_2 \xrightarrow{\alpha} y}$$

$$\frac{x_1 \xrightarrow{\ell} y}{x_1 \cdot x_2 \xrightarrow{\ell} y \cdot x_2} \qquad \frac{x_1 \xrightarrow{\ell_\surd} y}{x_1 \cdot x_2 \xrightarrow{\ell} x_2}$$

The label *tick* counts as internal action, and for this reason the labels *tick* and $tick_\surd$ can also be written τ and $\tau_\surd$, respectively. The label $\tau_\surd$ denotes termination and counts as a normal observable action. When this action occurs in the first component of a sequential composition, it changes into the internal action τ, so that this TSS is not abstraction-free. We do not have $a \leftrightarroweq_{rb} a \cdot tick$, as the former process performs one visible action and the latter two. For this reason we call the constant *tick* *tick*, rather than τ.

The TSS above is in rooted branching bisimulation format, if we take the arguments of alternative composition to be Λ-frozen, the first argument of sequential composition to be $\aleph$-liquid, and the second argument of sequential composition to be $\aleph$-frozen. For the sake of the application to the action refinement operator, in Sect. 5.4, we take the second argument of sequential composition to be Λ-liquid.

Corollary 1. *Rooted branching bisimulation is a congruence for BPA$_\delta^{tick}$.*

5.2 Binary Kleene star

The *binary Kleene star* $t_1{}^*t_2$ [18] repeatedly executes t_1 until it executes t_2. This operational behaviour is captured by the following rules, which are added to the rules for BPA$_\delta^{tick}$.

$$\frac{x_1 \xrightarrow{\ell} y}{x_1{}^*x_2 \xrightarrow{\ell} y\cdot(x_1{}^*x_2)} \qquad \frac{x_1 \xrightarrow{\ell}\surd}{x_1{}^*x_2 \xrightarrow{\ell} x_1{}^*x_2} \qquad \frac{x_2 \xrightarrow{\alpha} y}{x_1{}^*x_2 \xrightarrow{\alpha} y}$$

The resulting TSS is in rooted branching bisimulation format, if we take the arguments of the binary Kleene star to be Λ-frozen.

Corollary 2. *Rooted branching bisimulation is a congruence for BPA$_\delta^{tick}$ with the binary Kleene star.*

5.3 Initial Priority

Initial priority is a unary function symbol that assumes an ordering on labels (which is usually defined on the level of atomic actions). The term $\theta(t)$ executes the transitions of t, with the restriction that an initial transition $t \xrightarrow{\alpha} t_1$ only gives rise to an initial transition $\theta(t) \xrightarrow{\alpha} t_1$ if there does not exist an initial transition $t \xrightarrow{\beta} t_2$ with $\alpha < \beta$. This intuition is captured by the rule for the initial priority operator below, which is added to the rules for BPA$_\delta^{tick}$.

$$\frac{x \xrightarrow{\alpha} y \quad x \overset{\beta}{\not\longrightarrow} \ \text{for } \alpha < \beta}{\theta(x) \xrightarrow{\alpha} y}$$

The resulting TSS is in rooted branching bisimulation format, if we take the argument of initial priority to be Λ-frozen.

Corollary 3. *Rooted branching bisimulation is a congruence for BPA$_\delta^{tick}$ with initial priority.*

5.4 Action Refinement

In the previous applications, it sufficed to take $\Lambda = \aleph$. In the following example, however, this is not possible.

The binary *action refinement* operator $t_1[\ell \rightsquigarrow t_2]$, for $\ell \in Act\backslash\{tick\}$, replaces each ℓ-transition in t_1 by t_2. Its transition rules, presented below, are added to the rules for BPA$_\delta^{tick}$.

