Pushdown Automata and Context-Free Grammars in Bisimulation Semantics

Jos C. M. Baeten
CWI, Amsterdam, The Netherlands
University of Amsterdam, The Netherlands

Cesare Carissimo
University of Amsterdam, The Netherlands

Bas Luttik
Eindhoven University of Technology, The Netherlands

Abstract

The Turing machine models an old-fashioned computer, that does not interact with the user or with other computers, and only does batch processing. Therefore, we came up with a Reactive Turing Machine that does not have these shortcomings. In the Reactive Turing Machine, transitions have labels to give a notion of interactivity. In the resulting process graph, we use bisimilarity instead of language equivalence.

Subsequently, we considered other classical theorems and notions from automata theory and formal languages theory. In this paper, we consider the classical theorem of the correspondence between pushdown automata and context-free grammars. By changing the process operator of sequential composition to a sequencing operator with intermediate acceptance, we get a better correspondence in our setting. We find that the missing ingredient to recover the full correspondence is the addition of a notion of state awareness.

2012 ACM Subject Classification Theory of computation → Interactive computation; Theory of computation → Turing machines; Theory of computation → Grammars and context-free languages; Theory of computation → Process calculi

Keywords and phrases pushdown automaton, context-free grammar, bisimilarity, intermediate acceptance, state awareness

Digital Object Identifier 10.4230/LIPIcs.CALCO.2021.8

1 Introduction

A basic ingredient of any undergraduate curriculum in computer science is a course on automata theory and formal languages, as this gives students insight in the essence of a computer, and tells them what a computer can and cannot do. Usually, such a course contains the treatment of the Turing machine as an abstract model of a computer. However, the Turing machine is a very old-fashioned computer: it is deaf, dumb and blind, and all input from the user has to be put on the tape before the start. Computers behaved like this until the advent of the terminal in the mid 1970s. This is far removed from computers the students find all around them, which interact continuously with people, other computers and the internet. It is hard to imagine a self-driving car driven by a Turing machine that is deaf, dumb and blind, where all user input must be on the tape at the start of the trip.

In order to make the Turing machine more interactive, many authors have enhanced it with extra features, see e.g. [10, 16]. But an extra feature, we believe, is not the way to go. Interaction is an essential ingredient, such as it has been treated in many forms of concurrency theory. We seek a full integration of automata theory and concurrency theory, and proposed the Reactive Turing Machine in [4]. In the Reactive Turing Machine, transitions have labels to give a notion of interactivity. In the resulting process graphs, we use bisimilarity instead of language equivalence.

Subsequently, we considered other classical theorems and notions from automata theory and formal languages theory [2]. We find richer results and a finer theory. In this paper, we consider the classical theorem of the correspondence between pushdown automata and context-free grammars. Before [3], we did not get a good correspondence in the process setting. By changing the process operator of sequential composition to a sequencing operator with intermediate acceptance, we get a better correspondence in our setting [5, 7, 8]. We find that the missing ingredient to recover the full correspondence is the addition of a notion of state awareness, by means of a signal that can be passed along a sequencing operator.

2 Preliminaries

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

▶ **Definition 1.** *A* labelled transition system *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*,* $\tau \notin \mathcal{A}$*;*
3. $\rightarrow \subseteq \mathcal{S} \times \mathcal{A} \cup \{\tau\} \times \mathcal{S}$ *is an* $\mathcal{A} \cup \{\tau\}$*-labelled* transition relation*; and*
4. $\downarrow \subseteq \mathcal{S}$ *is the set of final or accepting states.*

A process graph *is a labelled transition system with a special designated* root 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 labelled transition system, and* $\uparrow \in \mathcal{S}$*. We write* $s \xrightarrow{a} s'$ *for* $(s, a, s') \in \rightarrow$ *and* $s\downarrow$ *for* $s \in \downarrow$*.*

By considering language equivalence classes of process graphs, 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 labelled transition system. 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* $s \mathrel{R} t$ *and for all* $a \in \mathcal{A} \cup \{\tau\}$*:*

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* $s' \mathrel{R} t'$*; and*
2. *if* $s\downarrow$*, then* $t\downarrow$*.*

The results of this paper do not rely on abstraction from internal computations, so we can use the *strong* version of bisimilarity defined above, which does not give special treatment to τ-labelled transitions. But in general 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 [12] for an overview of recent results).

A *process* is a bisimulation equivalence class of process graphs.

3 Pushdown Automata

We consider an abstract model of a computer with a memory in the form of a *stack*: this stack can be accessed only at the top: something can be added on top of the stack (push), or something can be removed from the top of the stack (pop).

▶ **Definition 3** (pushdown automaton). *A* pushdown automaton M *is a sextuple* $(\mathcal{S}, \mathcal{A}, \mathcal{D}, \rightarrow, \uparrow, \downarrow)$ *where:*

1. $\mathcal{S}$ *is a finite set of states,*
2. $\mathcal{A}$ *is a finite input alphabet,* $\tau \notin \mathcal{A}$ *is the unobservable step,*
3. $\mathcal{D}$ *is a finite data alphabet,*

4. $\rightarrow \subseteq \mathcal{S} \times (\mathcal{A} \cup \{\tau\}) \times (\mathcal{D} \cup \{\epsilon\}) \times \mathcal{D}^* \times \mathcal{S}$ *is a finite set of* transitions *or* steps*,*
5. $\uparrow \in \mathcal{S}$ *is the initial state,*
6. $\downarrow \subseteq \mathcal{S}$ *is the set of final or accepting states.*

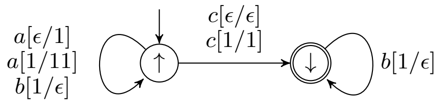

Figure 1 An example pushdown automaton.

If $(s, a, d, x, t) \in \rightarrow$ with $d \in \mathcal{D}$, we write $s \xrightarrow{a[d/x]} t$, and this means that the machine, when it is in state s and d is the top element of the stack, can consume input symbol a, replace d by the string x and thereby move to state t. Likewise, writing $s \xrightarrow{a[\epsilon/x]} t$ means that the machine, when it is in state s and the stack is empty, can consume input symbol a, put the string x on the stack and thereby move to state t. In steps $s \xrightarrow{\tau[d/x]} t$ and $s \xrightarrow{\tau[\epsilon/x]} t$, no input symbol is consumed, only the stack is modified.

For example, consider the pushdown automaton depicted in Figure 1. It represents the process that can start to read an a or a c, and after it has read at least one a, can also read b's. Upon acceptance, it will have read just one c, up to as many b's as it has read a's, and no a's after reading the c.

We do not consider the language of a pushdown automaton, but rather consider the process, i.e., the bisimulation equivalence class of the process graph of a pushdown automaton. A state of this process graph is a pair (s, x), where $s \in \mathcal{S}$ is the current state and $x \in \mathcal{D}^*$ is the current contents of the stack (the left-most element of x being the top of the stack). In the initial state, the stack is empty. In a final state, acceptance can take place irrespective of the contents of the stack. The transitions in the process graph are labeled by the inputs of the pushdown automaton or τ.

