PUSHDOWN AUTOMATA AND CONTEXT-FREE GRAMMARS IN BISIMULATION SEMANTICS [*]

JOS C. M. BAETEN [a,b], CESARE CARISSIMO [b], AND BAS LUTTIK [c]

[a] CWI, Amsterdam, The Netherlands
e-mail address: Jos.Baeten@cwi.nl

[b] University of Amsterdam, Amsterdam, The Netherlands
e-mail address: cesarecarissimo@gmail.com

[c] Eindhoven University of Technology, Eindhoven, The Netherlands
e-mail address: s.p.luttik@tue.nl

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.

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. [GW08, Weg97]. But an extra feature, we believe, is not the way to go. Interaction

Key words and phrases: pushdown automaton, context-free grammar, bisimilarity, intermediate acceptance, state awareness.

[*] Extended version of [BCL21].

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 [BLvT13]. 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 [BCLvT09, BLMT16]. 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: a formal language is accepted by a pushdown automaton if, and only if, it is generated by a context-free grammar. Our aim is to establish a process-theoretic variant of that correspondence theorem: a process is defined by a pushdown automaton if, and only if, can be specified in a process algebra comprising actions, choice, sequencing and recursion. There are several choices that we need to make both regarding the process-theoretic semantics of pushdown automata and regarding the process algebra.

Regarding the process-theoretic semantics of pushdown automata, in contrast to language semantics, in bisimulation semantics it makes a difference whether we consider acceptance by final state or acceptance by empty stack. Every process that is a pushdown process according to the acceptance-by-empty-stack interpretation is also a pushdown process according to the acceptance-by-final-state interpretation, but the converse does not hold [BCLvT09]. In this paper, we focus on the acceptance-by-final-state interpretation and set out to find the corresponding process algebra.

We start out from the seminal process algebra BPA of Bergstra and Klop [BK84]. Then, we need to add constants for acceptance and non-acceptance (deadlock), so we look at the process algebra TSP of [BBR10]. However, as we found earlier (starting from [BCvT08]), the process algebra TSP is not a good choice for the correspondence with pushdown auitomata modulo bisimulation: by means of a guarded recursive specification over TSP we can define processes that cannot be generated by any pushdown automaton, and some processes generated by a pushdown automaton cannot be defined by a guarded recursive specification over TSP.

By changing the process operator of sequential composition to a sequencing operator with intermediate acceptance, we get the theory $\mathrm{TSP}^{;}$ that provides a better correspondence in our setting [BLY17, Bel18, BLB19]. All processes defined by a guarded recursive specification over $\mathrm{TSP}^{;}$ can also be generated by a pushdown automaton. However, the other direction still does not hold.

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.

Our main contribution is to show that it suffices to add propositional signals and conditions in the style of [BB97] to restore the correspondence: we establish that a process is specified by a guarded sequential recursive specification with propositional signals and conditions if, and only if, it is the process associated with a pushdown automaton.

This paper extends [BCL21] by adding an axiomatisation of the process theory. This axiomatisation allows us to derive the head normal theorem already reported in [BCL21] purely equationally and is proved to be ground-complete for the recursion-free fragment of the process theory.

2. Preliminaries

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

Definition 2.1. 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 2.2. 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 $s\ R\ t$ 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 $s'\ R\ t'$; and
(2) if $s\downarrow$, then $t\downarrow$.

The results of this paper do not rely on abstraction from internal computations. This means we can use the *strong* version of bisimilarity defined above, which does not give special treatment to τ-labelled transitions. In general, when we do give special treatment to τ-labeled transitions, we use some form of *branching bisimulation* [vGW96].

A *process* is a (strong) 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*: the stack is last in and first out, something can be added on top of the stack (push), or something can be removed from the top of the stack (pop).

Definition 3.1 (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.

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.

Notice that we defined a pushdown automaton in such a way that it can be detected if the stack is empty, i.e. when there is no top element.

For example, consider the pushdown automaton depicted in Figure 1. It represents the process that can start to read an a, and after it has read at least one a, can read additional a's but can also read b's. Upon acceptance, it will have read up to as many b's as it has read a's. Interpreting symbol a as an increment and b as a decrement, we can see this process as a *counter*.

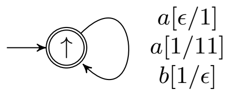

Figure 1: Pushdown automaton of a counter.

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 3.2. 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 process graph is meant.

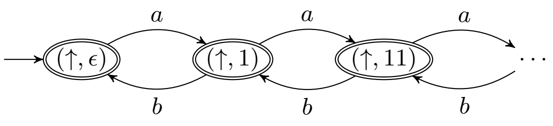

Figure 2: The process graph of the counter.

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

In language equivalence, the definition of acceptance in pushdown automata leads to the same set of languages when we define acceptance by final state (as we do here) and when we define acceptance by empty stack (not considering final states). In bisimilarity, these notions are different: acceptance by empty stack yields a smaller set of processes than acceptance by final state. This is illustrated by Figure 2: this process graph has infinitely many non-bisimilar final states. This cannot be realised if we define acceptance by empty stack. For details, see [BCLvT09].

A pushdown automaton has only finitely many transitions, so there is a maximum number of transitions from a given state, called its *branching degree*. Then, also the associated process graph has a branching degree, that cannot be larger than the branching degree of the underlying pushdown automaton. Thus, in a process graph associated with a pushdown automaton, the branching is always *bounded*.

4. Sequential Processes: TSP

In the process setting, a context-free grammar is a recursive specification over a process algebra comprising actions, choice and sequencing. We start out from the seminal process algebra BPA of Bergstra and Klop [BK84]. Later, it was extended to the process algebra $\text{BPA}_{\delta\varepsilon}$ by adding two constants δ (for deadlock) and ε (for termination). This process algebra was reformulated as the Theory of Sequential Processes (TSP) in [BBR10], using $\mathbf{0}$ and $\mathbf{1}$ for δ and ε. We present TSP next, and then will argue that it is not suitable to obtain a correspondence with pushdown automata.

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. We use symbols $a, b, \ldots$, possibly indexed, to range over $\mathcal{A} \cup \{\tau\}$, symbols $X, Y, \ldots$, possibly indexed, to range over $\mathcal{P}$. The set of *sequential process expressions* is generated by the following grammar ($a \in \mathcal{A} \cup \{\tau\}$, $X \in \mathcal{P}$):

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

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 $\cdot$ denote alternative composition and sequential composition, respectively. We adopt the convention that $a.\_$ binds strongest and $+$ binds weakest. The symbol $\cdot$ is often omitted when writing sequential process expressions.

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 $n > 0$.

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.

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 Figure 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.

$$\frac{}{\mathbf{1}\downarrow} \qquad \frac{}{a.p \xrightarrow{a} p}$$

$$\frac{p\downarrow}{(p+q)\downarrow} \qquad \frac{q\downarrow}{(p+q)\downarrow} \qquad \frac{p \xrightarrow{a} p'}{p+q \xrightarrow{a} p'} \qquad \frac{q \xrightarrow{a} q'}{p+q \xrightarrow{a} q'}$$

$$\frac{p\downarrow \quad q\downarrow}{p\cdot q\downarrow} \qquad \frac{p \xrightarrow{a} p'}{p\cdot q \xrightarrow{a} p'\cdot q} \qquad \frac{p\downarrow \quad q \xrightarrow{a} q'}{p\cdot q \xrightarrow{a} q'}$$

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

Figure 3: Operational semantics for sequential process expressions.

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

We proceed to define when two process expressions are behaviourally equivalent.

Definition 4.1. A binary relation R on the set of sequential process expressions is a *bisimulation* iff R is symmetric and for all process expressions p and q such that if $(p,q) \in R$:

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

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

The operational rules presented in Fig 3 are in the so-called *path format* from which it immediately follows that strong bisimilarity is a congruence [BV93]. Branching bisimulation, however, is not a congruence, but by adding a rootedness condition we get rooted branching bisimulation which is a congruence [vGW96].

As is customary in process theory, we restrict 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 rooted branching bisimulation.

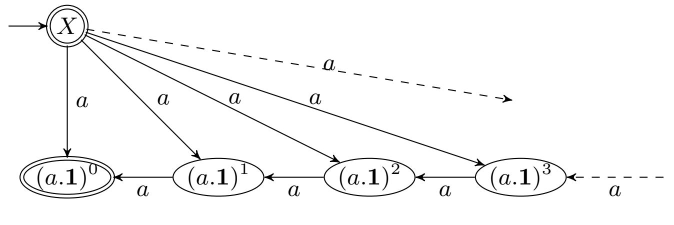

Figure 4: The process graph of X as defined in $X \overset{\text{def}}{=} \mathbf{1} + X \cdot a.\mathbf{1}$; the terms $(a.\mathbf{1})^n$ are inductively defined by $(a.\mathbf{1})^0 = \mathbf{1}$ and $(a.\mathbf{1})^{n+1} = (a.\mathbf{1})^n \cdot (a.\mathbf{1})$.

If we do not restrict to guarded recursion, unwanted behaviour can result, as the following example shows.

Example 4.2. Consider the (unguarded) recursive equation

$$X \overset{\text{def}}{=} \mathbf{1} + X \cdot a.\mathbf{1} \ .$$

We show the process graph generated by the operational rules in Figure 4

From the initial state, there are infinitely many transitions, to states that are all non-bisimilar. Thus, the process graph generated is infinitely branching, and it cannot be bisimilar to the graph of a pushdown automaton.

By restricting to guarded recursion, we guarantee that the associated process graphs are finitely branching.