$$\frac{x_1 \xrightarrow{\alpha} y}{x_1[\ell \rightsquigarrow x_2] \xrightarrow{\alpha} y[\ell \rightsquigarrow x_2]} \quad (\alpha \neq \ell, \ell_\surd)$$

$$\frac{x_1 \xrightarrow{\ell} y_1 \quad x_2 \xrightarrow{\ell'} y_2}{x_1[\ell \rightsquigarrow x_2] \xrightarrow{\ell'} y_2\cdot(y_1[\ell \rightsquigarrow x_2])} \qquad \frac{x_1 \xrightarrow{\ell_\surd} y_1 \quad x_2 \xrightarrow{\ell'} y_2}{x_1[\ell \rightsquigarrow x_2] \xrightarrow{\ell'} y_2}$$

$$\frac{x_1 \xrightarrow{\ell} y_1 \quad x_2 \xrightarrow{\ell'_\surd} y_2}{x_1[\ell \rightsquigarrow x_2] \xrightarrow{\ell'} y_1[\ell \rightsquigarrow x_2]} \qquad \frac{x_1 \xrightarrow{\ell_\surd} y_1 \quad x_2 \xrightarrow{\ell'_\surd} y_2}{x_1[\ell \rightsquigarrow x_2] \xrightarrow{\ell'_\surd} y_2}$$

The resulting TSS is in rooted branching bisimulation format, if we take the first argument of action refinement to be $\aleph$-liquid and the second argument to be Λ-frozen. For the second rule to be rooted branching bisimulation safe, it is essential that the second argument of sequential composition is Λ-liquid, for else it would violate restriction 2 of Def. 15.

Corollary 4. *Rooted branching bisimulation is a congruence for BPA$_\delta^{tick}$ with action refinement.*

6 Counterexamples

This section presents a series of counterexamples of complete TSSs in ntyft/ntyxt format, to show that none of the syntactic restrictions of our congruence formats can be omitted. (Of course it remains possible that certain restrictions can be refined.) In [16] a series of counterexamples can be found showing that the syntactic restrictions of the ntyft/ntyxt format are essential as well. Furthermore, in [5] a counterexample is given to show that completeness (there called positive after reduction) is essential.

It is well-known that branching bisimulation is not a congruence for BPA$_\delta^{tick}$. For instance, $a \leftrightarrows_b tick\cdot a$, but $a + c \not\leftrightarrows_b (tick\cdot a) + c$. Still we saw in Sect. 5.1 that the TSS for this process algebra is in rooted branching bisimulation format. This shows that universality of the predicate Λ cannot be omitted from the branching bisimulation format.

The examples in this section assume an action set $A = \{a, b, c\}$ and a TSS $P = \{\Sigma, R\}$, where the signature Σ contains the constant 0 and unary function symbols $\alpha\_$ for $\alpha \in A \cup \{\tau\}$, and R contains the rules $\alpha x \xrightarrow{\alpha} x$ for $\alpha \in A \cup \{\tau\}$. The argument of $\alpha\_$ is $\aleph$-frozen. Unlike before, in this section occurrences of

a, b, c as labels in rules are explicit action names, instead of parameters ranging over A.

Example 2. We extend P with the following rule:

$$\frac{x \xrightarrow{a} y \quad y \xrightarrow{b} z}{f(x) \xrightarrow{c} 0}$$

The rule above is not rooted branching bisimulation safe, because it contains lookahead, violating restriction 1 (of Def. 15). Clearly, $ab0 \leftrightarroweq_{rb} a\tau b0$ (and thus $ab0 \leftrightarroweq_{b} a\tau b0$). However, $f(ab0) \not\leftrightarroweq_{b} f(a\tau b0)$ (and thus $f(ab0) \not\leftrightarroweq_{rb} f(a\tau b0)$), since $f(ab0) \xrightarrow{c} 0$, while $f(a\tau b0) \stackrel{\alpha}{\not\longrightarrow}$ for $\alpha \in \{c, \tau\}$.

Example 3. We extend P with the following rule:

$$\frac{x \xrightarrow{a} y}{f(x) \xrightarrow{a} f(y)}$$

The argument of f must be $\aleph$-frozen, in view of the restriction in Def. 16 that for each $\aleph$-liquid argument there is an $\aleph$-patience rule. The rule above is not rooted branching bisimulation safe. Namely, if the argument of f is Λ-frozen, then y occurs both as the right-hand side of a premise and Λ-frozen in the right-hand side of the conclusion, violating restriction 2. And if the argument of f is Λ-liquid, then x occurs Λ-liquid and $\aleph$-frozen in the source and $\aleph$-liquid in the left-hand side of the premise, violating restriction 3(d). We have $f(aa0) \not\leftrightarroweq_{b} f(a\tau a0)$, since $f(aa0) \xrightarrow{a} f(a0) \xrightarrow{a} f(0)$, while $f(a\tau a0)$ can only do an a-transition to $f(\tau a0)$, and $f(\tau a0) \stackrel{\alpha}{\not\longrightarrow}$ for $\alpha \in \{a, \tau\}$.