▶ **Definition 4.** *Let $M = (\mathcal{S}, \mathcal{A}, \mathcal{D}, \rightarrow, \uparrow, \downarrow)$ be a pushdown automaton. The* process graph *$\mathcal{P}(M) = (\mathcal{S}_{\mathcal{P}(M)}, \mathcal{A}, \rightarrow_{\mathcal{P}(M)}, \uparrow_{\mathcal{P}(M)}, \downarrow_{\mathcal{P}(M)})$ associated with M is defined as follows:*
1. $\mathcal{S}_{\mathcal{P}(M)} = \{(s, x) \mid s \in \mathcal{S} \;\&\; x \in \mathcal{D}^*\}$*;*
2. $\rightarrow_{\mathcal{P}(M)} \subseteq \mathcal{S}_{\mathcal{P}(M)} \times \mathcal{A} \cup \{\tau\} \times \mathcal{S}_{\mathcal{P}(M)}$ *is the least relation such that for all $s, s' \in \mathcal{S}$, $a \in \mathcal{A} \cup \{\tau\}$, $d \in \mathcal{D}$ and $x, x' \in \mathcal{D}^*$ we have*

$$(s, dx) \xrightarrow{a}_{\mathcal{P}(M)} (s', x'x) \text{ if, and only if, } s \xrightarrow{a[d/x']} s' \;;$$

$$(s, \epsilon) \xrightarrow{a}_{\mathcal{P}(M)} (s', x) \text{ if, and only if, } s \xrightarrow{a[\epsilon/x]} s' \;;$$

3. $\uparrow_{\mathcal{P}(M)} = (\uparrow, \epsilon)$*;*
4. $\downarrow_{\mathcal{P}(M)} = \{(s, x) \mid s \in \downarrow \;\&\; x \in \mathcal{D}^*\}$.

To distinguish, in the definition above, the set of states, the transition relation, the initial state and the set of accepting states of the pushdown automaton from similar components of the associated process graph, we have attached a subscript $\mathcal{P}(M)$ to the latter. In the remainder of this paper, we will suppress the subscript whenever it is already clear from the context whether a component of the pushdown automaton or its associated transition system is meant.

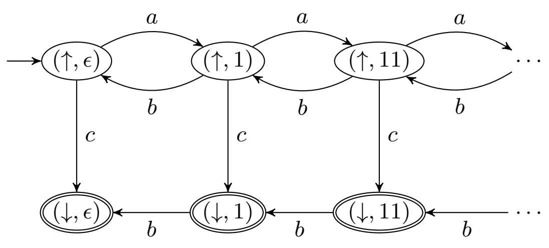

Figure 2 The process graph associated with the pushdown automaton in Figure 1.

Figure 2 depicts the process graph associated with the pushdown automaton depicted in Figure 1.

By adding additional states and τ-transitions, it is enough to consider only *push* and *pop* transitions: a push transition is of the form $s \xrightarrow{a[d/ed]} t$ or $s \xrightarrow{a[\epsilon/d]} t$, where one data element is added on top of the stack, and a pop transition is of the form $s \xrightarrow{a[d/\epsilon]} t$, where the top of the stack is removed ($a \in (\mathcal{A} \cup \{\tau\})$), see [2].

4 Sequential Processes

In our setting, a context-free grammar is denoted by a finite guarded recursive specification over the process theory of sequential expressions.

In this section we present the Theory of Sequential Processes adopting the revised operational semantics for sequential composition proposed in [5]. Sequential composition with the operational semantics of [6] is denoted by $\cdot$, and we call the operator with the revised operational semantics *sequencing* and denote it by ;.

Let $\mathcal{A}$ be a set of *actions* and $\tau \notin \mathcal{A}$ *the silent action*, symbols denoting atomic events, and let $\mathcal{P}$ be a finite set of *process identifiers*. The sets $\mathcal{A}$ and $\mathcal{P}$ serve as parameters of the process theory that we shall introduce below. The set of *sequential process expressions* is generated by the following grammar ($a \in \mathcal{A} \cup \{\tau\}$, $X \in \mathcal{P}$):

$$p ::= \mathbf{0} \mid \mathbf{1} \mid a.p \mid p + p \mid p\,;p \mid X \ .$$

The constants $\mathbf{0}$ and $\mathbf{1}$ respectively denote the *deadlocked* (i.e., inactive but not accepting) process and the *accepting* process. For each $a \in \mathcal{A} \cup \{\tau\}$ there is a unary action prefix operator $a.\_$. The binary operators $+$ and ; denote alternative composition and sequencing, respectively. We adopt the convention that $a.\_$ binds strongest and $+$ binds weakest. For a (possibly empty) sequence $p_1, \ldots, p_n$ we inductively define $\sum_{i=1}^{n} p_i = \mathbf{0}$ if $n = 0$ and $\sum_{i=1}^{n} p_i = (\sum_{i=1}^{n-1} p_i) + p_n$ if $n > 0$. The symbol ; is often omitted when writing process expressions. In particular, if $\alpha \in \mathcal{P}^*$, say $\alpha = X_1 \cdots X_n$, then α denotes the process expression inductively defined by $\alpha = \mathbf{1}$ if $n = 0$ and $\alpha = (X_1 \cdots X_{n-1})\,; X_n$ if $n > 0$.

A recursive specification over sequential process expressions is a mapping Δ from $\mathcal{P}$ to the set of sequential process expressions. The idea is that the process expression p 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* $X \overset{\text{def}}{=} p$, exactly one for every $X \in \mathcal{P}$. We shall, throughout the paper, presuppose a recursive specification Δ defining the process identifiers in $\mathcal{P}$, and we shall usually simply write $X \overset{\text{def}}{=} p$ for $\Delta(X) = p$. Note that, by our assumption that $\mathcal{P}$ is finite, Δ is finite too.

$$\frac{}{a.p \xrightarrow{a} p} \qquad \frac{p \xrightarrow{a} p'}{p+q \xrightarrow{a} p'} \qquad \frac{q \xrightarrow{a} q'}{p+q \xrightarrow{a} q'}$$

$$\frac{}{\mathbf{1}\downarrow} \qquad \frac{p\downarrow}{(p+q)\downarrow} \qquad \frac{q\downarrow}{(p+q)\downarrow}$$

$$\frac{p\downarrow \quad q\downarrow}{p\,;q\downarrow} \qquad \frac{p \xrightarrow{a} p'}{p\,;q \xrightarrow{a} p'\,;q} \qquad \frac{p\downarrow \quad p \nrightarrow \quad q \xrightarrow{a} q'}{p\,;q \xrightarrow{a} q'}$$

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

Figure 3 Operational semantics for sequential process expressions.

We associate behaviour with process expressions by defining, on the set of process expressions, a unary acceptance predicate $\downarrow$ (written postfix) and, for every $a \in \mathcal{A} \cup \{\tau\}$, a binary transition relation $\xrightarrow{a}$ (written infix), by means of the transition system specification presented in Fig. 3. We write $p \overset{a}{\nrightarrow}$ for "there does not exist p' such that $p \xrightarrow{a} p'$" and $p \nrightarrow$ for "$p \overset{a}{\nrightarrow}$ for all $a \in \mathcal{A} \cup \{\tau\}$".

For $w \in \mathcal{A}^*$ we define $p \overset{w}{\twoheadrightarrow} p'$ inductively, for all process expressions p, p', p'';

- $p \overset{\epsilon}{\twoheadrightarrow} p$;
- if $p \xrightarrow{a} p'$ and $p' \overset{w}{\twoheadrightarrow} p''$, then $p \overset{aw}{\twoheadrightarrow} p''$ $(a \in \mathcal{A})$;
- if $p \xrightarrow{\tau} p'$ and $p' \overset{w}{\twoheadrightarrow} p''$, then $p \overset{w}{\twoheadrightarrow} p''$.

