Pushdown Automata and Context-Free Grammars in Bisimulation Semantics

Jos C.M. Baeten\textsuperscript{1,2}, Cesare Carissimo\textsuperscript{2}, and Bas Luttik\textsuperscript{3}

\textsuperscript{1} CWI, Amsterdam, The Netherlands
\textsuperscript{2} University of Amsterdam, Amsterdam, The Netherlands
\textsuperscript{3} Eindhoven University of Technology, Eindhoven, The Netherlands

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

Keywords: pushdown automaton · context-free grammar · bisimilarity · intermediate acceptance · 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 the student insight in the essence of a computer, and tells him or her 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, that 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. [1, 2]. But an extra feature, we believe, is not the way to go. Interaction is an essential ingredient, such as it has been treated in many forms of concurrency theory. We seek a full integration of automata theory and concurrency
theory, and proposed the Reactive Turing Machine in [3]. In the Reactive Turing Machine, transitions have labels to give a notion of interactivity. In the resulting transition systems, we use bisimilarity instead of language equivalence.

Subsequently, we considered other classical theorems and notions from automata theory and formal languages theory [4]. We find richer results and a finer theory. In this paper, we consider the classical theorem of the correspondence between pushdown automata and context-free grammars. Before [5], we did not get a good correspondence in the process setting. By changing the process operator of sequential composition to a sequencing operator with intermediate acceptance, we get a better correspondence in our setting [6–8]. We find that the missing ingredient to recover the full correspondence is the addition of a notion of state awareness, by means of a signal that can be passed along a sequencing operator.

2 Preliminaries

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

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

1. \(S\) is a set of states;
2. \(A\) is a set of actions, \(\tau \not\in A\);
3. \(\rightarrow \subseteq S \times A \cup \{\tau\} \times S\) is an \(A \cup \{\tau\}\)-labelled transition relation; and
4. \(\downarrow \subseteq S\) is the set of final or accepting states.

A transition system is a transition system space with a special designated root state, i.e., it is a quintuple \((S, A, \rightarrow, \downarrow, s)\) such that \((S, A, \rightarrow, \downarrow)\) is a transition system space, and \(s \in S\). 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 transition systems, we recover languages as a semantics, but we can also consider other equivalence relations. Notable among these is bisimilarity.

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

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

The results of this paper do not rely on abstraction from internal computations, so we can use the strong version of bisimilarity defined above, which does not give special treatment to \(\tau\)-labelled transitions. But in general we have to use a version of bisimilarity that accommodates for abstraction from internal activity; the finest such notion of bisimilarity is divergence-preserving branching bisimilarity, which was introduced in [9] (see also [10] for an overview of recent results).
3 Pushdown Automata

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

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

1. $S$ is a finite set of states,
2. $A$ is a finite input alphabet, $\tau \not\in A$ the unobservable step,
3. $D$ is a finite data alphabet,
4. $\rightarrow \subseteq S \times (A \cup \{\tau\}) \times (D \cup \{\epsilon\}) \times D^* \times S$ is a finite set of transitions or steps,
5. $\uparrow \in S$ is the initial state,
6. $\downarrow \subseteq S$ is the set of final or accepting states.

![Diagram of a pushdown automaton](image.png)

Fig. 1. An example pushdown automaton.

If $(s, a, d, x, t) \in \rightarrow$ with $d \in D$, 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 $s \xrightarrow{\tau} t$ 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 $s \xrightarrow{\tau} t$ and $s \xrightarrow{\tau} t$, no input symbol is consumed, only the stack is modified.