It is convenient to present a guarded sequential specification in a normal form, the so-called Greibach normal form, see [BLB19].

Definition 4.3. A guarded sequential specification is in Greibach normal form, GNF, if every right-hand side of every equation has the following form:

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

Recall that the empty summation equals $\mathbf{0}$. The $\mathbf{1}$ summand may or may not occur. Furthermore, the empty sequence denotes $\mathbf{1}$.

It is well-known that 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, e.g., [BLB19]).

Since bisimilarity is a congruence, we can consider the equational theory of sequential expressions. From [BBR10], we know that the finite axiomatization in Table 1 is a sound and ground-complete (i.e. complete for all process expressions, not including variables) axiomatisation for the theory TSP with sequential composition and without recursion. In axiomatisations, we use variables x, y, z denoting arbitrary process expressions.

$x + y$	$=$	$y + x$	A1
$x + (y + z)$	$=$	$(x + y) + z$	A2
$x + x$	$=$	x	A3
$(x + y) \cdot z$	$=$	$x \cdot z + y \cdot z$	A4
$(x \cdot y) \cdot z$	$=$	$x \cdot (y \cdot z)$	A5
$x + \mathbf{0}$	$=$	x	A6
$\mathbf{0} \cdot x$	$=$	$\mathbf{0}$	A7
$x \cdot \mathbf{1}$	$=$	x	A8
$\mathbf{1} \cdot x$	$=$	x	A9
$(a.x) \cdot y$	$=$	$a.(x \cdot y)$	A10

Table 1: The axioms of TSP ($a \in \mathcal{A} \cup \{\tau\}$).

Now we show why the process algebra TSP is not suitable to establish a correspondence with pushdown automata. The problem is with the operational semantics of sequential composition. Consider the following example.

Example 4.4. Consider the recursive specification

$$X \overset{\text{def}}{=} \mathbf{1} + a.Y \cdot X \qquad Y \overset{\text{def}}{=} \mathbf{1} + b.\mathbf{1} + a.Y \cdot Y \ .$$

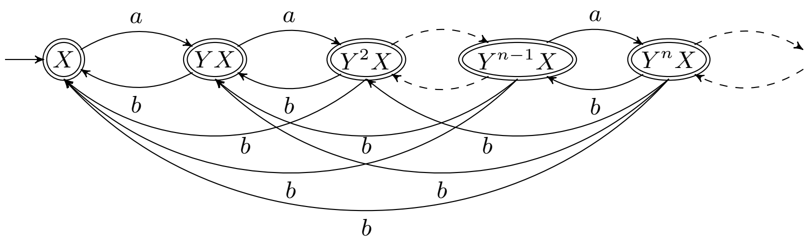

Figure 5: Process graph showing unbounded branching.

By following the operational rules, we obtain a process graph that is bisimilar to the one shown in Figure 5 (for simplicity we have identified states labelled with terms that are equal up to A5, A8 and A9). We see the similarity with Figure 2, but many more transitions are added.

We see that the state given by process expression $Y^n \cdot X$ has n outgoing transitions, to states $X, Y \cdot X, \ldots, Y^{n-1} \cdot X, Y^{n+1} \cdot X$. These states are all non-bisimilar. Thus, the branching in this process graph is unbounded, and it cannot be the process graph of a pushdown automaton.

We call the phenomenon occurring here *transparency*: it is possible to skip part of the terms in a sequential composition. It causes that TSP is too expressive: by means of a guarded recursive specification we can specify a process that is not pushdown. On the other hand, TSP is insufficiently expressive at the same time: also due to transparency, the pushdown automaton in Figure 1 cannot be specified in TSP.

Theorem 4.5. *There is no guarded recursive specification over* TSP *with a process graph bisimilar to the process graph in Figure 2.*

Proof. Suppose such a guarded recursive specification does exist. Without loss of generality, we can assume it is in Greibach normal form, such that every state of its process graph is given by a sequential composition of identifiers. As in Figure 2, from every state a maximal number of consecutive b-steps can be executed, this is also true in the process graph of the specification, and this is also true for each of the identifiers. As the specification only has finitely many identifiers, there is a maximal number of consecutive b-steps any identifier can execute. Take a number k larger than this maximum, and consider the state $(\uparrow, 1^k)$ of the counter, from which exactly k consecutive b-steps can be executed. This state is bisimilar to a sequential composition p of identifiers of the specification, so also from p, k consecutive b-steps can be executed. By choice of k, these b-steps cannot all come from the same component of p, so p contains two identifiers X, Y (X occurring before Y), each of which can execute at least one b-step. As $(\uparrow, 1^k)\downarrow$, also $p\downarrow$, and each identifier in p must be accepting, so $X\downarrow$ and $Y\downarrow$. But then, the last operational rule of sequential composition can be applied to X, and the b-step(s) in X can be skipped. This gives a sequence of less than k consecutive b-steps from p, after which no further b can be executed. But the resulting state must be bisimilar to a state $(\uparrow, 1^n)$ of the counter, with $n > 0$. From this state a further b can be executed, which gives a contradiction. □

Transparency occurs because of the last rule of sequential composition in the operational semantics: process expression $p \cdot q$ can execute a step from q, thereby forgetting p if p is accepting. When p has intermediate acceptance, so p can still execute a step, this step is lost.

In order to recover the correspondence between pushdown automata and sequential process algebra, we need to change this operational rule. In order not to cause confusion, we introduce a new operator ;, called *sequencing*, that will replace the sequential composition operator $\cdot$ of TSP. We call the resulting theory TSP;. It was introduced in [BLY17].

5. Sequential processes: $\mathrm{TSP}^{;}$

We replace the sequential composition $\cdot$ of TSP by the sequencing operator ; of which the operational rules are shown in Figure 6.

$$\frac{p\downarrow \quad q\downarrow}{p\,;q\downarrow} \qquad \frac{p \xrightarrow{a} p'}{p\,;q \xrightarrow{a} p'\,;q} \qquad \frac{p\downarrow \quad p \not\rightarrow \quad q \xrightarrow{a} q'}{p\,;q \xrightarrow{a} q'}$$

Figure 6: Operational semantics for sequencing.

The last rule for sequencing has a so-called negative premise. 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 [Gro93, BG96, vG04]. Indeed, in [BLY17] it was already pointed out that the transition system specification in Figure 3 gives rise to such anomalies in case we use unguarded recursion. Consider recursive equation $X \stackrel{\text{def}}{=} X\,; a.\mathbf{1} + \mathbf{1}$. If $X \not\rightarrow$, then according to the rules for sequencing and recursion we find that $X \xrightarrow{a} \mathbf{1}$, which is a contradiction. On the other hand, $X \rightarrow$ is not provable from the transition system specification.

This is the second reason for restricting our attention to *guarded* recursive specifications. If specification 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 $(X \stackrel{\text{def}}{=} p) \in \Delta$ 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 [Gro93] 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 recursive specification

$$X \stackrel{\text{def}}{=} \mathbf{1} + a.Y\,; X \qquad Y \stackrel{\text{def}}{=} \mathbf{1} + b.\mathbf{1} + a.Y\,; Y\ .$$

now yields a process graph bisimilar to the one in Figure 2, which has only binary branching. We see that it is the process of a pushdown automaton. In the next section, we will prove that every recursive specification over $\mathrm{TSP}^{;}$ generates a process graph that is bisimilar to the process graph of a pushdown automaton. We find that the expressiveness of $\mathrm{TSP}^{;}$ is *incomparable* to the expressiveness of TSP: by means of a guarded specification over $\mathrm{TSP}^{;}$ we can define the always accepting counter (see Figure 1) but not an unboundedly branching process, and for TSP it is the other way around.

The operational rules presented in Fig 6 are in the so-called *panth format* from which it immediately follows that strong bisimilarity is a congruence [Ver95]. Due to the sequencing operator, rooted branching bisimulation is no longer a congruence, and we have to add an extra condition: rooted *divergence-preserving* branching bisimulation [vGW96, Lut20] is a congruence, which can be proved using [FvGL19].

Using sequencing instead of sequential composition, the distributive law A4 is no longer valid as the following example shows.

Example 5.1. Consider the process expressions $(a.\mathbf{1}+\mathbf{1}) \mathbin{;} b.\mathbf{1}$ and $a.\mathbf{1} \mathbin{;} b.\mathbf{1}+\mathbf{1} \mathbin{;} b.\mathbf{1}$. On the one hand, since $a.\mathbf{1}+\mathbf{1} \xrightarrow{a} \mathbf{1}$, the last operational rule for ; does not apply to $(a.\mathbf{1}+\mathbf{1}) \mathbin{;} b.\mathbf{1}$, and hence $(a.\mathbf{1}+\mathbf{1}) \mathbin{;} b.\mathbf{1} \overset{b}{\nrightarrow}$. On the other hand, $\mathbf{1} \mathbin{;} b.\mathbf{1} \xrightarrow{b} \mathbf{1}$, so $a.\mathbf{1} \mathbin{;} b.\mathbf{1}+\mathbf{1} \mathbin{;} b.\mathbf{1} \xrightarrow{b} \mathbf{1}$. It follows that $(a.\mathbf{1}+\mathbf{1}) \mathbin{;} b.\mathbf{1} \not\leftrightarrow a.\mathbf{1} \mathbin{;} b.\mathbf{1}+\mathbf{1} \mathbin{;} b.\mathbf{1}$.