We see that τ-steps do not contribute to the string w. We write $p \longrightarrow p'$ for there exists $a \in \mathcal{A} \cup \{\tau\}$ such that $p \xrightarrow{a} p'$. Similarly, we write $p \twoheadrightarrow p'$ for there exists $w \in \mathcal{A}^*$ such that $p \overset{w}{\twoheadrightarrow} p'$ and say that p' is *reachable* from p.

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 [11, 9, 13]. Indeed, in [5] it was already pointed out that the transition system specification in Fig. 3 gives rise to such anomalies, e.g., if Δ includes for X the defining equation $X \overset{\text{def}}{=} X\,; a.\mathbf{1} + \mathbf{1}$. For then, if $X \nrightarrow$, 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.

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. Note that we allow τ as a guard. This is possible since we use strong bisimulation, not branching bisimulation. If Δ is guarded, then it is straightforward to prove that the mapping S from process expressions to natural numbers inductively defined by $S(\mathbf{1}) = S(\mathbf{0}) = S(a.p) = 0$, $S(p_1 + p_2) = S(p_1\,; p_2) = S(p_1) + S(p_2) + 1$, and $S(X) = S(p)$ if $(X \overset{\text{def}}{=} p) \in \Delta$ gives rise to a so-called *stratification* S' from transitions to natural numbers defined by $S'(p \xrightarrow{a} p') = S(p)$ for all $a \in \mathcal{A} \cup \{\tau\}$ and process expressions p and p'. In [11] it is proved that whenever such a stratification exists, then the transition system specification defines a unique transition relation that agrees with provability in the transition system specification.

The operational rules in Fig. 3 deviate from the operational rules for the Theory of Sequential Processes discussed in [6] in only two ways: to get that set of rules, the symbol ; should be replaced by $\cdot$, and the negative premise $p \nrightarrow$ should be removed from the third sequencing rule. The replacement of ; by $\cdot$ is, of course, insignificant; the removal of the negative premise $p \nrightarrow$, however, does have a significant effect on the semantics of sequencing. The negative premise ensures that a sequencing can only proceed to execute its second argument when its first argument not only satisfies the acceptance predicate, but also cannot perform any further activity. The semantic difference between ; and $\cdot$ is illustrated in the following example.

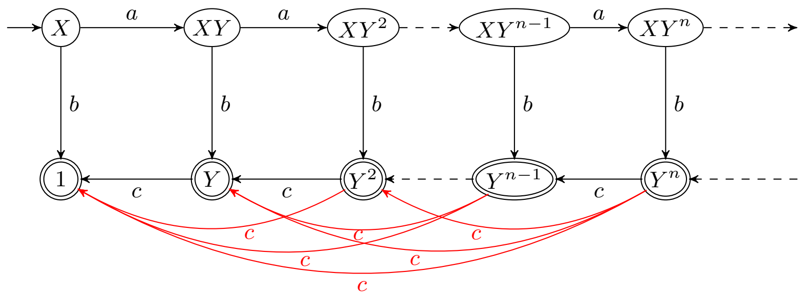

Figure 4 The difference between ; and $\cdot$.

▶ **Example 5.** Consider the recursive specification

$$X \overset{\text{def}}{=} a.(XY) + b.\mathbf{1} \qquad Y \overset{\text{def}}{=} c.\mathbf{1} + \mathbf{1} \ .$$

We have deliberately omitted the occurrence of the sequencing operator between X and Y from the right-hand side of the defining equation for X (as is, actually, standard practice). Depending on whether we interpret the sequencing of X and Y using the semantics for $\cdot$ or for ;, we obtain the process graph shown in Fig. 4 with or without the red c-transitions. Note that, under the $\cdot$-interpretation, the phenomenon of *transparency* plays a role: from Y^n we have c-transitions to every Y^k with $k < n$, by executing the c-transition of the kth occurrence of Y, thus skipping the first $k-1$ occurrences of Y. This behaviour is prohibited by the negative premise in the rule for ;, for, since $Y \xrightarrow{c} \mathbf{1}$, none of the occurrences of Y can be skipped.

When a term p satisfies both $p\downarrow$ and $p \longrightarrow$ we say p has *intermediate acceptance*. We will need to take special care of such terms in the sequel.

We proceed to define when two closed terms are behaviourally equivalent.

▶ **Definition 6.** *A binary relation R on the set of sequential process expressions is a* bisimulation *iff R is symmetric and for all closed terms p and q such that if $(p, q) \in R$:*

1. *If $p \xrightarrow{a} p'$, then there exists a term q', such that $q \xrightarrow{a} q'$, and $(p', q') \in R$.*
2. *If $p\downarrow$, then $q\downarrow$.*

The terms p and q are bisimilar (notation: $p \leftrightarroweq q$) iff there exists a bisimulation R such that $(p, q) \in R$.

The operational rules presented in Fig 3 are in the so-called *panth format* from which it immediately follows that bisimilarity is a congruence [15].

▶ **Proposition 7.** *The relation $\leftrightarroweq$ is a congruence on sequential process expressions.*

5 The correspondence

The classical theorem states that a language can be defined by a push-down automaton just in case it can be defined by a context-free grammar. In our setting, we do have that the process of a given guarded sequential specification (i.e., the equivalence class of process graphs bisimilar to the process graph associated with the sequential specification) coincides with the process of some push-down automaton (i.e., the equivalence class of process graphs bisimilar to the process graph associated with the push-down automaton), but not the other way around: there is a push-down automaton of which the process is different from the process of any guarded sequential specification. In this section, we will prove these facts, in the next section, we investigate what is needed in addition to recover the full correspondence.

First of all, we look at the failing direction. It can fail if the push-down automaton has at least two states. For one state, it does work.

▶ **Theorem 8.** *For every one-state pushdown automaton there is a guarded sequential specification of which the process coincides with the process of the automaton.*

Proof. Let $M = (\{\uparrow\}, \mathcal{A}, \mathcal{D}, \rightarrow, \uparrow, \downarrow)$ be a pushdown automaton. We can assume $\rightarrow$ only has push and pop transitions. If there is no transition $\uparrow \xrightarrow{a[\epsilon/d]} \uparrow$, then we can take either $X = \mathbf{1}$ or $X = \mathbf{0}$ as the resulting specification (in case $\downarrow = \{\uparrow\}$ resp. $\downarrow = \emptyset$). Otherwise, add a summand $a.X_d \mathbin{;} X$ for each such transition to the equation of the initial identifier X. Next, the equation for the added identifier X_d has a summand $a.\mathbf{1}$ for each transition $\uparrow \xrightarrow{a[d/\epsilon]} \uparrow$ and a summand $a.X_e \mathbin{;} X_d$ for each transition $\uparrow \xrightarrow{a[d/ed]} \uparrow$, apart from a summand $\mathbf{1}$ or $\mathbf{0}$, depending on whether $\uparrow \in \downarrow$ or not. ◀

▶ **Example 9.** The stack that is accepting in every state has a pushdown automaton with state $\uparrow$, data some finite set D, actions $\{push_d, pop_d \mid d \in D\}$, $\downarrow = \{\uparrow\}$ and transitions $\uparrow \xrightarrow{push_d[\epsilon/d]} \uparrow$ and $\uparrow \xrightarrow{pop_d[d/\epsilon]} \uparrow$ for each $d \in D$ and transitions $\uparrow \xrightarrow{push_e[d/ed]} \uparrow$ for each $d, e \in D$.