Example 4. We extend P with the following rules:

$$\frac{x \xrightarrow{\tau} y}{f(x) \xrightarrow{\tau} f(y)} \qquad \frac{x \xrightarrow{a} y}{f(x) \xrightarrow{a} f(y)} \qquad \frac{x \stackrel{a}{\not\longrightarrow}}{f(x) \xrightarrow{c} 0}$$

The argument of f has to be Λ-liquid, for else the first and second rule would violate restriction 2. It even has to be $\aleph$-liquid, for otherwise these rules would violate requirement 3(d). However, with the argument of f $\aleph$-liquid, the third rule is not rooted branching bisimulation safe, because x occurs $\aleph$-liquid both in the source and in the left-hand side of the negative premise, violating restriction 3(b). We have $f(aa0) \not\leftrightarroweq_{b} f(a\tau a0)$, since $f(a\tau a0) \xrightarrow{a} f(\tau a0) \xrightarrow{c} 0$, while $f(aa0)$ can only do an a-transition to $f(a0)$, and $f(a0) \stackrel{\alpha}{\not\longrightarrow}$ for $\alpha \in \{c, \tau\}$.

Example 5. We extend P with the following rules:

$$\frac{x \xrightarrow{\tau} y}{f(x) \xrightarrow{\tau} f(y)} \qquad \frac{x \xrightarrow{a} y}{f(x) \xrightarrow{a} f(y)} \qquad \frac{x \xrightarrow{\tau} y}{f(x) \xrightarrow{c} 0}$$

As in the previous example, the argument of f has to be $\aleph$-liquid. The third rule is not rooted branching bisimulation safe, because x occurs $\aleph$-liquid both in the source and in the left-hand side of the positive premise with label τ, violating restriction 3(c). We have $f(aa0) \not\leftrightarroweq_b f(a\tau a0)$, since $f(a\tau a0) \xrightarrow{a} f(\tau a0) \xrightarrow{c} 0$, while $f(aa0)$ can only do an a-transition to $f(a0)$, and $f(a0) \stackrel{\alpha}{\not\longrightarrow}$ for $\alpha \in \{c, \tau\}$.

Example 6. We extend P with the following rules:

$$\eta \xrightarrow{a} 0 \qquad \zeta \xrightarrow{b} 0 \qquad \eta \xrightarrow{\tau} \zeta \qquad \zeta \xrightarrow{\tau} \eta \qquad \nu \xrightarrow{a} 0 \qquad \nu \xrightarrow{b} 0$$

$$\frac{x \xrightarrow{\tau} y}{f(x) \xrightarrow{\tau} f(y)} \qquad \frac{x \xrightarrow{a} y \quad x \xrightarrow{b} z}{f(x) \xrightarrow{c} 0}$$

Again, the argument of f is $\aleph$-liquid. The last rule is not rooted branching bisimulation safe, because x has an $\aleph$-liquid occurrence in the source, and two $\aleph$-liquid occurrences in the left-hand sides of the premises, violating restriction 3(c). Clearly, $\tau\nu \leftrightarroweq_{rb} \tau\eta$. However, $f(\tau\nu) \not\leftrightarroweq_b f(\tau\eta)$, since $f(\tau\nu) \xrightarrow{\tau} f(\nu) \xrightarrow{c} 0$, while $f(\tau\eta)$ only exhibits an infinite sequence of τ-transitions: $f(\tau\eta) \xrightarrow{\tau} f(\eta) \xrightarrow{\tau} f(\zeta) \xrightarrow{\tau} f(\eta) \xrightarrow{\tau} \cdots$.

Example 7. In the TSS from Example 6, we replace the last rule with the following rules:

$$f(x) \xrightarrow{\tau} g(x) \qquad \frac{x \xrightarrow{a} y \quad x \xrightarrow{b} z}{g(x) \xrightarrow{c} 0}$$

The argument of f is again $\aleph$-liquid, and the argument of g must be Λ-frozen, as otherwise the second rule would violate restriction 3(c). The first rule above is not rooted branching bisimulation safe, because x occurs $\aleph$-liquid (hence Λ-liquid) in the source and Λ-frozen in the right-hand side of the conclusion, violating restriction 3(a). We have $f(\tau\nu) \not\leftrightarroweq_b f(\tau\eta)$, since $f(\tau\nu) \xrightarrow{\tau} f(\nu) \xrightarrow{\tau} g(\nu) \xrightarrow{c} 0$, while $f(\tau\eta)$ can only perform τ-transitions.