In fact, due to this failure, there is no finite sound and ground-complete axiomatization of bisimilarity for (the recursion-free fragment of) TSP$^{;}$ [Bel18, BLB19]. In order to find an axiomatization, nevertheless, it suffices to add an auxiliary unary operator NA, denoting *non-acceptance*. Intuitively, $NA(p)$ behaves as p except that it does not have the option to accept immediately. The operational semantics of NA is obtained by adding one very simple rule, see Figure 7.

$$\frac{p \xrightarrow{a} p'}{NA(p) \xrightarrow{a} p'}$$

Figure 7: Operational semantics for non-acceptance.

Now, we can formulate a sound and ground-complete axiomatization of bisimilarity for the recursion-free fragment of TSP$^{;}$ in three parts. First, there are the axioms in Table 1 without the distributive law A4. Second, the axiomatization of the *NA* operator is straightforward: see axioms NA1–NA4 in Table 2.

Finally, instead of the distributive law we include axioms A11–A13 shown in Table 2.

$NA(\mathbf{0})$	$=$	$\mathbf{0}$	NA1
$NA(\mathbf{1})$	$=$	$\mathbf{0}$	NA2
$NA(a.x)$	$=$	$a.x$	NA3
$NA(x+y)$	$=$	$NA(x)+NA(y)$	NA4
$NA(x+y) \mathbin{;} z$	$=$	$NA(x) \mathbin{;} z + NA(y) \mathbin{;} z$	A11
$(a.x+y+\mathbf{1}) \mathbin{;} NA(z)$	$=$	$(a.x+y) \mathbin{;} NA(z)$	A12
$(a.x+y+\mathbf{1}) \mathbin{;} (z+\mathbf{1})$	$=$	$(a.x+y) \mathbin{;} (z+\mathbf{1})+\mathbf{1}$	A13

Table 2: The axioms for the auxiliary operator NA and weaker forms of distributivity.

The ground-completeness argument in [Bel18, BLB19] proceeds via an *elimination* theorem: it is established that for every recursion-free process expression p there exists a recursion-free process expression q without occurrences of the operators ; and NA such that the equation $p = q$ is derivable from the axioms above using equational logic. Thus the ground-completeness of the axiomatisation of bisimilarity for (the recursion-free fragment) of TSP$^{;}$ follows from the ground-completeness of the axiomatisation of bisimilarity for (the recursion-free fragment of) BSP (see [BBR10, Theorem 4.4.12]).

In the context of a guarded recursive specification Δ, we can, moreover, obtain the following result.

Theorem 5.2 (head normal form theorem)**.** *Let Δ be a guarded recursive specification. For every process expression p there exists a natural number n and sequences of actions $a_1, \ldots, a_n$ and process expressions $p_1, \ldots, p_n$ such that the equation*

$$p = (\mathbf{1}+)\sum_{i=1}^{n} a_i.p_i$$

(where $(\mathbf{1}+\ )$ *indicates that there may or may not be a summand* $\mathbf{1}$*), is derivable from the axioms for* TSP$^{;}$ *and the equations in* Δ *using equational logic. The process expression* $(\mathbf{1}+)\sum_{i=1}^{n} a_i.p_i$ *is called the* head normal form *of* p*.*

Proof. Let p be a process expression. Without loss of generality we may assume that p is guarded. For we can replace every unguarded occurrence of a process identifier in p by the right-hand side of its defining equation in Δ which, since Δ is guarded, results in process expression with one fewer unguarded occurrence of a process identifier. We now proceed by induction on the structure of p.

Note that p cannot be itself a process identifier, since then p would not be guarded.

If $p = \mathbf{0}$, then p is a head normal form.

If $p = \mathbf{1}$, then by A6 we have $p = \mathbf{1} + \mathbf{0}$, which is a head normal form.

If $p = a.p'$, then by A6 and A1 we have that $p = \mathbf{0} + a.p' = \sum_{i=1}^{0} a_i p_i + a.p' = \sum_{i=1}^{1} a_i.p_i$, with $a_1 = a$ and $p_1 = p'$, which is a head normal form.

Suppose that $p = p' + p''$ and (using the induction hypothesis) that

$$p' = (\mathbf{1}+\ )\sum_{i=1}^{n'} a'_i.p'_i \text{ and } p'' = (\mathbf{1}+\ )\sum_{i=1}^{n''} a''_i.p''_i \ .$$

Then, using A1 and A2 and, if necessary, A3 to get rid of a superfluous occurrence of $\mathbf{1}$, we get that

$$p = (\mathbf{1}+\ )\sum_{i=1}^{n'+n''} a_i.p_i \ ,$$

where $a_i = a'_i$ and $p_i = p'_i$ if $1 \leq i \leq n'$ and $a_i = a''_{i-n'}$ and $p_i = p''_{i-n'}$ if $n' < i \leq n' + n''$, and so p has a head normal form.

Suppose that $p = p'\ ;p''$, and (using the induction hypothesis) that

$$p' = (\mathbf{1}+\ )\sum_{i=1}^{n'} a'_i.p'_i \text{ and } p'' = (\mathbf{1}+\ )\sum_{i=1}^{n''} a''_i.p''_i \ .$$

We distinguish three cases:

(1) Suppose that the head normal form of p' does not have the $\mathbf{1}$-summand. If $n' = 0$, then $p = p'\ ;p'' = \mathbf{0}\ ;p'' = \mathbf{0}$ and the latter is a head normal form. Otherwise, if $n' > 0$, then, by axioms NA1, NA3 and NA4 we have that $p' = NA(\sum_{i=1}^{n'} a'_i.p'_i)$. It then follows with applications A11, A5, NA1, NA3 and NA4 that

$$p = p'\ ;p'' = NA(\sum_{i=1}^{n'} a'_i.p'_i)\ ;p'' = \sum_{i=1}^{n'} NA(a'_i.p'_i)\ ;p'' = \sum_{i=1}^{n'} a'_i.p'_i\ ;p'' \ .$$

(2) Suppose that the head normal form of p' has the $\mathbf{1}$-summand, but the head normal form of p'' does not. If $n' = 0$, then, by A6 we have that $p' = \mathbf{1}$, so by A9 it follows that $p = p'\ ;p'' = 1\ ;p'' = p''$. Otherwise, if $n' > 0$, then we first note that, since the head normal form of p'' does not have the $\mathbf{1}$-summand, by the axioms for NA we get that $p'' = NA(p'')$. Then we find with an application of the axiom A12 (and applications of A1 and A2 if necessary) that

$$p = p'\ ;p'' = p'\ ;NA(p'') = \left(\sum_{i=1}^{n'} a'_i.p'_i\right)\ ;NA(p'') = \left(\sum_{i=1}^{n'} a'_i.p'_i\right)\ ;p'' \ .$$

We can then proceed as in the first case to find the head normal form for $p = \left(\sum_{i=1}^{n'} a'_i.p'_i\right)\ ;p''$.

(3) Suppose that both the head normal forms of p' and p'' have the $\mathbf{1}$-summand. Again, if $n' = 0$, then, by A6 we have that $p' = \mathbf{1}$, so by A9 it follows that $p = p' \,;p'' = \mathbf{1}\,;p'' = p''$. Otherwise, if $n' > 0$, then we apply the axiom A13 to get $p = p'\,;p'' = \mathbf{1} + \left(\sum_{i=1}^{n'} a'_i.p'_i\right);p''$, we proceed as in the first case to find a head normal form $\left(\sum_{i=1}^{n'} a'_i.p'_i\right)\,;p''$ for $\left(\sum_{i=1}^{n'} a'_i.p'_i\right)\,;p''$, and observe that $p = \mathbf{1} + \left(\sum_{i=1}^{n'} a'_i.p'_i\right)\,;p''$.

Suppose that $p = NA(p')$, and (using the induction hypothesis) that

$$p' = (\mathbf{1}+)\sum_{i=1}^{n} a_i.p_i \quad .$$

Then by the axioms for NA, A1 and A6 we have

$$p = NA(p') = (\mathbf{0}+)\sum_{i=1}^{n} a_i.p_i = \sum_{i=1}^{n} a_i.p_i \quad . \qquad \square$$

6. 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 specification over TSP; (i.e., the equivalence class of process graphs bisimilar to the process graph associated with the 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.

We use the following extra notation. 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 $n > 0$. We will also use this construct indexed by a word $x \in \mathcal{D}^*$, so if we have $X_d \in \mathcal{P}$ ($d \in \mathcal{D}$), then α_x is inductively defined by $\alpha_\epsilon = \mathbf{1}$ and $\alpha_{xd} = \alpha_x\,;X_d$.

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 6.1. *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 have identifiers X and X_d for $d \in \mathcal{D}$. If there is no transition $\uparrow \xrightarrow{a[\epsilon/x]} \uparrow$, 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 $a.\alpha_x\,;X$ for each such transition to the equation of the initial identifier X. Next, the equation for the added identifier X_d has a summand $a.\alpha_x$ for each transition $\uparrow \xrightarrow{a[d/x]} \uparrow$, and a summand $\mathbf{1}$ or $\mathbf{0}$, depending on whether $\uparrow \in \downarrow$ or not.

