Unique Solutions of Guarded Recursive Equations

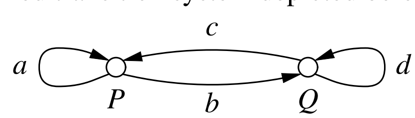
Rob van Glabbeek*
School of Informatics, University of Edinburgh, UK
School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
rvg@cs.stanford.edu

This paper shows that guarded systems of recursive equations have unique solutions up to strong bisimilarity for any process algebra with a structural operation semantics in the ready simulation format. A similar result holds for simulation equivalence, for ready simulation equivalence and for the (ready) simulation preorder. As a consequence, these equivalences and preorders are full (pre) congruences for guarded recursion. Moreover, the unique-solutions result yields a sound and ground-complete axiomatisation of strong bisimilarity for any finitary GSOS language.

1 Introduction

In many process algebras, including CCS [30], CSP [10] and ACP [3], infinite processes are specified by systems $\mathcal{S}$ of recursive equations $\{X = \mathcal{S}_X \mid X \in V_{\mathcal{S}}\}$. Such systems are called *recursive specifications*. Here $V_{\mathcal{S}}$ is a set of recursion variables specific to a recursive specification, and each $X \in V_{\mathcal{S}}$ is defined to behave as the expression $\mathcal{S}_X$. Crucially, X itself, and other variables from $V_{\mathcal{S}}$, may occur in the expression $\mathcal{S}_X$. An important tool in equational reasoning is that a large class of recursive specifications have unique solutions. Here a *solution* is a tuple $\vec{P}$ of processes, one for each variable X in the set $V_{\mathcal{S}}$, such that filling in the X-component of $\vec{P}$ for X, at both the left- and the right-hand side of the recursive equations in $\mathcal{S}$, yields a set of semantically valid statements.

Example 1 Using the syntax of CCS [30], let $\mathcal{S}$ be the recursive specification consisting of the two recursive equations $\{X = a.X + b.Y \ , \ Y = c.X + d.Y\}$. In the model of labelled transition systems [30], the pair (P,Q) of states in the labelled transition system depicted below constitutes a solution of $\mathcal{S}$.



In fact, this is the only solution of $\mathcal{S}$ up to strong bisimilarity [30].

Unfortunately, not all recursive specifications have unique solutions. The recursive equation $X = X$, for instance, allows any process as a solution. MILNER [30] formulated a syntactic criterion on recursive specifications called *guardedness*, and showed that in the process algebra CCS, systems of guarded recursive equations have unique solutions up to strong bisimilarity. In the setting of ACP [2, 3] two reasoning principles are defined: the *Recursive Definition Principle* (RDP) states that any recursive specification has a solution, and the *Recursive Specification Principle* (RSP) states that guarded recursive specifications have at most one solution, up to a semantic equivalence. Whether these principles hold may depend on the process algebra and on the semantic equivalence employed. First of all it takes a semantic equivalence to determine whether a given tuple $\vec{P}$ is a solution of a recursive specification $\mathcal{S}$. Secondly,

*Supported by Royal Society Wolfson Fellowship RSWF\R1\221008

© R.J. van Glabbeek
This work is licensed under the
Creative Commons Attribution License.

if $\mathcal{S}$ is guarded and $\vec{P}$ and $\vec{Q}$ both are solutions, RSP demands that $\vec{P}$ and $\vec{Q}$ are semantically equivalent. RDP and RSP are equational and conditional equational laws that can be added as axioms to the equational theories of various process algebras. Doing so has been a crucial step in achieving complete axiomatisations [29, 4, 12, 21]—see also Section 11.

Subtly different definitions of guardedness appear in the literature. In [30] it was required that within the bodies $\mathcal{S}_X$ of recursive equations the recursion variables $Y \in V_\mathcal{S}$ may appear only within the scope of a CCS action prefixing operator $a.\_$. Here this action *guards* the recursion variable. In Example 1, $\mathcal{S}$ is guarded in this sense. In other process algebras one might allow other operators as guards. In ACP [3] for instance, the left merge $⫽$ is a good candidate. However, it guards only its second argument: the recursive specification $X = b ⫽ X$ is guarded, but $Y = Y ⫽ b$ is not. In general, a suitable definition of guardedness is whatever syntactic criterion allows one to obtain RSP for a chosen semantic equivalence. Proofs of RSP [30, 2] tend to depend to a large extent on the syntax and semantics of the process algebra involved.

The goal of this paper is to give a proof of RSP for strong bisimulation semantics for a large class of process algebras, including CCS and ACP, at once. I do this by characterising a format for transition system specifications in structural operational semantics, proposing the most general definition of guardedness I can find in that context, and formally show that all process algebras whose operational semantics fits that format satisfy RSP, for strong bisimilarity. They also satisfy RDP, but that is fairly trivial.

The most general format for standard process algebras (without probabilities, time, name-binding or other special features) that guarantees strong bisimilarity to be congruence is the ntyft/ntyxt format of GROOTE [23], augmented with recursion as in [20]. Let f (pronounced "after a") be a unary operator defined through the transition rules

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

for the specific action a and each action b. It has lookahead, in the sense that the initial actions of f are determined by looking ahead at the second action performed by its argument. Now the recursive equation $X = f(a.X)$ has any process as solution, even though the recursion variable X appears guarded (by the guard a) in the body $f(a.X)$ of the recursion equation. Because of this I rule out lookahead in this paper. Aiming for maximal generality, I will obtain RSP for strong bisimilarity in the ready simulation format [16], which is the ntyft/ntyxt format without lookahead, augmented with recursion.

Most applications of this work will fall in the GSOS format, which restricts the ready simulation format mainly by allowing only variables to occur in premises, except in the rules for recursion. In this context I favour a notion of guardedness due to VAANDRAGER [35]: an n-ary operator f guards its i^{th} argument iff there is no transition rule of the form

$$\frac{H}{f(x_1,\ldots,x_n) \xrightarrow{a} E}$$

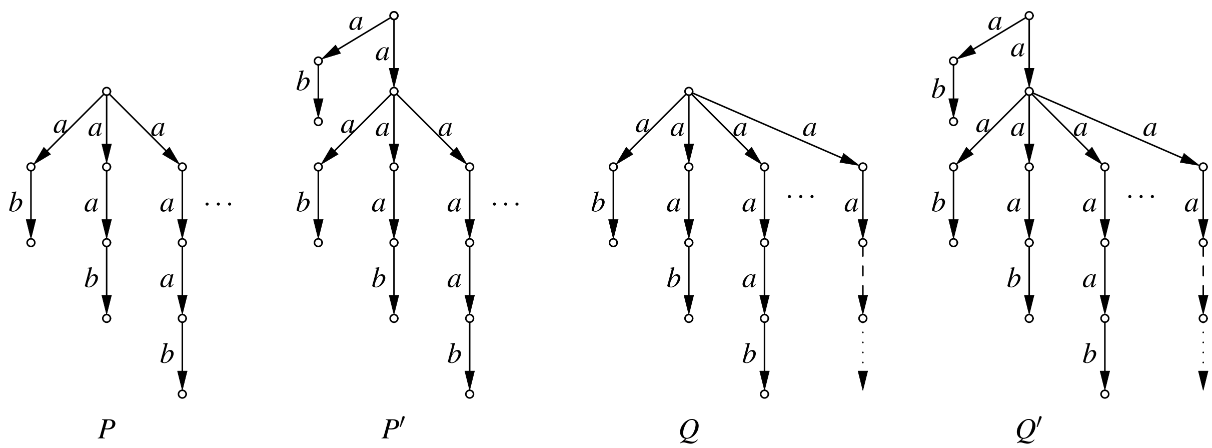
such that the variable x_i occurs as the left-hand side of one of the premises in H. I will generalise this idea to the ready simulation format by allowing x_i to occur in the left-hand sides of premises (which may be complex terms) but only in guarded positions.

After establishing RSP for strong bisimilarity, I observe that by means of an almost identical proof it can also be shown for other strong semantic equivalences at the branching time side of the linear time – branching time spectrum [18], namely simulation equivalence (in the absence of negative premises) and ready simulation equivalence. Naturally I would like to show it for preorders as well. Using the notation $\vec{P} \equiv \mathcal{S}[\vec{P}]$ to indicate that the tuple $\vec{P}$ is a solution of the recursive specification $\mathcal{S}$ up to the semantic equivalence $\equiv$, there are two ways to replace $\equiv$ by a preorder $\sqsubseteq$. When $\vec{P} \sqsubseteq \mathcal{S}[\vec{P}]$ holds I call $\vec{P}$ a presolution of $\mathcal{S}[\vec{P}]$, and when $\mathcal{S}[\vec{P}] \sqsubseteq \vec{P}$ holds I call it a postsolution. The form of RSP for $\equiv$ simply says that if $\vec{P}$ and $\vec{Q}$ are both solutions of $\mathcal{S}$ then $\vec{P} \equiv \vec{Q}$. It turns out that no interesting conclusions can be drawn from the assumption that $\vec{P}$ and $\vec{Q}$ are both presolutions, or both postsolutions. However, an asymmetric form of RSP holds if $\vec{P}$ is a presolution and $\vec{Q}$ a postsolution. In that case $\vec{P} \sqsubseteq \vec{Q}$, at least when taking for $\sqsubseteq$ the strong simulation preorder or the ready simulation preorder. I thus formulate RSP as

$$\frac{\vec{P} \equiv \mathcal{S}[\vec{P}] \qquad \mathcal{S}[\vec{Q}] \equiv \vec{Q}}{\vec{P} \equiv \vec{Q}} \qquad \frac{\vec{P} \sqsubseteq \mathcal{S}[\vec{P}] \qquad \mathcal{S}[\vec{Q}] \sqsubseteq \vec{Q}}{\vec{P} \sqsubseteq \vec{Q}}. \tag{1}$$

In most cases a semantic equivalence $\equiv$ (say simulation equivalence) arises as the kernel of a preorder, meaning that $P \equiv Q$ iff $P \sqsubseteq Q$ and $Q \sqsubseteq P$. In this setting, RSP for $\equiv$ is a simple consequence of RSP for $\sqsubseteq$. Thus, one merely needs to prove the latter.

For linear time equivalences, a generic counterexample to RSP is given by these processes:



In CCS [30], process P can be expressed as $P := \sum_{i=1}^{\infty} a^i.b.\mathbf{0}$, where the abbreviation $a^i.R$ is defined by $a^0.R := R$ and $a^{i+1}.R := a.a^i.R$. This process has branches featuring any finite positive number of a-transitions in succession, and each of those branches ends with a b-transition. Process $Q := P + a^\infty$ is like P, except that it also has a branch with infinitely many a-transitions, thus not ending with a b. The process a^∞ can be defined as the unique solution of the guarded recursive equation $X = a.X$. Finally, $P' := a.b + a.P$ and $Q' := a.b + a.Q$.

In all semantics $\equiv$ of [18] between trace and ready trace equivalence one has $P' \equiv P$ and $Q \equiv Q'$. Thus, the guarded recursive specification $\mathcal{S} = \{X = a.b + a.X\}$ has both P and Q as solutions. Hence for all such semantic equivalences that distinguish between P and Q in models of concurrency where P and Q both exist, RSP fails. This applies to semantics that take infinite traces (or infinite ready traces) into account. A reason to distinguish between P and Q is that P is guaranteed to eventually perform a b-transition, whereas Q is not. For linear time semantics that do not take infinite traces into account, and identify P and Q, RSP may hold. But that is best established by means of fixed point arguments [34], which is out of scope for this paper.

After establishing RSP and RDP for bisimilarity and (ready) simulation equivalence, as well as for the corresponding preorders, I establish two corollaries, namely (1) that these equivalences and preorders are full (pre)congruences for guarded recursion, and (2) a sound and ground-complete axiomatisation of strong bisimilarity for any finitary GSOS language.

2 Transition system specifications and their meaning

In this paper *Var* and A are two sets of *variables* and *actions*. Many concepts that will appear are parameterised by the choice of *Var* and A, but as here this choice is fixed, a corresponding index is suppressed.

Definition 1 (*Terms*) A *signature* is a set Σ of operator symbols $f \notin Var$, each equipped with an *arity* $ar(f) \in \mathbb{N}$.[1] The set $\mathbb{T}(\Sigma)$ of *terms with recursion* over a signature Σ is defined inductively by:

- $Var \subseteq \mathbb{T}(\Sigma)$,
- if $f \in \Sigma$ and $E_1,\dots,E_{ar(f)} \in \mathbb{T}(\Sigma)$ then $f(E_1,\dots,E_{ar(f)}) \in \mathbb{T}(\Sigma)$,
- If $V_{\mathcal{S}} \subseteq Var$, $\mathcal{S} : V_{\mathcal{S}} \rightarrow \mathbb{T}(\Sigma)$ and $X \in V_{\mathcal{S}}$, then $\langle X|\mathcal{S}\rangle \in \mathbb{T}(\Sigma)$.