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

We do not consider the language of a pushdown automaton, but rather consider the branching bisimulation equivalence class of the transition system of a pushdown automaton. A state of this transition system is a pair $(s, x)$, where $s \in S$ is the current state and $x \in D^*$ 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 transition system are labeled by the inputs of the pushdown automaton or $\tau$. 
Definition 4. Let $M = (S, A, D, \to, \uparrow, \downarrow)$ be a pushdown automaton. The transition system $T(M) = (S_T(M), A, \rightarrow_T(M), \downarrow_T(M), \uparrow_T(M))$ associated with $M$ is defined as follows:

1. $S_T(M) = \{(s, x) | s \in S \& x \in D^*\}$;
2. $\rightarrow_T(M) \subseteq S_T(M) \times A \cup \{\tau\} \times S_T(M)$ is the least relation such that for all $s, s' \in S$, $a \in A \cup \{\tau\}$, $d \in D$ and $x, x' \in D^*$ we have
   $$\left((s, dx) \xrightarrow{a}_{T(M)} (s', x'x)\right) \text{ if, and only if, } s \xrightarrow{a[d/x']} s'$$
   $$\left((s, \epsilon) \xrightarrow{a}_{T(M)} (s', x)\right) \text{ if, and only if, } s \xrightarrow{a[\epsilon/x]} s'$$
3. $\downarrow_T(M) = \{(s, x) | s \in \downarrow \& x \in D^*\}$;
4. $\uparrow_T(M) = (\uparrow, \epsilon)$.

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 transition system, we have attached a subscript $T(M)$ to the latter. In the remainder of this paper, we will suppress the subscript whenever it is already clear from the context whether a component of the pushdown automaton or its associated transition system is meant.

![Fig. 2. The transition system associated with the pushdown automaton in Figure 1.](image)

Figure 2 depicts the transition system associated with the pushdown automaton depicted in Figure 1.

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

4 Sequential Processes

In our setting, a context-free grammar is denoted by a finite guarded recursive specification over the process algebra $TSP^+(A)$. 
In this section we present the Theory of Sequential Processes adopting the revised operational semantics for sequential composition proposed in [6]. Sequential composition with the operational semantics of [11] is denoted by ·, and we call the operator with the revised operational semantics sequencing and denote it by ;.

Let \( \mathcal{A} \) be a set of actions, symbols denoting atomic events, and let \( \mathcal{P} \) be a finite set of process identifiers. The sets \( \mathcal{A} \) and \( \mathcal{P} \) serve as parameter of the process theory \( \text{TSP}'(\mathcal{A}, \mathcal{P}) \) that we shall introduce below. The set of process expressions associated with \( \text{TSP}'(\mathcal{A}, \mathcal{P}) \) is generated by the following grammar (\( a \in \mathcal{A}, X \in \mathcal{P} \)):

\[
p ::= 0 \mid 1 \mid a.p \mid p + p \mid p ; p \mid X.
\]

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

A recursive specification over \( \text{TSP}'(\mathcal{A}, \mathcal{P}) \) is a mapping \( \Delta \) from \( \mathcal{P} \) to the set of process expressions associated with \( \text{TSP}'(\mathcal{A}, \mathcal{P}) \). The idea is that the process expression \( p \) associated with a process identifier \( X \in \mathcal{P} \) by \( \Delta \) defines the behaviour of \( X \). We prefer to think of \( \Delta \) 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 \( \Delta \) 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, \( \Delta \) 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 \( a \in \mathcal{A} \), a
binary transition relation $\xrightarrow{a}$ (written infix), by means of the transition system specification presented in Fig. 3. We write $p \xrightarrow{a}$ for “there does not exist $p'$ such that $p \xrightarrow{a} p'$ and $p \xrightarrow{a}$ for all $a \in A$”. Furthermore, when $w \in A^*$, say $w = a_1 \ldots a_n$, then we write $p \xrightarrow{w} p'$ if there exist $p_0, \ldots, p_n$ such that $p = p_0, p_1 \xrightarrow{a_1} p_1 (1 \leq i \leq n)$ and $p_n = p'$. Also, we write $p \xrightarrow{w} p'$ for there exists $a \in A$ such that $p \xrightarrow{a} p'$. Similarly, we write $p \xrightarrow{w} p'$ for there exists $w \in A^*$ such that $p \xrightarrow{w} p'$ and say that $p'$ is reachable from $p$.

