Sequential Value Passing yields a Kleene Theorem for Processes

Jos C.M. Baeten[1] and Bas Luttik[2]

[1] CWI, Amsterdam, The Netherlands
[2] Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract. Communication with value passing has received ample attention in process theory. Value passing through a sequential composition has received much less attention. In recent work, we found that sequential value passing is the essential ingredient to prove the analogue of the classical theorem of the equivalence of pushdown automata and context-free grammars in a setting of interactive processes and bisimulation. Subsequently, we found that the treatment of sequential value passing in the process setting can be simplified considerably. We report on this simplification here, and find another application of sequential value passing, viz. a Kleene theorem for processes.

Keywords: sequential value passing · process theory · pushdown automaton · context-free grammar · bisimilarity · Kleene theorem.

We dedicate this paper to our valued colleague Herman Geuvers on the occasion of his sixtieth birthday. We admire his meticulous and thorough style.

1 Introduction

This paper contributes to our ongoing project to integrate the theory of automata and formal languages on the one hand and concurrency theory on the other hand. We do not treat automata as language acceptors. Instead, we treat them as processes. That is, we view automata as transition systems, and consider them modulo bisimilarity.

It is well-known that Kleene's theorem [20], which states that a language is accepted by a finite automaton if and only if it is denoted by a regular expression, does not have a direct process-theoretic pendant modulo bisimilarity. Milner showed that there exist finite automata that are not bisimilar to the transition system associated with a regular expression [23]. We shall prove in this paper that it suffices to extend regular expressions with a simple notion of sequential value passing to obtain a process-theoretic variant of Kleene's theorem.

In [5], it was shown that a Kleene theorem can also be obtained in bisimulation semantics if the syntax of regular expressions is enriched with parallel composition, synchronisation and encapsulation. The Kleene theorem we establish here provides an alternative to that result.

In [4], we already looked at the classical theorem that a language is accepted by a pushdown automaton if and only if it is defined by a context-free grammar. In the process setting, a context-free grammar is a process algebra with actions, choice, sequencing and recursion. We proved that every process given by a finite guarded recursive specification over this algebra is also a process defined by a pushdown automaton, but not the other way around, the algebra is not sufficiently expressive to specify all processes defined by pushdown automata. Adding sequential value passing suffices: the set of processes given by a finite guarded recursive specification over the extended algebra coincides with the set of processes defined by pushdown automata. The variant of sequential value passing used in [4], however, is semantically significantly more involved than the variant that we propose here. Thus, we also present a simplification of the result in [4].

Communication with value passing has received ample attention in process theory (see, e.g., [19]). Value passing through a sequential composition has received much less attention (but see, e.g., [25, 13]). Kleene algebra with tests, proposed in [21], gives an axiomatic treatment of regular expressions with conditionals. There are two important differences between this theory and the theory presented in this article. First, Kleene algebra with tests is catered towards language equivalence; indeed, it includes axioms that are not valid in bisimulation semantics. Second, it does not include a facility to specify sequential passing which, as we will show, is an essential ingredient to get a Kleene theorem in bisimulation semantics. In [18], a process algebra with guards is proposed, which yields a treatment of conditionals in bisimulation semantics. Also this work, however, lacks a facility to specify sequential value passing; state attributes are, instead, changed implicitly through an effect associated with actions. This approach makes it unsuitable for establishing the type of correspondence results we obtain in this article.

2 Preliminaries

As a common semantic framework we use the notion of a *transition system*.

Definition 1. *A* transition system space *is a quadruple* $(\mathcal{S}, \mathcal{A}, \rightarrow, \downarrow)$*, where*

1. $\mathcal{S}$ *is a set of* states*;*
2. $\mathcal{A}$ *is a set of* actions*;*
3. $\rightarrow \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ *is an $\mathcal{A}$-labelled* transition relation*; and*
4. $\downarrow \subseteq \mathcal{S}$ *is the set of final or accepting states.*

A transition system *is a transition system space with a special designated* root state *or* initial state $\uparrow$*, i.e., it is a quintuple* $(\mathcal{S}, \mathcal{A}, \rightarrow, \uparrow, \downarrow)$ *such that* $(\mathcal{S}, \mathcal{A}, \rightarrow, \downarrow)$ *is a transition system space, and* $\uparrow \in \mathcal{S}$*.*

Note that, by the requirement that there is a designated root state, a transition system has a non-empty set of states. It will be technically convenient to also consider the structure with an empty set of states, an empty transition relation, an empty set of *accepting states and an undefined root state. This structure will be referred to as the* inconsistent *transition system.*

We write $s \xrightarrow{a} s'$ for $(s, a, s') \in \rightarrow$ and $s\downarrow$ for $s \in \downarrow$. We say that s' is reachable *from s if, for some $n \in \mathbb{N}$, there exist $s_0, \ldots, s_n \in \mathcal{S}$ and $a_1, \ldots, a_n \in \mathcal{A}$ such that $s = s_0$, $s_i \xrightarrow{a_{i+1}} s_{i+1}$ for all $0 \leq i < n$, and $s_n = s'$. A transition system that has finitely many states reachable from the root state (if it exists) and finitely many transitions between them is called a* finite automaton*.*

By considering language equivalence classes of transition systems, we recover languages as a semantics, but we can also consider other equivalence relations. Notable among these is *bisimilarity*.

Definition 2. *Let $(\mathcal{S}, \mathcal{A}, \rightarrow, \downarrow)$ be a transition system space. A symmetric binary relation R on $\mathcal{S}$ is a* bisimulation *if it satisfies the following conditions for every $s, t \in \mathcal{S}$ such that $s\ R\ t$ and for all $a \in \mathcal{A}$:*

1. *if $s \xrightarrow{a} s'$ for some $s' \in \mathcal{S}$, then there is a $t' \in \mathcal{S}$ such that $t \xrightarrow{a} t'$ and $s'\ R\ t'$; and*
2. *if $s\downarrow$, then $t\downarrow$.*

We write $s \leftrightarrow t$ if and only if there a bisimulation relating s and t, and say that s is bisimilar to t.

The results of this paper do not rely on abstraction from internal computations, so we do not consider the silent step τ here, and we can use the *strong* version of bisimilarity defined above, which does not give special treatment to τ-labelled transitions. But in general (in other work) we have to use a version of bisimilarity that accomodates for abstraction from internal activity; the finest such notion of bisimilarity is *divergence-preserving branching bisimilarity*, which was introduced in [14] (see also [22] for an overview of recent results).

We see that bisimilarity is an equivalence relation on a transition system space, so it divides a transition system space into a number of equivalence classes.

3 Regular Expressions

In [23], Robin Milner considered regular expressions in the process setting (we define these below), and found that not all finite automata can be defined by a regular expression (modulo bisimulation). He posed the question how the set of finite automata defined by a regular expression can be characterized, a question that was solved in [2]. Here, we answer the question which ingredient needs to be added to regular expressions to characterize all finite automata modulo bisimulation.