A term $c()$ is abbreviated as c. A term $\langle X|\mathcal{S}\rangle$ as appears in the last clause is a *recursive call*, and the function $\mathcal{S}$ therein is called a *recursive specification*. It is often displayed as $\{X = \mathcal{S}_X \mid X \in V_{\mathcal{S}}\}$. So $V_{\mathcal{S}}$ is the domain of $\mathcal{S}$ and $\mathcal{S}_X$ represents $\mathcal{S}(X)$. Each term $\mathcal{S}_Y$ for $Y \in V_{\mathcal{S}}$ counts as a subterm of $\langle X|\mathcal{S}\rangle$. An occurrence of a variable y in a term E is *free* if it does not occur in a subterm of E of the form $\langle X|\mathcal{S}\rangle$ with $y \in V_{\mathcal{S}}$. Let $var(E)$ denote the set of variables occurring free in a term $E \in \mathbb{T}(\Sigma)$, and for $W \subseteq Var$ let $\mathbb{T}(\Sigma,W)$ be the set of terms E over Σ with $var(E) \subseteq W$. $\mathrm{T}(\Sigma) := \mathbb{T}(\Sigma,\emptyset)$ is the set of *closed* terms over Σ, modelling processes.

Example 2 Let Σ contain three unary operators $a.\_$, $b.\_$ and $d.\_$, and an infix-written binary operator $\|$. Let $X,Y,z \in Var$. Then $\mathcal{S} = \{X = (a.X)\|(b.Y),\ Y = (d.Y)\|(X\|z)\}$ is a recursive specification, so $\langle X|\mathcal{S}\rangle \in \mathbb{T}(\Sigma)$. Since $V_{\mathcal{S}} = \{X,Y\}$, the only variable that occurs free in this term is z.

As illustrated here, I often choose upper case letters for bound variables (the ones occurring in a set $V_{\mathcal{S}}$) and lower case ones for variables occurring free; this is a convention only.

A recursive specification $\mathcal{S}$ is meant to denote a $V_{\mathcal{S}}$-tuple (in the example above a pair) of processes that—when filled in for the variables in $V_{\mathcal{S}}$—forms a solution to the equations in $\mathcal{S}$.[2] The term $\langle X|\mathcal{S}\rangle$ denotes the X-component of such a tuple.

Definition 2 (*Substitution*) A Σ-*substitution* σ is a partial function from *Var* to $\mathbb{T}(\Sigma)$; it is *closed* if it is a total function from *Var* to $\mathrm{T}(\Sigma)$. If σ is a substitution and S any syntactic object, then $S[\sigma]$ denotes the object obtained from S by replacing, for x in the domain of σ, every free occurrence of x in S by $\sigma(x)$, while renaming bound variables if necessary to prevent name-clashes. In that case $S[\sigma]$ is called a *substitution instance* of S. A substitution instance $S[\sigma]$ where σ is given by $\sigma(x_i) = u_i$ for $i \in I$ is denoted as $S[u_i/x_i]_{i\in I}$, and for a recursive specification $\mathcal{S}$, the expression $\langle E|\mathcal{S}\rangle$ abbreviates $E[\langle Y|\mathcal{S}\rangle/Y]_{Y\in V_{\mathcal{S}}}$.

Example 3 Extend Σ from Example 2 with a constant c. Then
$\langle X|\mathcal{S}\rangle[b.c/z] = \langle X|\{X{=}(a.X)\|(b.Y),\ Y{=}(d.Y)\|(X\|b.c)\}\rangle$,
$\langle X|\mathcal{S}\rangle[X/z] = \langle Z|\{Z{=}(a.Z)\|(b.Y),\ Y{=}(d.Y)\|(Z\|X)\}\rangle$ and
$\langle X|\mathcal{S}\rangle[b.c/Y] = \langle X|\mathcal{S}\rangle$.

Structural operational semantics [32] defines the meaning of system description languages whose syntax is given by a signature Σ. It generates a transition system in which the states, or *processes*, are the closed terms over Σ—representing the remaining system behaviour from that state—and transitions between processes are supplied with labels. The transitions between processes are obtained from a transition system specification, which consists of a set of transition rules.

[1] This work, prior to Section 9, generalises seamlessly to operators with infinitely many arguments. Such operators occur, for instance, in [9, Appendix A.2]. Hence one may take $ar(f)$ to be any ordinal. An operator, like the *summation* or *choice* of CCS [30], that actually takes any *set* of arguments, needs to be simulated by a family of operators with a *sequence* of arguments (but yielding the same value upon reshuffling of the arguments), one for each cardinality of this set.

[2] When $\mathcal{S}$ contains free variables from a set W, this solution is parameterised by the choice of a valuation of these variables as processes, thereby becoming a W-ary function.

Definition 3 (***Transition system specifications***) Let Σ be a signature. A *positive* Σ*-literal* is an expression $E \xrightarrow{a} E'$ and a *negative* Σ*-literal* an expression $E \stackrel{a}{\nrightarrow}$ with $E, E' \in \mathbb{T}(\Sigma)$ and $a \in A$. For $E, E' \in \mathbb{T}(\Sigma)$ the literals $E \xrightarrow{a} E'$ and $E \stackrel{a}{\nrightarrow}$ are said to *deny* each other. A *transition rule* over Σ is an expression of the form $\frac{H}{\alpha}$ with H a set of Σ-literals (the *premises* or *antecedents* of the rule) and α a positive Σ-literal (the *conclusion*). The terms at the left- and right-hand side of α are the *source* and *target* of the rule. A rule $\frac{H}{\alpha}$ with $H = \emptyset$ is also written α. A literal or transition rule is *closed* if it contains no free variables. A *transition system specification (TSS)* is a pair $(\Sigma, \mathcal{R})$ with Σ a signature and $\mathcal{R}$ a set of transition rules over Σ; it is *positive* if all antecedents of its rules are positive.

The concept of a (positive) TSS presented above was introduced in GROOTE & VAANDRAGER [24]; the negative premises $E \stackrel{a}{\nrightarrow}$ were added in GROOTE [23]. The notion generalises the *GSOS rule systems* of [7] and constitutes a formalisation of PLOTKIN's *Structural Operational Semantics (SOS)* [32] that is sufficiently general to cover many of its applications.

Example 4 Let Σ feature a constant 0, unary operators $a.\_$ for a ranging over the set A of actions, binary operators $\_ \triangleright \_$ and $\_ \|_S \_$ for each $S \subseteq A$, and unary operators $en_R(\_)$ for each $R \subseteq A$. Their semantics is given by the following set of transition rules:

$$a.x \xrightarrow{a} x \qquad \frac{x \xrightarrow{a} x'}{x \|_S y \xrightarrow{a} x' \|_S y}\ {\scriptstyle (a \notin S)} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \|_S y \xrightarrow{a} x' \|_S y'}\ {\scriptstyle (a \in S)} \qquad \frac{y \xrightarrow{a} y'}{x \|_S y \xrightarrow{a} x \|_S y'}\ {\scriptstyle (a \notin S)} \qquad \frac{\langle \mathcal{S}_X | \mathcal{S} \rangle \xrightarrow{a} z}{\langle X | \mathcal{S} \rangle \xrightarrow{a} z}$$

$$\frac{x \xrightarrow{a} x'}{en_R(x) \xrightarrow{a} en_R(x')} \qquad \frac{x \stackrel{a}{\nrightarrow}}{en_R(x) \xrightarrow{a} en_R(x)}\ {\scriptstyle (a \in R)} \qquad \frac{x \xrightarrow{a} x'}{x \triangleright y \xrightarrow{a} x' \triangleright y}\ {\scriptstyle (a \neq sw)} \qquad \frac{x \xrightarrow{sw} x'}{x \triangleright y \xrightarrow{sw} y \triangleright x'}$$

Here all rules displayed are really rule templates, with one instance for each choice of $a \in A$, except when this choice is restricted by the side condition. The operators 0 and $a.\_$ stem from CCS [30] and $\|_S$ from CSP [31]. The operators en_R and $\triangleright$ were invented to create this example.

The process 0 cannot perform any actions, and the process $a.P$ first performs the action a and then behaves as P. The process $P \|_S Q$ is a partially synchronous parallel composition of the processes P and Q, where actions $a \notin S$ from P and Q may occur independently, whereas actions $a \in S$ can occur only when both P and Q partake in such a synchronisation. The process $en_R(P)$ behaves like P, but with the modification that the actions from R must be enabled in any state. In a state where P cannot perform such an action, $en_R(P)$ adds it as a self-loop. The purpose of this operator is to take R to be those actions that are interpreted as receiving a message, so that the process $en_R(P)$ can receive a message in any state. In case P itself could not receive the message (by having an appropriate outgoing transition), then $en_R(P)$ adds the possibility of receiving and dropping said message. As a consequence, whereas in $P \|_S Q$ the process P can be blocked when it wants to "send" a message $b \in S$ that Q is not ready to receive, in $P \|_S en_S(Q)$ no such blocking can occur. Finally, the operator $P \triangleright Q$ lets P work while Q rests, until P performs the action sw(itch) $\in A$, after which Q works and P rests.

The following definition (from [16]) tells when a transition is provable from a TSS. It generalises the standard definition (see e.g. [24]) by (also) allowing the derivation of transition rules. The derivation of a transition $E \xrightarrow{a} E'$ corresponds to the derivation of the transition rule $\frac{H}{E \xrightarrow{a} E'}$ with $H = \emptyset$. The case $H \neq \emptyset$ corresponds to the derivation of $E \xrightarrow{a} E'$ under the assumptions H.

Definition 4 (***Proof***) Let $\mathcal{T} = (\Sigma, \mathcal{R})$ be a TSS. A *proof* of a transition rule $\frac{H}{\alpha}$ from $\mathcal{T}$ is a well-founded, upwardly branching tree of which the nodes are labelled by Σ-literals, such that:

- the root is labelled by α, and
- if β is the label of a node q and K is the set of labels of the nodes directly above q, then

- either $K = \emptyset$ and $\beta \in H$,
- or $\frac{K}{\beta}$ is a substitution instance of a rule from $\mathcal{R}$.

If a proof of $\frac{H}{\alpha}$ from $\mathcal{T}$ exists, then $\frac{H}{\alpha}$ is *provable* from $\mathcal{T}$, notation $\mathcal{T} \vdash \frac{H}{\alpha}$.

Example 5 In Example 4 one can prove that $a.b.0 \,\|_{\{a,b\}}\, en_{\{a,b\}}(b.a.0) \xrightarrow{a} b.0 \,\|_{\{a,b\}}\, en_{\{a,b\}}(b.a.0)$ and $b.0 \,\|_{\{a,b\}}\, en_{\{a,b\}}(b.a.0) \xrightarrow{b} 0 \,\|_{\{a,b\}}\, en_{\{a,b\}}(a.0)$. No further transitions from these three processes are provable. Let $\mathcal{S}$ be the recursive specification $X = b.0 \,\|_b\, en_b(X)$. Under the assumption that $\langle X|\mathcal{S}\rangle \xrightarrow{b} P$ one can now prove that $\langle X|\mathcal{S}\rangle \xrightarrow{b} 0 \,\|_b\, en_b(P)$, whereas under the assumption that $\langle X|\mathcal{S}\rangle \not\xrightarrow{b}$ one proves that $\langle X|\mathcal{S}\rangle \xrightarrow{b} 0 \,\|_b\, en_b(\langle X|\mathcal{S}\rangle)$. However, there is no outright proof of any transition $\langle X|\mathcal{S}\rangle \xrightarrow{b} Q$.

A TSS is meant to specify a transition system in which the transitions are closed positive literals. A positive TSS specifies a transition relation in a straightforward way as the set of all provable transitions.[3] But as pointed out in GROOTE [23], it is not so easy to associate a transition relation to a TSS with negative premises. In [19] several solutions to this problem were reviewed and evaluated. Arguably, the best method to assign a meaning to all TSSs is the *well-founded semantics* of VAN GELDER, ROSS & SCHLIPF [15], which in general yields a *3-valued transition relation* $T : \mathsf{T}(\Sigma) \times A \times \mathsf{T}(\Sigma) \to \{\mathit{present}, \mathit{undetermined}, \mathit{absent}\}$. I present such a relation as a pair $\langle CT, PT\rangle$ of 2-valued transition relations—the sets of *certain* and *possible transitions*—with $CT \subseteq PT$. When insisting on 2-valued transition relations, the best method is the same, declaring meaningful only those TSSs whose well-founded semantics is 2-valued, meaning that $CT = PT$. Such a TSS is called *complete* [19].

Below I follow the presentation from [20], which was strongly inspired by earlier accounts [33, 8, 19]. As Definition 4 does not allow the derivation of negative literals, to arrive at an approximation AT^+ of the set of transitions that are in the transition relation intended by a TSS $\mathcal{T}$, one could start from an approximation AT^- of the closed negative literals that ought to be generated, and define AT^+ as the set of closed positive literals provable from $\mathcal{T}$ under the hypotheses AT^-. Intuitively,

1. if AT^- is an under- (resp. over-)approximation of the closed negative literals that "really" hold, then AT^+ will be an under- (resp. over-)approximation of the intended (2-valued) transition relation, and
2. if AT^+ is an under- (resp. over-)approximation of the intended transition relation, then the set of all closed negative literals that do not deny any literal in AT^+ is an over- (resp. under-)approximation of the closed negative literals that agree with the intended transition relation.

Based on this insight, for each ordinal λ I define overapproximations PT^+_λ and PT^-_λ of the closed positive and negative literals that the transition relation intends to generate, and underapproximations CT^+_λ and CT^-_λ. The approximations get better with increasing λ. I start by taking the set of *all* closed negative literals as PT^-_0 and then apply 1 and 2 above to generate these approximations in the order $PT^-_\lambda \to PT^+_\lambda \to CT^-_\lambda \to CT^+_\lambda \to PT^-_{\lambda+1} \to \cdots$.