It is well-known that transition system specifications with negative premises may not define a unique transition relation that agrees with provability from the transition system specification [12–14]. Indeed, in [6] it was already pointed out that the transition system specification in Fig. 3 gives rise to such anomalies, e.g., if $\varDelta$ system specification \[12–14\]. Indeed, in [6] it was already pointed out that the transition system specification in Fig. 3 gives rise to such anomalies, e.g., if $\varDelta$ includes for $X$ the defining equation $X \overset{0}{\equiv} X : a.1 + 1$. For then, if $X \xrightarrow{\cdot}$, according to the rules for sequencing and recursion we find that $X \xrightarrow{\cdot} 1$, which is a contradiction. On the other hand, the transition $X \xrightarrow{\cdot} 1$ is not provable from the transition system specification.

We remedy the situation by restricting our attention to guarded recursive specifications, i.e., we require that every occurrence of a process identifier in the definition of some (possibly different) process identifier occurs within the scope of an action prefix. If $\varDelta$ is guarded, then it is straightforward to prove that the mapping $S$ from process expressions to natural numbers inductively defined by $S(1) = S(0) = S(a.p) = 0$, $S(p_1 + p_2) = S(p_1 + p_2) = S(p_1) + S(p_2) + 1$, and $S(X) = S(p)$ if $(X \overset{0}{\equiv} p) \in \varDelta$ gives rise to a so-called stratification $S'$ from transitions to natural numbers defined by $S'(p \xrightarrow{a} p') = S(p)$ for all $a \in A$ and process expressions $p$ and $p'$. In [12] it is proved that whenever such a stratification exists, then the transition system specification defines a unique transition relation that agrees with provability in the transition system specification.

The operational rules in Fig. 3 deviate from the operational rules for the process theory TSP($A$) discussed in [11] in only two ways: to get the rules for TSP($A$), the symbol ; should be replaced by $\cdot$, and the negative premise $p \xrightarrow{\cdot}$ should be removed from Rule 9. The replacement of ; by $\cdot$ is, of course, insignificant; the removal of the negative premise $p \xrightarrow{\cdot}$, however, does have a significant effect on the semantics of sequencing. The negative premise ensures that a sequencing can only proceed to execute its second argument when its first argument not only satisfies the acceptance predicate, but also cannot perform any further activity. The semantic difference between ; and $\cdot$ is illustrated in the following example.

**Example 1.** Consider the recursive specification

$$X \overset{0}{\equiv} a.(XY) + b.1 \quad Y \overset{0}{\equiv} c.1 + 1.$$  

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

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

**Definition 5.** A binary relation $R$ on the set of process expressions associated with TSP$^i(A, P)$ is a bisimulation iff $R$ is symmetric and for all closed terms $p$ and $q$ such that if $(p, q) \in R$:

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

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

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

**Proposition 1.** The relation $\leftrightarrow$ is a congruence for TSP$^i(A, P)$.

## 5 The correspondence

**Theorem 1.** For every one-state pushdown automaton there is a guarded sequential specification that is bisimilar to the process of the automaton.

**Proof.** Let $M = (\{s\}, A, D, \rightarrow, \uparrow, \downarrow)$ be a pushdown automaton with state $s$. We can assume $\rightarrow$ only has push and pop transitions. If there is no transition $s \xrightarrow{[e/d]} s$, then we can take either $X = 1$ or $X = 0$ as the resulting specification (in case $\downarrow = \{s\}$ resp. $\downarrow = \emptyset$). Otherwise, add a summand $a.X_d$ for each such transition to the equation of the initial variable $X$. Next, the equation for the added variable $X_d$ has a summand $a.1$ for each transition $s \xrightarrow{[d]} s$ and a summand $a.X_e ; X_d$ for each transition $s \xrightarrow{[d/\epsilon]} s$, apart from a summand $1$ or $0$, depending on whether $s \in \downarrow$ or not.
Example 2. The stack that is accepting in every state has pushdown automaton with state \( s \), data some finite set \( D \), actions \{\text{push}_d, \text{pop}_d \mid d \in D\}, \uparrow = s, \downarrow = \{s\} and transitions \( s \xrightarrow{\text{push}_d[d/e]} s \) and \( s \xrightarrow{\text{pop}_d[d/\epsilon]} s \) for each \( d \in D \) and transitions \( s \xrightarrow{\text{push}_e[d/e]} s \) for each \( d, e \in D \).