The recursive specification becomes

$$X \stackrel{\text{def}}{=} \mathbf{1} + \sum_{d \in D} push_d.X_d \mathbin{;} X \qquad X_d \stackrel{\text{def}}{=} \mathbf{1} + pop_d.\mathbf{1} + \sum_{e \in D} push_e.X_e \mathbin{;} X_d \ \ (d \in D)$$

Note that if we use the sequential composition operator $\cdot$ of [6] instead of the present sequencing operator ;, then Theorem 8 fails because of the transparency illustrated in Figure 4. With the sequential composition operator, we cannot find a recursive specification of the stack accepting in every state of Example 9, because of the same phenomenon.

In order to prove that there is a push-down automaton, of which the process cannot be specified by a guarded sequential specification, it is convenient to present a guarded sequential specification in a normal form, the so-called Greibach normal form, see [8].

▶ **Definition 10.** *A guarded sequential specification is in Greibach normal form, GNF, if every right-hand side of every equation has one of the following two forms:*

- $\sum_{i=1}^{n} a_i.\alpha_i$ *for actions* $a_i \in \mathcal{A} \cup \{\tau\}$ *and identifiers* $\alpha_i \in \mathcal{P}^*$, $n \geq 0$.
- $\mathbf{1} + \sum_{i=1}^{n} a_i.\alpha_i$ *for actions* $a_i \in \mathcal{A} \cup \{\tau\}$ *and identifiers* $\alpha_i \in \mathcal{P}^*$, $n \geq 0$.

Recall that the empty summation equals $\mathbf{0}$ *and the empty sequence is* $\mathbf{1}$.

By adding a finite number of process identifiers, every guarded sequential specification can be brought into Greibach normal form (i.e. the behaviour associated with a process identifier by the original specification is strongly bisimilar to the behaviour associated with it by the transformed specification, see [8]).

▶ **Theorem 11.** *There is a pushdown automaton with two states, such that there is no guarded sequential specification with the same process.*

Proof. Consider the example pushdown automaton in Figure 1. Suppose there is a finite guarded sequential specification with the same process, depicted by the representative in Figure 2. Without loss of generality we can assume that this specification is in Greibach Normal Form (see [8]). As a consequence, each state of the process graph generated by the automaton corresponds to a sequence of identifiers of the specification (as defined earlier). Take k a natural number that is larger than the number of process identifiers of the specification, take $i \leq k$ and consider the state $(\uparrow, 1^i)$ reached after executing i a-steps. From this state, consider any sequence of steps $\xrightarrow{w}\!\!\!\!\rightarrow$ where $a \notin w$. Thus, w contains at most one c and at most i b's.

In the process graph generated by the recursive specification, this same sequence of steps $\xrightarrow{w}\!\!\!\!\rightarrow$ is possible from the sequence of identifiers α_i bisimilar to state $(\uparrow, 1^i)$. Let X_i be the first element of α_i. From X_i, we can also execute at most one c-step and i b-steps, without executing an a-step.

Since k is larger than the number of process identifiers of the specification, there must be a repetition in the identifiers X_i $(i \leq k)$. Thus, there are numbers n, m, $n < m \leq k$, with $X_n = X_m$. The process identifier X_m can execute at most one c and $n < m$ b's without executing an a. But $\alpha_m \xrightarrow{b^m}\!\!\!\!\rightarrow$, so the additional b-steps must come from the second and following identifiers of the sequence. As the second identifier is reached by just executing b's, this is a state reached by just executing a's and b's, so it must allow an initial c-step. Now we can consider $\alpha_m \xrightarrow{cb^m}\!\!\!\!\rightarrow$. This sequence of steps must also reach the second identifier, but then, a second c can be executed, which is a contradiction.

Thus, our assumption was wrong, and the theorem is proved. ◀

We see that the contradiction is reached, because when we reach the second identifier in the sequence α_i, we do not know whether we are in a state relating to the initial state or the final state of the pushdown automaton. Going from the first identifier to the second identifier by means of the sequencing operator, no extra information can be passed along. We will describe a mechanism that allows the passing of extra information along the sequencing operator.

Theorem 11 holds for the sequencing operator we introduced, but it holds in the same way for the sequential composition operator of [6]. No intermediate acceptance is involved in the proof.

In the other direction, we can find a pushdown automaton with the same process as a given guarded sequential specification. The proof we give is more complicated than the classical proof, where it is only needed to find a pushdown automaton with the same language. There, we can handle all acceptance in a single state, which is only entered when the stack memory is empty. In bisimulation semantics, it is not true that every pushdown automaton is equivalent to one with such a single accepting state. We carefully need to consider every instance of intermediate acceptance. Consider the sequencing $(a.\mathbf{1} + \mathbf{1})\,;b.\mathbf{1}$. The first term in this sequencing shows intermediate acceptance, the second does not. As $(a.\mathbf{1}+\mathbf{1})\,;b.\mathbf{1} \leftrightarrows a.\mathbf{1}\,;b.\mathbf{1}$, the intermediate acceptance in the first term is redundant, and can be removed. We have to restrict the notion of Greibach normal form, in order to remove all redundant intermediate acceptance.

▶ **Definition 12.** *A guarded sequential specification is in Acceptance Irredundant Greibach normal form, AIGNF, if it is in Greibach normal form, and moreover, every state of the resulting process graph is given by a sequence of identifiers of the specification of the form $\alpha\beta$ $(\alpha, \beta \in \mathcal{P}^*)$, where all identifiers in α do not have intermediate acceptance, and all identifiers in β do have intermediate acceptance (or equal $\mathbf{1}$). The sequence α or β may be empty.*

It is proven in [8] that every guarded sequential specification can be transformed to one in Acceptance Irredundant Greibach normal form, so that all redundant intermediate acceptance is removed.

▶ **Theorem 13.** *For every guarded sequential specification there is a pushdown automaton with a bisimilar process graph, with at most two states.*

Proof. Let Δ be a guarded sequential specification over $\mathcal{P}$. Without loss of generality, we can assume Δ is in Acceptance Irredundant Greibach Normal Form [8]. Every state of the specification is given by a sequence of identifiers that is acceptance irredundant. The corresponding pushdown automaton has two states $\{n, t\}$. The initial state is n iff the initial identifier $S \not\downarrow$ and t iff the initial identifier $S \downarrow$ (as defined by the operational semantics), and the final state is t.

- For each summand $a.\alpha$ of an identifier X with $X \downarrow$ and the first identifier of α an identifier with $\downarrow$, add a step $t \xrightarrow{a[X/\alpha]} t$. Moreover, in case X is initial, a step $t \xrightarrow{a[\epsilon/\alpha]} t$;
- For each summand $a.\alpha$ of an identifier X with $X \downarrow$ and the first identifier of α an identifier with $\not\downarrow$, add a step $t \xrightarrow{a[X/\alpha]} n$. Moreover, in case X is initial, a step $t \xrightarrow{a[\epsilon/\alpha]} n$;
- For each summand $a.\alpha$ of an identifier X with $X \not\downarrow$ and the first identifier of α an identifier with $\downarrow$, add a step $n \xrightarrow{a[X/\alpha]} t$. Moreover, in case X is initial, a step $n \xrightarrow{a[\epsilon/\alpha]} t$;
- For each summand $a.\alpha$ of an identifier X with $X \not\downarrow$ and the first identifier of α an identifier with $\downarrow$, add a step $n \xrightarrow{a[X/\alpha]} n$. Moreover, in case X is initial, a step $n \xrightarrow{a[\epsilon/\alpha]} n$.