Definition 5 (*Over- and underapproximations of transition relations* [20]) Let $\mathcal{T}$ be a TSS. For ordinals λ the sets CT^+_λ and PT^+_λ of closed positive literals, and CT^-_λ, PT^-_λ of closed negative literals are defined inductively by:

PT^-_λ	is the set of negative literals that do not deny any $\beta \in CT^+_\kappa$ with $\kappa < \lambda$	$\beta \in PT^+_\lambda$ iff $\mathcal{T} \vdash \frac{PT^-_\lambda}{\beta}$
CT^-_λ	is the set of negative literals that do not deny any $\beta \in PT^+_\lambda$	$\beta \in CT^+_\lambda$ iff $\mathcal{T} \vdash \frac{CT^-_\lambda}{\beta}$.

[3] Readers interested only in the restriction of my results to TSSs without negative premises—giving rise to 2-valued transition relations—can safely skip the remainder of this section, and identify $p \overset{a}{\dashrightarrow} p'$ with $p \xrightarrow{a} p'$. In the proof of Lemma 2 the induction on λ can be skipped, as well as Claims 1, 2 and 3, and the proof proceeds directly by induction on π.

Lemma 1 ([20]) $CT^-_\kappa \subseteq CT^-_\lambda \subseteq PT^-_\lambda \subseteq PT^-_\kappa$ and $CT^+_\kappa \subseteq CT^+_\lambda \subseteq PT^+_\lambda \subseteq PT^+_\kappa$ for $\kappa < \lambda$.

Definition 6 (*Well-founded semantics* [20]) Define $PT^- := \bigcap_\lambda PT^-_\lambda$, taking the intersection over all ordinals. Likewise, $PT^+ := \bigcap_\lambda PT^+_\lambda$, $CT^- := \bigcup_\lambda CT^-_\lambda$ and $CT^+ := \bigcup_\lambda CT^+_\lambda$.

The 3-valued transition relation $\langle CT^+, PT^+\rangle$ constitutes the well-founded semantics of a TSS.

Since the closed literals over Σ form a proper set, there must be an ordinal κ such that $PT^-_\lambda = PT^-_\kappa$ for all $\lambda > \kappa$, and hence also $PT^+_\lambda = PT^+_\kappa$, $CT^-_\lambda = CT^-_\kappa$ and $CT^+_\lambda = CT^+_\kappa$. Such an ordinal κ is called a *closure ordinal*. Now $PT^- = PT^-_\kappa$, $PT^+ = PT^+_\kappa$, $CT^- = CT^-_\kappa$ and $CT^+ = CT^+_\kappa$. Trivially, one obtains:

Observation 1 $CT^- \subseteq PT^-$ $\qquad$ $CT^+ \subseteq PT^+$

PT^- is the set of negative literals that do not deny any $\beta \in CT^+$ $\qquad$ $\beta \in PT^+$ iff $\mathcal{T} \vdash \frac{PT^-}{\beta}$

CT^- is the set of negative literals that do not deny any $\beta \in PT^+$ $\qquad$ $\beta \in CT^+$ iff $\mathcal{T} \vdash \frac{CT^-}{\beta}$.

In [20] it is shown that the above account of the well-founded semantics is consistent with the one in [19], and thereby with the ones in [8, 33, 15].

Below the statements $P \xrightarrow{a}_\lambda Q$, $P \stackrel{a}{\not\longrightarrow}_\lambda$, $P \stackrel{a}{\dashrightarrow}_\lambda Q$ and $P \stackrel{a}{\not\dashrightarrow}_\lambda$ will abbreviate $(P \xrightarrow{a} Q) \in CT^+_\lambda$, $(P \stackrel{a}{\not\longrightarrow}) \in CT^-_\lambda$, $(P \xrightarrow{a} Q) \in PT^+_\lambda$ and $(P \stackrel{a}{\not\longrightarrow}) \in PT^-_\lambda$, respectively. Likewise, $P \xrightarrow{a} Q$ and $P \stackrel{a}{\dashrightarrow} Q$ abbreviate $(P \xrightarrow{a} Q) \in CT^+$, and $(P \xrightarrow{a} Q) \in PT^+$.

Example 6 For $\mathcal{S}$ the recursive specification of Example 5, we have $\langle X|\mathcal{S}\rangle \stackrel{a}{\not\longrightarrow}$ for all $a \neq b$, $\langle X|\mathcal{S}\rangle \stackrel{b}{\not\dashrightarrow}$ and $\langle X|\mathcal{S}\rangle \stackrel{b}{\dashrightarrow} 0 \parallel_b en_b(\langle X|\mathcal{S}\rangle)$. For this application the ordinal 0 is already a closure ordinal.

3 The ready simulation format with recursion

Definition 7 (*Ready simulation format with recursion*) An *ntyft/ntyxt rule* is a transition rule whose source contains at most one operator symbol and no multiple occurrences of the same variable, and in which the right-hand sides of positive premises are variables that are all distinct and do not occur in the source. It has *lookahead* if some variable both occurs in the right-hand side of a premise and in the left-hand side of a premise. A TSS is in the *ready simulation format with recursion* if for every recursive specification $\mathcal{S}$ and $X \in V_\mathcal{S}$ it has a rule

$$\frac{\langle \mathcal{S}_X|\mathcal{S}\rangle \xrightarrow{a} z}{\langle X|\mathcal{S}\rangle \xrightarrow{a} z}$$

with $z \in Var$ and all of its other rules are recursion-free[4] ntyft/ntyxt rules without lookahead.

Example 7 The TSS of Example 4 is in the GSOS format of [6] (cf. Definition 16), and thus certainly in the ready simulation format. To make it a non-GSOS example of a TSS in ready simulation format, add for instance a ternary operator h with a rule $\frac{en_b(x \triangleright y) \xrightarrow{a} x'}{h(x,y,z) \xrightarrow{a} h(z,x',y)}$ for each $a \in A$.

[4] i.e. without containing recursive calls at all. I conjecture that this restriction can be lifted at the expense of extra bookkeeping regarding bound variables. As I am not aware of any applications of such a generalisation I did not explore this further.

4 Guardedness

Definition 8 (Frozen arguments of operators [14]) Let Σ be a signature and Γ a unary predicate on the collection $\{(f,i) \mid 1 \leq i \leq ar(f),\ f \in \Sigma\}$ of arguments of operators. If $\Gamma(f,i)$, then argument i of f is $\overline{\Gamma}$-*frozen*; otherwise it is $\overline{\Gamma}$-*liquid*. An occurrence of a variable x in a term E is $\overline{\Gamma}$-*frozen* iff E contains a subterm $f(E_1,\dots,E_{ar(f)})$ such that the occurrence of x is in E_i for a $\overline{\Gamma}$-frozen argument i of f.

To prepare for the definition of guardedness below, I here use a predicate Γ that marks the frozen arguments of operators. However, I do not want to break the convention of [14], where a predicate Γ marks the liquid arguments. This is why I define frozen and liquid arguments and variable occurrences in terms of the complement $\overline{\Gamma}$ of my Γ.

Definition 9 Let $\mathcal{T}$ be a TSS in the ready simulation format with recursion. A predicate Γ as above *respects guardedness* for $\mathcal{T}$ if for each rule r of $\mathcal{T}$ with source $f(x_1,\dots,x_{ar(f)})$ and i a $\overline{\Gamma}$-frozen argument of f, the variable x_i only occurs $\overline{\Gamma}$-frozen in the left-hand sides of the premises of r.

Note that the union of any collection of predicates Γ that respect guardedness for $\mathcal{T}$ is itself a predicate that respects guardedness for $\mathcal{T}$. Hence there exists a largest predicate $\Gamma_{\mathcal{T}}$ that respects guardedness for $\mathcal{T}$. Henceforth, $\mathcal{T}$-*guarded* (or *guarded* when $\mathcal{T}$ is understood from context) will mean $\overline{\Gamma_{\mathcal{T}}}$-frozen.

Thus, if r is a rule of $\mathcal{T}$ with source $f(x_1,\dots,x_{ar(f)})$ and i is a guarded (= $\overline{\Gamma_{\mathcal{T}}}$-frozen) argument of f, then x_i only occurs guarded (= $\overline{\Gamma_{\mathcal{T}}}$-frozen) in the left-hand sides of the premises of r.

Definition 10 Let $\mathcal{T}$ be a TSS in the ready simulation format with recursion. An occurrence of a variable x in a term E is $\mathcal{T}$-*guarded* if it is $\overline{\Gamma}$-frozen for some predicate Γ on arguments of operators that respects guardedness for $\mathcal{T}$. A term E is $\mathcal{T}$-*guarded* if all free occurrences of variables in E are $\mathcal{T}$-guarded.

Example 8 For the LTS $\mathcal{T}$ of Example 7, the predicate $\Gamma_{\mathcal{T}}$ marks as $\overline{\Gamma_{\mathcal{T}}}$-frozen (1) the argument of the unary action prefixing operators $a.\_$, (2) the second argument of the binary operator $\triangleright$, (3) the third argument of h, and (4) the second argument of h. In case (2) this is because there is no rule with source $x \triangleright y$ in which y occurs in the left-hand side of a premise. Likewise, for (1), there is no rule with source $a.x$ in which x occurs in a premise—in fact, the only rule with source $a.x$ has no premises at all. Similarly, for (3), the only rule with source $h(x,y,z)$ lacks a premise containing z. Case (4) is slightly harder, as the only rule with source $h(x,y,z)$ does have a premise containing y. However, in the left-hand side of that premise y merely occurs $\overline{\Gamma_{\mathcal{T}}}$-frozen. Namely, in the term $en_b(x \triangleright y)$ the operator $\triangleright$ guards the occurrence of y. As a consequence, the term $h(a.x,x,x)$ is $\mathcal{T}$-guarded, as in it the variable x occurs $\mathcal{T}$-guarded only.

The next lemma, that for CCS goes back to MILNER [30], is a crucial tool in proving RSP. When creating or adapting a definition of guardedness, the main consideration is to make sure that this lemma holds. Below $\vec{P},\vec{Q} \in \mathrm{T}^W$ are W-tuples of processes; such tuples are also substitutions—hence the notation $G[\vec{P}]$.

Lemma 2 Let $G \in \mathbb{T}(\Sigma,W)$ be guarded and have free variables from $W \subseteq \mathit{Var}$ only, and let $\vec{P},\vec{Q} \in \mathrm{T}^W$. If $G[\vec{P}] \xrightarrow{a} R$ with $a \in A$, then R has the form $G'[\vec{P}]$ for some $G' \in \mathbb{T}(\Sigma,W)$. Moreover $G[\vec{Q}] \xrightarrow{a} G'[\vec{Q}]$. If $G[\vec{Q}] \stackrel{a}{\dashrightarrow} R$ with $a \in A$, then R has the form $G'[\vec{Q}]$ for some $G' \in \mathbb{T}(\Sigma,W)$. Moreover $G[\vec{P}] \stackrel{a}{\dashrightarrow} G'[\vec{P}]$.

Proof: Given $\vec{P},\vec{Q} \in \mathrm{T}^W$, I will show, for each ordinal λ, that for each guarded $G \in \mathbb{T}(\Sigma,W)$ and $a \in A$,

1. If $G[\vec{Q}] \stackrel{a}{\not\dashrightarrow}_\lambda$ then $G[\vec{P}] \stackrel{a}{\not\dashrightarrow}_\lambda$,
2. If $G[\vec{Q}] \stackrel{a}{\dashrightarrow}_\lambda R$ then R has the form $G'[\vec{Q}]$ for some $G' \in \mathbb{T}(\Sigma,W)$. Moreover $G[\vec{P}] \stackrel{a}{\dashrightarrow}_\lambda G'[\vec{P}]$,
3. If $G[\vec{P}] \stackrel{a}{\not\longrightarrow}_\lambda$ then $G[\vec{Q}] \stackrel{a}{\not\longrightarrow}_\lambda$, and
4. If $G[\vec{P}] \xrightarrow{a}_\lambda R$ then R has the form $G'[\vec{P}]$ for some $G' \in \mathbb{T}(\Sigma,W)$. Moreover $G[\vec{Q}] \xrightarrow{a}_\lambda G'[\vec{Q}]$.

The statements of the lemma then follow by taking λ to be a closure ordinal. I apply induction on λ.