Now a bisimulation between the process graph of M and the process graph of the constructed specification can be obtained by relating a state $(\uparrow, x)$ ($x \in \mathcal{D}^*$) to the state given by the sequence of identifiers $\alpha_x\,;X$. The initial states are related (by identifying X and $\mathbf{1}\,;X$) and $(\uparrow, dx) \xrightarrow{a} (\uparrow, yx)$ just in case $X_d\,;\alpha_x\,;X \xrightarrow{a} \alpha_y\,;\alpha_x\,;X$. Also, $(\uparrow, \epsilon) \xrightarrow{a} (\uparrow, x)$ just in case $X \xrightarrow{a} \alpha_x\,;X$. $\square$

Example 6.2. The counter of Figure 2 can be seen as a stack over a singleton data set $\{1\}$. For a general finite data set $\mathcal{D}$, the stack that is accepting in every state has a pushdown automaton with state $\uparrow$, actions $\{push_d, pop_d \mid d \in \mathcal{D}\}$, $\downarrow = \{\uparrow\}$ and transitions $\uparrow \xrightarrow{push_d[\epsilon/d]} \uparrow$ and $\uparrow \xrightarrow{pop_d[d/\epsilon]} \uparrow$ for each $d \in \mathcal{D}$ and transitions $\uparrow \xrightarrow{push_e[d/ed]} \uparrow$ for each $d, e \in \mathcal{D}$.

The following guarded recursive specification is obtained for this pushdown automaton in accordance with the procedure described in the proof of Theorem 6.1.

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

Note that if we use the sequential composition operator $\cdot$ of TSP [BBR10] instead of the present sequencing operator ; of TSP$^{;}$, then Theorem 6.1 fails because of the transparency illustrated in Figure 5. With the sequential composition operator, we cannot find a recursive specification of the stack accepting in every state of Example 6.2, because of the same phenomenon.

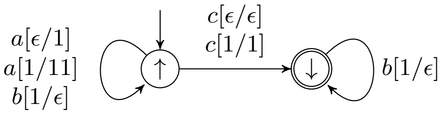

Figure 8: Pushdown automaton used in the proof of Theorem 6.3.

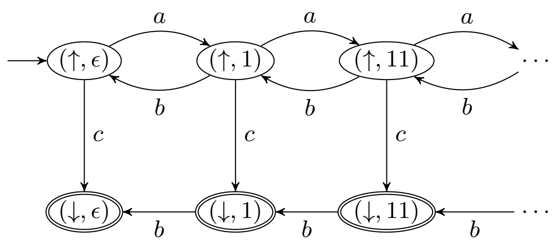

Figure 9: The process graph associated with the pushdown automaton in Figure 8.

Theorem 6.3. *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 8. Suppose there is a finite guarded sequential specification with the same process, depicted by the representative in Figure 9. Without loss of generality we can assume that this specification is in Greibach Normal Form (see [BLB19]). As a consequence, each state of the process graph generated by the automaton is bisimilar 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, and for $0 \leq i \leq k$ consider the state $(\uparrow, 1^i)$ reached after executing i a-steps. From this state, consider any sequence of steps $\overset{w}{\twoheadrightarrow}$ 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 $\overset{w}{\twoheadrightarrow}$ 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 $\alpha_m \overset{b^m}{\twoheadrightarrow}$, 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 $\alpha_m \overset{cb^m}{\twoheadrightarrow}$. This sequence of steps must also reach the second identifier, but then, a second c can be executed, contradicting that α_i is bisimilar to $(\uparrow, 1^i)$.

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. In the next section we will add a mechanism that allows the passing of extra information along the sequencing operator.

Theorem 6.3 holds for the sequencing operator we introduced, but it holds in the same way for the sequential composition operator of TSP [BBR10]. 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. The classical proof uses the equivalence of acceptance by final state and acceptance by empty stack, which is not true in bisimulation semantics.

Considered in terms of equations, we again have to deal with the failure of the distributive law. Due to this, we carefully need to consider every instance of intermediate acceptance. Consider the sequencing $(a.\mathbf{1} + \mathbf{1})\ ;\ b.\mathbf{1}$. The left argument of this sequencing shows intermediate acceptance, the right argument does not. As $(a.\mathbf{1} + \mathbf{1})\ ;\ b.\mathbf{1} \leftrightarrow a.\mathbf{1}\ ;\ b.\mathbf{1}$, the intermediate acceptance in the first argument 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 6.4. A guarded sequential specification is in Acceptance Irredundant Greibach normal form, AIGNF, if it is in Greibach normal form (see Definition 4.3), 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 sequences α and β may be empty. A sequence of this form is called *acceptance irredundant*.

It is proven in [BLB19, Proposition 20] 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 6.5. *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 [BLB19]. Every state of the specification is given by a sequence of identifiers that is acceptance irredundant. As a result, the state is accepting if and only if the lead variable is accepting. 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 $\alpha = \mathbf{1}$ or the first identifier of α is 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 $\alpha = \mathbf{1}$ or the first identifier of α is 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 the bisimulation relation will relate $\mathbf{1}$ to (t, ϵ) and will relate $X_d\ ;\ \alpha_x$ to (t, x) if $X_d \downarrow$ and to (n, x) if $X_d \not\downarrow$ for each $x \in \mathcal{P}^*$. 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. $\square$

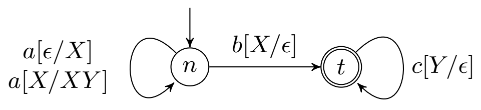

Figure 10: Pushdown automaton illustrating the construction in the proof of Theorem 6.5.

Example 6.6. Consider the recursive specification

$$X \stackrel{\text{def}}{=} a.X\ ;Y + b.\mathbf{1} \qquad Y \stackrel{\text{def}}{=} \mathbf{1} + c.\mathbf{1}$$

Notice it is in Acceptance Irredundant GNF. We obtain the pushdown automaton shown in Figure 10.

In the case of the sequential composition operator of TSP [BBR10], Theorem 6.5 fails because a guarded recursive specification over TSP can generate a process graph with unbounded branching, that cannot be bisimilar to a process graph generated by a pushdown automaton (see Example 4.4).

Note that Theorem 6.5 also fails when we use pushdown automata with acceptance by empty stack, for by means of a guarded specification over TSP$^{;}$ (or TSP) we can generate a process graph with infinitely many non-bisimilar accepting states.

The conclusion of this section is, that the replacement of sequential composition $\cdot$ by sequencing ; results in a calculus that is not *too* expressive: every guarded specification yields the process of a pushdown automaton. On the other hand, this calculus is not expressive *enough*: we still cannot specify all pushdown processes. In order to do that, we need an extra ingredient.

7. Signals and conditions

In order to obtain the missing correspondence, we need a mechanism to pass state information along a sequencing operator. We shall prove that it suffices to add the mechanism provided by propositional signals together with a conditional statement as given in [BB97], see also [BBR10].

To keep the focus of attention on the process calculus, especially also when we consider a sound and ground-complete axiomatization for it, we want to concern ourselves as little as possible with the formalities of propositional logic. To this end, we presuppose a Boolean algebra $\mathcal{B}$, with distinguished elements $false$ and $true$, a unary operator $\neg$ and binary operators $\vee$ and $\wedge$, freely generated by some suitable (finite or countably infinite) set of generators. The elements of this Boolean algebra can be thought of as abstract propositions modulo logical equivalence, and each generator as the logical equivalence class of some propositional variable. A *valuation* is a homomorphism from $\mathcal{B}$ into the two-element Boolean algebra; it is completely determined by how it maps the generators. Moreover, we have that if ϕ and ψ are elements of $\mathcal{B}$ and $v(\phi) = v(\psi)$ for all valuations v, then $\phi = \psi$. Henceforth, for the sake of readability, we will commit a minor abuse of language by referring to the elements of $\mathcal{B}$ as *propositions* and to the generators as *propositional variables*.

Given an element ϕ of $\mathcal{B}$ and a process expression p, we write $\phi :\rightarrow p$, with the intuitive meaning '*if* ϕ *then* p'; the construct is referred to as *conditional* or *guarded command*. The operational behaviour of $\phi :\rightarrow p$ depends on a valuation that associates a truth value with ϕ. 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 Figure 11; it presupposes a function *effect* that associates with every action a and every valuation v its effect. We define when a process expression together with a certain valuation can take a step or be in a final state.

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

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

$$\frac{\langle p, v\rangle \downarrow \quad \langle q, v\rangle \downarrow}{\langle p\,;q, v\rangle \downarrow} \qquad \frac{\langle p, v\rangle \xrightarrow{a} \langle p', v'\rangle}{\langle p\,;q, v\rangle \xrightarrow{a} \langle p'\,;q, v'\rangle}$$

$$\frac{\langle p, v\rangle \downarrow \quad \langle p, v\rangle \not\rightarrow \quad \langle q, v\rangle \xrightarrow{a} \langle q', v'\rangle}{\langle p\,;q, v\rangle \xrightarrow{a} \langle q', v'\rangle}$$

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

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

Figure 11: Operational rules for guarded command ($a \in \mathcal{A} \cup \{\tau\}$ and ϕ ranging over propositions).

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.

Definition 7.1. A binary relation R on the set of sequential process expressions with conditionals is a *stateless bisimulation* iff R is symmetric and for all process expressions p and q such that if $(p,q) \in R$:

(1) If for some $a \in \mathcal{A} \cup \{\tau\}$ and valuations v, v' we have $\langle p, v\rangle \xrightarrow{a} \langle p', v'\rangle$, then there exists a process expression q', such that $\langle q, v\rangle \xrightarrow{a} \langle q', v'\rangle$, and $(p', q') \in R$.

(2) If for some valuation v we have $\langle p, v\rangle\downarrow$, then $\langle q, v\rangle\downarrow$.

The process expressions p and q are stateless bisimilar (notation: $p \leftrightarroweq_s q$) iff there exists a stateless bisimulation R such that $(p, q) \in R$.