We present regular expressions as the closed terms in the theory TSP+IT of [8]. The syntax of this theory has the following elements:

- $\mathbf{0}$ is the inactive and not accepting process (deadlock), the one-state automaton without transitions where the state is not final;
- $\mathbf{1}$ is the inactive and accepting process, the one-state automaton without transitions where the state is final;

– for a given set $\mathcal{A}$ of actions we have the prefix operators $a.\_$ for each $a \in \mathcal{A}$;
– the binary operator $+$ is alternative composition or choice;
– the binary operator $\cdot$ is sequential composition;
– the unary operator $\_^*$ is iteration or Kleene star.

We give the behaviour of terms over this algebra by means of structural operational semantics (see [1]): we define a unary acceptance or termination predicate $\downarrow$ (written postfix) and, for every $a \in \mathcal{A}$, a binary transition relation $\xrightarrow{a}$ (written infix), by means of the transition system specification in Table 1. The rules in Table 1 should be read as follows: if the premises above the line are satisfied for a certain substitution, then the conclusion(s) below the line are also valid for the same substitution. These rules turn the set of regular expressions into a transition system space. Each regular expression has a transition system in which every step and every termination is provable from this specification. Each regular expression has a transition system with finitely many reachable states and finitely many transitions, so is a finite automaton.

A *regular process* is a bisimulation equivalence class of finite automata.

$$\frac{}{a.x \xrightarrow{a} x} \qquad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x' \quad y + x \xrightarrow{a} x'}$$

$$\frac{}{\mathbf{1}\downarrow} \qquad \frac{x\downarrow}{x + y\downarrow \quad y + x\downarrow}$$

$$\frac{x\downarrow \quad y\downarrow}{x \cdot y\downarrow} \qquad \frac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y} \qquad \frac{x\downarrow \quad y \xrightarrow{a} y'}{x \cdot y \xrightarrow{a} y'}$$

$$\frac{}{x^*\downarrow} \qquad \frac{x \xrightarrow{a} x'}{x^* \xrightarrow{a} x' \cdot x^*}$$

Table 1. Operational semantics for regular expressions.

Since the rules in Table 1 are in *path* format (see [7]), bisimilarity is a congruence relation for the operators of regular expressions. Consequently, we can consider the equational theory of regular expressions. From [10, 8], we know that there is no finite axiomatization that is sound and ground-complete. For more information, see [16]. For all regular expressions x, we have $\mathbf{1} \cdot x \mathrel{\underline{\leftrightarrow}} x \mathrel{\underline{\leftrightarrow}} x \cdot \mathbf{1}$ and $\mathbf{0} \cdot x \mathrel{\underline{\leftrightarrow}} \mathbf{0}$ (but not $x \cdot \mathbf{0} \mathrel{\underline{\leftrightarrow}} \mathbf{0}$!), which gives us the motivation to use the symbols $\mathbf{1}$ and $\mathbf{0}$.

We see alternative composition is commutative and associative, and sequential composition is associative but not commutative, and so we can leave out brackets as usual.

We further note that, modulo bisimilarity, sequential composition distributes from the right over choice ($(x + y) \cdot z \mathrel{\underline{\leftrightarrow}} x \cdot z + y \cdot z$), but not from the left ($x \cdot (y + z) \not\mathrel{\underline{\leftrightarrow}} x \cdot y + x \cdot z$). If we consider the classical algorithm that finds a regular expression with the same language as a given finite automaton, we see that it makes essential use of the distributivity law that is not valid for bisimulation.

Each regular expression generates a finite automaton. The reverse direction is valid in language equivalence, but not in bisimilarity, as the following example shows (here, the state on the left is the root state, and both states are accepting).

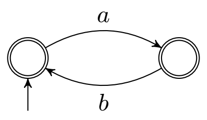

Fig. 1. A finite automaton not denoted by a regular expression.

Theorem 1. *There is no regular expression of which the transition system is bisimilar to the finite automaton in Fig. 1.*

Proof. Milner [23] proves this for a slightly more complicated example than the one in Fig. 1. His proof can be easily transposed to the present situation.

Thus, we look for an extension of TSP+IT in order to find expressions for all finite automata. In [5], extensions with various forms of parallel composition are studied. There, it is found that an extension with parallel composition and value passing communication yields all regular processes. Here, we present an extension that is more straightforward: just extending with sequential value passing suffices. In order to define sequential value passing, we use operators for signals and conditions, based on [3, 8].

4 Signals and Conditions

The key to the extension is that information is needed about the state which the process is in at a given time. In [4], we used expressions in propositional logic to express a property of the current state of a process. This gives considerable overhead, as we need to evaluate the expressions again at each step in a process, assigning truth values to the propositional variables that occur in an expression. Here, we just assume that the states of a process have an attribute that takes a unique value from a finite data set $\mathcal{D}$. We need that $\mathcal{D}$ is finite, since we will use in examples indexed summations over elements of $\mathcal{D}$, and we can use $\sum_{d\in\mathcal{D}}$ notation as an abbreviation. A difficulty is that we need to ensure that the attribute of a state has a unique value: it is impossible to have two different values of an attribute at the same time.

First, we introduce an operator, called the *root-signal emission operator* $\wedge\!\!\!\rightarrow$ in [3, 8], that exposes the value of the attribute. A term $d \wedge\!\!\!\rightarrow x$ represents the process x that starts out in a state in which the attribute has the value d. Only one value can be shown at a time, so we need to declare terms like . We associate the inconsistent transition system with these inconsistent terms. In the operational semantics, we add an extra predicate $\mathcal{C}$, indicating that the term is consistent, and giving the value if there is one.

Thus, for $d \in \mathcal{D}$, $x\ \mathcal{C}d$ means that (the initial state of) expression x is consistent, and has value d. On the other hand, $x\ \mathcal{C}\ominus$ means that (the initial state of) expression x is consistent, and no value is given. We write $\mathcal{D}_\ominus$ for $\mathcal{D} \cup \{\ominus\}$. We use $d, e \in \mathcal{D}$ and $\delta, \varepsilon \in \mathcal{D}_\ominus$. We have a very simple partial ordering on $\mathcal{D}_\ominus$: $\ominus$ is below all elements of $\mathcal{D}$, and elements of $\mathcal{D}$ are incomparable. We write $\delta \preceq \varepsilon$ if $\delta = \varepsilon$ or $\delta = \ominus$. We define the predicate $\mathcal{C}$ by the operational rules in Table 2 for regular expressions and the root-signal emission operator.