Now it is not difficult to check that the process of this pushdown automaton is the same as the process of the given guarded sequential specification. ◀

In the case of the sequential composition operator of [6], Theorem 13 also holds, the proof is simpler because of the more straightforward handling of intermediate acceptance, but it is not as simple as in the classical case.

6 Signals and conditions

In order to obtain the missing correspondence, we need a mechanism to pass state information along a sequencing operator. This mechanism is provided by propositional signals together with a conditional statement as given in [1], see also [6].

▶ **Definition 14.** *First of all, we add the data domain $\mathcal{B}$ of the* Booleans*, with two constants true and false, operators $\neg$ for negation and $\vee, \wedge$ for or, and. We use a set $P_1, \ldots, P_n$ as propositional variables. In this data type, we can make propositional formulas.*

Next, we can define a conditional statement or *guarded command*. Given a propositional formula ϕ, we write $\phi :\to x$, with the intuitive meaning '*if ϕ then x*'. In order to give an operational semantics, it is important to note that it is needed to know the values of the propositional variables in order to decide on possible transitions. Moreover, values of propositional variables can change during the execution of a process expression. Thus, we need a *valuation* that in each state of a process graph assigns *true* or *false* to each propositional variable. Upon executing an action a in a state with valuation v, a state with a possibly different valuation v' results. The resulting valuation v' is called the *effect* of the execution of a in a state with valuation v.

We present operational rules for guarded command in Fig. 5; it presupposes a function *effect* that associates with every action a and every valuation v its effect. We define when a term in a certain valuation can take a step or be in a final state.

$$\frac{}{\langle \mathbf{1}, v\rangle \downarrow} \qquad \frac{v' = \mathit{effect}(a, v)}{\langle a.x, v\rangle \xrightarrow{a} \langle x, v'\rangle}$$

$$\frac{\langle x, v\rangle \xrightarrow{a} \langle x', v'\rangle}{\langle x + y, v\rangle \xrightarrow{a} \langle x', v'\rangle \quad \langle y + x, v\rangle \xrightarrow{a} \langle x', v'\rangle} \qquad \frac{\langle x, v\rangle \downarrow}{\langle x + y, v\rangle \downarrow \quad \langle y + x, v\rangle \downarrow}$$

$$\frac{\langle x, v\rangle \xrightarrow{a} \langle x', v'\rangle \quad v(\phi) = \mathit{true}}{\langle \phi :\rightarrow x, v\rangle \xrightarrow{a} \langle x', v'\rangle} \qquad \frac{\langle x, v\rangle \downarrow \quad v(\phi) = \mathit{true}}{\langle \phi :\rightarrow x, v\rangle \downarrow}$$

$$\frac{\langle x, v\rangle \downarrow \quad \langle y, v\rangle \downarrow}{\langle x \mathbin{;} y, v\rangle \downarrow} \qquad \frac{\langle x, v\rangle \xrightarrow{a} \langle x', v'\rangle}{\langle x \mathbin{;} y, v\rangle \xrightarrow{a} \langle x' \mathbin{;} y, v'\rangle}$$

$$\frac{\langle x, v\rangle \downarrow \quad \langle x, v\rangle \not\rightarrow \quad \langle y, v\rangle \xrightarrow{a} \langle y', v'\rangle}{\langle x \mathbin{;} y, v\rangle \xrightarrow{a} \langle y', v'\rangle}$$

$$\frac{\langle x, v\rangle \xrightarrow{a} \langle x', v'\rangle \quad X \stackrel{\text{def}}{=} x}{\langle X, v\rangle \xrightarrow{a} \langle x', v'\rangle} \qquad \frac{\langle x, v\rangle\downarrow \quad X \stackrel{\text{def}}{=} x}{\langle X, v\rangle\downarrow}$$

Figure 5 Operational rules for guarded command ($a \in \mathcal{A} \cup \{\tau\}$).

On the basis of these rules, we can define a notion of bisimulation. We use *stateless* bisimulation, which means that two process graphs are bisimilar iff there is a bisimulation relation that relates two process expressions iff they are related under every possible valuation. The stateless bisimulation also allows non-determinism, as the effect of the execution of an action will allow every possible resulting sequel. See the example further on, after we also introduce the root signal operator. Notice that $\langle \mathit{false} :\rightarrow \mathbf{1}, v\rangle$ is not final for *any* valuation v.

Next, we introduce an operator that allows the observation of aspects of the current state of a process graph. The central assumption is that the visible part of the state of a process graph is a proposition, an expression over the booleans. We introduce the *root-signal emission operator* $\wedge\!\!\!\blacktriangle$. A term $\phi \wedge\!\!\!\blacktriangle x$ represents the process x that shows the signal ϕ in its initial state. In order to define this operator by operational rules, we need to define an additional predicate on terms, namely *consistency*. $\mathit{Cons}(\langle x, v\rangle)$ will not hold when the valuation of the root signal of x is false. A step $\xrightarrow{a}$ can only be between consistent states, and a state can only be final when it is consistent. Thus, the term $a.(\mathit{false} \wedge\!\!\!\blacktriangle x)$ can under no valuation execute action a.

The operational rules are defined in Fig. 6. First, we define the consistency predicate. Next, we find that the second, third, fourth and eighth rule of Table 5 require an extra condition. The other rules of Table 5 can remain unchanged. Finally, we give the operational rules of the root signal emission operator. To emphasise again the difference between guarded commands and root signal emission, term $\mathit{false} :\rightarrow (\mathbf{1} + a.\mathbf{1})$ is consistent and has, under any valuation, the same process graph as $\mathbf{0}$, whereas $\mathit{false} \wedge\!\!\!\blacktriangle (\mathbf{1} + a.\mathbf{1})$ is inconsistent, this state cannot be reached.

We again have a stateless bisimulation, where two terms are related iff any valuation that makes the root signal of one term *true* also makes the root signal of the other term *true* and for each such valuation, the process graphs of the terms are bisimilar.

The information given by the truth of the signals allows to determine the truth of some of the guarded commands. In this way, we can give a semantics for terms and specifications as regular process graphs, leaving out the valuations.

$$\frac{}{\mathit{Cons}(\langle \mathbf{0}, v\rangle)} \qquad \frac{}{\mathit{Cons}(\langle \mathbf{1}, v\rangle)} \qquad \frac{}{\mathit{Cons}(\langle a.x, v\rangle)}$$

$$\frac{\mathit{Cons}(\langle x, v\rangle)\ \mathit{Cons}(\langle y, v\rangle)}{\mathit{Cons}(\langle x + y, v\rangle)} \qquad \frac{\mathit{Cons}(\langle x, v\rangle)}{\mathit{Cons}(\langle \phi :\rightarrow x, v\rangle)} \qquad \frac{\mathit{Cons}(\langle x, v\rangle) \quad v(\phi) = \mathit{true}}{\mathit{Cons}(\langle \phi {\wedge}\!\blacktriangle x, v\rangle)}$$

$$\frac{\mathit{Cons}(\langle x, v\rangle) \quad \langle x, v\rangle \not\downarrow}{\mathit{Cons}(\langle x \,;\, y, v\rangle)} \qquad \frac{\langle x, v\rangle \downarrow \quad \mathit{Cons}(\langle y, v\rangle)}{\mathit{Cons}(\langle x \,;\, y, v\rangle)}$$