1. Suppose $G[\vec{P}] \stackrel{a}{\not\dashrightarrow}_\lambda$ does not hold. Then, by Definition 5, the literal $G[\vec{P}] \stackrel{a}{\not\dashrightarrow}$ denies a $\beta \in CT^+_\kappa$ with $\kappa < \lambda$. The literal β must have the form $G[\vec{P}] \xrightarrow{a}_\kappa R$. By induction, using Claim 4 above, $G[\vec{Q}] \xrightarrow{a}_\kappa R'$ for some $R' \in \mathrm{T}(\Sigma)$. Consequently, $G[\vec{Q}] \stackrel{a}{\not\dashrightarrow}_\lambda$ does not hold.
2. The proof of Claim 2 proceeds exactly like the one of Claim 4 below.
3. Suppose $G[\vec{Q}] \stackrel{a}{\not\longrightarrow}_\lambda$ does not hold. Then, by Definition 5, the literal $G[\vec{Q}] \stackrel{a}{\not\longrightarrow}$ denies a $\beta \in PT^+_\lambda$. The literal β must have the form $G[\vec{Q}] \stackrel{a}{\dashrightarrow}_\lambda R$. Using Claim 2 above, $G[\vec{Q}] \stackrel{a}{\dashrightarrow}_\lambda R'$ for some $R' \in \mathrm{T}(\Sigma)$. Consequently, $G[\vec{P}] \stackrel{a}{\not\dashrightarrow}_\lambda$ does not hold.
4. The proof of this claim proceeds by a nested induction on the derivation π of $G[\vec{P}] \xrightarrow{a} R$ from the hypotheses CT^-_λ, making a case distinction on the shape of G. Trivially, if a closed positive literal is provable from a set of closed literals, all literals occurring in the proof can be chosen closed. This is the case because applying the same closed substitution to all literals in a proof, preserves the property of it being a proof. I thus may assume that all literals in π are closed.
   The case that $G = x \in Var$ cannot occur, as G is guarded.
   Let $G = f(G_1,\dots,G_{ar(f)})$ for some $f \in \Sigma$ and $G_1,\dots,G_{ar(f)} \in \mathbb{T}(\Sigma,W)$. Let $r = \frac{H}{f(x_1,\dots,x_{ar(f)}) \xrightarrow{a} F}$ be the ntyft/ntyxt-rule without lookahead and $\sigma : V \rightarrow \mathrm{T}(\Sigma)$ the substitution that are used in the last step of π. Here V is the set of free variables occurring in r. Then $F[\sigma] = R$ and $f(x_1,\dots,x_{ar(f)})[\sigma] = G[\vec{P}]$, so $\sigma(x_i) = G_i[\vec{P}]$ for $i = 1,\dots,ar(f)$. Now define the substitution $\rho : V \rightarrow \mathbb{T}(\Sigma,W)$ by
   $$\begin{array}{lll} \rho(x_i) = G_i & & \text{for } i = 1,\dots,ar(f), \text{ and} \\ \rho(y) = \sigma(y) & & \text{for all } y \in V \backslash \{x_1,\dots,x_{ar(f)}\}. \end{array}$$
   Then $E[\sigma] = E[\rho][\vec{P}]$ for any recursion-free term $E \in \mathbb{T}(\Sigma,V)$.
   Given a positive premise $E_y \xrightarrow{c} y$ in H, one finds that $E_y[\rho][\vec{P}] = E_y[\sigma] \xrightarrow{c}_\lambda \sigma(y)$, and the proof of this transition is a subproof of π. Consequently, I may apply the induction hypothesis on this transition, provided I can show that $E_y[\rho]$ is guarded.
   Any free occurrence of a variable y in $E_y[\rho]$ must lay in a term G_i, for some $i \in \{1,\dots,ar(f)\}$, that is substituted by ρ for an occurrence of x_i in E_y. In case x_i is a guarded argument of f, each occurrence of x_i in E_y is guarded, and hence the occurrence of y is guarded in $E_y[\rho]$ as well. In case x_i is an unguarded argument of f, as this occurrence of y must be guarded in G, it is guarded in G_i, and thus also in $E_y[\rho]$.
   So $E_y[\rho]$ is guarded indeed. By induction, $\sigma(y)$ has the form $E'_y[\vec{P}]$ for some $E'_y \in \mathbb{T}(\Sigma,W)$. Moreover $E_y[\rho][\vec{Q}] \xrightarrow{c}_\lambda E'_y[\vec{Q}]$. Let V_{RHS} be the set of variables occurring as right-hand sides of premises in H, and define the substitution $\overline{\rho} : V \rightarrow \mathbb{T}(\Sigma,W)$ by
   $$\begin{array}{ll} \overline{\rho}(x_i) = G_i & \text{for } i = 1,\dots,ar(f), \\ \overline{\rho}(y) = E'_y & \text{for } y \in V_{\mathrm{RHS}}, \text{ and} \\ \overline{\rho}(y) = \sigma(y) & \text{for all } y \in V \backslash (\{x_1,\dots,x_{ar(f)}\} \cup V_{\mathrm{RHS}}). \end{array}$$
   Then $E[\sigma] = E[\overline{\rho}][\vec{P}]$ for any recursion-free term $E \in \mathbb{T}(\Sigma,V)$. Thus $R = F[\sigma] = F[\overline{\rho}][\vec{P}]$, which is the first statement that needed to be proved.
   Towards the second, let $\overline{\sigma} : V \rightarrow \mathrm{T}(\Sigma)$ be the substitution given by
   $$\begin{array}{ll} \overline{\sigma}(x_i) = G_i[\vec{Q}] & \text{for } i = 1,\dots,ar(f), \\ \overline{\sigma}(y) = E'_y[\vec{Q}] & \text{for } y \in V_{\mathrm{RHS}}, \text{ and} \\ \overline{\sigma}(y) = \sigma(y) & \text{for all } y \in V \backslash (\{x_1,\dots,x_{ar(f)}\} \cup V_{\mathrm{RHS}}). \end{array}$$
   Then $E[\overline{\sigma}] = E[\overline{\rho}][\vec{Q}]$ for any recursion-free term $E \in \mathbb{T}(\Sigma,V)$. For each premise $E_y \xrightarrow{c} y$ in H one has $E_y[\rho] = E_y[\overline{\rho}]$ since the rule r has no lookahead. Thus
   $$E_y[\overline{\sigma}] = E_y[\overline{\rho}][\vec{Q}] = E_y[\rho][\vec{Q}] \xrightarrow{c}_\lambda E'_y[\vec{Q}] = y[\overline{\sigma}].$$

For each premise $E \stackrel{c}{\not\longrightarrow}$ in H one has $E[\rho][\vec{P}] = E[\sigma] \stackrel{c}{\not\longrightarrow}_\lambda$, and $E[\rho]$ is guarded by the same reasoning as above. By Claim 3 above, $E[\rho][\vec{Q}] \stackrel{c}{\not\longrightarrow}_\lambda$. Again $E[\rho] = E[\overline{\rho}]$, as r has no lookahead, so $E[\overline{\sigma}] = E[\overline{\rho}][\vec{Q}] = E[\rho][\vec{Q}] \stackrel{c}{\not\longrightarrow}_\lambda$. Applying rule r, it follows that

$$G[\vec{Q}] = f(x_1,\dots,x_{ar(f)})[\overline{\sigma}] \xrightarrow{a}_\lambda F[\overline{\sigma}] = F[\overline{\rho}][\vec{Q}],$$

which had to be shown.

Finally, let $G = \langle X|\mathcal{S}\rangle$, so that $G[\vec{P}] = \langle X|\mathcal{S}[\vec{P}^\dagger]\rangle$, where $\vec{P}^\dagger$ is the $W\backslash V_\mathcal{S}$-tuple that is left of $\vec{P}$ after deleting the Y-components for $Y \in V_\mathcal{S}$. The transition $\langle \mathcal{S}_X[\vec{P}^\dagger]|\mathcal{S}[\vec{P}^\dagger]\rangle \xrightarrow{a}_\lambda R$ is derivable through a subderivation of the one for $\langle X|\mathcal{S}[\vec{P}^\dagger]\rangle \xrightarrow{a}_\lambda R$. Moreover, $\langle \mathcal{S}_X[\vec{P}^\dagger]|\mathcal{S}[\vec{P}^\dagger]\rangle = \langle \mathcal{S}_X|\mathcal{S}\rangle[\vec{P}]$. So by induction, R has the form $E[\vec{P}]$ for some $E \in \mathbb{T}(\Sigma,W)$, and $\langle \mathcal{S}_X|\mathcal{S}\rangle[\vec{Q}] \xrightarrow{a}_\lambda E[\vec{Q}]$. Since $\langle \mathcal{S}_X|\mathcal{S}\rangle[\vec{Q}] = \langle \mathcal{S}_X[\vec{Q}^\dagger]|\mathcal{S}[\vec{Q}^\dagger]\rangle$, it follows that $G[\vec{Q}] = \langle X|\mathcal{S}\rangle[\vec{Q}] = \langle X|\mathcal{S}[\vec{Q}^\dagger]\rangle \xrightarrow{a}_\lambda G'[\vec{Q}]$. □

Definition 11 (Guarded recursive specifications) A recursive specification $\mathcal{S}$ is *manifestly guarded* if for each $X,Y \in V_\mathcal{S}$ all occurrences of the variable Y in the term $\mathcal{S}_X$ are guarded. It is *guarded* if it can be converted into a manifestly guarded recursive specification by repeated substitution of expressions $\mathcal{S}_Y$ for variables $Y \in V_\mathcal{S}$ occurring in expressions $\mathcal{S}_Z$ for $Z \in V_\mathcal{S}$.

Example 9 The recursive specification $\{X = h(a.b.sw.X, X, X)\}$ is manifestly guarded. The specification $\mathcal{S} := \{X = Y \triangleright X,\ Y = (a.b.sw.X) \triangleright Y\}$ is not manifestly guarded, since Y occurs unguarded in $\mathcal{S}_X$. However, $\mathcal{S}$ is guarded, because substituting $\mathcal{S}_Y = (a.b.sw.X) \triangleright Y$ for Y in $\mathcal{S}_X = Y \triangleright X$ yields the recursive specification $\{X = ((a.b.sw.X) \triangleright Y) \triangleright X,\ Y = (a.b.sw.X) \triangleright Y\}$, which *is* manifestly guarded.

5 The bisimulation preorder

Traditionally [30], bisimilarity is defined on 2-valued transition systems only, whereas the structural operational semantics of a language specified by a TSS can be 3-valued. Rather than limit my results to languages specified by complete TSSs, I follow [20] in using an extension of the notion of bisimilarity to 3-valued transition systems. This extension stems from [28] under the name *modal refinement*; there 3-valued transition systems are called *modal transition systems*.

Definition 12 (Bisimilarity) Let $\mathcal{T} = (\Sigma, \mathcal{R})$ be a TSS. A *bisimulation* $\mathscr{B}$ is a binary relation on $\mathrm{T}(\Sigma)$ such that, for $P,Q \in \mathrm{T}(\Sigma)$ and $a \in A$,

- if $P \mathscr{B} Q$ and $P \xrightarrow{a} P'$, then there is a Q' with $Q \xrightarrow{a} Q'$ and $P' \mathscr{B} Q'$,
- if $P \mathscr{B} Q$ and $Q \stackrel{a}{\dashrightarrow} Q'$, then there is a P' with $P \stackrel{a}{\dashrightarrow} P'$ and $P' \mathscr{B} Q'$.

A process $Q \in \mathrm{T}(\Sigma)$ is a *modal refinement* of $P \in \mathrm{T}(\Sigma)$, notation $P \sqsubseteq_B Q$, if there exists a bisimulation $\mathscr{B}$ with $P \mathscr{B} Q$. I call $\sqsubseteq_B$ the *bisimulation preorder*, or *bisimilarity*. The kernel of $\sqsubseteq$, given by $\equiv_B := \sqsubseteq_B \cap \sqsupseteq_B$, is *bisimulation equivalence*.

Clearly, modal refinement is reflexive and transitive, and hence a preorder. The underlying idea is that a process P with a 3-valued transition relation $\langle CT, PT\rangle$ is a *specification* of a process with a 2-valued transition relation, in which the presence or absence of certain transitions is left open. CT contains the transitions that are *required* by the specification, and PT the ones that are *allowed*. If $P \sqsubseteq_B Q$, then Q may be closer to the eventual implementation, in the sense that some of the undetermined transitions have been resolved to present or absent. The requirements of Definition 12 now say that any transition that is required by P should be (matched by a transition) required by Q, whereas any transition allowed by Q, should certainly be (matched by a transition) allowed by P.

In case P and Q are 2-valued (i.e. *implementations*) the modal refinement relation is just the traditional notion of bisimilarity [30] (and thus symmetric).

Theorem 1 (Lean precongruence [20]) Let $\mathcal{T} = (\Sigma, \mathcal{R})$ be a TSS in the ready simulation format with recursion. Let $E \in \mathbb{T}(\Sigma, W)$ be an expression, and let $\vec{P}, \vec{Q} \in \mathrm{T}(\Sigma)^W$ be W-tuples of processes for some $W \subseteq \mathit{Var}$, such that $\vec{P}(x) \sqsubseteq_B \vec{Q}(x)$ for each $x \in W$. Then $E[\vec{P}] \sqsubseteq_B E[\vec{Q}]$.

In [20] this result was actually obtained for the more general ntyft/ntyxt-format with recursion, obtained by lifting the restriction ruling out lookahead. The special case that E is a recursion-free expression stems from [8] (under a side condition on the TSS called *well-foundedness*, which was lifted in [13]). The special subcase in which $\mathcal{T}$ has no negative premises goes back to [24].

MILNER [30] proposed an important tool in proving bisimilarity between two processes: the notion of a bisimulation up to bisimilarity. Here I extend it to the bisimulation preorder on 3-valued TSSs.

Definition 13 (Bisimulation upto) Let $\mathcal{T} = (\Sigma, \mathcal{R})$ be a TSS. A *bisimulation $\mathscr{B}$ up to bisimilarity* is a binary relation on $\mathrm{T}(\Sigma)$ such that, for $P, Q \in \mathrm{T}(\Sigma)$ and $a \in A$,

- if $P \mathscr{B} Q$ and $P \xrightarrow{a} P'$, then there is a Q' with $Q \xrightarrow{a} Q'$ and $P' \sqsubseteq_B \mathscr{B} \sqsubseteq_B Q'$,
- if $P \mathscr{B} Q$ and $Q \dashrightarrow{a} Q'$, then there is a P' with $P \dashrightarrow{a} P'$ and $P' \sqsubseteq_B \mathscr{B} \sqsubseteq_B Q'$.