$$\frac{}{\mathbf{0}\ \mathcal{C}\ominus} \qquad \frac{}{\mathbf{1}\ \mathcal{C}\ominus} \qquad \frac{}{a.x\ \mathcal{C}\ominus}$$

$$\frac{x\ \mathcal{C}\delta \quad y\ \mathcal{C}\varepsilon \quad \delta \preceq \varepsilon}{x + y\ \mathcal{C}\varepsilon \quad y + x\ \mathcal{C}\varepsilon} \qquad \frac{x\ \mathcal{C}\delta \quad x\not\downarrow}{x \cdot y\ \mathcal{C}\delta}$$

$$\frac{x\downarrow \quad x\ \mathcal{C}\delta \quad y\ \mathcal{C}\varepsilon \quad \delta \preceq \varepsilon}{x \cdot y\ \mathcal{C}\varepsilon} \qquad \frac{x\downarrow \quad x\ \mathcal{C}\varepsilon \quad y\ \mathcal{C}\delta \quad \delta \preceq \varepsilon}{x \cdot y\ \mathcal{C}\varepsilon}$$

$$\frac{x\ \mathcal{C}\delta}{x^*\ \mathcal{C}\delta} \qquad \frac{x\ \mathcal{C}\delta \quad \delta \preceq d}{d\ {}^\wedge x\ \mathcal{C}d}$$

Table 2. Operational semantics of consistency and state values ($d \in \mathcal{D}, \delta, \varepsilon \in \mathcal{D}_\ominus$).

In Table 3, we repeat the rules of Table 1, with extra conditions to ensure consistency. In addition, we give the operational rules for the signal emission operator.

$$\frac{}{\mathbf{1}\downarrow} \qquad \frac{x\ \mathcal{C}\delta}{a.x \xrightarrow{a} x} \qquad \frac{x \xrightarrow{a} x' \quad x + y\ \mathcal{C}\delta}{x + y \xrightarrow{a} x' \quad y + x \xrightarrow{a} x'}$$

$$\frac{x\downarrow \quad x + y\ \mathcal{C}\delta}{x + y\downarrow \quad y + x\downarrow} \qquad \frac{x\downarrow \quad y\downarrow \quad x \cdot y\ \mathcal{C}\delta}{x \cdot y\downarrow}$$

$$\frac{x \xrightarrow{a} x' \quad x \cdot y\ \mathcal{C}\delta \quad x' \cdot y\ \mathcal{C}\varepsilon}{x \cdot y \xrightarrow{a} x' \cdot y} \qquad \frac{x\downarrow \quad y \xrightarrow{a} y' \quad x \cdot y\ \mathcal{C}\delta}{x \cdot y \xrightarrow{a} y'}$$

$$\frac{x\ \mathcal{C}\delta}{x^*\downarrow} \qquad \frac{x \xrightarrow{a} x' \quad x' \cdot x^*\ \mathcal{C}\delta}{x^* \xrightarrow{a} x' \cdot x^*}$$

$$\frac{x\downarrow \quad d\ {}^\wedge x\ \mathcal{C}d}{d\,{}^\wedge x\downarrow} \qquad \frac{x \xrightarrow{a} x' \quad d\ {}^\wedge x\ \mathcal{C}d}{d\,{}^\wedge x \xrightarrow{a} x'}$$

Table 3. Operational semantics for regular expressions and signal emission with consistency conditions ($\delta, \varepsilon \in \mathcal{D}_\ominus, d \in \mathcal{D}$).

Note that these rules ensure that a step can only occur between consistent terms, so if we can derive $x \xrightarrow{a} x'$, then terms x and x' are consistent. Also, if we can derive $x\downarrow$, then x is consistent.

Notice that the fifth rule in Table 2, the first rule for sequential composition, has a so-called *negative premise*: we can conclude $x \cdot y\ \mathcal{C}\delta$ provided $x{\downarrow}$ does not hold. It is well-known that transition system specifications with negative premises may not define a unique transition relation that agrees with provability from the transition system specification [17, 12, 15]. To show that the transition system specification presented here does indeed define a unique transition relation that agrees with provability, it suffices to define a *stratification* (see [17, Definition 2.11]). First note that, since the rules defining the predicates $\mathcal{C}\delta$ and $\downarrow$ do not have premises referring to $\xrightarrow{a}$, and the negative premise only occurs in a rule defining $\mathcal{C}\delta$, so we can ignore the rules with $\xrightarrow{a}$ in the conclusion. The mapping S from expressions of the form $x\ \mathcal{C}\delta$ and $x{\downarrow}$ to natural numbers defined by

$$
\begin{aligned}
&S(\mathbf{0}\ \mathcal{C}\delta) = S(\mathbf{1}\ \mathcal{C}\delta) = S(a.x\ \mathcal{C}\delta) = 0\\
&S(x + y\ \mathcal{C}\delta) = S(x \cdot y\ \mathcal{C}\delta) = S(x\ \mathcal{C}\delta) + S(y\ \mathcal{C}\delta) + 1\ \text{, and}\\
&S(d\ {}^\wedge x\ \ \mathcal{C}\delta) = S(x^*\ \mathcal{C}\delta) = S(x{\downarrow}) = S(x\ \mathcal{C}\delta)
\end{aligned}
$$

is a stratification. In [17] it is proved that whenever a stratification exists, then the transition system specification defines a unique transition relation that agrees with provability in the transition system specification.

The rules in Tables 2 and 3 turn the set of consistent expressions over the extended syntax into a transition system space, and thus Definition 2 yields a notion of bisimilarity on the consistent expressions. We will have no need to consider inconsistent expressions in this paper, or to do calculations on expressions that may be inconsistent, so we will not define bisimilarity on inconsistent expressions. By interpreting all inconsistent expressions as the inconsistent transition system, that is not hard to do, however.

Note that the set of consistent expressions is not closed under alternative composition: $d{}^\wedge\mathbf{1}$ and $e{}^\wedge\mathbf{1}$ are both consistent, but, if $d \neq e$, then $d{}^\wedge\mathbf{1} + e{}^\wedge\mathbf{1}$ is not consistent. Further note that bisimilarity as defined in Definition 2 is not a congruence on the set of all expressions, as $d{}^\wedge\mathbf{1} \leftrightarrows e{}^\wedge\mathbf{1}$, but $d{}^\wedge\mathbf{1} + d{}^\wedge\mathbf{1}$ is not bisimilar to $d{}^\wedge\mathbf{1} + e{}^\wedge\mathbf{1}$. It is not difficult to define a variant of bisimulation on the set of all expressions that is a congruence. All that is required is that a term satisfying $\mathcal{C}\delta$ can only be related to a term also satisfying $\mathcal{C}\delta$. As all operational rules are in *panth* format, as defined in [24], the resulting bisimilarity is a congruence. We leave the precise formulation as further work, as we do not use this notion in the present paper.

An expression over this extended syntax that satisfies $\mathcal{C}\delta$ for some $\delta \in \mathcal{D}_\ominus$ is consistent, and exposes the attribute d of the state when it satisfies $\mathcal{C}d$ for some $d \in \mathcal{D}$. We can depict the attribute values in a state, as shown in Fig. 2 for the term $d{}^\wedge(\mathbf{1} + a.(e{}^\wedge\mathbf{0}))$, but we emphasize that these values are not part of the transition system, they just occur to help the reader.