$$\frac{\mathit{Cons}(\langle x, v\rangle) \quad X \overset{\text{def}}{=} x}{\mathit{Cons}(\langle X, v\rangle)} \qquad \frac{\mathit{Cons}(\langle x, v'\rangle) \quad v' = \mathit{effect}(a, v)}{\langle a.x, v\rangle \xrightarrow{a} \langle x, v'\rangle}$$

$$\frac{\langle x, v\rangle \xrightarrow{a} \langle x', v'\rangle \quad \mathit{Cons}(\langle y, v\rangle)}{\langle x + y, v\rangle \xrightarrow{a} \langle x', v'\rangle \ \ \langle y + x, v\rangle \xrightarrow{a} \langle x', v'\rangle} \qquad \frac{\langle x, v\rangle \downarrow \quad \mathit{Cons}(\langle y, v\rangle)}{\langle x + y, v\rangle \downarrow \ \ \langle y + x, v\rangle \downarrow}$$

$$\frac{\langle x, v\rangle \xrightarrow{a} \langle x', v'\rangle \quad \mathit{Cons}(\langle x' \,;\, y, v'\rangle)}{\langle x \,;\, y, v\rangle \xrightarrow{a} \langle x' \,;\, y, v'\rangle}$$

$$\frac{\langle x, v\rangle \xrightarrow{a} \langle x', v'\rangle \quad v(\phi) = \mathit{true}}{\langle \phi {\wedge}\!\blacktriangle x, v\rangle \xrightarrow{a} \langle x', v'\rangle} \qquad \frac{\langle x, v\rangle \downarrow \quad v(\phi) = \mathit{true}}{\langle \phi {\wedge}\!\blacktriangle x, v\rangle \downarrow}$$

Figure 6 Operational rules for root-signal emission ($a \in \mathcal{A} \cup \{\tau\}$).

▶ **Definition 15.** *Let t, s be sequential terms with signals and conditions, let $a \in \mathcal{A} \cup \{\tau\}$ and suppose the root signal of t is not* false.

- $t \xrightarrow{a} s$ *iff for all valuations v such that $\mathit{Cons}(\langle t, v\rangle)$ we have $\langle t, v\rangle \xrightarrow{a} \langle s, \mathit{effect}(a, v)\rangle$,*
- $t \downarrow$ *iff for all valuations v such that $\mathit{Cons}(\langle t, v\rangle)$ we have $\langle t, v\rangle \downarrow$.*

The above definition makes all undetermined guarded commands *false*, as is illustrated in the following example.

▶ **Example 16.** Let P be a proposition variable, let a and b be actions without noticeable effect (i.e. $\mathit{effect}(a, v) = \mathit{effect}(b, v) = v$ for all valuations v), and let $s = P :\rightarrow a.\mathbf{1}$ and $t = P :\rightarrow b.\mathbf{1}$. Note that we have $\mathit{Cons}(\langle s, v\rangle)$ for all valuations v (since s does not emit any signal), but $\langle s, v\rangle \xrightarrow{a} \langle \mathbf{1}, v'\rangle$ only if $v(P) = \mathit{true}$. Hence, s does not have any outgoing transitions according to Definition 15. By the same reasoning, also t does not have any outgoing transitions. Therefore, s and t are bisimilar with respect to the transition relation induced on them by Definition 15.

When two terms are stateless bisimilar, then they are also bisimilar with respect to the transition relation induced on them by the Definition 15. The converse, however, does not hold: although the terms s and t in Example 16 are bisimilar, they are not stateless bisimilar.

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 17.** A coin toss can be described by the following term:

$$T \overset{\text{def}}{=} \mathit{toss}.(\mathit{heads} {\wedge}\!\blacktriangle \mathbf{1}) + \mathit{toss}.(\mathit{tails} {\wedge}\!\blacktriangle \mathbf{1}),$$

where $\mathit{effect}(\mathit{toss}, v) = v'$ is such that $v'(\mathit{heads}) = v'(\mathit{tails}) = \mathit{true}$ (for all v). Now, consider the expression

$$S \overset{\text{def}}{=} T \,;\, (\mathit{heads} :\rightarrow \mathit{hurray}.\mathbf{1} + \mathit{tails} :\rightarrow S).$$

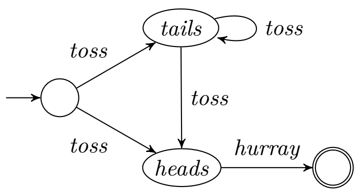

Figure 7 The process graph associated with the specification in Example 17.

Then S represents the process of tossing a coin until heads comes up, and its process graph is shown in Figure 7, as we shall now explain. Let

$$\mathit{Heads} = (\mathit{heads}\,{}^\wedge\,\mathbf{1})\ ;\ (\mathit{heads} :\rightarrow \mathit{hurray}.\mathbf{1} + \mathit{tails} :\rightarrow S)$$

and let

$$\mathit{Tails} = (\mathit{tails}\,{}^\wedge\,\mathbf{1})\ ;\ (\mathit{heads} :\rightarrow \mathit{hurray}.\mathbf{1} + \mathit{tails} :\rightarrow S)\ \ .$$

To see that $S \xrightarrow{\mathit{toss}} \mathit{Tails}$ and $S \xrightarrow{\mathit{toss}} \mathit{Heads}$, note that $\langle S, v\rangle \xrightarrow{\mathit{toss}} \langle \mathit{Heads}, \mathit{effect}(\mathit{toss}, v)\rangle$ and $\langle S, v\rangle \xrightarrow{\mathit{toss}} \langle \mathit{Heads}, \mathit{effect}(\mathit{toss}, v)\rangle$ for every valuation v.

To see that $\mathit{Tails} \xrightarrow{\mathit{toss}} \mathit{Tails}$ and $\mathit{Tails} \xrightarrow{\mathit{toss}} \mathit{Heads}$, first observe that $\mathit{Cons}(\langle \mathit{Tails}, v\rangle)$ if, and only if, $v(\mathit{tails}) = \mathit{true}$, and then note that for all such valuations v we, indeed, have $\langle \mathit{Tails}, v\rangle \xrightarrow{\mathit{toss}} \langle \mathit{Tails}, \mathit{effect}(\mathit{toss}, v)\rangle$ and $\langle \mathit{Tails}, v\rangle \xrightarrow{\mathit{toss}} \langle \mathit{Heads}, \mathit{effect}(\mathit{toss}, v)\rangle$.

It is instructive to see why we do *not* have that $\mathit{Tails} \xrightarrow{\mathit{hurray}} \mathbf{1}$. This is because if v is a valuation that satisfies $v(\mathit{tails}) = \mathit{true}$ and $v(\mathit{heads}) = \mathit{false}$, then we also have $\mathit{Cons}(\langle \mathit{Tails}, v\rangle)$, whereas $\langle \mathit{Tails}, v\rangle \overset{\mathit{hurray}}{\not\rightarrow} \langle \mathbf{1}, \mathit{effect}(\mathit{hurray}, v)\rangle$.

Finally, to see that $\mathit{Heads} \xrightarrow{\mathit{hurray}} \mathbf{1}$, first observe that $\mathit{Cons}(\langle \mathit{heads}, v\rangle)$ if, and only if, $v(\mathit{heads}) = \mathit{true}$, and then note that, indeed, $\langle \mathit{Heads}, v\rangle \xrightarrow{\mathit{hurray}} \langle \mathbf{1}, \mathit{effect}(\mathit{hurray}, v)\rangle$ for all such valuations v.

7 The full correspondence

We prove that signals and conditions make it possible to find a guarded sequential specification with the same process as a given pushdown automaton.