Example 8. In the TSS from Example 6, we replace the last rule with the following rules:

$$\frac{x \xrightarrow{a} y \quad x \xrightarrow{b} z}{g(x) \xrightarrow{c} 0} \qquad \frac{g(x) \xrightarrow{c} y}{f(x) \xrightarrow{c} 0}$$

As in the previous example, the argument of f must be $\aleph$-liquid, and the argument of g Λ-frozen. The last rule above is not rooted branching bisimulation safe, because x occurs $\aleph$-liquid in the source and Λ-frozen in the left-hand side of the premise, violating restriction 3(a). We have $f(\tau\nu) \not\leftrightarroweq_b f(\tau\eta)$, since $f(\tau\nu) \xrightarrow{\tau} f(\nu) \xrightarrow{c} 0$, while $f(\tau\eta)$ can only perform τ-transitions.

7 Related Work

The first congruence formats for branching and rooted branching bisimulation were presented in [2], and reformulated in [13]. Those formats, which are contained in the GSOS format [4], distinguish so-called "principal" operators and "abbreviations". The latter can be regarded as syntactic sugar, adding nothing that could not be expressed with principal operators. Our formats are incomparable with the ones of [2, 13]. However, our formats generalise the result of simplifying the formats of [2, 13] by requiring all operators to be principal.

For the branching bisimulation format our generalisation consists of allowing transition rules outside the GSOS format; the simplified format of [2, 13] is exactly the intersection of our branching bisimulation format and the GSOS format. However, the intersection of our rooted branching bisimulation format and the GSOS format is still a proper generalisation of the simplified format for rooted branching bisimulation of [2, 13]. The latter can be described as the intersection of our rooted branching bisimulation format and the GSOS format in which all arguments of all operators that occur in right-hand sides of conclusions of transition rules are required to be Λ-liquid.

The format of [2, 13] for rooted branching bisimulation distinguishes "tame" and "wild" function symbols. In terms of our approach, wild operators have only Λ-frozen arguments, and tame operators only Λ-liquid arguments. The idea to allow operators with both kinds of arguments stems from [7].

In [7] a format for rooted branching bisimulation was proposed that generalises the simplified format of [2, 13]. Given that it applies to TSSs with predicates, it is incomparable with our current rooted branching bisimulation format. However, predicates can easily be encoded in terms of transitions, and when disregarding predicates, our current format is more general than the format of [7]. Still, the format of [7] strictly contains the intersection of our format with the GSOS format, and all applications of our work discussed in Sect. 5 fall within that intersection.

In [10] we apply the techniques of the current paper to derive congruence formats for η- and rooted η-bisimulation. These formats differ from the ones of the current paper only in restriction 2 of Def. 15. There it is required that right-hand sides of premises occur only $\aleph$-liquid in u, whereas here we merely require Λ-liquidity. That the rooted branching bisimulation format is essentially more general than the rooted η-bisimulation format is illustrated by the action refinement example of Sec. 5.4. $\mathrm{BPA}_{\delta}^{tick}$ with action refinement falls outside the rooted η-bisimulation format, due to the fact that the second argument of sequential composition needs to be $\aleph$-liquid in order for the second action refinement rule to be rooted η-bisimulation safe. Indeed, this operator fails to be compositional for rooted η-bisimulation [14].

References

1. J.A. Bergstra & J.W. Klop (1984): *Process algebra for synchronous communication. Information and Control* 60(1/3), pp. 109–137.
2. B. Bloom (1995): *Structural operational semantics for weak bisimulations. Theoretical Computer Science* 146(1/2), pp. 25–68.
3. B. Bloom, W.J. Fokkink & R.J. van Glabbeek (2004): *Precongruence formats for decorated trace semantics. ACM Transactions on Computational Logic* 5(1), pp. 26–78.