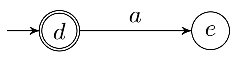

Fig. 2. Example showing attribute values.

Next, we define the *guarded command*. Given a attribute value d, we write $d :\rightarrow x$, with the intuitive meaning '*if the attribute of the current state has value* d *then* x *can be executed*'. Thus, d is a guard: x can only be executed in a state with attribute value d.

In the operational semantics, we use as additional relations the conditional steps $\xrightarrow{d,a}$ ($d \in \mathcal{D}, a \in \mathcal{A}$) and as additional predicates the conditional acceptance ${}^{d}{\downarrow}$ ($d \in \mathcal{D}$). In Table 4, we give the operational rules for guarded command.

$$\frac{x\ \mathcal{C}\delta \qquad \delta \preceq d}{d :\rightarrow x\ \mathcal{C}\ominus}$$

$$\frac{x \xrightarrow{a} x' \qquad d :\rightarrow x\ \mathcal{C}\ominus}{d :\rightarrow x \xrightarrow{d,a} x'} \qquad \frac{x \xrightarrow{d,a} x'}{d :\rightarrow x \xrightarrow{d,a} x'}$$

$$\frac{x\downarrow \qquad d :\rightarrow x\ \mathcal{C}\ominus}{d :\rightarrow x\ {}^{d}{\downarrow}} \qquad \frac{x\ {}^{d}{\downarrow}}{d :\rightarrow x\ {}^{d}{\downarrow}}$$

Table 4. Operational semantics for guarded command ($\delta \in \mathcal{D}_{\ominus}, d \in \mathcal{D}$).

In Table 5, we give the conditional steps and conditional acceptance for regular expressions and root signal emission.

Notice that if $d \neq e$, then the term $d :\rightarrow (e\,\hat{}\,\mathbf{1})$ is inconsistent, whereas the term $d\,\hat{}\,(e :\rightarrow \mathbf{1})$ is consistent, and bisimilar to $d\,\hat{}\,\mathbf{0}$. Notice that a consistent term of the form $d\,\hat{}\,x$ can do no conditional steps and no conditional termination: all steps $x \xrightarrow{e,a} x'$ and $x\ {}^{e}{\downarrow}$ for $e \neq d$ disappear, and for all steps $x \xrightarrow{d,a} x'$ and $x\ {}^{d}{\downarrow}$ the conditions are removed.

We can easily extend the stratification given above to include the guarded command operator by defining $S(d :\rightarrow x\ \mathcal{C}\delta) = S(x\ {}^{d}{\downarrow}) = S(x\ \mathcal{C}\delta)$; as above, there is no need to consider rules with $\xrightarrow{d,a}$ in the conclusion. So the extended set of rules define a transition system space on the consistent expressions, and we also have a notion of bisimilarity on the extended set of consistent terms.

Now in order to get a notion of bisimilarity that is a congruence on the extended syntax, we also need to relate the conditional steps and conditional acceptance of bisimilar terms. We have no use for this bisimilarity in the present paper, so do not give the details here. We emphasize that the conditional steps and conditional accceptance are only needed to generate the (unconditional) steps and acceptance of the transition system of a term. Thus, the conditional steps and conditional acceptance do not appear in the generated transition system space.

In order to illustrate the interplay of root signal emission and guarded command, and to show how nondeterminism can be dealt with, we give the following example.

Example 1. A coin toss can be described by the following term:

$$\mathit{Toss} = \mathit{toss}.(\mathit{heads}\,\hat{}\,\mathbf{1}) + \mathit{toss}.(\mathit{tails}\,\hat{}\,\mathbf{1}).$$

$$\frac{x \xrightarrow{d,a} x' \quad y\,\mathcal{C}d}{x+y \xrightarrow{a} x' \quad y+x \xrightarrow{a} x'} \qquad \frac{x \xrightarrow{d,a} x' \quad y\,\mathcal{C}\ominus}{x+y \xrightarrow{d,a} x' \quad y+x \xrightarrow{d,a} x'}$$

$$\frac{x\,{}^{d}\!\downarrow \quad y\,\mathcal{C}d}{x+y\downarrow \quad y+x\downarrow} \qquad \frac{x\,{}^{d}\!\downarrow \quad y\,\mathcal{C}\ominus}{x+y\,{}^{d}\!\downarrow \quad y+x\,{}^{d}\!\downarrow}$$

$$\frac{x\,{}^{d}\!\downarrow \quad y\,{}^{d}\!\downarrow}{x\cdot y\,{}^{d}\!\downarrow} \qquad \frac{x\,{}^{d}\!\downarrow \quad y\downarrow \quad y\,\mathcal{C}\delta \quad \delta \preceq d}{x\cdot y\,{}^{d}\!\downarrow}$$

$$\frac{x\downarrow \quad x\,\mathcal{C}d \quad y\,{}^{d}\!\downarrow}{x\cdot y\downarrow} \qquad \frac{x\downarrow \quad x\,\mathcal{C}\ominus \quad y\,{}^{d}\!\downarrow}{x\cdot y\,{}^{d}\!\downarrow}$$

$$\frac{x \xrightarrow{d,a} x' \quad x\cdot y\,\mathcal{C}d \quad x'\cdot y\,\mathcal{C}\delta}{x\cdot y \xrightarrow{a} x'\cdot y} \qquad \frac{x \xrightarrow{d,a} x' \quad x\cdot y\,\mathcal{C}\ominus \quad x'\cdot y\,\mathcal{C}\delta}{x\cdot y \xrightarrow{d,a} x'\cdot y}$$

$$\frac{x\downarrow \quad x\,\mathcal{C}d \quad y \xrightarrow{d,a} y'}{x\cdot y \xrightarrow{a} y'} \qquad \frac{x\downarrow \quad x\,\mathcal{C}\ominus \quad y \xrightarrow{d,a} y'}{x\cdot y \xrightarrow{d,a} y'}$$

$$\frac{x\,{}^{d}\!\downarrow \quad y \xrightarrow{d,a} y'}{x\cdot y \xrightarrow{d,a} y'} \qquad \frac{x\,{}^{d}\!\downarrow \quad y\,\mathcal{C}\delta \quad \delta \preceq d \quad y \xrightarrow{a} y'}{x\cdot y \xrightarrow{d,a} y'}$$

$$\frac{x \xrightarrow{d,a} x' \quad x'\cdot x^*\,\mathcal{C}\delta}{x^* \xrightarrow{d,a} x'\cdot x^*} \qquad \frac{x\,{}^{d}\!\downarrow}{d \mathbin{{}^\wedge\!\!\blacktriangle} x\downarrow} \qquad \frac{x \xrightarrow{d,a} x'}{d \mathbin{{}^\wedge\!\!\blacktriangle} x \xrightarrow{a} x'}$$

Table 5. Conditional steps and conditional termination for regular expressions and root signal emission ($d \in \mathcal{D}, \delta \in \mathcal{D}_\ominus$).