Here $\sqsubseteq_B \mathscr{B} \sqsubseteq_B := \{(R,S) \mid \exists R', S'.\ R \sqsubseteq_B R' \mathscr{B} S' \sqsubseteq_B S\}$.

Proposition 1 If $P \mathscr{B} Q$ for some bisimulation $\mathscr{B}$ up to bisimilarity, then $P \sqsubseteq_B Q$.

Proof: Using the reflexivity of $\sqsubseteq_B$ it suffices to show that $\sqsubseteq_B \mathscr{B} \sqsubseteq_B$ is a bisimulation. Using transitivity of $\sqsubseteq_B$ this is straightforward. □

6 The recursive specification principle

Let Σ be a signature. Any preorder $\sqsubseteq \subseteq \mathrm{T}(\Sigma) \times \mathrm{T}(\Sigma)$ on the closed terms over Σ extends to a preorder $\sqsubseteq \subseteq \mathbb{T}(\Sigma) \times \mathbb{T}(\Sigma)$ on open terms by defining $E \sqsubseteq F$ iff $E[\rho] \sqsubseteq F[\rho]$ for each closed substitution ρ. It extends to substitutions $\rho, \nu : \mathit{Var} \rightharpoonup \mathbb{T}(\Sigma)$ by $\rho \sim \nu$ iff $\mathrm{dom}(\rho) = \mathrm{dom}(\nu)$ and $\rho(x) \sim \nu(x)$ for each $x \in \mathrm{dom}(\rho)$. As W-tuples $\vec{E}, \vec{F} \in \mathbb{T}(\Sigma)^W$ for $W \subseteq \mathit{Var}$ are substitutions, this gives $\vec{E} \sqsubseteq \vec{F}$ iff $\vec{E}(x) \sqsubseteq \vec{F}(x)$ for each $x \in W$. A recursive specification $\mathcal{S} : V_{\mathcal{S}} \to \mathbb{T}(\Sigma)$ is a $V_{\mathcal{S}}$-tuple; if $\vec{E} \in \mathbb{T}(\Sigma)^{V_{\mathcal{S}}}$ is another $V_{\mathcal{S}}$-tuple then with $\mathcal{S}[\vec{E}]$ I denote the $V_{\mathcal{S}}$-tuple with elements $\mathcal{S}_X[\vec{E}]$ for each $X \in V_{\mathcal{S}}$, that is, the body $\mathcal{S}_X$ of the recursive equation of X in which $\vec{E}(Y)$ is substituted for each variable $Y \in V_{\mathcal{S}}$ occurring freely in $\mathcal{S}_X$.

Definition 14 (Solution) For $\mathcal{S}$ a recursive specification and $\equiv$ a semantic equivalence, a $V_{\mathcal{S}}$-tuple of expressions $\vec{E} \in \mathbb{T}(\Sigma)^{V_{\mathcal{S}}}$ is a *solution* of $\mathcal{S}$ if $\vec{E} \equiv \mathcal{S}[\vec{E}]$.

When generalising this definition with a preorder $\sqsubseteq$ in the rôle of $\equiv$, one obtains *pre-* and *postsolutions*.

Definition 15 For $\mathcal{S}$ a recursive specification and $\sqsubseteq$ a semantic equivalence, a $V_{\mathcal{S}}$-tuple of expressions $\vec{E} \in \mathbb{T}(\Sigma)^{V_{\mathcal{S}}}$ is a *presolution* (resp. *postsolution*) of $\mathcal{S}$ if $\vec{E} \sqsubseteq \mathcal{S}[\vec{E}]$ (resp. $\mathcal{S}[\vec{E}] \sqsubseteq \vec{E}$).

The following theorem establishes RSP for the bisimulation preorder.

Theorem 2 Let $\mathcal{T} = (\Sigma, \mathcal{R})$ be a TSS in the ready simulation format with recursion, and $\mathcal{S}$ be a $\mathcal{T}$-guarded recursive specification. If $\vec{E} \sqsubseteq_B \mathcal{S}[\vec{E}]$ and $\mathcal{S}[\vec{F}] \sqsubseteq_B \vec{F}$ with $\vec{E}, \vec{F} \in \mathbb{T}(\Sigma)^{V_{\mathcal{S}}}$, then $\vec{E} \sqsubseteq_B \vec{F}$.

Proof: It suffices to prove Theorem 2 under the assumptions that $\vec{E}, \vec{F} \in \mathrm{T}(\Sigma)^{V_{\mathcal{S}}}$ and only the variables from $V_{\mathcal{S}}$ occur free in the expressions $\mathcal{S}_X$ for $X \in V_{\mathcal{S}}$. For in the general case I have to establish that $\vec{E}[\sigma] \sqsubseteq_B \vec{F}[\sigma]$ for an arbitrary closed substitution $\sigma : \mathit{Var} \to \mathrm{T}(\Sigma)$. Let $\sigma^\dagger : \mathit{Var} \backslash V_{\mathcal{S}} \to \mathrm{T}(\Sigma)$ be given by $\sigma^\dagger(X) = \sigma(X)$ for all $X \in \mathit{Var} \backslash V_{\mathcal{S}}$. Then $\vec{E} \sqsubseteq_B \mathcal{S}[\vec{E}]$ and $\mathcal{S}[\vec{F}] \sqsubseteq_B \vec{F}$ imply $\vec{E}[\sigma] \sqsubseteq_B \mathcal{S}[\vec{E}][\sigma] = \mathcal{S}[\sigma^\dagger][\vec{E}[\sigma]]$ and $\mathcal{S}[\sigma^\dagger][\vec{F}[\sigma]] = \mathcal{S}[\vec{F}][\sigma] \sqsubseteq_B \vec{F}[\sigma]$. Hence, I merely have to prove the theorem with $\vec{E}[\sigma]$, $\vec{F}[\sigma]$and $\mathcal{S}[\sigma^\dagger]$ in place of $\vec{E}$, $\vec{F}$ and $\mathcal{S}$.

It also suffices to prove Theorem 2 under the assumption that $\mathcal{S}$ is a manifestly guarded recursive specification. Namely, for a general guarded recursive specification $\mathcal{S}$, let $\mathcal{S}'$ be the manifestly guarded recursive specification into which $\mathcal{S}$ can be converted. Then $\vec{E} \sqsubseteq_B \mathcal{S}[\vec{E}]$ implies $\vec{E} \sqsubseteq_B \mathcal{S}'[\vec{E}]$ by Theorem 1 and transitivity, and likewise $\mathcal{S}[\vec{F}] \sqsubseteq_B \vec{F}$ implies $\mathcal{S}'[\vec{F}] \sqsubseteq_B \vec{F}$.

So let $\mathcal{S}$ be manifestly guarded with free variable from $V_\mathcal{S}$ only, and let $\vec{P}, \vec{Q} \in \mathrm{T}^{V_\mathcal{S}}(\Sigma)$ be a pre- and postsolution of $\mathcal{S}$, that is, $\vec{P} \sqsubseteq_B \mathcal{S}[\vec{P}]$ and $\mathcal{S}[\vec{Q}] \sqsubseteq_B \vec{Q}$. I will show that

$$\mathscr{B} := \{G[\mathcal{S}[\vec{P}]], G[\mathcal{S}[\vec{Q}]] \mid G \in \mathbb{T}(\Sigma) \text{ is recursion-free and has variables from } V_\mathcal{S} \text{ only}\}$$

is a bisimulation up to bisimilarity. Once I have that, taking $G := X \in V_\mathcal{S}$ yields $\mathcal{S}_X[\vec{P}] \sqsubseteq_B \mathcal{S}_X[\vec{Q}]$ by Proposition 1, and thus $\vec{P}(X) \sqsubseteq_B \mathcal{S}_X[\vec{P}] \sqsubseteq_B \mathcal{S}_X[\vec{Q}] \sqsubseteq_B \vec{Q}(X)$ for all $X \in V_\mathcal{S}$. So $\vec{P} \sqsubseteq_B \vec{Q}$.

- Let $R \mathrel{\mathscr{B}} T$ and $R \xrightarrow{a} R'$ with $a \in A$. I have to find a T' with $T \xrightarrow{a} T'$ and $R' \sqsubseteq_B \mathscr{B} \sqsubseteq_B T'$. Let $R = G[\mathcal{S}[\vec{P}]]$ and $T = G[\mathcal{S}[\vec{Q}]]$, where $G \in \mathbb{T}(\Sigma)$ is recursion-free and has variables from $V_\mathcal{S}$ only. Note that $G[\mathcal{S}[\vec{P}]]$ can also be written as $G[\mathcal{S}][\vec{P}]$. Since the expressions $\mathcal{S}_X$ for $X \in V_\mathcal{S}$ have free variables from $V_\mathcal{S}$ only, so does $G[\mathcal{S}]$. Moreover, since $\mathcal{S}$ is manifestly guarded, the expression $G[\mathcal{S}]$ must be guarded. By Lemma 2, R' must have the form $G'[\vec{P}]$, where $G' \in \mathbb{T}(\Sigma, V_\mathcal{S})$. Moreover, $T = G[\mathcal{S}[\vec{Q}]] = G[\mathcal{S}][\vec{Q}] \xrightarrow{a} G'[\vec{Q}] =: T'$. Furthermore, by Theorem 1, $G'[\vec{P}] \sqsubseteq_B G'[\mathcal{S}[\vec{P}]]$ and $G'[\mathcal{S}[\vec{Q}]] \sqsubseteq_B G'[\vec{Q}]$. By definition $G'[\mathcal{S}[\vec{P}]] \mathrel{\mathscr{B}} G'[\mathcal{S}[\vec{Q}]]$. Thus, $R' = G'[\vec{P}] \sqsubseteq_B \mathscr{B} \sqsubseteq_B G'[\vec{Q}] = T'$.
- Let $R \mathrel{\mathscr{B}} T$ and $T \overset{a}{\dashrightarrow} T'$ with $a \in A$. I have to find an R' with $R \overset{a}{\dashrightarrow} R'$ and $R' \sqsubseteq_B \mathscr{B} \sqsubseteq_B T'$. Let $R = G[\mathcal{S}[\vec{P}]]$ and $T = G[\mathcal{S}[\vec{Q}]]$, where $G \in \mathbb{T}(\Sigma)$ is recursion-free and has variables from $V_\mathcal{S}$ only. Note that $G[\mathcal{S}[\vec{Q}]]$ can also be written as $G[\mathcal{S}][\vec{Q}]$. Since the expressions $\mathcal{S}_X$ for $X \in V_\mathcal{S}$ have free variables from $V_\mathcal{S}$ only, so does $G[\mathcal{S}]$. Moreover, since $\mathcal{S}$ is manifestly guarded, the expression $G[\mathcal{S}]$ must be guarded. By Lemma 2, T' must have the form $G'[\vec{Q}]$, where $G' \in \mathbb{T}(\Sigma, V_\mathcal{S})$. Moreover, $R = G[\mathcal{S}[\vec{P}]] = G[\mathcal{S}][\vec{P}] \overset{a}{\dashrightarrow} G'[\vec{P}] =: R'$. Furthermore, by Theorem 1, $G'[\vec{P}] \sqsubseteq_B G'[\mathcal{S}[\vec{P}]]$ and $G'[\mathcal{S}[\vec{Q}]] \sqsubseteq_B G'[\vec{Q}]$. By definition $G'[\mathcal{S}[\vec{P}]] \mathrel{\mathscr{B}} G'[\mathcal{S}[\vec{Q}]]$. Thus, $R' = G'[\vec{P}] \sqsubseteq_B \mathscr{B} \sqsubseteq_B G'[\vec{Q}] = T'$. □

7 The recursive definition principle

The recursive definition principle, saying that recursive specifications have a solution, trivially holds for bisimulation equivalence, and thereby also for all coarser equivalences. Namely the recursive call $\langle X|\mathcal{S}\rangle$ itself is a solution of the recursive specification $\mathcal{S}$.

$$\langle X|\mathcal{S}\rangle \equiv_B \langle \mathcal{S}_X|\mathcal{S}\rangle \quad \text{for all } X \in V_\mathcal{S} \tag{2}$$

This follows immediately from the operational rules for recursion.

Write $\langle V_\mathcal{S}|\mathcal{S}\rangle$ for the $V_\mathcal{S}$-tuple with elements $\langle X|\mathcal{S}\rangle$ for $X \in V_\mathcal{S}$. Connecting this with the notation of Definition 2, and seeing $V_\mathcal{S}$-tuples as substitutions, this yields $\langle E|\mathcal{S}\rangle = E[\langle V_\mathcal{S}|\mathcal{S}\rangle]$ for any $E \in \mathbb{T}(\Sigma)$. Moreover, $\mathcal{S}[\langle V_\mathcal{S}|\mathcal{S}\rangle]$ is the $V_\mathcal{S}$-tuple with elements $\langle \mathcal{S}_X|\mathcal{S}\rangle$ for $X \in V_\mathcal{S}$.