4. B. Bloom, S. Istrail & A.R. Meyer (1995): *Bisimulation can't be traced. Journal of the ACM* 42(1), pp. 232–268.
5. R.N. Bol & J.F. Groote (1996): *The meaning of negative premises in transition system specifications. Journal of the ACM* 43(5), pp. 863–914.
6. R. De Nicola & F.W. Vaandrager (1995): *Three logics for branching bisimulation. Journal of the ACM* 42(2), pp. 458–487.
7. W.J. Fokkink (2000): *Rooted branching bisimulation as a congruence. Journal of Computer and System Sciences* 60(1), pp. 13–37.
8. W.J. Fokkink & R.J. van Glabbeek (1996): *Ntyft/ntyxt rules reduce to ntree rules. Information and Computation* 126(1), pp. 1–10.
9. W.J. Fokkink, R.J. van Glabbeek & P. de Wind (2005): *Compositionality of Hennessy-Milner logic by structural operational semantics.* Available at `http://theory.stanford.edu/~rvg/abstracts.html#61`. To appear in *Theoretical Computer Science*. Extended abstract appeared in: *Proc. FCT'03*, LNCS 2751, Springer, 2003, pp. 412–422.
10. W.J. Fokkink, R.J. van Glabbeek & P. de Wind (2005): *Divide and congruence applied to η-bisimulation.* In *Proc. SOS'05*, To appear, ENTCS. Elsevier.
11. R.J. van Glabbeek (1993): *The linear time-branching time spectrum II: The semantics of sequential systems with silent moves.* In *Proc. CONCUR'93*, LNCS 715, pp. 66–81. Springer.
12. R.J. van Glabbeek (2004): *The meaning of negative premises in transition system specifications II. Journal of Logic and Algebraic Programming* 60/61, pp. 229–258.
13. R.J. van Glabbeek (2005): *On cool congruence formats for weak bisimulations (extended abstract).* In *Proc. ICTAC'05*, LNCS 3722, pp. 331–346. Springer.
14. R.J. van Glabbeek & W.P. Weijland (1996): *Branching time and abstraction in bisimulation semantics. Journal of the ACM* 43(3), pp. 555–600.
15. J.F. Groote (1993): *Transition system specifications with negative premises. Theoretical Computer Science* 118(2), pp. 263–299.
16. J.F. Groote & F.W. Vaandrager (1992): *Structured operational semantics and bisimulation as a congruence. Information and Computation* 100(2), pp. 202–260.
17. M.C.B. Hennessy & R. Milner (1985): *Algebraic laws for non-determinism and concurrency. Journal of the ACM* 32(1), pp. 137–161.
18. S.C. Kleene: *Representation of events in nerve nets and finite automata.* In (C. Shannon and J. McCarthy, eds.) *Automata Studies*, pp. 3–41. Princeton University Press, 1956.
19. K.G. Larsen & X. Liu (1991): *Compositionality through an operational semantics of contexts. Journal of Logic and Computation* 1(6), pp. 761–795.
20. G.D. Plotkin (2004): *A structural approach to operational semantics. Journal of Logic and Algebraic Programming* 60/61, pp. 17–139. Originally appeared in 1981.
21. R. de Simone (1985): *Higher-level synchronising devices in* Meije–*SCCS. Theoretical Computer Science* 37(3), pp. 245–267.

A Modal Characterisation of Branching Bisimulation

We prove the first part of Thm. 1, which states that $\mathbb{O}_b$ is a modal characterisation of branching bisimulation equivalence. The proof is based on [6]. We need to prove, given an LTS $(\mathbb{P}, \rightarrow)$, that $p \leftrightarroweq_b q \Leftrightarrow p \sim_{\mathbb{O}_b} q$ for all $p, q \in \mathbb{P}$.

Proof. ($\Rightarrow$) Suppose $p \leftrightarroweq_b q$, and $p \models \varphi$ for some $\varphi \in \mathbb{O}_b$. We prove $q \models \varphi$, by structural induction on φ. The reverse implication ($q \models \varphi$ implies $p \models \varphi$) follows by symmetry.

- $\varphi = \bigwedge_{i\in I} \varphi_i$. Then $p \models \varphi_i$ for $i \in I$. By induction $q \models \varphi_i$ for $i \in I$, so $q \models \bigwedge_{i\in I} \varphi_i$.
- $\varphi = \neg\varphi'$. Then $p \not\models \varphi'$. By induction $q \not\models \varphi'$, so $q \models \neg\varphi'$.
- $\varphi = \langle\epsilon\rangle(\varphi_1\langle\hat{\tau}\rangle\varphi_2)$. Then for some n there are $p_0, \ldots, p_n \in \mathbb{P}$ with $p_0 = p$, $p_i \xrightarrow{\tau} p_{i+1}$ for $i \in \{0, \ldots, n-1\}$, and $p_n \models \varphi_1\langle\hat{\tau}\rangle\varphi_2$. We apply induction on n.