The behaviour of a player who wins one dollar when heads comes up and loses one dollar when tails comes up is specified by the term

$$\mathit{Player} = \mathit{heads} :\to \mathit{win}1\$ + \mathit{tails} :\to \mathit{lose}1\$.$$

Process $\mathit{Toss} \cdot \mathit{Player}$ shows how sequential value passing is achieved.

For a more involved example consider the process of tossing a coin until heads comes up:

$$\mathit{Toss} \cdot (\mathit{tails} :\to \mathit{Toss})^* \cdot (\mathit{heads} :\to \mathbf{1}).$$

We show the transition system of this process in Fig. 3. It is a finite automaton. We have labelled two states with their attribute values to clarify the correspondence with the given expressions, but the attributes are not formally part of the transition system.

Let us call the extended syntax TSP$^*_{sc}$, TSP with iteration, signals and conditions. Obviously, every consistent closed term over this syntax denotes a finite transition system. We use the conditional steps, conditional acceptance and signals in the operational rules to generate this transition system, but formally, they are not included in it. We now have all ingredients to prove the Kleene theorem.

Theorem 2. *Let t be a finite automaton. Then there is a consistent closed term over TSP$^*_{sc}$ of which the transition system is isomorphic to t.*

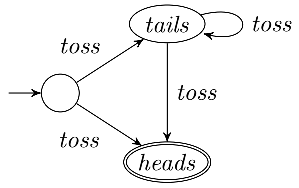

Fig. 3. The transition system of the coin toss.

Proof. Let t be a finite automaton. Assign a distinct value from $\mathcal{D}$ to each state, with i for the initial state. The term in TSP$^*_{sc}$ consists of three parts, and has the form $\mathit{Init} \cdot \mathit{Loop}^* \cdot \mathit{Exit}$.

- The initial part Init has a summand $a.(d^{\blacktriangle}\mathbf{1})$ whenever $i \xrightarrow{a} d$, and a summand $\mathbf{1}$ if $i\downarrow$;
- The looping part Loop has a summand $d :\rightarrow a.(e^{\blacktriangle}\mathbf{1})$ for every step $d \xrightarrow{a} e$ ($d, e \in \mathcal{D}$);
- The exit part Exit has a summand $d :\rightarrow \mathbf{1}$ whenever $d\downarrow$.

Notice that all three terms are consistent. The term $\mathit{Init} \cdot \mathit{Loop}^* \cdot \mathit{Exit}$ is also consistent; the transition system associated with it is isomorphic to t, so it is certainly bisimilar to t. Notice that the star height of this term is 1, i.e. there is no nesting of iterations. In fact, there is only a single iteration. As a corrolary, each consistent TSP$^*_{sc}$ term is bisimilar to a term with only one iteration, a result well-known for while programs, see [21].

To illustrate the general procedure, we give a consistent TSP$^*_{sc}$-term for the regular process of Fig. 1. We label the initial state by i, and the other state by j.

$$\mathbf{1} + a.(j^{\blacktriangle}\mathbf{1}) \cdot (i :\rightarrow a.(j^{\blacktriangle}\mathbf{1}) + j :\rightarrow b.(i^{\blacktriangle}\mathbf{1}))^* \cdot (i :\rightarrow \mathbf{1} + j :\rightarrow \mathbf{1})$$

The term starts with the options of the initial state. Next, there is the iteration of the behaviour of the process, with the final states encoded as the possible exits of the iteration.

5 Sequencing

In [4], sequential composition is not used but sequencing, as this fits better with stacks and pushdown automata (see Example 2 in the next section). It requires a number of adaptations in the theory, but we can still obtain the Kleene theorem. As the operational semantics of the iteration operator (Kleene star) is defined in terms of sequential composition, iteration based on sequencing is a different operator. When we use sequencing, we can only continue with the second component if the first component cannot do any further steps; e.g., the process $(a.\mathbf{1}+\mathbf{1});b.\mathbf{1}$ cannot do a b-step. Therefore, a loop cannot be exited by a term composed with sequencing, and we need to incorporate iteration as a *binary* operator (as, indeed, Kleene's original iteration operator, see [10]).

In this section we present TSP+IT with sequential composition $\cdot$ replaced by sequencing ;, and Kleene star $\_^*$ replaced by $\_^{*;}\_$. We denote this theory by TSP;IT. Sequencing with its operational rules was first considered in [11], and studied extensively in [6, 9]. The Kleene star based on sequencing is new here.

In Table 6, we give the operational rules for sequencing and the binary Kleene star. We write $x \overset{a}{\nrightarrow}$ for "there does not exist x' such that $x \xrightarrow{a} x'$" and $x \nrightarrow$ for "$x \overset{a}{\nrightarrow}$ for all $a \in \mathcal{A}$".

$$\frac{x\downarrow \quad y\downarrow}{x\,;y\downarrow} \qquad \frac{x \xrightarrow{a} x'}{x\,;y \xrightarrow{a} x'\,;y} \qquad \frac{x\downarrow \quad x \nrightarrow \quad y \xrightarrow{a} y'}{x\,;y \xrightarrow{a} y'}$$

$$\frac{x \xrightarrow{a} x'}{x^{*;}y \xrightarrow{a} x'\,;x^{*;}y} \qquad \frac{y\downarrow}{x^{*;}y\downarrow} \qquad \frac{y \xrightarrow{a} y'}{x^{*;}y \xrightarrow{a} y'}$$

Table 6. Operational semantics for sequencing and binary Kleene star.

The crucial difference with the sequential composition operator is the third rule for sequencing: it is only allowed to continue with the second component if the first component cannot do any step. In the rules for binary Kleene star, we see again that there is no premise of the form $x\downarrow$, so termination of the body is irrelevant, and in the last rule we see that y can take a step irrespectively of the fact whether or not x can take a step.

Again, we see the occurrence of a negative premise here, so the rules may not define a unique transition relation that agrees with provability from the transition system specification. Since the negative premises refer to the transition relation, and due to the presence of (binary) Kleene star, defining a stratification in this case is considerably more involved than before, and we leave it to future work to spell it out in detail.

To define sequential value passing, we use signals and conditions to pass state information along a sequencing operator, similar to what we did in the previous section. First of all, we consider signal emission, and the $\mathcal{C}$ predicate, in Table 7. There, we write $x \rightarrow$ for "there exist a and x' such that $x \xrightarrow{a} x'$". Note that the term $(d\,{}^\wedge a.\mathbf{1})^{*;}(e\,{}^\wedge\mathbf{1})$ is inconsistent if $d \neq e$.

Next, we modify the rules of Table 6 by adding consistency conditions, in Table 8.

Subsequently, we add the guarded command, and conditional steps and conditional termination. In Table 9, we use an extra abbreviation: we write $x \overset{d,}{\nrightarrow}$ for "there does not exist x' and $a \in \mathcal{A}$ such that $x \xrightarrow{d,a} x'$" .