Stateless bisimulation is a congruence for sequential process expressions with conditionals (proven along the lines of [BBR10]), and so we can investigate the equational theory. We add the axioms in Table 3.

$$\begin{array}{lcll}
\mathit{true} :\rightarrow x & = & x & \text{C1}\\
\mathit{false} :\rightarrow x & = & \mathbf{0} & \text{C2}\\
(\phi \vee \psi) :\rightarrow x & = & (\phi :\rightarrow x) + (\psi :\rightarrow x) & \text{C3}\\
(\phi \wedge \psi) :\rightarrow x & = & \phi :\rightarrow (\psi :\rightarrow x) & \text{C4}\\
\phi :\rightarrow (x + y) & = & (\phi :\rightarrow x) + (\phi :\rightarrow y) & \text{C5}\\
\phi :\rightarrow (x \,;\, y) & = & (\phi :\rightarrow x) \,;\, y & \text{C6}\\
 & & & \\
\phi :\rightarrow \mathit{NA}(x) & = & \mathit{NA}(\phi :\rightarrow x) & \text{C7}\\
\multicolumn{3}{l}{(\mathit{NA}(x) + \phi :\rightarrow \mathbf{1}); (\mathit{NA}(y) + \psi :\rightarrow \mathbf{1})} & \\
 & = & \mathit{NA}(x); (\mathit{NA}(y) + \psi :\rightarrow \mathbf{1}) + (\phi \wedge \psi) :\rightarrow \mathbf{1} & \text{C8}
\end{array}$$

Table 3: The axioms for conditionals.

Note that there is no elimination theorem here: due to the presence of unresolved propositional variables, conditionals cannot be removed from all closed terms.

Next, we introduce an operator that allows the observation of aspects of the current state of a process graph. The central idea is that the observable part of the state of a process graph is represented by a proposition. We introduce the *root-signal emission operator* $\wedge\!\!\!\blacktriangle$. A process expression $\phi \wedge\!\!\!\blacktriangle x$ 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 process expressions, namely *consistency*. $\mathit{Cons}(\langle p, v\rangle)$ will not hold when the valuation of the root signal of p is false. A step $\xrightarrow{a}$ can only be between consistent states, and a state can only be accepting when it is consistent. Thus, if the effect of executing the action a is to set the value of the proposition ϕ to *false* (i.e., $\mathit{effect}(a, v)(\phi) = \mathit{false}$ for all valuations v), then the process expression $a.(\phi \wedge\!\!\!\blacktriangle p)$ can under no valuation execute action a.

The operational rules are defined in Fig. 12. First, we define the consistency predicate. Next, we find that the rule for action prefix, the rules for choice and the second rule for ; in Fig. 11 require an extra condition. The other rules of Table 11 can remain unchanged. Finally, we give the operational rules of the root signal emission operator. To emphasise the difference between guarded commands and root signal emission, process expression $\phi :\rightarrow \mathbf{1}$ is consistent under any valuation, whereas $\phi \wedge\!\!\!\blacktriangle \mathbf{1}$ is inconsistent under a valuation that assigns false to ϕ. So, if the effect of the action a is to set the value of ϕ to false, then $a.(\phi :\rightarrow \mathbf{1})$, in any valuation, can execute an a-transition, whereas $a.\phi \wedge\!\!\!\blacktriangle \mathbf{1}$ cannot.

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

Definition 7.2. A binary relation R on the set of sequential process expressions with conditionals and signals is a *stateless bisimulation* iff R is symmetric and for all process expressions p and q such that if $(p, q) \in R$:

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

$$\frac{Cons(\langle p, v\rangle) \;\; Cons(\langle q, v\rangle)}{Cons(\langle p + q, v\rangle)} \qquad \frac{Cons(\langle p, v\rangle)}{Cons(\langle \phi :\rightarrow p, v\rangle)} \qquad \frac{Cons(\langle p, v\rangle) \quad v(\phi) = true}{Cons(\langle \phi \mathbin{{}^{\wedge\!\blacktriangle}} p, v\rangle)}$$

$$\frac{Cons(\langle p, v\rangle) \quad \langle p, v\rangle \not\downarrow}{Cons(\langle p \,;\, q, v\rangle)} \qquad \frac{\langle p, v\rangle \downarrow \quad Cons(\langle q, v\rangle)}{Cons(\langle p \,;\, q, v\rangle)}$$

$$\frac{Cons(\langle p, v\rangle)}{Cons(\langle NA(p), v\rangle)} \qquad \frac{Cons(\langle p, v\rangle) \quad X \overset{\text{def}}{=} p}{Cons(\langle X, p\rangle)}$$

$$\frac{Cons(\langle p, v'\rangle) \quad v' = \mathit{effect}(a, v)}{\langle a.p, v\rangle \xrightarrow{a} \langle p, v'\rangle}$$

$$\frac{\langle p, v\rangle \xrightarrow{a} \langle p', v'\rangle \quad Cons(\langle q, v\rangle)}{\langle p + q, v\rangle \xrightarrow{a} \langle p', v'\rangle \;\; \langle q + p, v\rangle \xrightarrow{a} \langle p', v'\rangle} \qquad \frac{\langle p, v\rangle \downarrow \quad Cons(\langle q, v\rangle)}{\langle p + q, v\rangle \downarrow \;\; \langle q + p, v\rangle \downarrow}$$

$$\frac{\langle p, v\rangle \xrightarrow{a} \langle p', v'\rangle \quad Cons(\langle p' \,;\, q, v'\rangle)}{\langle p \,;\, q, v\rangle \xrightarrow{a} \langle p' \,;\, q, v'\rangle}$$

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

Figure 12: Operational rules for root-signal emission ($a \in \mathcal{A} \cup \{\tau\}$).

(1) If for some valuation v we have $Cons(\langle p, v\rangle)$, then also $Cons(\langle q, v\rangle)$.
(2) If for some $a \in \mathcal{A} \cup \{\tau\}$ and valuations v, v' we have $\langle p, v\rangle \xrightarrow{a} \langle p', v'\rangle$, then there exists a process expression q', such that $\langle q, v\rangle \xrightarrow{a} \langle q', v'\rangle$, and $(p', q') \in R$.
(3) If for some valuation v we have $\langle p, v\rangle\downarrow$, then $\langle q, v\rangle\downarrow$.

The process expressions p and q are bisimilar (notation: $p \mathrel{\underline{\leftrightarrow}}_s q$) iff there exists a stateless bisimulation R such that $(p, q) \in R$.

Again, stateless bisimulation is a congruence for sequential process expressions with conditionals and signals (see [BBR10]), and we have the following additional axioms.

$\mathit{true} \mathbin{{}^{\wedge\!\blacktriangle}} x$	$=$	x	S1
$\mathit{false} \mathbin{{}^{\wedge\!\blacktriangle}} x$	$=$	$\mathit{false} \mathbin{{}^{\wedge\!\blacktriangle}} \mathbf{0}$	S2
$a.(\mathit{false} \mathbin{{}^{\wedge\!\blacktriangle}} x)$	$=$	$\mathbf{0}$	S3
$(\phi \mathbin{{}^{\wedge\!\blacktriangle}} x) + y$	$=$	$\phi \mathbin{{}^{\wedge\!\blacktriangle}} (x + y)$	S4
$\phi \mathbin{{}^{\wedge\!\blacktriangle}} (\psi \mathbin{{}^{\wedge\!\blacktriangle}} x)$	$=$	$(\phi \wedge \psi) \mathbin{{}^{\wedge\!\blacktriangle}} x$	S5
$\phi :\rightarrow (\psi \mathbin{{}^{\wedge\!\blacktriangle}} x)$	$=$	$(\neg\phi \vee \psi) \mathbin{{}^{\wedge\!\blacktriangle}} (\phi :\rightarrow x)$	S6
$\phi \mathbin{{}^{\wedge\!\blacktriangle}} (\phi :\rightarrow x)$	$=$	$\phi \mathbin{{}^{\wedge\!\blacktriangle}} x$	S7
$\phi \mathbin{{}^{\wedge\!\blacktriangle}} (x \,;\, y)$	$=$	$(\phi \mathbin{{}^{\wedge\!\blacktriangle}} x) \,;\, y$	S8
$\phi \mathbin{{}^{\wedge\!\blacktriangle}} NA(x)$	$=$	$NA(\phi \mathbin{{}^{\wedge\!\blacktriangle}} x)$	S9

Table 4: The axioms for signals.

It is tedious but straightforward to verify that the extended theory is still sound. Below we shall prove first a head normal form theorem and then also a ground-completeness theorem.

Theorem 7.3 (head normal form theorem)**.** *Let Δ be a guarded recursive specification. For every process expression p there exists a natural number n, sequences of propositions $\phi_1, \ldots, \phi_n$, actions $a_1, \ldots, a_n$, and process expressions $p_1, \ldots, p_n$, and propositions ψ and χ such that*

$$p = \sum_{i=1}^{n} \phi_i :\rightarrow a_i.p_i + \psi {}^{\wedge}\!{}^{\blacktriangle} \chi :\rightarrow \mathbf{1} \tag{7.1}$$

is derivable from the axioms in Tables 1–4 and the equations in Δ using equational logic. The process expression at the right-hand side of the equation is called a head normal form *of p, ψ is called its* root signal *and χ is called its* acceptance condition.

Proof. Let p be a process expression. Without loss of generality we may assume that p is guarded, for we can replace every unguarded occurrence of a process identifier in p by the right-hand side of its defining equation in Δ which, since Δ is guarded, results in process expression with one fewer unguarded occurrence of a process identifier. We now proceed by induction on the structure of p.