  $n = 0$ Then $p \models \varphi_1$, so by induction on formula size, $q \models \varphi_1$. Furthermore, either (1) $p \models \varphi_2$ or (2) there is a $p' \in \mathbb{P}$ with $p \xrightarrow{\tau} p'$ and $p' \models \varphi_2$. In case (1), by induction on formula size, $q \models \varphi_2$, so $q \models \langle\epsilon\rangle(\varphi_1\langle\hat{\tau}\rangle\varphi_2)$. In case (2), since $p \leftrightarroweq_b q$, by Def. 1 either (2.1) $p' \leftrightarroweq_b q$ or (2.2) $q \overset{\epsilon}{\Longrightarrow} q' \xrightarrow{\tau} q''$ with $p \leftrightarroweq_b q'$ and $p' \leftrightarroweq_b q''$. In case (2.1), by induction on formula size, $q \models \varphi_2$. In case (2.2), by induction on formula size, $q' \models \varphi_1$ and $q'' \models \varphi_2$. In both cases, $q \models \langle\epsilon\rangle(\varphi_1\langle\hat{\tau}\rangle\varphi_2)$.

  $n > 0$ Since $p \xrightarrow{\tau} p_1$, and $p \leftrightarroweq_b q$, according to Def. 1 there are two possibilities.

  1. Either $p_1 \leftrightarroweq_b q$. Since $p_1 \models \langle\epsilon\rangle(\varphi_1\langle\hat{\tau}\rangle\varphi_2)$, by induction on n, $q \models \langle\epsilon\rangle(\varphi_1\langle\hat{\tau}\rangle\varphi_2)$.
  2. Or $q \overset{\epsilon}{\Longrightarrow} q' \xrightarrow{\tau} q''$ with $p_1 \leftrightarroweq_b q''$. Since $p_1 \models \langle\epsilon\rangle(\varphi_1\langle\hat{\tau}\rangle\varphi_2)$, by induction on n, $q'' \models \langle\epsilon\rangle(\varphi_1\langle\hat{\tau}\rangle\varphi_2)$. Hence $q \models \langle\epsilon\rangle(\varphi_1\langle\hat{\tau}\rangle\varphi_2)$.
- $\varphi = \langle\epsilon\rangle(\varphi_1\langle a\rangle\varphi_2)$. Then for some n there are $p_0, \ldots, p_n \in \mathbb{P}$ with $p_0 = p$, $p_i \xrightarrow{\tau} p_{i+1}$ for $i \in \{0, \ldots, n-1\}$, and $p_n \models \varphi_1\langle a\rangle\varphi_2$. We apply induction on n.

  $n = 0$ Then $p \models \varphi_1$, and there is a $p' \in \mathbb{P}$ with $p \xrightarrow{a} p'$ and $p' \models \varphi_2$. Since $p \leftrightarroweq_b q$, by Def. 1 $q \overset{\epsilon}{\Longrightarrow} q' \xrightarrow{a} q''$ with $p \leftrightarroweq_b q'$ and $p' \leftrightarroweq_b q''$. By induction on formula size, $q' \models \varphi_1$ and $q'' \models \varphi_2$. Hence $q \models \langle\epsilon\rangle(\varphi_1\langle a\rangle\varphi_2)$.

  $n > 0$ Since $p \xrightarrow{\tau} p_1$, and $p \leftrightarroweq_b q$, according to Def. 1 there are two possibilities.

  1. Either $p_1 \leftrightarroweq_b q$. Since $p_1 \models \langle\epsilon\rangle(\varphi_1\langle a\rangle\varphi_2)$, by induction on n, $q \models \langle\epsilon\rangle(\varphi_1\langle a\rangle\varphi_2)$.
  2. Or $q \overset{\epsilon}{\Longrightarrow} q' \xrightarrow{\tau} q''$ with $p_1 \leftrightarroweq_b q''$. Since $p_1 \models \langle\epsilon\rangle(\varphi_1\langle a\rangle\varphi_2)$, by induction on n, $q'' \models \langle\epsilon\rangle(\varphi_1\langle a\rangle\varphi_2)$. Hence $q \models \langle\epsilon\rangle(\varphi_1\langle a\rangle\varphi_2)$.