Let us call the extended syntax TSP;$^*_{sc}$. Again, we have a Kleene theorem.

Theorem 3. *Let t be a finite automaton. Then there is a consistent closed term over TSP;$^*_{sc}$ of which the transition system is isomorphic to t.*

Proof. As before. The term now becomes $\mathit{Init}\,;\mathit{Loop}^{*;}\mathit{Exit}$.

$$\frac{x\,\mathcal{C}\delta \quad x\not\downarrow}{x \mathbin{;} y\,\mathcal{C}\delta} \qquad \frac{x\downarrow \quad x \rightarrow \quad x\,\mathcal{C}\delta \quad y\not\downarrow}{x \mathbin{;} y\,\mathcal{C}\delta}$$

$$\frac{x\downarrow \quad x \not\rightarrow \quad x\,\mathcal{C}\delta \quad y\,\mathcal{C}\varepsilon \quad \delta \preceq \varepsilon}{x \mathbin{;} y\,\mathcal{C}\varepsilon} \qquad \frac{x\downarrow \quad x \not\rightarrow \quad x\,\mathcal{C}\varepsilon \quad y\,\mathcal{C}\delta \quad \delta \preceq \varepsilon}{x \mathbin{;} y\,\mathcal{C}\varepsilon}$$

$$\frac{x\downarrow \quad y\downarrow \quad x\,\mathcal{C}\delta \quad y\,\mathcal{C}\varepsilon \quad \delta \preceq \varepsilon}{x \mathbin{;} y\,\mathcal{C}\varepsilon} \qquad \frac{x\downarrow \quad y\downarrow \quad x\,\mathcal{C}\varepsilon \quad y\,\mathcal{C}\delta \quad \delta \preceq \varepsilon}{x \mathbin{;} y\,\mathcal{C}\varepsilon}$$

$$\frac{x\,\mathcal{C}\delta \quad y\,\mathcal{C}\varepsilon \quad \delta \preceq \varepsilon}{x^{*;}y\,\mathcal{C}\varepsilon} \qquad \frac{x\,\mathcal{C}\varepsilon \quad y\,\mathcal{C}\delta \quad \delta \preceq \varepsilon}{x^{*;}y\,\mathcal{C}\varepsilon}$$

Table 7. Consistency and signals for sequencing and binary Kleene star ($d \in \mathcal{D}, \delta, \varepsilon \in \mathcal{D}_{\ominus}$).

$$\frac{x\downarrow \quad y\downarrow \quad x \mathbin{;} y\,\mathcal{C}\delta}{x \mathbin{;} y\downarrow} \qquad \frac{x \xrightarrow{a} x' \quad x \mathbin{;} y\,\mathcal{C}\delta \quad x' \mathbin{;} y\,\mathcal{C}\varepsilon}{x \mathbin{;} y \xrightarrow{a} x' \mathbin{;} y}$$

$$\frac{x\downarrow \quad x \not\rightarrow \quad y \xrightarrow{a} y' \quad x \mathbin{;} y\,\mathcal{C}\delta}{x \mathbin{;} y \xrightarrow{a} y'}$$

$$\frac{x^{*;}y\,\mathcal{C}\delta \quad y\downarrow}{x^{*;}y\downarrow} \qquad \frac{x \xrightarrow{a} x' \quad x^{*;}y\,\mathcal{C}\delta \quad x' \mathbin{;} x^{*;}y\,\mathcal{C}\varepsilon}{x^{*;}y \xrightarrow{a} x' \mathbin{;} x^{*;}y}$$

$$\frac{y \xrightarrow{a} y' \quad x^{*;}y\,\mathcal{C}\delta}{x^{*;}y \xrightarrow{a} y'}$$

Table 8. Operational rules for sequencing and binary Kleene star with consistency conditions ($\delta, \varepsilon \in \mathcal{D}_{\ominus}$).

6 Pushdown processes

In [4], we considered the theory TSP;$_{sc}$ obtained by leaving out the binary Kleene star operator from the theory of the previous section, and considering recursion over this theory. Take $\mathcal{P}$ to be a finite set of *process identifiers*. The set $\mathcal{P}$ is a parameter of the theory.

A recursive specification over TSP;$_{sc}$ is a mapping Δ from $\mathcal{P}$ to the set of process expressions, that may contain elements of $\mathcal{P}$. The idea is that the process expression y associated with a process identifier $X \in \mathcal{P}$ by Δ *defines* the behaviour of X. We prefer to think of Δ as a collection of *defining equations* $X \overset{\text{def}}{=} y$, exactly one for every $X \in \mathcal{P}$.

In Table 10, we provide operational rules for recursion. As stated before, the occurring negative premises may not define a unique transition relation that agrees with provability from the transition system specification. This occurs in the defining equation $X \overset{\text{def}}{=} X \mathbin{;} a.\mathbf{1} + \mathbf{1}$. For then, if $X \not\rightarrow$, according to the rules for sequencing and recursion we find that $X \xrightarrow{a} \mathbf{1}$, which is a contradiction. On the other hand, the transition $X \xrightarrow{a} \mathbf{1}$ is not provable from the transition system specification.

$$\frac{x \,{}^{d}{\downarrow} \quad y \,{}^{d}{\downarrow}}{x \,;\, y \,{}^{d}{\downarrow}} \qquad \frac{x \,{}^{d}{\downarrow} \quad y \downarrow \quad y\, \mathcal{C}\delta \quad \delta \preceq d}{x \,;\, y \,{}^{d}{\downarrow}}$$

$$\frac{x \downarrow \quad x\, \mathcal{C}d \quad y \,{}^{d}{\downarrow}}{x \,;\, y{\downarrow}} \qquad \frac{x \downarrow \quad x\, \mathcal{C}\ominus \quad y \,{}^{d}{\downarrow}}{x \,;\, y \,{}^{d}{\downarrow}}$$

$$\frac{x \xrightarrow{d,a} x' \quad x \,;\, y\, \mathcal{C}d \quad x' \,;\, y\, \mathcal{C}\delta}{x \,;\, y \xrightarrow{a} x' \,;\, y} \qquad \frac{x \xrightarrow{d,a} x' \quad x \,;\, y\, \mathcal{C}\ominus \quad x' \,;\, y\, \mathcal{C}\delta}{x \,;\, y \xrightarrow{d,a} x' \,;\, y}$$

$$\frac{x \downarrow \quad x \nrightarrow \quad x\, \mathcal{C}d \quad y \xrightarrow{d,a} y'}{x \,;\, y \xrightarrow{a} y'} \qquad \frac{x \downarrow \quad x \nrightarrow \quad x \overset{d,}{\nrightarrow} \quad x\, \mathcal{C}\ominus \quad y \xrightarrow{d,a} y'}{x \,;\, y \xrightarrow{d,a} y'}$$