Now (2), or RDP, for an arbitrary semantic equivalence $\equiv$, can be restated as $\langle V_\mathcal{S}|\mathcal{S}\rangle \equiv \mathcal{S}[\langle V_\mathcal{S}|\mathcal{S}\rangle]$. When $\equiv$ is the kernel of a preorder $\sqsubseteq$, this implies $\mathcal{S}[\langle V_\mathcal{S}|\mathcal{S}\rangle] \sqsubseteq \langle V_\mathcal{S}|\mathcal{S}\rangle$ and $\langle V_\mathcal{S}|\mathcal{S}\rangle \sqsubseteq \mathcal{S}[\langle V_\mathcal{S}|\mathcal{S}\rangle]$. Using this, RSP (1) can be restated as

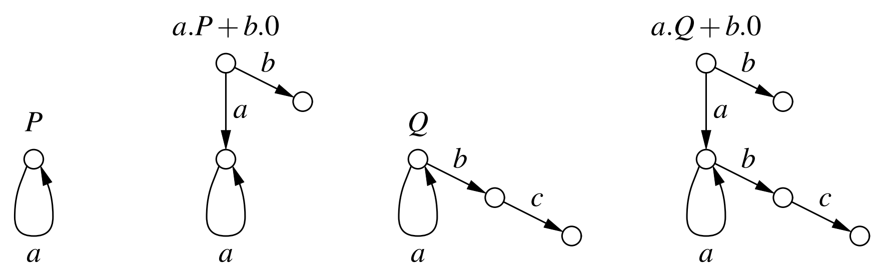
$$\frac{\vec{P} \equiv \mathcal{S}[\vec{P}]}{\vec{P} \equiv \langle V_\mathcal{S}|\mathcal{S}\rangle} \qquad \frac{\vec{P} \sqsubseteq \mathcal{S}[\vec{P}]}{\vec{P} \sqsubseteq \langle V_\mathcal{S}|\mathcal{S}\rangle} \qquad \frac{\mathcal{S}[\vec{Q}] \sqsubseteq \vec{Q}}{\langle V_\mathcal{S}|\mathcal{S}\rangle \sqsubseteq \vec{Q}}\,. \tag{3}$$

8 The simulation and ready simulation preorders

The ready simulation preorder $\sqsubseteq_{RS}$ [7, 18] has been generalised to 3-valued transition systems in [5]. Its definition is just like Definition 12, but without the targets Q' and P' in the second clause. Here $Q \stackrel{a}{\dashrightarrow}$ means that there exists a Q' such that $Q \stackrel{a}{\dashrightarrow} Q'$. In [5] it is also shown to be a congruence (cf. Theorem 1) on TSSs in the ready simulation format, albeit without recursion. Using this, the proof of Theorem 2 generalises smoothly to $\sqsubseteq_{RS}$. There merely is a straightforward simplification in the second clause, dealing with transitions $\dashrightarrow$. I conclude that RSP holds for the ready simulation preorder for TSSs in the ready simulation format.

The simulation preorder $\sqsubseteq_S$ [18] is defined just like the bisimulation preorder, but without its second clause. It is a congruence only for TSSs in the positive ready simulation format, that is, ruling out negative premises [5]. As a counterexample, in the TSS of Example 4, but extended with the familiar choice operator $+$ of CCS [30], we have $a.0 \sqsubseteq_S a.0+b.0$, yet $en_{\{b\}}(a.0) \not\sqsubseteq_S en_{\{b\}}(a.0+b.0)$, since only $en_{\{b\}}(a.0)$ can do first a b and then an a-transition. Again a straightforward simplification of the above proof shows that RSP holds for the simulation preorder for TSSs in the positive ready simulation format.

Example 10 Let $\mathcal{S} := \{X = a.X + b.0\}$. Take $P := \langle Y|\{Y = a.Y\}\rangle$ and $Q := \langle Z|\{Z = a.Z \ + a.b.0\}\rangle$.



Now $P \sqsubseteq_S \mathcal{S}[P] = a.P+b.0$ and $\mathcal{S}[Q] = a.Q+b.0 \sqsubseteq_S Q$. Hence, by RSP, $P \sqsubseteq_S Q$.

The literature is divided about the orientation of the simulation preorder, i.e., whether $a.0 \sqsubseteq_S a.0+b.0$ or $a.0+b.0 \sqsubseteq_S a.0$, and likewise for all semantic preorders. Intuitively, the argument for $a.0 \sqsubseteq_S a.0+b.0$ is that there exists a simulation *from* $a.0$ *to* $a.0+b.0$, in the sense that behaviour of the former process can be mimicked by the latter. Related to that, the process $a.0$ has *less* behaviours or runs than $a.0+b.0$.

Below I will follow the argument of Tony Hoare [25] that $P \sqsubseteq Q$ should mean that Q is a (non-strict) *improvement* of P, in the sense that all good properties of P should hold for Q as well. As a specification S tells us the minimal set of requirements that an implementation I should satisfy, this pleads for the orientation $S \sqsubseteq I$.

Usually this argument is used to advocate the direction $a.0+b.0 \sqsubseteq_S a.0$. A good property could be a *satisfy* property, saying that nothing bad will ever happen, or a *liveness* property, saying that something good will happen eventually [26]. An example of a safety property could be that the action b will never occur, and an example of a liveness property could be that each run contains the action a or b. Both safety and liveness properties are *linear time properties*: they are defined to hold for certain runs of a system, and then are said to hold for a system iff they hold for all its runs. Now a system with less runs will automatically satisfy more linear time properties, and thus be a more suitable implementation. This pleads for the direction $a.0+b.0 \sqsubseteq_S a.0$.

Nevertheless, theoretically one could have quite different intentions with specifications, modelled as 2- or 3-valued labelled transition systems. It could be that the specification stipulates which transitions should be present at least, allowing room for adding more transitions. This fits with allowing *possibility* *properties* [27] in the above argument about good properties preserved by semantic preorders. Such properties may say that a process has the possibility to behave in a certain way.

In [17], for each semantic preorder $\sqsubseteq$ surveyed there I propose two variants, called $\sqsubseteq^{\text{may}}$ and $\sqsubseteq^{\text{must}}$. Although these preorders differ from each other in non-trivial ways when dealing with divergence and underspecification, when restricted to processes P and Q without divergence and underspecification, we simply have $P \sqsubseteq^{\text{may}} Q$ iff $Q \sqsubseteq^{\text{must}} P$. For the simulation preorder we have $a.0 \sqsubseteq_S^{\text{may}} a.0+b.0$ and $a.0+b.0 \sqsubseteq_S^{\text{must}} a.0$, and more in general $P \sqsubseteq^{\text{may}} Q$ says that possible behaviour of P must also be possible for Q, whereas $P \sqsubseteq^{\text{must}} Q$ says that required behaviour of P must also be required of Q. The annotations $^{\text{may}}$ and $^{\text{must}}$ were inspired by the may- and must-testing preorders of De Nicola & Hennessy [11], which also are oriented this way.

When applying preorders $\sqsubseteq^{\text{may}}$ and $\sqsubseteq^{\text{must}}$ to modal transition systems, in which transitions $\xrightarrow{a}$ are required, and transitions $\overset{a}{\dashrightarrow}$ are allowed, it makes most sense to define $\sqsubseteq^{\text{may}}$ in terms of the necessary transitions $\xrightarrow{a}$, and $\sqsubseteq^{\text{must}}$ in terms of the possible transitions $\overset{a}{\dashrightarrow}$. Thus $\sqsubseteq_S^{\text{may}}$ is defined like $\sqsubseteq_B$ in Definition 12, but without the second clause, whereas $\sqsubseteq_S^{\text{must}}$ is defined like $\sqsubseteq_B$, but without the first clause. In the same spirit, $\sqsubseteq_{RS}^{\text{may}}$ is as $\sqsubseteq_B$ but without the targets in the second clause, so that this second clause can be restated as

- if $P \mathscr{B} Q$ and $P \overset{a}{\not\longrightarrow}$, then $Q \overset{a}{\not\longrightarrow}$.

Similarly, $\sqsubseteq_{RS}^{\text{must}}$ is as $\sqsubseteq_B$ but without the targets in the first clause. Hence the preorders $\sqsubseteq_{RS}$ and $\sqsubseteq_S$ from [18, 5] that were featured at the beginning of this section are the may-versions of the ready simulation and simulation preorders. The must-versions of these preorders, which may be preferable when concentrating on safety and liveness properties, satisfy RSP by entirely symmetric arguments.

Corollary 1 RSP holds for $\sqsubseteq_{RS}^{\text{must}}$ for any TSS in the ready simulation format.
RSP also holds for $\sqsubseteq_S^{\text{must}}$ for any TSS in the positive ready simulation format.

9 Full congruence results for guarded recursion

A semantic equivalence $\equiv$ is called a *full congruence* for recursion [20] if

$$\mathcal{S} \equiv \mathcal{S}' \Rightarrow \langle V_\mathcal{S}|\mathcal{S}\rangle \equiv \langle V_\mathcal{S}|\mathcal{S}'\rangle$$

for all recursive specifications $\mathcal{S}$ and $\mathcal{S}'$ with $V_\mathcal{S} = V'_\mathcal{S}$. When expanding the application of $\equiv$ to $V_\mathcal{S}$-tuples, defined at the beginning of Section 6, and the tuple $\langle V_\mathcal{S}|\mathcal{S}\rangle$ of Section 7, this translates to

$$\text{if } \mathcal{S}_Y \equiv \mathcal{S}'_Y \text{ for all } Y \in V_\mathcal{S} \text{ then } \langle X|\mathcal{S}\rangle \equiv \langle X|\mathcal{S}'\rangle \text{ for all } X \in \mathcal{S},$$

where $\mathcal{S}_Y \equiv \mathcal{S}'_Y$ means that $\mathcal{S}_Y[\sigma] \equiv \mathcal{S}'_Y[\sigma]$ for all closed substitutions σ. In [20] this was merely required for recursive specifications $\mathcal{S},\mathcal{S}' : V_\mathcal{S} \to \mathbb{T}(\Sigma, V_\mathcal{S})$, that is, without free variables (cf. Example 2), so that $\langle X|\mathcal{S}\rangle$ and $\langle X|\mathcal{S}'\rangle$ are closed terms. However, when we have that, the general form follows trivially, just as in the first paragraph of the proof of Theorem 2. In the same way, a preorder $\sqsubseteq$ is a *full precongruence* for recursion [20] if

$$\mathcal{S} \sqsubseteq \mathcal{S}' \Rightarrow \langle V_\mathcal{S}|\mathcal{S}\rangle \sqsubseteq \langle V_\mathcal{S}|\mathcal{S}'\rangle$$

for all recursive specifications $\mathcal{S}$ and $\mathcal{S}'$ with $V_\mathcal{S} = V'_\mathcal{S}$. A preorder $\sqsubseteq$ is a *precongruence* for an n-ary operator $f \in \Sigma$ if

$$\vec{P} \sqsubseteq \vec{Q} \Rightarrow f(\vec{P}) \sqsubseteq f(\vec{Q})$$

where $\vec{P}$ and $\vec{Q}$ denote the tuples $(P_i)_{i=1}^n$ and $(Q_i)_{i=1}^n$. Clearly, if $\sqsubseteq$ is a precongruence for all operators f in a language given by a signature Σ, as well as a full precongruence for recursion, then it surely is a *lean precongruence* [20] (cf. Theorem 1), meaning that

$$\vec{P} \sqsubseteq \vec{Q} \Rightarrow E[\vec{P}] \sqsubseteq E[\vec{Q}]$$

for all expressions $E \in \mathbb{T}(\Sigma)$. Moreover, being a lean precongruence implies that a preorder is a precongruence for all operators. However, as illustrated in [20], being a lean precongruence does not imply that a preorder is a full precongruence for recursion. In [20] it was shown that (strong) bisimilarity is a lean precongruence for all TSSs in the ready simulation format with recursion—in fact, this was shown for the even more general ntyft/ntyxt format with recursion. However, a full precongruence result could be obtained only for the *positive* ntyft/ntyxt format with recursion, avoiding all negative premises. For the ready simulation format with recursion, allowing negative premises, and even for the more restrictive GSOS format (Definition 16), it remained an open question whether bisimilarity is a full precongruence. Here I partly answer that question: for TSSs in the ready simulation format with recursion, bisimilarity is a full precongruence for *guarded* recursion. This is a straightforward consequence of RSP.

Theorem 3 Let $\mathcal{T} = (\Sigma, \mathcal{R})$ be a TSS in the ready simulation format with recursion, and $\mathcal{S}, \mathcal{S}'$ be two recursive specifications with $V_{\mathcal{S}} = V_{\mathcal{S}'}$ and one of them $\mathcal{T}$-guarded. If $\mathcal{S} \sqsubseteq_B \mathcal{S}'$ then $\langle V_{\mathcal{S}}|\mathcal{S}\rangle \sqsubseteq_B \langle V_{\mathcal{S}}|\mathcal{S}'\rangle$.

Proof: I consider first the case that $\mathcal{S}$ and then the one that $\mathcal{S}'$ is guarded.

By RDP, $\langle V_{\mathcal{S}}|\mathcal{S}'\rangle \equiv_B \mathcal{S}'[\langle V_{\mathcal{S}}|\mathcal{S}'\rangle]$, and thus $\mathcal{S}'[\langle V_{\mathcal{S}}|\mathcal{S}'\rangle] \sqsubseteq_B \langle V_{\mathcal{S}}|\mathcal{S}'\rangle$. Since $\mathcal{S}[\sigma] \sqsubseteq_B \mathcal{S}'[\sigma]$ for all substitutions σ, in particular $\mathcal{S}[\langle V_{\mathcal{S}}|\mathcal{S}'\rangle] \sqsubseteq_B \mathcal{S}'[\langle V_{\mathcal{S}}|\mathcal{S}'\rangle]$. By transitivity, $\mathcal{S}\langle V_{\mathcal{S}}|\mathcal{S}'\rangle \sqsubseteq_B \langle V_{\mathcal{S}}|\mathcal{S}'\rangle$, that is, $\langle V_{\mathcal{S}}|\mathcal{S}'\rangle$ is a postsolution of $\mathcal{S}$. Given that $\mathcal{S}$ is guarded, this yields $\langle V_{\mathcal{S}}|\mathcal{S}\rangle \sqsubseteq_B \langle V_{\mathcal{S}}|\mathcal{S}'\rangle$ by RSP (3).