Note that p cannot be itself a process identifier, since then p would not be guarded.

If $p = \mathbf{0}$, then $p = \mathbf{0} + \mathbf{0} = \mathbf{0} + \mathit{true} {}^{\wedge}\!{}^{\blacktriangle} \mathit{false} :\rightarrow \mathbf{1}$ by axioms A3, C2 and S1. Note that $\mathbf{0} + \mathit{true} {}^{\wedge}\!{}^{\blacktriangle} \mathit{false} :\rightarrow \mathbf{1}$ fits the shape of the right-hand side of Equation (7.1): we take $n = 0$ and we let $\psi = \mathit{true}$ and $\chi = \mathit{false}$.

If $p = \mathbf{1}$, then by A6, A1, C1 and S1 we have $p = \mathbf{1}+\mathbf{0} = \mathbf{0}+\mathbf{1} = \mathbf{0}+\mathit{true} {}^{\wedge}\!{}^{\blacktriangle} \mathit{true} :\rightarrow \mathbf{1}$, which fits the shape of the right-hand side of Equation (7.1) if we take $n = 0$, $\psi = \mathit{true}$ and $\chi = \mathit{true}$.

If $p = a.p'$, then by A6, C2 and S1 we have that $p = a.p' + \mathbf{0} = a.p' + \mathit{true} {}^{\wedge}\!{}^{\blacktriangle} \mathit{false} :\rightarrow \mathbf{1}$.

Suppose, now, by way of induction hypothesis, that

$$p' = \sum_{i=1}^{m} \phi'_i :\rightarrow a'_i.p'_i + \psi' {}^{\wedge}\!{}^{\blacktriangle} \chi' :\rightarrow \mathbf{1}$$

and

$$p'' = \sum_{i=1}^{n} \phi''_i :\rightarrow a''_i.p''_i + \psi'' {}^{\wedge}\!{}^{\blacktriangle} \chi'' :\rightarrow \mathbf{1} \quad .$$

First note that if $p = p' + p''$, then, by A1–A3, C3, S4 and S5, we get that

$$p = \sum_{i=1}^{m+n} \phi_i :\rightarrow a_i.p_i + (\psi' \wedge \psi'') {}^{\wedge}\!{}^{\blacktriangle} (\chi' \vee \chi'') :\rightarrow \mathbf{1} \quad ,$$

where $\phi_i = \phi'_i$, $a_i = a'_i$ and $p_i = p'_i$ if $1 \leq i \leq m$, and $\phi_i = \phi''_i$, $a_i = a''_{i-m}$ and $p_i = p''_{i-m}$ if $m < i \leq m+n$.

Next, we consider $p = p' \,;\, p''$. By axioms A1, NA1 (used if $m = 0$ or $n = 0$), NA3, NA4, C7, S4 and S8 we have

$$p' = NA(\psi' {}^{\wedge}\!{}^{\blacktriangle} \sum_{i=1}^{m} \phi'_i :\rightarrow a'_i.p'_i) + \chi' :\rightarrow \mathbf{1}$$

and

$$p'' = NA(\psi'' {}^{\wedge}\!{}^{\blacktriangle} \sum_{i=1}^{n} \phi''_i :\rightarrow a''_i.p''_i) + \chi'' :\rightarrow \mathbf{1} \quad .$$

Hence, by C8, we have

$$p = p' \,;p'' = \mathit{NA}(\psi' {}^{\wedge}\!\!\!\blacktriangle \sum_{i=1}^{m} \phi'_i :\rightarrow a'_i.p'_i) \,; p'' + (\chi' \wedge \chi'') :\rightarrow \mathbf{1} \quad .$$

The root signal can be transferred to the other summand with applications of S8, S9, S4 and A1:

$$p = \mathit{NA}(\sum_{i=1}^{m} \phi'_i :\rightarrow a'_i.p'_i) \,; p'' + \psi' {}^{\wedge}\!\!\!\blacktriangle (\chi' \wedge \chi'') :\rightarrow \mathbf{1} \quad .$$

We then find a head normal form for p with applications of A11, C6, NA3, C7, and A5:

$$p = \sum_{i=1}^{m} \phi'_i :\rightarrow a'_i.(p'_i \,; p'') + \psi' {}^{\wedge}\!\!\!\blacktriangle (\chi' \wedge \chi'') :\rightarrow \mathbf{1} \quad .$$

If $p = \mathit{NA}(p')$, then NA1–NA4, A1, A6, C7, S9, C2 and C4 we have

$$p = \mathit{NA}(p') = \sum_{i=1}^{m} \phi'_i :\rightarrow a'_i.p'_i + \psi' {}^{\wedge}\!\!\!\blacktriangle \chi' :\rightarrow \mathbf{0} = \sum_{i=1}^{m} \phi'_i :\rightarrow a'_i.p'_i + \psi' {}^{\wedge}\!\!\!\blacktriangle \mathit{false} :\rightarrow \mathbf{1} \quad .$$

Suppose that $p = \phi :\rightarrow p'$, with the head normal form of p' as given above. Let us first consider the case that $m > 0$, then by C5, S6 and C4 we get the following head normal form:

$$p = \sum_{i=1}^{m} (\phi \wedge \phi'_i) :\rightarrow a'_i.p'_i + (\neg\phi \vee \psi') {}^{\wedge}\!\!\!\blacktriangle (\phi \wedge \chi') :\rightarrow \mathbf{1} \quad .$$

If $m = 0$, then the same head normal form for p can be obtained, since by C2 and C4 we have

$$\phi :\rightarrow \sum_{i=1}^{m} \phi'_i :\rightarrow a'_i.p'_i = \phi :\rightarrow \mathbf{0} = \phi :\rightarrow (\mathit{false} :\rightarrow \mathbf{0}) =$$

$$\mathit{false} :\rightarrow \mathbf{0} = \mathbf{0} = \sum_{i=1}^{m} (\phi \wedge \phi'_i) :\rightarrow a'_i.p'_i \quad .$$

Suppose that $p = \phi {}^{\wedge}\!\!\!\blacktriangle p'$, again with the head normal form of p' as given above. Then by A1, S4 and S5 we obtain the following head normal form:

$$p' = \sum_{i=1}^{m} \phi'_i :\rightarrow a'_i.p'_i + (\phi \wedge \psi') {}^{\wedge}\!\!\!\blacktriangle \chi' :\rightarrow \mathbf{1}$$

□

There may be some redundancy in a head normal form. Consider, for instance, the head normal form

$$p = \mathbf{0} + P :\rightarrow a.\mathbf{1} + P :\rightarrow b.(\neg P) {}^{\wedge}\!\!\!\blacktriangle \mathbf{1} + (\neg P) :\rightarrow c.\mathbf{1} + P {}^{\wedge}\!\!\!\blacktriangle \mathit{true} :\rightarrow \mathbf{1} \quad .$$

Then the summand $\neg P :\rightarrow c.\mathbf{1}$ does not contribute any behaviour, since there does not exist a valuation that is consistent with the root signal P and at the same time makes condition $\neg P$ evaluate to true. Moreover, if the *effect* function satisfies, e.g., the property that $\mathit{effect}(b, v)(P) = v(P)$ for all valuations v, then the effect of b is not consistent with the root signal of $\neg P {}^{\wedge}\!\!\!\blacktriangle \mathbf{1}$, and so the summand $P :\rightarrow b.(\neg P) {}^{\wedge}\!\!\!\blacktriangle \mathbf{1}$ does not contribute any behaviour.

Definition 7.4. A head normal form $\sum_{i=1}^{n} \phi_i :\rightarrow a_i.p_i + \psi {}^{\wedge}\!\!\!\blacktriangle \chi :\rightarrow \mathbf{1}$ is *reduced* if, for every $1 \leq i \leq n$, there exists a valuation v such that $v(\phi_i \wedge \psi) = \mathit{true}$ and $\mathit{Cons}(\langle p_i, \mathit{effect}(a_i, v)\rangle)$.

In the remainder we shall only be interested in a very specific type of *effect* function. Let us denote by v_{true} the valuation that assigns *true* to every propositional variable. We say that the *effect* function has the *reset property* if $\mathit{effect}(a, v) = v_{true}$ for every action a and every valuation v. If the *effect* function has the reset property, then the axiom in Table 5 is valid.

$$a.(\neg P_1 \vee \cdots \vee \neg P_k)^\wedge x \quad = \quad \mathbf{0} \quad \mathrm{R}$$

Table 5: The axiom for the *effect* function with the reset property ($a \in \mathcal{A} \cup \{\tau\}$ and $P_1, \ldots, P_k$ is any sequence of propositional variables).

Lemma 7.5. *For every head normal form p with a consistent root signal there exists a reduced head normal form q such that $p = q$ is derivable from the axioms in Tables 1–5.*

Proof. Let $p = \sum_{i=1}^{n} \phi_i :\rightarrow a_i.p_i + \psi^\wedge\chi :\rightarrow \mathbf{1}$. If for some $1 \leq i \leq n$, there does not exist a valuation v such that $\phi_i \wedge \psi = \mathit{true}$, then $\phi_i \wedge \psi = \mathit{false}$, so by axioms S7, C4, and C2 we have that

$$\begin{aligned}\psi^\wedge(\phi_i :\rightarrow a_i.p_i) = \psi^\wedge(\psi :\rightarrow (\phi_i :\rightarrow a_i.p_i)) &= \psi^\wedge(\phi_i \wedge \psi) :\rightarrow a_i.p_i \\ &= \psi^\wedge(\mathit{false} :\rightarrow a_i.p_i) = \psi^\wedge\mathbf{0} = \mathbf{0}\ .\end{aligned}$$

Hence, by S4, A1–3 and A6 such a summand can be eliminated from the head normal form.