$$\frac{x \,{}^{d}{\downarrow} \quad x \nrightarrow \quad x \overset{d,}{\nrightarrow} \quad y \xrightarrow{d,a} y'}{x \,;\, y \xrightarrow{d,a} y'} \qquad \frac{x \,{}^{d}{\downarrow} \quad x \nrightarrow \quad x \overset{d,}{\nrightarrow} \quad y\, \mathcal{C}\delta \quad \delta \preceq d \quad y \xrightarrow{a} y'}{x \,;\, y \xrightarrow{d,a} y'}$$

$$\frac{x \xrightarrow{d,a} x' \quad y\, \mathcal{C}\ominus \quad x' \,;\, x^{*;}y\, \mathcal{C}\delta}{x^{*;}y \xrightarrow{d,a} x' \,;\, x^{*;}y} \qquad \frac{x \xrightarrow{d,a} x' \quad y\, \mathcal{C}d \quad x' \,;\, x^{*;}y\, \mathcal{C}\delta}{x^{*;}y \xrightarrow{a} x' \,;\, x^{*;}y}$$

$$\frac{y \,{}^{d}{\downarrow} \quad x\, \mathcal{C}\ominus}{x^{*;}y \,{}^{d}{\downarrow}} \qquad \frac{y \,{}^{d}{\downarrow} \quad x\, \mathcal{C}d}{x^{*;}y \downarrow}$$

$$\frac{y \xrightarrow{d,a} y' \quad x\, \mathcal{C}\ominus}{x^{*;}y \xrightarrow{d,a} y'} \qquad \frac{y \xrightarrow{d,a} y' \quad x\, \mathcal{C}d}{x^{*;}y \xrightarrow{a} y'}$$

Table 9. Conditional steps and conditional termination for sequencing and binary Kleene star ($d \in \mathcal{D}, \delta \in \mathcal{D}_{\ominus}$).

We remedy the situation by restricting our attention to *guarded* recursive specifications, i.e., we require that every occurrence of a process identifier in the definition of some (possibly different) process identifier occurs within the scope of an action prefix. If $\varDelta$ is guarded, then it is straightforward to prove that the mapping S from process expressions to natural numbers inductively defined by $S(\mathbf{1}) = S(\mathbf{0}) = S(a.x) = 0$, $S(x_1 + x_2) = S(x_1 \,;\, x_2) = S(x_1) + S(x_2) + 1$, and $S(X) = S(y)$ if $(X \stackrel{\text{def}}{=} y) \in \varDelta$ gives rise to a stratification S' from transitions to natural numbers defined by $S'(x \xrightarrow{a} x') = S(x)$ for all $a \in \mathcal{A}$ and process expressions x and x'.

Example 2. Let us consider the stack S of unbounded capacity that is only accepting when it is empty. In [8, Section 6.6], we give the following guarded recursive specification.

$$S \stackrel{\text{def}}{=} \mathbf{1} + \sum_{d \in \mathcal{D}} push(d).T_d \cdot S \qquad T_d \stackrel{\text{def}}{=} pop(d).\mathbf{1} + \sum_{e \in \mathcal{D}} push(e).T_e \cdot T_d$$

Now suppose that we want to define the stack that is always accepting, irrespective of the contents. We show the one-state pushdown automaton of this stack in Fig. 4. The transitions on the left show that we can execute $push(d)$ for some $d \in \mathcal{D}$ whenever

$$\frac{y \xrightarrow{a} y' \quad X \overset{\text{def}}{=} y}{X \xrightarrow{a} y'} \qquad \frac{y{\downarrow} \quad X \overset{\text{def}}{=} y}{X{\downarrow}}$$

$$\frac{y\ \mathcal{C}\delta \quad X \overset{\text{def}}{=} y}{X\ \mathcal{C}\delta} \qquad \frac{y \xrightarrow{d,a} y' \quad X \overset{\text{def}}{=} y}{X \xrightarrow{d,a} y'} \qquad \frac{y\ {}^{d}{\downarrow} \quad X \overset{\text{def}}{=} y}{X\ {}^{d}{\downarrow}}$$

Table 10. Operational rules for recursion ($\delta \in \mathcal{D}_\ominus$).

the stack is empty (denoted by the empty string) or has some $e \in \mathcal{D}$ on top, and the transition on the right shows we can execute $pop(d)$ for some $d \in \mathcal{D}$ whenever the stack has d on top, replacing it by the empty string. Consider now the following specification.

$$S' \overset{\text{def}}{=} \mathbf{1} + \sum_{d \in \mathcal{D}} push(d).T'_d \cdot S' \qquad T'_d \overset{\text{def}}{=} \mathbf{1} + pop(d).\mathbf{1} + \sum_{e \in \mathcal{D}} push(e).T'_e \cdot T'_d$$

This specification will not define the always accepting stack, as it is forgetful (can lose part of the stack contents) and moreover, its transition system is unboundedly branching, see [4]. The following specification does give the right result.

$$S'' \overset{\text{def}}{=} \mathbf{1} + \sum_{d \in \mathcal{D}} push(d).T''_d \,;\, S'' \qquad T''_d \overset{\text{def}}{=} \mathbf{1} + pop(d).\mathbf{1} + \sum_{e \in \mathcal{D}} push(e).T''_e \,;\, T''_d$$

This provides the motivation to use sequencing rather than sequential composition for stacks and pushdown automata.

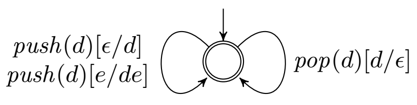

Fig. 4. Pushdown automaton of the always terminating stack ($d, e \in \mathcal{D}$).

In [4], we proved that a process (i.e., a bisimulation equivalence class of transition systems) is defined by a pushdown automaton if and only if it can be specified by a finite guarded recursive specification over TSP with sequencing, propositional signals and conditions. The following theorem shows that the simpler theory of sequential value passing presented here can be used instead.

Theorem 4. *A process is defined by a pushdown automaton if and only if it is defined by a finite guarded recursive specification over TSP;$_{sc}$.*

The proof of this theorem is not so much different from the proof in [4]. The simplification is in the setup by means of the operational semantics and the two-sorted syntax.

7 Conclusion

We investigated sequential value passing. By taking an unstructured finite data set for signals and conditions instead of terms over propositional logic, the treatment is simplified considerably. We presented this simplification both in a setting with sequential composition and in a setting with sequencing.

In [3], propositional signals and conditions were introduced, and studied in a setting with parallel composition, in order to study signal observation between processes in a parallel or distributed setting. We think that there, the combination of different signals is more important, and our simplification does not work so well. This needs to be investigated further.

We proved a Kleene theorem for processes: extending regular expressions with sequential value passing suffices to denote all finite automata.

In this paper we focussed on the correspondence results. We leave as future work to investigate the equational theory of the theory presented here. We see that signal emission and guarded command can lead to inconsistency, in the equational theory we need to introduce the inaccessible process $\perp$ that can never be reached, see [3, 8, 4].