▶ **Theorem 18.** *For every pushdown automaton there is a guarded sequential recursive specification with signals and conditions with the same process.*

Proof. Let $M = (\mathcal{S}, \mathcal{A}, \mathcal{D}, \rightarrow, \uparrow, \downarrow)$ be a pushdown automaton. We can assume M only has push and pop transitions. For every state $s \in \mathcal{S}$ we have a propositional variable $\mathit{state}(s)$. The *effect* function for every action invariantly results in a valuation that assigns *true* to every propositional variable $\mathit{state}(s)$, to make sure that the execution of an action from a consistent state always results in a consistent state. Note that it is the interplay between the emitted root signal and the guards that ensures appropriate continuations (cf. also Example 17). We proceed to define the recursive specification with initial identifier X and additional identifiers $\{X_d \mid d \in \mathcal{D}\}$.

- If M does not contain any transition of the form $\uparrow \xrightarrow{a[\epsilon/d]} s$, and the initial state is final, we can take $X \overset{\mathrm{def}}{=} \mathbf{1}$ as the specification, and we do not need the additional identifiers.

- If M does not contain any transition of the form $\uparrow \xrightarrow{a[\epsilon/d]} s$, and the initial state is not final, we can take $X \overset{\text{def}}{=} \mathbf{0}$ as the specification, and we do not need the additional identifiers.
- Otherwise, there is some transition $\uparrow \xrightarrow{a[\epsilon/d]} s$ in M. For each such transition, add a summand

  $$a.(\mathit{state}(s) \,{}^\wedge\, X_d \,;\, (\mathit{state}(\uparrow) :\to X + \neg \mathit{state}(\uparrow) :\to \mathbf{1}))$$

  to the equation of X. The effect v of a in any valuation satisfies $v(\mathit{state}(s)) = \mathit{true}$. Besides these summands, add a summand $\mathbf{1}$ iff the initial state of M is final. Notice that no restriction on the initial valuation is necessary.
- Next, the equation for the added identifier X_d has a summand $\mathit{state}(s) :\to a.(\mathit{state}(t) \,{}^\wedge \mathbf{1})$ for each transition $s \xrightarrow{a[d/\epsilon]} t$, for every $s, t \in \mathcal{S}$. The effect v of a in any valuation satisfies $v(\mathit{state}(t)) = \mathit{true}$.

  In addition, the equation for the added identifier X_d has a summand

  $$\mathit{state}(s) :\to a.((\mathit{state}(t) \,{}^\wedge X_e \,;\, X_d)$$

  for each transition $s \xrightarrow{a[d/ed]} t$, for every $s, t \in \mathcal{S}$. The effect v of a in any valuation satisfies $v(\mathit{state}(t)) = \mathit{true}$.

  Finally, the equation for the added identifier X_d has summands $\mathit{state}(s) :\to \mathbf{1}$ (whenever $s \in \downarrow$) or $\mathit{state}(s) :\to \mathbf{0}$ (otherwise). ◀

▶ **Example 19.** For the pushdown automaton in Fig. 1, we find the following guarded recursive specification:

$$S = a.(\mathit{state}{\uparrow} \,{}^\wedge A \,;(\mathit{state}{\uparrow} :\to S + \mathit{state}{\downarrow} :\to \mathbf{1})) + c.(\mathit{state}{\downarrow} \,{}^\wedge \mathbf{1})$$

$$A = \mathit{state}{\downarrow} :\to b.(\mathit{state}{\downarrow} \,{}^\wedge \mathbf{1}) +$$

$$+ \mathit{state}{\uparrow} :\to (a.(\mathit{state}{\uparrow} \,{}^\wedge A; A) + b.(\mathit{state}{\uparrow} \,{}^\wedge \mathbf{1}) + c.(\mathit{state}{\downarrow} \,{}^\wedge A)).$$

Finally, we are interested in the question whether the mechanism of signals and conditions is not *too* powerful: can we find a pushdown automaton with the same process for any guarded sequential specification with signals and conditions? We will show the answer is positive. This means we recover the perfect analogue of the classical theorem: the set of processes given by pushdown automata coincides with the set of processes given by guarded sequential specifications with signals and conditions.

In order to prove this final theorem, we need another refinement of Greibach normal forms, now in the presence of signals and conditions. We start out from a result from [6]. Actually, we modify this result, using the sequencing ; instead of the sequential composition $\cdot$. The proof goes the same way. We use identities $\chi \,{}^\wedge x \leftrightarrow \chi \,{}^\wedge \mathbf{0} + x$ and $\chi \,{}^\wedge \mathbf{0} + \psi :\to \mathbf{1} \leftrightarrow \chi \,{}^\wedge \mathbf{0} + (\chi \wedge \psi) :\to \mathbf{1}$.

▶ **Theorem 20.** *Every sequential term with signals and conditions is bisimilar to one in* head normal form*, i.e. of the form*

$$\chi \,{}^\wedge (\psi :\to \mathbf{1}) + \sum_{i=1}^{n} \phi_i :\to a_i . p_i \qquad a_i \in \mathcal{A} \cup \{\tau\}$$

for terms p_i, and is consistent in any state with a valuation v satisfying $v(\chi) = true$. Here, χ is the root signal *of the term, and $\chi \wedge \psi$ is the* acceptance condition *of the term. If the acceptance condition $\chi \wedge \psi = false$, we write this as*

$$\chi \,\hat{}\, \mathbf{0} + \sum_{i=1}^{n} \phi_i :\rightarrow a_i.p_i \qquad a_i \in \mathcal{A} \cup \{\tau\}.$$

In the summation, we only write summands satisfying $\chi \wedge \phi_i \neq false$.

We see a term in head normal form is accepting in any state with a valuation v that makes the acceptance condition *true*, and it shows intermediate acceptance whenever the sum is nonempty and there is i with $v(\chi \wedge \psi \wedge \phi_i) = true$.

Based on this, we can write each equation in a guarded sequential specification with signals and conditions in a Greibach normal form, as follows:

$$X = \chi \,\hat{}\, (\psi :\rightarrow \mathbf{1}) + \sum_{i=1}^{n} \phi_i :\rightarrow a_i.\alpha_i \qquad a_i \in \mathcal{A} \cup \{\tau\}, X \in \mathcal{P}, \alpha_i \in \mathcal{P}^*.$$

Here, χ is the root signal of X, and ψ the acceptance condition of X, writing again

$$X = \chi \,\hat{}\, \mathbf{0} + \sum_{i=1}^{n} \phi_i :\rightarrow a_i.\alpha_i \qquad a_i \in \mathcal{A} \cup \{\tau\}, X \in \mathcal{P}, \alpha_i \in \mathcal{P}^*$$

if the acceptance condition is *false*. We define the Acceptance Irredundant Greibach normal form as before, disregarding the remaining signals and conditions.

▶ **Theorem 21.** *For every guarded sequential recursive specification with signals and conditions there is a pushdown automaton with the same process.*

Proof. Suppose a finite guarded sequential specification with signals and conditions is given. Without loss of generality we can assume this specification is in Acceptance Irredundant Greibach normal form, so every state of the specification is given by a sequence of identifiers that is acceptance irredundant. Consider the set of propositional variables $P_1, ..., P_n$ occurring in this specification. For each possible valuation $v : \{P_1, ..., P_n\} \to \{true, false\}$, we create two states in the pushdown automaton to be constructed, the state $\langle v, t\rangle$ is final and the state $\langle v, n\rangle$ is not. Then, we go along the lines of Theorem 13.