It remains to argue that if $\mathit{Cons}(\langle p_i, v_{true}\rangle)$ does not hold, then the summand $\phi_i :\rightarrow a_i.p_i$ can be eliminated. Note that by Theorem 7.3, p_i is derivably equal to a head normal form, say

$$p_i = \sum_{j=1}^{n_i} \phi_{i,j} :\rightarrow a_{i,j}.p_{i,j} + \psi_i{}^\wedge\chi_i :\rightarrow \mathbf{1} \quad .$$

From the rules in Figure 12 it follows that we have $\mathit{Cons}(\langle p_i, v_{true}\rangle)$ if, and only if, $v_{true}(\psi_i) = \mathit{true}$. Suppose that $v_{true}(\psi_i) = \mathit{false}$. Since ψ_i is an element of a Boolean algebra $\mathcal{B}$ that is freely generated by the propositional variables, there exists a finite sequence $P_1, \ldots, P_k$ of propositional variables such that, for every valuation v, $v(\psi_i)$ is completely determined by the truth values it assigns to these propositional variables. From $v_{true}(\psi_i) = \mathit{false}$, it then follows that for all valuations v such that $v(\psi_i) = \mathit{true}$ there exists $1 \leq i \leq k$ such that $v(P_i) = \mathit{false}$. Therefore, $\psi_i = \psi_i \wedge (\neg P_1 \vee \cdots \vee \neg P_k)$. Applications of axioms S5 and R then yield

$$\phi_i :\rightarrow a_i.p_i = \phi_i :\rightarrow a_i.(\neg P_1 \vee \cdots \vee \neg P_k)^\wedge p_i = \mathbf{0}.$$

We conclude that if $\mathit{Cons}(\langle p_i, v_{true}\rangle)$ does not hold, then the summand $\phi_i :\rightarrow a_i.p_i$ can be eliminated by A6. □

Below, we shall prove that, assuming an *effect* function with the reset property, the axioms in Tables 1–5 are complete for recursion-free process expressions. The proof will use induction on the depth of the process expressions involved.

Definition 7.6. We define the *depth* $|p|$ of a process expression p by

$$\begin{aligned}|p| = \sup\{n \mid \exists p_0, \ldots, p_n.\exists v_1, \ldots, v_n.\exists a_1, \ldots, a_n.\exists v_1', \ldots, v_n'.& \\ p = p_0\ \&\ \forall_{1 \leq i < n}.\langle p_i, v_i\rangle \xrightarrow{a_{i+1}} \langle p_{i+1}, v_i'\rangle\}&\ .\end{aligned}$$

From the operational semantics and the definition of stateless bisimilarity it follows that if $p \leftrightarroweq_s q$, then $|p| = |q|$. Moreover, if p is recursion-free, then $|p|$ is finite. The following lemma facilitates our ground-completeness proof to proceed by induction on depth.

Lemma 7.7. *Let $p = \sum_{i=1}^{n} \phi_i :\to a_i.p_i + \psi{}^{\wedge\!\blacktriangle}\chi :\to \mathbf{1}$ be a recursion-free and reduced head normal form and suppose that effect has the reset property; then $|p| > |p_i|$ for all $1 \leq i \leq n$.*

Proof. Let $1 \leq i \leq n$. Then since p is reduced, there exists a valuation v such that $v(\phi_i \wedge \psi) \neq \mathit{false}$. Since effect has the reset property, we have that $\mathit{effect}(a_i, v) = v_{true}$, and since p is reduced we have that $\mathit{Cons}(\langle p_i, v_{true}\rangle)$. Therefore $\langle p, v\rangle \xrightarrow{a_i} \langle p_i, v_{true}\rangle$. Since the depth of recursion-free processes is finite, it follows that $|p| > |p_i|$. □

We can now establish that our axiomatisation is ground-complete.

Theorem 7.8 (Ground-completeness)**.** *Suppose that the effect function has the reset property. For all recursion-free process expressions p and p', if $p \leftrightarroweq_s p'$ then $p = p'$ is derivable from the axioms in Tables 1–5 using equational logic.*

Proof. We proceed by induction on the sum of the depths of p and q. By Theorem 5.2 and Lemma 7.5 we may assume that p and p' are reduced head normal forms:

$$p = \sum_{i=1}^{n} \phi_i :\to a_i.p_i + \psi{}^{\wedge\!\blacktriangle}\chi :\to \mathbf{1}$$

and

$$p' = \sum_{j=1}^{n'} \phi'_j :\to a'_j.p'_j + \psi'{}^{\wedge\!\blacktriangle}\chi' :\to \mathbf{1} \quad .$$

First note that by condition 1 of Definition 7.2 we have that $\psi = \psi'$ and by condition 3 we have that $\chi = \chi'$. Hence, it remains to establish that $p' = p' + \phi_i :\to a_i.p_i$ for all $1 \leq i \leq n$.

Note that if v is some valuation such that $v(\phi_i) = \mathit{true}$, then $\langle p, v\rangle \xrightarrow{a_i} \langle p_i, v_{true}\rangle$. It then follows by condition 2 of Definition 7.2 that there exists $1 \leq j \leq n'$ such that $\langle p', v\rangle \xrightarrow{a_j} \langle p'_j, v_{true}\rangle$, $a_j = a_i$ and $p_i \leftrightarroweq_s p'_j$, and hence $v(\phi'_j) = \mathit{true}$. Define

$$J' = \{1 \leq j \leq n' \mid a_j = a_i \;\&\; p_j \leftrightarroweq_s p_i\} \quad .$$

We have just argued that $v(\phi_i) = \mathit{true}$ implies that there exists $j \in J'$ such that $v(\phi'_j) = \mathit{true}$, and hence

$$\bigvee_{j \in J'} (\phi_i \wedge \phi'_j) = \phi_i \quad . \tag{7.2}$$

Using that $\phi'_j = (\phi_i \wedge \phi'_j) \vee (\phi_i \wedge \neg\phi'_j)$, note that by axioms C3 and A3 we have

$$\phi'_j :\to a'_j.p'_j = \phi'_j :\to a'_j.p'_j + (\phi_i \wedge \phi'_j) :\to a'_j.p'_j \quad ,$$

and so we have

$$p' = p' + \sum_{j \in J'} (\phi_i \wedge \phi'_j) :\to a'_j.p'_j \quad .$$

By Lemma 7.7 we have $|p_i| < |p|$ and $|p'_j| < |p'|$, so by the induction hypothesis $p'_j = p_i$ is derivable from the axioms in Tables 1–5 . It follows that

$$p' = p' + \sum_{j \in J'} (\phi_i \wedge \phi'_j) :\to a_i.p_i \quad .$$

Finally, we apply C3 and Eqn. (7.2) to get

$$p' = p' + \phi_i :\to a_i.p_i \quad .$$

□

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

Definition 7.9. Let p, q be sequential process expressions with signals and conditions, let $a \in \mathcal{A} \cup \{\tau\}$ and suppose the root signal of t is not *false*.

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

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

Example 7.10. Let P be a proposition variable, and let $p = P :\rightarrow a.\mathbf{1}$ and $q = P :\rightarrow b.\mathbf{1}$. Note that we have $Cons(\langle p, v\rangle)$ for all valuations v (since p does not emit any signal), but $\langle p, v\rangle \xrightarrow{a} \langle \mathbf{1}, v'\rangle$ only if $v(P) = true$. Hence, p does not have any outgoing transitions according to Definition 7.9. By the same reasoning, also q does not have any outgoing transitions. Therefore, p and q are bisimilar with respect to the transition relation induced on them by Definition 7.9.

When two process expressions are stateless bisimilar, then they are also bisimilar with respect to the transition relation induced on them by the Definition 7.9. The converse, however, does not hold: although the process expressions p and q in Example 7.10 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.

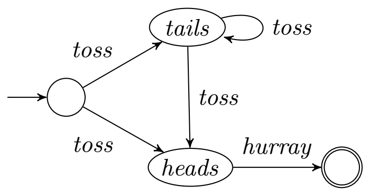

Figure 13: The process graph associated with the specification of a coin toss.

Example 7.11. A coin toss can be described by the following process expression:

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

where the *effect* function has the reset property, i.e. for every valuation v we have

$$\mathit{effect}(toss, v)(heads) = \mathit{effect}(toss, v)(tails) = true \quad .$$

Now, consider the expression

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

Then S represents the process of tossing a coin until heads comes up, and its process graph is shown in Figure 13, as we shall now explain. Let

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

and let

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

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.

8. 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 8.1. *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. For every state $s \in \mathcal{S}$ we have a propositional variable $\mathit{state}(s)$. We assume an *effect* function with the reset property, 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 7.11). We proceed to define the recursive specification with initial identifier X and additional identifiers X_ϵ and $\{X_d \mid d \in \mathcal{D}\}$.

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

  $$a.(\mathit{state}(s) \,{}^{\blacktriangle}\, \alpha_x \,;\, X_\epsilon)$$

  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

  $$\mathit{state}(s) :\rightarrow a.(\mathit{state}(t) \,{}^{\blacktriangle}\, \alpha_x)$$

  for each transition $s \xrightarrow{a[d/x]} t$, 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 a summand $\mathit{state}(s) :\rightarrow \mathbf{1}$ whenever $s \in \downarrow$.