By RDP, $\langle V_{\mathcal{S}}|\mathcal{S}\rangle \equiv_B \mathcal{S}[\langle V_{\mathcal{S}}|\mathcal{S}\rangle]$, and thus $\langle V_{\mathcal{S}}|\mathcal{S}\rangle \sqsubseteq_B \mathcal{S}[\langle V_{\mathcal{S}}|\mathcal{S}\rangle]$. Since $\mathcal{S}[\sigma] \sqsubseteq_B \mathcal{S}'[\sigma]$ for all substitutions σ, in particular $\mathcal{S}[\langle V_{\mathcal{S}}|\mathcal{S}\rangle] \sqsubseteq_B \mathcal{S}'[\langle V_{\mathcal{S}}|\mathcal{S}\rangle]$. By transitivity, $\langle V_{\mathcal{S}}|\mathcal{S}\rangle \sqsubseteq_B \mathcal{S}'\langle V_{\mathcal{S}}|\mathcal{S}\rangle$, that is, $\langle V_{\mathcal{S}}|\mathcal{S}\rangle$ is a presolution of $\mathcal{S}'$. Given that $\mathcal{S}'$ is guarded, this yields $\langle V_{\mathcal{S}}|\mathcal{S}\rangle \sqsubseteq_B \langle V_{\mathcal{S}}|\mathcal{S}'\rangle$ by RSP (3). □

Similarly, bisimulation equivalence, $\equiv_B$, is a full congruence for guarded recursion. This can be shown in exactly the same way, or follows directly from Theorem 3, using that $\equiv_B$ is the kernel of $\sqsubseteq_B$.

Also in the same way it follows that $\sqsubseteq_{RS}^{\text{may}}$ and $\sqsubseteq_{RS}^{\text{must}}$ are full precongruences for guarded recursion on TSSs in the ready simulation format with recursion, and that $\sqsubseteq_{S}^{\text{may}}$ and $\sqsubseteq_{S}^{\text{must}}$ are full precongruences for guarded recursion on TSSs in the positive ready simulation format with recursion.

10 GSOS languages

Definition 16 A *GSOS rule* is a ntyft/ntyxt rule without lookahead, such that the left-hand sides of all premises are variables that also occur in the source of the rule, and its target contains only variables that also occur in its source or premises. A TSS is in *GSOS format* if has the recursion rules from Definition 7, and all of its other rules are recursion-free GSOS rules. A *GSOS language* is language whose syntax is given by a signature Σ and its semantics by TSS $\mathcal{T}$ in GSOS format, but only allowing recursive calls $\langle X|\mathcal{S}\rangle$ for $\mathcal{T}$-guarded recursive specifications $\mathcal{S}$.

GSOS languages, or *GSOS rule systems*, were introduced by BLOOM, ISTRAIL & MEYER in [6]. Here "GSOS" stands for *structural operational semantics with guarded recursion*. In [6] the action prefixing operators $a.\_$ of CCS and Example 4 are required to be present in each GSOS language, so that guarded recursion could be defined as in CCS, using only such operators as guards. As here I have proposed a general definition of guardedness for GSOS languages, the requirement that the operators $a.\_$ must be present can be dropped. In [7], which could be seen as the journal version of [6], recursion has been dropped altogether, and there the "G" of "GSOS" 'might as well stand for "Grand."'

Theorem 4 The well-founded semantics of a GSOS language is always 2-valued.

Proof: Let $\mathcal{T} = (\Sigma, \mathcal{R})$ be the TSS of a GSOS language. If $\langle CT^+, PT^+\rangle$ is its well-founded semantics, I have to show that $CT^+ = PT^+$. This amounts to $PT^+ \subseteq CT^+$, as the other direction holds always.

Let $\rightsquigarrow$ be the smallest binary relation on $\mathrm{T}(\Sigma)$ such that (i) $f(P_1,\dots,P_k) \rightsquigarrow P_i$ for each k-ary operator $f \in \Sigma$ and each $i \in \{1,\dots,k\}$ such that the i^{th} argument of f is unguarded, and (ii) $\langle X|\mathcal{S}\rangle \rightsquigarrow \langle \mathcal{S}_X|\mathcal{S}\rangle$.

Claim: $\rightsquigarrow$ has no forward infinite chain $P_0 \rightsquigarrow P_1 \rightsquigarrow \dots$.

Proof of claim: Let $P_0 \rightsquigarrow P_1 \rightsquigarrow \dots$ be a forward infinite chain. First consider the possibility that P_{h_0} has the form $\langle X|\mathcal{S}\rangle$ for some $h_0 \geq 0$, and all P_m with $m > h_0$ have a subterm of the form $\langle Y|\mathcal{S}\rangle$. In that case there must exist indices h_i for $i \in \mathbb{N}$ such that $h_0 < h_1 < h_2 < \cdots$ and, for all $i \in \mathbb{N}$, each P_{h_i} has the form $\langle X_i|\mathcal{S}\rangle$ for some $X_i \in V_\mathcal{S}$, and X_{i+1} occurs unguarded in the expression $P_{h_i+1} = \langle \mathcal{S}_{X_i}|\mathcal{S}\rangle$.[5] It follows that the recursive specification $\mathcal{S}$ is not manifestly guarded, and neither can it be converted into a manifestly guarded recursive specification by repeated substitution of expressions $\mathcal{S}_Y$ for variables $Y \in V_\mathcal{S}$ occurring in expressions $\mathcal{S}_Z$ for $Z \in V_\mathcal{S}$. This contradicts the assumption that all recursive specifications $\mathcal{S}$ in the language must be guarded. It follows that each P_h in the chain is followed by an expression P_m with $m > h$ that is a strict subterm of P_h. But this contradicts the inductive process of building terms.

Application of the claim: The relation $\rightsquigarrow$ is finitely branching, meaning that for any $P \in \mathrm{T}(\Sigma)$ there are only finitely many $P^\dagger$ with $P \rightsquigarrow P^\dagger$. For any process $P \in \mathrm{T}(\Sigma)$, let $e(P)$ be the length of the longest forward chain $P \rightsquigarrow P_1 \rightsquigarrow \cdots \rightsquigarrow P_{e(P)}$. I show with induction on $e(P)$ that

$$(P \xrightarrow{a} P') \in PT^+ \quad \Rightarrow \quad (P \xrightarrow{a} P') \in CT^+$$

which suffices to conclude the proof. Here I make a case distinction on the shape of P.

Let $P = f(P_1,\dots,P_{ar(f)})$. Using that $\beta \in PT^+$ iff $\mathcal{T} \vdash \frac{PT^-}{\beta}$, the transition $(P \xrightarrow{a} P') \in PT^+$ must be derivable from PT^- through a substitution instance of a proof rule $r = \frac{H}{f(x_1,\dots,x_{ar(f)}) \xrightarrow{a} F}$ from $\mathcal{R}$. Since r is a GSOS rule, the premises of that substitution instance must have the form $P_i \xrightarrow{a} R$ and/or $P_i \stackrel{a}{\not\longrightarrow}$ with $i \in \{1,\dots,ar(f)\}$. Moreover, in case i is be a guarded argument of f, the rule r cannot have premises $x_i \xrightarrow{a} y$ or $x_i \stackrel{a}{\not\longrightarrow}$, using that the occurrence of the variable x in the term x is unguarded. Thus, if there is a premise $P_i \xrightarrow{a} R$ or $P_i \stackrel{a}{\not\longrightarrow}$ then i must be an unguarded argument of f, and hence $P \rightsquigarrow P_i$. In that case $e(P_i) < e(P)$, and I may apply the induction hypothesis.

If the substitution instance of r has a premise $P_i \xrightarrow{a} R$, then $(P_i \xrightarrow{a} R) \in PT^+$, and by induction $(P_i \xrightarrow{a} R) \in CT^+$. If it has a premise $P_i \stackrel{a}{\not\longrightarrow}$, then $(P_i \stackrel{a}{\not\longrightarrow}) \in PT^-$. I show that $(P_i \stackrel{a}{\not\longrightarrow}) \in CT^-$ by contradiction. Namely, if $(P_i \stackrel{a}{\not\longrightarrow}) \notin CT^-$, then $(P_i \xrightarrow{a} Q) \in PT^+$ for some $Q \in \mathrm{T}(\Sigma)$, by Observation 1. In that case $(P_i \xrightarrow{a} Q) \in CT^+$ by induction, which implies $(P_i \stackrel{a}{\not\longrightarrow}) \notin PT^-$, again by Observation 1, so that a contradiction is reached.

Applying the rule r again, I obtain $(P \xrightarrow{a} P') \in CT^+$.

In case $P = \langle X|\mathcal{S}\rangle$, the transition $(P \xrightarrow{a} P') \in PT^+$ must be derivable from PT^- using a proof rule for recursion with premise $\langle \mathcal{S}_X|\mathcal{S}\rangle \xrightarrow{a} P'$. Since $e(\langle \mathcal{S}_X|\mathcal{S}\rangle) < e(\langle X|\mathcal{S}\rangle)$ I may apply the induction hypothesis. So from $(\langle \mathcal{S}_X|\mathcal{S}\rangle \xrightarrow{a} P') \in PR^+$ I conclude $(\langle \mathcal{S}_X|\mathcal{S}\rangle \xrightarrow{a} P') \in CR^+$ by induction, and this yields $(P \xrightarrow{a} P') \in CT^+$. □

In [6] it was observed that processes in GSOS languages $(\Sigma, \mathcal{R})$ are finitely branching, in the sense that for each $P \in \mathrm{T}(\Sigma)$ there are only finitely many pairs $(a, P^\dagger) \in A \times \mathrm{T}(\Sigma)$ such that $P \xrightarrow{a} P^\dagger$. Here I show

[5] Here I use that a term $\langle \mathcal{S}_{X_i}|\mathcal{S}\rangle$ cannot contain, as a subterm, a recursive call $\langle Z|\mathcal{S}'\rangle$ with $\mathcal{S}' \neq \mathcal{S}$, such that $\langle Z|\mathcal{S}'\rangle$ has a subterm of the form $\langle Y|\mathcal{S}\rangle$. This is because terms and recursive specifications are constructed inductively, and if $\mathcal{S}$ is a recursive specification with a subterm $\langle Z|\mathcal{S}'\rangle$ then $\mathcal{S}'$ cannot have a subterm $\langle Y|\mathcal{S}\rangle$.

that this observation remains valid now that I have generalised the definition of guardedness. In [6, 7] GSOS languages are required to feature only finitely many GSOS rules, and each of these rules has only finitely many premises. In the following I need a weaker assumption, namely that each operator $f \in \Sigma$ occurs in the source of only finitely many abstract GSOS rules, and each of these rules has only finitely many positive premises. Here I define an *abstract GSOS rule* as a (possibly infinite) family of GSOS rules that differ only in their transition labels, and with the property that for each choice of transition labels in the premises there are only finitely many choices for the transition label of the conclusion. GSOS languages that satisfy this requirement I here call *finitary*.

Proposition 2 Each process in a finitary GSOS language is finitely branching.

Proof: Let $\mathcal{T} = (\Sigma, \mathcal{R})$ be the TSS of a finitary GSOS language. Again I apply the metric $e(P)$ defined in the previous proof. I show with induction on $e(P)$ that there are only finitely many proofs π of transitions with source P that may use negative literals from CT^- as assumptions. This is sufficient, since $(P \xrightarrow{a} P') \in CT^+$ iff $\mathcal{T} \vdash \frac{CT^-}{P \xrightarrow{a} P'}$. If π derives a transition $Q \xrightarrow{a} R$, then I call Q the *source* of π.

In case $P = f(P_1, \ldots, P_{ar(f)})$, a proof π with source P is completely determined by the concluding abstract rule from $\mathcal{R}$, the label of the concluding transition, and the subderivations of π with source P_i for some of the $i \in \{1, \ldots, ar(f)\}$. Moreover, as already argued in the previous proof, such subderivations only exists when i is an unguarded argument of f, so that $e(P_i) < e(P)$. The choice of the concluding abstract rule r depends on f, and for each f there are only finitely many choices. Moreover, using that $e(P_i) < e(P)$, by induction there are only finitely many subderivations of π with source P_i. Finally, for each collection of premises of an instance of r (determined by these subderivations) there are only finitely many possibilities for the label of the concluding transition. This shows there are only finitely many choices for π.

In case $P = \langle X|\mathcal{S}\rangle$, the last step in π must be application of the rules for recursion, so π is completely determined by a subderivation π' of a transition with source $\langle \mathcal{S}_X|\mathcal{S}\rangle$. By induction there are only finitely many choices for π', and hence also for π. □

11 A complete axiomatisation of bisimilarity for GSOS languages

Let BCCSP [18] be the common kernel of the process algebras CCS [30] and CSP [10], consisting of the operators 0, $a.\_$ for $a \in A$ and $\_+\_$. It is well-known [30, 3, 12] that the collection Ax of the four axioms A1–4 constitutes a sound and ground-complete axiomatisation of bisimilarity over BCCSP.