This paper contributes to our ongoing project to integrate automata theory and process theory. As a result, we can present the foundations of computer science using a computer model with explicit interaction (as opposed to viewing automata just as language acceptors). Such a computer model relates more closely to the computers we see all around us.

Acknowledgements We are grateful to the two anonymous reviewers for their careful reviews; their comments and suggestions led to improvements in the presentation.

References

1. Aceto, L., Fokkink, W.J., Verhoef, C.: Structural operational semantics. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, pp. 197–292. North-Holland / Elsevier (2001). https://doi.org/10.1016/b978-044482830-9/50021-7
2. Baeten, J.C.M., Corradini, F., Grabmayer, C.A.: A characterization of regular expressions under bisimulation. J. ACM **54**(2), 6–28 (apr 2007). https://doi.org/10.1145/1219092.1219094
3. Baeten, J.C.M., Bergstra, J.A.: Process algebra with propositional signals. Theor. Comput. Sci. **177**(2), 381–405 (1997). https://doi.org/10.1016/S0304-3975(96)00253-8
4. Baeten, J.C.M., Carissimo, C., Luttik, B.: Pushdown automata and context-free grammars in bisimulation semantics. Logical Methods in Computer Science **19**, 15:1–15.32 (2023). https://doi.org/10.46298/LMCS-19(1:15)2023
5. Baeten, J.C.M., Luttik, B., Muller, T., van Tilburg, P.: Expressiveness modulo bisimilarity of regular expressions with parallel composition. Math. Struct. Comput. Sci. **26**(6), 933–968 (2016). https://doi.org/10.1017/S0960129514000309
6. Baeten, J.C.M., Luttik, B., Yang, F.: Sequential composition in the presence of intermediate termination (extended abstract). In: Peters, K., Tini, S. (eds.) Proceedings Combined 24th International Workshop on Expressiveness in Concurrency and 14th Workshop on Structural Operational Semantics and 14th Workshop on Structural Operational Semantics, EXPRESS/SOS 2017, Berlin, Germany, 4th September 2017. EPTCS, vol. 255, pp. 1–17 (2017). https://doi.org/10.4204/EPTCS.255.1, http://arxiv.org/abs/1709.00049

7. Baeten, J.C.M., Verhoef, C.: A congruence theorem for structured operational semantics with predicates. In: Best, E. (ed.) CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings. Lecture Notes in Computer Science, vol. 715, pp. 477–492. Springer (1993). https://doi.org/10.1007/3-540-57208-2_33
8. Baeten, J.C., Basten, T., Reniers, M.: Process algebra: equational theories of communicating processes, vol. 50. Cambridge university press (2010). https://doi.org/10.1017/CBO9781139195003
9. Belder, A., Luttik, B., Baeten, J.: Sequencing and intermediate acceptance: axiomatisation and decidability of bisimilarity. In: Roggenbach, M., Sokolova, A. (eds.) 8th Conference on Algebra and Coalgebra in Computer Science, CALCO 2019. Leibniz International Proceedings in Informatics, LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.CALCO.2019.11
10. Bergstra, J.A., Fokkink, W., Ponse, A.: Chapter 5 - process algebra with recursive operations. In: Bergstra, J., Ponse, A., Smolka, S. (eds.) Handbook of Process Algebra, pp. 333–389. Elsevier Science, Amsterdam (2001). https://doi.org/https://doi.org/10.1016/B978-044482830-9/50023-0
11. Bloom, B.: When is partial trace equivalence adequate? Formal Aspects Comput. **6**(3), 317–338 (1994). https://doi.org/10.1007/BF01215409
12. Bol, R.N., Groote, J.F.: The meaning of negative premises in transition system specifications. J. ACM **43**(5), 863–914 (1996). https://doi.org/10.1145/234752.234756
13. Garavel, H.: Nested-unit petri nets: A structural means to increase efficiency and scalability of verification on elementary nets. In: Devillers, R.R., Valmari, A. (eds.) Application and Theory of Petri Nets and Concurrency - 36th International Conference, PETRI NETS 2015, Brussels, Belgium, June 21-26, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9115, pp. 179–199. Springer (2015). https://doi.org/10.1007/978-3-319-19488-2_9
14. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. J. ACM **43**(3), 555–600 (1996). https://doi.org/10.1145/233551.233556
15. van Glabbeek, R.J.: The meaning of negative premises in transition system specifications II. J. Log. Algebr. Program. **60-61**, 229–258 (2004). https://doi.org/10.1016/j.jlap.2004.03.007
16. Grabmayer, C., Fokkink, W.J.: A complete proof system for 1-free regular expressions modulo bisimilarity. CoRR **abs/2004.12740** (2020), https://arxiv.org/abs/2004.12740
17. Groote, J.F.: Transition system specifications with negative premises. Theor. Comput. Sci. **118**(2), 263–299 (1993). https://doi.org/10.1016/0304-3975(93)90111-6
18. Groote, J.F., Ponse, A.: Process algebra with guards: Combining hoare logic with process algebra. Formal Aspects Comput. **6**(2), 115–164 (1994). https://doi.org/10.1007/BF01221097
19. Hennessy, M.: Value-passing in process algebras. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR '90 Theories of Concurrency: Unification and Extension. pp. 31–31. Springer Berlin Heidelberg, Berlin, Heidelberg (1990)
20. Kleene, S.C.: Representation of events in nerve nets and finite automata. Automata Studies pp. 3–41 (1956)
21. Kozen, D.: Kleene algebra with tests. ACM Trans. Program. Lang. Syst. **19**(3), 427–443 (1997). https://doi.org/10.1145/256167.256195
22. Luttik, B.: Divergence-preserving branching bisimilarity. In: Dardha, O., Rot, J. (eds.) Proceedings Combined 27th International Workshop on Expressiveness in Concurrency and 17th Workshop on Structural Operational Semantics, EXPRESS/SOS 2020, and 17th Workshop on Structural Operational Semantics, Online, 31 August 2020. EPTCS, vol. 322, pp. 3–11 (2020). https://doi.org/10.4204/EPTCS.322.2
23. Milner, R.: A complete inference system for a class of regular behaviours. Journal of Computer and System Sciences **28**(3), 439–466 (1984). https://doi.org/10.1016/0022-0000(84)90023-0

24. Verhoef, C.: A congruence theorem for structured operational semantics with predicates and negative premises. Nord. J. Comput. **2**(2), 274–302 (1995)
25. Visser, E., Benaissa, Z.: A core language for rewriting. In: Kirchner, C., Kirchner, H. (eds.) 1998 International Workshop on Rewriting Logic and its Applications, WRLA 1998, Abbaye des Prémontrés at Pont-à-Mousson, France, September 1998. Electronic Notes in Theoretical Computer Science, vol. 15, pp. 422–441. Elsevier (1998). https://doi.org/10.1016/S1571-0661(05)80027-1