Take a valuation v such that the root signal of the initial identifier is *true*. If $v(\psi) = true$, where ψ is the acceptance condition of the initial identifier, then the initial state is $\langle v, t\rangle$. Otherwise, the initial state is $\langle v, n\rangle$. Next, for any identifier X of the specification with root signal χ and acceptance condition ψ, and any valuation v satisfying $v(\chi) = true$, we consider two cases.

- *Case 1.* Suppose $v(\psi) = true$. Look at a summand $\phi :\rightarrow a.\alpha$ of X. If $v(\phi) = false$ or the effect of executing a from v, the valuation v', makes the root signal of the first identifier of α *false*, we add no steps; otherwise, we consider two subcases.
  - *Subcase 1.a.* Suppose v' makes all root signals of the identifiers α and all acceptance conditions of the identifiers α *true*. Add a step $\langle v, t\rangle \xrightarrow{a[X/\alpha]} \langle v', t\rangle$. Moreover, in case X is initial, a step $\langle v, t\rangle \xrightarrow{a[\epsilon/\alpha]} \langle v', t\rangle$;
  - *Subcase 1.b.* Suppose v' makes some root signal of α or some acceptance condition of α *false*. Add a step $\langle v, t\rangle \xrightarrow{a[X/\alpha]} \langle v', n\rangle$. Moreover, in case X is initial, a step $\langle v, t\rangle \xrightarrow{a[\epsilon/\alpha]} \langle v', n\rangle$;

Next, repeat this procedure for the remaining summands of X.

- *Case 2.* Suppose $v(\psi) = \mathit{false}$. Look at a summand $\phi :\rightarrow a.\alpha$ of X. If $v(\phi) = \mathit{false}$ or the effect of executing a from v, the valuation v' makes the root signal of the first identifier of α *false*, we add no steps; otherwise, we consider two subcases.
  - *Subcase 2.a.* Suppose v' makes all root signals of α and all acceptance conditions of α *true*. Add a step $\langle v, n\rangle \xrightarrow{a[X/\alpha]} \langle v', t\rangle$. Moreover, in case X is initial, a step $\langle v, n\rangle \xrightarrow{a[\epsilon/\alpha]} \langle v', t\rangle$;
  - *Subcase 2.b.* Suppose v' makes some root signal of α or some acceptance condition of α *false*. Add a step $\langle v, n\rangle \xrightarrow{a[X/\alpha]} \langle v', n\rangle$. Moreover, in case X is initial, a step $\langle v, n\rangle \xrightarrow{a[\epsilon/\alpha]} \langle v', n\rangle$;

  Next, repeat this procedure for the remaining summands of X.

Now it is not difficult to check that the process of this pushdown automaton coincides with the process of the given guarded sequential specification. ◀

8 Conclusion

We looked at the classical theorem, that the set of languages given by a pushdown automaton coincides with the set of languages given by a context-free grammar. A language is an equivalence class of process graphs modulo language equivalence. A process is an equivalence class of process graphs modulo bisimulation. The set of processes given by a pushdown automaton coincides with the set of processes given by a finite guarded sequential recursive specification, if and only if we add a notion of state awareness, that allows to pass on some information during sequencing.

We see that signals and conditions add expressive power to TSP, since a signal can be passed along the sequencing operator. If we go to the theory BCP, so without sequencing but with parallel composition, then we know from [1] that value passing can be replaced by signal observation. We leave it as an open problem, whether or not signals and conditions add to the expressive power of BCP.

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 interaction. Such a computer model relates more closely to the computers we see all around us.

References

1 J. C. M. Baeten and J. A. Bergstra. Process algebra with propositional signals. *Theor. Comput. Sci.*, 177(2):381–405, 1997. doi:10.1016/S0304-3975(96)00253-8.

2 J. C. M. Baeten, P. J. L. Cuijpers, B. Luttik, and P. J. A. van Tilburg. A process-theoretic look at automata. In F. Arbab and M. Sirjani, editors, *Fundamentals of Software Engineering, Third IPM International Conference, FSEN 2009, Kish Island, Iran, April 15-17, 2009, Revised Selected Papers*, volume 5961 of *Lecture Notes in Computer Science*, pages 1–33. Springer, 2009. doi:10.1007/978-3-642-11623-0_1.

3 J. C. M. Baeten, P. J. L. Cuijpers, and P. J. A. van Tilburg. A context-free process as a pushdown automaton. In F. van Breugel and M. Chechik, editors, *Proceedings CONCUR'08*, number 5201 in Lecture Notes in Computer Science, pages 98–113, 2008.

4 J. C. M. Baeten, B. Luttik, and P. J. A. van Tilburg. Reactive Turing machines. *Information and Computation*, 231:143–166, 2013. Fundamentals of Computation Theory. doi:10.1016/j.ic.2013.08.010.

5 J. C. M. Baeten, B. Luttik, and F. Yang. Sequential composition in the presence of intermediate termination (extended abstract). In K. Peters and S. Tini, editors, *Proceedings Combined 24th International Workshop on Expressiveness in Concurrency and 14th Workshop on Structural Operational Semantics, EXPRESS/SOS 2017, Berlin, Germany, 4th September 2017.*, volume 255 of *EPTCS*, pages 1–17, 2017. `doi:10.4204/EPTCS.255.1`.

6 J. C.M. Baeten, A. A. Basten, and M. A. Reniers. *Process algebra: equational theories of communicating processes*, volume 50. Cambridge university press, 2010.

7 A. Belder. Decidability of bisimilarity and axiomatisation for sequential processes in the presence of intermediate termination. Master's thesis, Eindhoven University of Technology, 2018. Available from `https://research.tue.nl/en/studentTheses/decidability-of-bisimilarity-and-axiomatisation-for-sequential-pr`.

8 A. Belder, B. Luttik, and J. C. M. Baeten. Sequencing and intermediate acceptance: axiomatisation and decidability of bisimilarity. In Markus Roggenbach and Ana Sokolova, editors, *8th Conference on Algebra and Coalgebra in Computer Science, CALCO 2019*, Leibniz International Proceedings in Informatics, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.CALCO.2019.11`.

9 R. N. Bol and J. F. Groote. The meaning of negative premises in transition system specifications. *J. ACM*, 43(5):863–914, 1996. `doi:10.1145/234752.234756`.

10 D. Goldin and P. Wegner. The interactive nature of computing: Refuting the strong Church-Turing thesis. *Minds and Machines*, 18(1):17–38, 2008.

11 J. F. Groote. Transition system specifications with negative premises. *Theor. Comput. Sci.*, 118(2):263–299, 1993. `doi:10.1016/0304-3975(93)90111-6`.

12 B. Luttik. Divergence-preserving branching bisimilarity. In O. Dardha and J. Rot, editors, *Proceedings Combined 27th International Workshop on Expressiveness in Concurrency and 17th Workshop on Structural Operational Semantics, EXPRESS/SOS 2020, Online, 31 August 2020*, volume 322 of *EPTCS*, pages 3–11, 2020. `doi:10.4204/EPTCS.322.2`.

13 R. J. van Glabbeek. The meaning of negative premises in transition system specifications II. *J. Log. Algebr. Program.*, 60-61:229–258, 2004. `doi:10.1016/j.jlap.2004.03.007`.

14 R. J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996. `doi:10.1145/233551.233556`.

15 C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nord. J. Comput.*, 2(2):274–302, 1995.

16 P. Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, 1997.