A1	$x+y = y+x$		$\pi_0(x) = 0$
A2	$(x+y)+z = x+(y+z)$	$\dfrac{x \xrightarrow{a} y}{\pi_{n+1} \xrightarrow{a} \pi_n(y)}$	$\pi_{n+1}(0) = 0$
A3	$x+x = x$		$\pi_{n+1}(a.x) = a.\pi_n(x)$
A4	$x+0 = x$		$\pi_{n+1}(x+y) = \pi_{n+1}(x)+\pi_{n+1}(y)$

This means that $P \equiv_B Q \Leftrightarrow Ax \vdash P = Q$ for all BCCSP processes P and Q. Here provability is w.r.t. equational logic. To reason in an algebraic way about BCCSP processes extended with guarded recursion, the *Approximation Induction Principle* (AIP) [2, 3, 12] can be used. It employs *projection operators* π_n for all $n \in \mathbb{N}$ that cut off a process after n transitions. There transition rules are displayed above. Complete axioms for these operators, when added to BCCSP with recursion, are displayed above as well. Now AIP says that equivalence of processes is determined by the equivalence of their finite projections. It can be formulated as

$$\frac{\forall n.\ \ \pi_n(x) = \pi_n(y)}{x = y}\ .$$

As shown, e.g., in [12], *Ax* together with RDP and AIP, and of course the projection axioms, constitutes a sound and ground-complete axiomatisation of BCCSP with guarded recursion. Here, provability is w.r.t. infinitary conditional equational logic [22].

ACETO, BLOOM & VAANDRAGER [1] extended this sound and ground-complete axiomatisation to finite GSOS languages $(\Sigma, \mathcal{R})$ that contain BCCSP as a sublanguage. Their key contribution is to provide for each such language a set of axioms that allow to convert each process into a head-normal form.

Definition 17 A *head formal form* is a process of the form $\Sigma_{i\in I} a_i.P_i$ with $a_i \in A$ and $P_i \in \mathrm{T}(\Sigma)$ for all i from a finite index set I.

In generating these head-normalising axioms some auxiliary operators need to be added to language. In [1] only recursion-free GSOS languages are studied with finitely many GSOS rules, in which each rule has only finite many premises. However, their method of generating axioms that allow to convert each process into a head-normal form generalises straightforwardly to finitary GSOS languages, allowing guarded recursion, now also using RDP among the axioms. As a consequence, *Ax* together with RDP, AIP, the projection axioms and the head-normalising axioms from [1] constitutes a sound and ground-complete axiomatisation of any finitary GSOS language that contains BCCSP as a sublanguage.

I here contribute that in the above ground-complete axiomatisation result one can replace AIP by RSP. In [12, Section 4.4] it is shown that the basic axioms that are ground-complete for recursion-free ACP processes, together with RDP and RSP, fail to constitute a ground-complete axiomatisation of bisimilarity for ACP with guarded recursion. However, this is because in [12] only recursive specifications with finitely many recursion equations are allowed. When allowing arbitrary many recursive equations, the counterexample of [12] goes away. In fact, [12, Theorem 4.4.1] shows that the basic axioms that are ground-complete for recursion-free ACP processes, together with RDP and RSP, do form a ground-complete axiomatisation for ACP processes with finite, linear, guarded recursive specifications. Here *linear* says in essence that within these recursive specifications only BCCSP operators are allowed (except that in the setting of ACP one employs a variant that distinguishes deadlock from successful termination). As observed in [21], nothing in the proof of [12, Theorem 4.4.1] depends on the finiteness of recursive specifications, so the result holds without that disclaimer. This does turn RSP into an infinitary conditional equation, just like AIP, so that also here infinitary conditional equational logic needs to be employed.

VAN GLABBEEK & MIDDELBURG [21] also show that using RSP, any guarded recursive specification can be converted into a linear, guarded recursive specification. This shows that the basic axioms that are ground-complete for recursion-free ACP processes, together with RDP and RSP, do form a sound and ground-complete axiomatisation of ACP with guarded recursion. Moreover, nothing in the reasoning of [21] is specific to the language ACP. The only relevant fact about ACP that is incorporated in the proof is [21, Proposition 1], saying that each ACP process with guarded recursion can be brought into head normal form. Thus, combining the insights of [1, 12, 21], it follows that *Ax* together with RDP, RSP and the head-normalising axioms from [1] constitutes a sound and ground-complete axiomatisation of any finitary GSOS language that contains BCCSP as a sublanguage.

12 Future work

It would be great to extend this work to branching bisimilarity and/or other weak equivalences.

References

[1] L. Aceto, B. Bloom & F.W. Vaandrager (1994): *Turning SOS Rules into Equations*. *Information and Computation* 111(1), pp. 1–52, doi:10.1006/inco.1994.1040.

[2] J.C.M. Baeten, J.A. Bergstra & J.W. Klop (1987): *On the consistency of Koomen's fair abstraction rule*. *Theoretical Computer Science* 51(1/2), pp. 129–176, doi:10.1016/0304-3975(87)90052-1.

[3] J.C.M. Baeten & W.P. Weijland (1990): *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, doi:10.1017/CBO9780511624193.

[4] J.A. Bergstra & J.W. Klop (1988): *A complete inference system for regular processes with silent moves*. In F.R. Drake & J.K. Truss, editors: *Proceedings Logic Colloquium 1986* Hull, North-Holland, pp. 21–81, doi:10.1016/S0049-237X(09)70651-2. First appeared as: Report CS-R8420, CWI, Amsterdam, 1984.

[5] B. Bloom, W.J. Fokkink & R.J. van Glabbeek (2004): *Precongruence Formats for Decorated Trace Semantics*. *Transactions on Computational Logic* 5(1), pp. 26–78, doi:10.1145/963927.963929.

[6] B. Bloom, S. Istrail & A.R. Meyer (1988): *Bisimulation Can't be Traced: Preliminary Report*. In: *Conference Record of the 15^{th} ACM Symposium on Principles of Programming Languages*, San Diego, California, pp. 229–239, doi:10.1145/73560.73580.

[7] B. Bloom, S. Istrail & A.R. Meyer (1995): *Bisimulation Can't be Traced*. *Journal of the ACM* 42(1), pp. 232–268, doi:10.1145/200836.200876.

[8] 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, doi:10.1145/234752.234756.

[9] E. Bres, R.J. van Glabbeek & P. Höfner (2016): *A Timed Process Algebra for Wireless Networks with an Application in Routing*. Technical Report 9145, NICTA. Available at `http://arxiv.org/abs/1606.03663`. Extended abstract in P. Thiemann, editor: *Programming Languages and Systems:* Proceedings 25th *European Symposium on Programming*, ESOP'16; held as part of the *European Joint Conferences on Theory and Practice of Software*, ETAPS'16, Eindhoven, The Netherlands, April 2016, LNCS 9632, Springer, 2016, pp. 95-122, doi:10.1007/978-3-662-49498-1_5.

[10] S.D. Brookes, C.A.R. Hoare & A.W. Roscoe (1984): *A theory of communicating sequential processes*. *Journal of the ACM* 31(3), pp. 560–599, doi:10.1145/828.833.

[11] R. De Nicola & M. Hennessy (1984): *Testing equivalences for processes*. *Theoretical Computer Science* 34, pp. 83–133, doi:10.1016/0304-3975(84)90113-0.

[12] W. J. Fokkink (2000): *Introduction to Process Algebra*. Texts in Theoretical Computer Science, An EATCS Series, Springer, doi:10.1007/978-3-662-04293-9.

[13] W.J. Fokkink & R.J. van Glabbeek (1996): *Ntyft/ntyxt rules reduce to ntree rules*. *Information and Computation* 126(1), pp. 1–10, doi:10.1006/inco.1996.0030.

[14] W.J. Fokkink, R.J. van Glabbeek & P. de Wind (2012): *Divide and congruence: From decomposition of modal formulas to preservation of branching and η-bisimilarity*. *Information and Computation* 214, pp. 59–85, doi:10.1016/j.ic.2011.10.011.

[15] A. van Gelder, K. Ross & J.S. Schlipf (1991): *The well-founded semantics for general logic programs*. *Journal of the ACM* 38(3), pp. 620–650, doi:10.1145/116825.116838.

[16] R.J. van Glabbeek (1993): *Full Abstraction in Structural Operational Semantics (extended abstract)*. In M. Nivat, C. Rattray, T. Rus & G. Scollo, editors: Proceedings of the 3^{rd} International Conference on *Algebraic Methodology and Software Technology*, AMAST'93, Twente, The Netherlands, June l993, Workshops in Computing, Springer, pp. 75–82, doi:10.1007/978-1-4471-3227-1_7. Available at `http://theory.stanford.edu/~rvg/abstracts.html#28`.

[17] R.J. van Glabbeek (1993): *The Linear Time – Branching Time Spectrum II; The semantics of sequential systems with silent moves (extended abstract)*. In E. Best, editor: Proceedings *CONCUR'93*, 4^{th} International Conference on *Concurrency Theory*, Hildesheim, Germany, August 1993, LNCS 715, Springer, pp. 66–81, doi:10.1007/3-540-57208-2_6.

[18] R.J. van Glabbeek (2001): *The Linear Time – Branching Time Spectrum I; The Semantics of Concrete, Sequential Processes*. In J.A. Bergstra, A. Ponse & S.A. Smolka, editors: *Handbook of Process Algebra*, chapter 1, Elsevier, pp. 3–99, doi:10.1016/B978-044482830-9/50019-9.

[19] 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, doi:10.1016/j.jlap.2004.03.007.

[20] R.J. van Glabbeek (2017): *Lean and Full Congruence Formats for Recursion*. In: Proceedings 32^{nd} Annual ACM/IEEE Symposium on *Logic in Computer Science*, LICS'17, Reykjavik, Iceland, June 2017, IEEE Computer Society Press, doi:10.1109/LICS.2017.8005142.

[21] R.J. van Glabbeek & C.A. Middelburg (2020): *On Infinite Guarded Recursive Specifications in Process Algebra*. Available at `http://arxiv.org/abs/2005.00746`.

[22] R.J. van Glabbeek & F.W. Vaandrager (1993): *Modular Specification of Process Algebras*. *Theoretical Computer Science* 113(2), pp. 293–348, doi:10.1016/0304-3975(93)90006-F.

[23] J.F. Groote (1993): *Transition System Specifications with Negative Premises*. *Theoretical Computer Science* 118, pp. 263–299, doi:10.1016/0304-3975(93)90111-6.

[24] J.F. Groote & F.W. Vaandrager (1992): *Structured Operational Semantics and Bisimulation as a Congruence*. *Information and Computation* 100(2), pp. 202–260, doi:10.1016/0890-5401(92)90013-6.

[25] C.A.R. Hoare (1985): *Communicating Sequential Processes*. Prentice Hall, Englewood Cliffs.

[26] L. Lamport (1977): *Proving the correctness of multiprocess programs*. *IEEE Transactions on Software Engineering* 3(2), pp. 125–143, doi:10.1109/TSE.1977.229904.

[27] L. Lamport (1998): *Proving Possibility Properties*. *Theoretical Computer Science* 206(1-2), pp. 341–352, doi:10.1016/S0304-3975(98)00129-7. See also `http://research.microsoft.com/users/lamport/pubs/pubs.html#lamport-possibility`.

[28] K.G. Larsen & B. Thomsen (1988): *A Modal Process Logic*. In: Proc. LICS'88, IEEE Computer Society Press, pp. 203–210, doi:10.1109/LICS.1988.5119.

[29] R. Milner (1984): *A complete inference system for a class of regular behaviours*. *Journal of Computer and System Sciences* 28, pp. 439–466, doi:10.1016/0022-0000(84)90023-0.

[30] R. Milner (1990): *Operational and algebraic semantics of concurrent processes*. In J. van Leeuwen, editor: *Handbook of Theoretical Computer Science*, chapter 19, Elsevier Science Publishers B.V. (North-Holland), pp. 1201–1242. Alternatively see *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, 1989, of which an earlier version appeared as *A Calculus of Communicating Systems*, LNCS 92, Springer, 1980, doi:10.1007/3-540-10235-3.

[31] E.-R. Olderog & C.A.R. Hoare (1986): *Specification-oriented semantics for communicating processes*. *Acta Informatica* 23, pp. 9–66, doi:10.1007/BF00268075.

[32] G.D. Plotkin (2004): *A Structural Approach to Operational Semantics*. *The Journal of Logic and Algebraic Programming* 60–61, pp. 17–139, doi:10.1016/j.jlap.2004.05.001. Originally appeared in 1981.

[33] T.C. Przymusinski (1990): *The Well-founded Semantics Coincides with the Three-valued Stable Semantics*. *Fundamenta Informaticae* XIII(4), pp. 445–463, doi:10.3233/FI-1990-13404.

[34] A.W. Roscoe (1997): *The Theory and Practice of Concurrency*. Prentice-Hall. Available at `http://www.comlab.ox.ac.uk/bill.roscoe/publications/68b.pdf`.

[35] F.W. Vaandrager (1993): *Expressiveness Results for Process Algebras*. In J.W. de Bakker, W.P. de Roever & G. Rozenberg, editors: *Proceedings REX Workshop on Semantics: Foundations and Applications*, Beekbergen, The Netherlands, 1992, LNCS 666, Springer, pp. 609–638, doi:10.1007/3-540-56596-5_49.