- Lastly, the equation for the added identifier X_ϵ has a summand

  $$\mathit{state}(s) :\rightarrow a.(\mathit{state}(t)\,{}^{\wedge}\alpha_x \,;\, X_\epsilon)$$

  for each transition $s \xrightarrow{a[\epsilon/x]} t$, 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_ϵ has a summand $\mathit{state}(s) :\rightarrow \mathbf{1}$ whenever $s \in \downarrow$.

Then the bisimulation relation will relate the initial states, and will relate every state (s, x) of the process graph of the pushdown automaton to state $\mathit{state}(s)\,{}^{\wedge}\alpha_x \,;\, X_\epsilon$ of the process graph of the constructed specification. We see that the process graph of the pushdown automaton has a step $(s, dy) \xrightarrow{a} (s, xy)$ (coming from a step $s \xrightarrow{a[d/x]} t$ in the automaton) if, and only if, $\mathit{state}(s)\,{}^{\wedge}X_d \,;\, \alpha_y \,;\, X_\epsilon \xrightarrow{a} \mathit{state}(t)\,{}^{\wedge}\alpha_x \,;\, \alpha_y \,;\, X_\epsilon$ in the process graph of the specification.

Also, the process graph of the pushdown automaton has a step $(s, \epsilon) \xrightarrow{a} (s, x)$ (coming from a step $s \xrightarrow{a[\epsilon/x]} t$ in the automaton) if, and only if, $\mathit{state}(s)\,{}^{\wedge}X_\epsilon \xrightarrow{a} \mathit{state}(t)\,{}^{\wedge}\alpha_x \,;\, X_\epsilon$ in the process graph of the specification. □

Example 8.2. For the pushdown automaton in Fig. 8, we find the following guarded recursive specification:

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

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

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. Recall the head normal form of the previous section. Using Theorem 7.3 and, if necessary, adding a finite number of process identifiers with appropriate defining equations, we can write each equation in a guarded sequential specification with signals and conditions in Greibach normal form, as follows:

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

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

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

if $\psi \wedge \chi$ is *false*. We see that X is accepting in any state with a valuation v such that $v(\psi \wedge \chi) = \mathit{true}$, and it shows intermediate acceptance whenever there exists $1 \leq i \leq n$ with $v(\psi \wedge \chi \wedge \phi_i) = \mathit{true}$.

We define the Acceptance Irredundant Greibach normal form as before, disregarding the remaining signals and conditions.

Theorem 8.3. *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.

Because the specification is in AIGNF, it becomes easier to establish the acceptance and steps for any state of the process graph of the specification. For any variable of the specification, we have $X \downarrow$ iff for all valuations v that make the root signal of X true also make the acceptance condition of X true, and $X \xrightarrow{a} \alpha$ iff for all valuations v that make the root condition of X also make the guard of a true and $Cons(v, a)$.

Because of the acceptance irredundancy, it holds that if $X \downarrow$ then also $X;\beta \downarrow$ for every reachable state of the specification. Also, if $X \xrightarrow{a} \alpha$ then also $X ; \beta \xrightarrow{a} \alpha ; \beta$ for every reachable state of the specification $X ; \beta$.

This means we can follow the proof of Theorem 6.5 exactly, and we are done. □

9. 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.

This paper solves the question how we can characterize the set of processes given by a pushdown automaton. In the process setting, a context-free grammar is a process algebra with actions, choice, sequential composition and recursion. We need to limit to guarded recursion in the process setting. Starting out from the seminal process algebra BPA of [BK84], we need to add constants for the inactive and accepting process and for the inactive non-accepting (deadlock) process. We look at the candidate process algebra TSP of [BBR10], but find it is not suitable: by means of a guarded recursive specification we find a process that cannot be the process of a pushdown automaton, and also, there is a process of a pushdown automaton that cannot be given by a guarded recursive specification.

If we replace the sequential composition operator of TSP by a sequencing operator [BLY17], we get a better situation: every guarded specification yields the process of a pushdown automaton, but still, not the other way around, there are processes of pushdown automata that cannot be given by a guarded specification. We obtain a complete correspondence by adding a notion of state awareness, that allows to pass on some information during sequencing.

A process is defined by a pushdown automaton if and only if it is defined by a finite guarded recursive specification over a process algebra with actions, choice, sequencing, conditions and signals.

We see that signals and conditions add expressive power to the process algebras considered, since a signal can be passed along the sequencing (or sequential composition) operator. If we go to the theory BCP, so without sequencing but with parallel composition, then we know from [BB97] 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.

A pushdown automaton is a model of a computer with a memory in the form of a stack. A *parallel pushdown automaton* is like a pushdown automaton, but with a memory in the form of a multiset (a set, where the elements have a multiplicity, telling how often they occur). It is interesting to compare the set of processes defined by a parallel pushdown automaton with the *basic parallel* processes, the processes that are defined by a guarded recursive specification over the language, where sequencing is replaced by parallel composition. We conjecture that every process defined by a basic parallel recursive specification can also be defined by a parallel pushdown automaton, and in the other direction that a process defined by a one-state parallel pushdown automaton can also be defined by a basic parallel recursive specification. In order to recover a full correspondence, we need to add an extra feature. We speculate this extra feature can be communication between parallel processes, without abstraction. Some details can be found in [vT11].

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.

Acknowledgement

Thanks to the anonymous referees we were able to correct some errors and make some clarifications, improving on [BCL21].

References

[BB97] Jos C. M. Baeten and Jan A. Bergstra. Process algebra with propositional signals. *Theor. Comput. Sci.*, 177(2):381–405, 1997. URL: `https://doi.org/10.1016/S0304-3975(96)00253-8`, `doi:10.1016/S0304-3975(96)00253-8`.

[BBR10] Jos C. M. Baeten, Twan Basten, and Michel Reniers. *Process algebra: equational theories of communicating processes*, volume 50. Cambridge university press, 2010.

[BCL21] Jos C. M. Baeten, Cesare Carissimo, and Bas Luttik. Pushdown automata and context-free grammars in bisimulation semantics. In Fabio Gadducci and Alexandra Silva, editors, *9th Conference on Algebra and Coalgebra in Computer Science, CALCO 2021, August 31 to September 3, 2021, Salzburg, Austria*, volume 211 of *LIPIcs*, pages 8:1–8:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.CALCO.2021.8`.

[BCLvT09] Jos C. M. Baeten, Pieter Cuijpers, Bas Luttik, and Paul van Tilburg. A process-theoretic look at automata. In Farhad Arbab and Marjan 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`.

[BCvT08] Jos C. M. Baeten, Pieter Cuijpers, and Paul 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.

[Bel18] Astrid 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`.

[BG96] Roland N. Bol and Jan Friso Groote. The meaning of negative premises in transition system specifications. *J. ACM*, 43(5):863–914, 1996. `doi:10.1145/234752.234756`.

[BK84] Jan A. Bergstra and Jan Willem Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984. `doi:10.1016/S0019-9958(84)80025-X`.

[BLB19] Astrid Belder, Bas Luttik, and Jos 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`.

[BLMT16] Jos C. M. Baeten, Bas Luttik, Tim Muller, and Paul J. A. van Tilburg. Expressiveness modulo Bisimilarity of Regular Expressions with Parallel Composition. *Mathematical Structures in Computer Science*, 26:933–968, 2016. doi:10.1017/S0960129514000309.

[BLvT13] Jos C. M. Baeten, Bas Luttik, and Paul van Tilburg. Reactive Turing machines. *Information and Computation*, 231:143 – 166, 2013. Fundamentals of Computation Theory. doi:https://doi.org/10.1016/j.ic.2013.08.010.

[BLY17] Jos C. M. Baeten, Bas Luttik, and Fei Yang. Sequential composition in the presence of intermediate termination (extended abstract). In Kirstin Peters and Simone Tini, editors, *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.*, volume 255 of *EPTCS*, pages 1–17, 2017. URL: http://arxiv.org/abs/1709.00049, doi:10.4204/EPTCS.255.1.

[BV93] Jos C. M. Baeten and Chris Verhoef. A congruence theorem for structured operational semantics with predicates. In Eike Best, editor, *Proceedings CONCUR 1993, Hildesheim, Germany, August 1993.*, number 715 in Lecture Notes in Computer Science, pages 477–492, 1993.

[FvGL19] Wan Fokkink, Rob van Glabbeek, and Bas Luttik. Divide and congruence III: from decomposition of modal formulas to preservation of stability and divergence. *Inf. Comput.*, 268, 2019. doi:10.1016/j.ic.2019.104435.

[Gro93] Jan Friso Groote. Transition system specifications with negative premises. *Theor. Comput. Sci.*, 118(2):263–299, 1993. doi:10.1016/0304-3975(93)90111-6.

[GW08] Dina Goldin and Peter Wegner. The interactive nature of computing: Refuting the strong church–turing thesis. *Minds and Machines*, 18(1):17–38, 2008.

[Lut20] Bas Luttik. Divergence-preserving branching bisimilarity. In Ornela Dardha and Jurriaan Rot, editors, *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*, volume 322 of *EPTCS*, pages 3–11, 2020. doi:10.4204/EPTCS.322.2.

[Ver95] Chris Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nord. J. Comput.*, 2(2):274–302, 1995.

[vG04] Rob 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.

[vGW96] Rob van Glabbeek and Peter Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996. doi:10.1145/233551.233556.

[vT11] Paul van Tilburg. *From Computability to Executability*. PhD thesis, Eindhoven University of Technology, 2011.

[Weg97] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, 1997.