We conclude that $p \sim_{\mathbb{O}_b} q$.

($\Leftarrow$) We prove that $\sim_{\mathbb{O}_b}$ is a branching bisimulation relation. The relation is clearly symmetric. Let $p \sim_{\mathbb{O}_b} q$. Suppose $p \xrightarrow{\alpha} p'$. If $\alpha = \tau$ and $p' \sim_{\mathbb{O}_b} q$, then the first condition of Def. 1 is fulfilled. So we can assume that either (i) $\alpha \neq \tau$ or (ii) $p' \not\sim_{\mathbb{O}_b} q$. We define two sets:

$$\begin{aligned}
Q' &= \{q' \in \mathbb{P} \mid q \overset{\epsilon}{\Longrightarrow} q' \wedge p \not\sim_{\mathbb{O}_b} q'\} \\
Q'' &= \{q'' \in \mathbb{P} \mid \exists q' \in \mathbb{P} : q \overset{\epsilon}{\Longrightarrow} q' \xrightarrow{\alpha} q'' \wedge p' \not\sim_{\mathbb{O}_b} q''\}
\end{aligned}$$

For each $q' \in Q'$, let $\varphi_{q'}$ be a formula in $\mathbb{O}_b$ such that $p \models \varphi_{q'}$ and $q' \not\models \varphi_{q'}$. (Such a formula always exists because $\mathbb{O}_b$ is closed under negation $\neg$.) We define

$$\varphi = \bigwedge_{q' \in Q'} \varphi_{q'}$$

Similarly, for each $q'' \in Q''$, let $\psi_{q''}$ be a formula in $\mathbb{O}_b$ such that $p' \models \psi_{q''}$ and $q'' \not\models \psi_{q''}$. We define

$$\psi = \bigwedge_{q'' \in Q''} \psi_{q''}$$

Clearly, $\varphi, \psi \in \mathbb{O}_b$, $p \models \varphi$ and $p' \models \psi$. We distinguish two cases.

1. $\alpha \neq \tau$. Since $p \models \langle\epsilon\rangle(\varphi\langle\alpha\rangle\psi) \in \mathbb{O}_b$ and $p \sim_{\mathbb{O}_b} q$, also $q \models \langle\epsilon\rangle(\varphi\langle\alpha\rangle\psi)$. Hence $q \overset{\epsilon}{\Longrightarrow} q' \xrightarrow{\alpha} q''$ with $q' \models \varphi$ and $q'' \models \psi$. By the definition of φ and ψ it follows that $p \sim_{\mathbb{O}_b} q'$ and $p' \sim_{\mathbb{O}_b} q''$.
2. $\alpha = \tau$ and $p' \not\sim_{\mathbb{O}_b} q$. Let $\tilde{\varphi} \in \mathbb{O}_b$ such that $p' \models \tilde{\varphi}$ and $p, q \not\models \tilde{\varphi}$. Since $p \models \langle\epsilon\rangle(\varphi\langle\hat{\tau}\rangle(\tilde{\varphi} \wedge \psi)) \in \mathbb{O}_b$ and $p \sim_{\mathbb{O}_b} q$, also $q \models \langle\epsilon\rangle(\varphi\langle\hat{\tau}\rangle(\tilde{\varphi} \wedge \psi))$. So $q \overset{\epsilon}{\Longrightarrow} q'$ with $q' \models \varphi\langle\hat{\tau}\rangle(\tilde{\varphi} \wedge \psi)$. By definition of φ it follows that $p \sim_{\mathbb{O}_b} q'$. Thus $q' \not\models \tilde{\varphi}$, so $q' \xrightarrow{\tau} q''$ with $q'' \models \tilde{\varphi} \wedge \psi$. By the definition of ψ it follows that $p' \sim_{\mathbb{O}_b} q''$.

Both cases imply that the second condition of Def. 1 is fulfilled. We therefore conclude that $\sim_{\mathbb{O}_b}$ is a branching bisimulation relation. $\square$

Using the first part of Thm. 1, which was proved above, it is not hard to derive the second part of Thm. 1, i.e. that $\mathbb{O}_{rb}$ is a modal characterisation of rooted branching bisimulation equivalence.