The recursive specification becomes

\[
X \overset{\text{def}}{=} 1 + \sum_{d \in D} \text{push}_d X_d \quad X_d \overset{\text{def}}{=} 1 + \text{pop}_d 1 + \sum_{e \in D} \text{push}_e X_e \quad X_d \quad (d \in D)
\]

Theorem 2. There is a pushdown automaton with two states, such that there is no guarded sequential specification that is bisimilar to its process.

Proof. Consider the example pushdown automaton in Figure 1. Suppose there is a finite guarded sequential specification bisimilar to the process of this automaton. Without loss of generality we can assume that this specification is in Greibach Normal Form (see \[7\]). As a consequence, each state of the transition system generated by the automaton corresponds to a sequence of variables of the specification. Take \( k \) some natural number, and consider the state \((\uparrow, 1^k)\) reached after executing \( k \) \text{a}-steps. From this state, consider any sequence of steps \( w \xrightarrow{\text{a}} \) where \( a \notin w \). Thus, \( w \) contains at most one \( c \), and at most \( k \) \text{b}'s.

In the transition system generated by the recursive specification, this same sequence of steps \( \xrightarrow{\xi_k} \) is possible from the sequence of variables \( \xi_k \) corresponding to state \((\uparrow, 1^k)\).

Let \( X_k \) be the first element of \( \xi_k \). From \( X_k \), we can also execute at most one \text{c}-step and \( k \) \text{b}-steps, without executing an \text{a}-step.

Since the recursive specification is finite, there must be a repetition in the sequence \( X_k \) \((k \geq 0)\). Thus, there are numbers \( n, m, n < m \), with \( X_n = X_m \). \( X_m \) can execute at most one \text{c} and \( n < m \) \text{b}'s without executing an \text{a}. But \( \xi_m \xrightarrow{b^m} \), so the additional \text{b}-steps must come from the second and following variables of the sequence. As the second variable is reached by just executing \text{b}'s, it must allow an initial \text{c}-step. Now we can consider \( \xi_m \xrightarrow{c b^m} \). This sequence of steps must also reach the second variable, but then, a second \text{c} can be executed, which is a contradiction.

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

We see that the contradiction is reached, because when we pass a sequencing operator, we do not know whether we are in the initial state or the final state of the pushdown automaton. We need a mechanism to pass this information along the sequencing.

Theorem 3. For every guarded sequential specification there is a pushdown automaton with a bisimilar process, with at most two states.

Proof. Let \( \Delta \) be a guarded sequential specification over \( P \). Wlog, we can assume \( \Delta \) is in Acceptance Irredundant Greibach Normal Form [8]. Every state of the specification is given by a sequence of variables that is acceptance irredundant. The corresponding pushdown automaton has two states \{\( n, t \}\}. The initial state is \( n \) if the initial variable \( S \not\in \) and \( t \) if the initial variable \( S \in \) (as defined by the operational semantics), the final state is \( t \).
For each summand \( a \cdot \chi \) of a variable \( X \) with \( X \downarrow \) and the first variable of \( \chi \) a variable with \( \downarrow \), add a step \( t \xrightarrow{a[X/\chi]} t \). Moreover, in case \( X \) is initial, a step \( t \xrightarrow{a[\epsilon/\chi]} t \).

- For each summand \( a \cdot \chi \) of a variable \( X \) with \( X \downarrow \) and the first variable of \( \chi \) a variable with \( \not\downarrow \), add a step \( n \xrightarrow{a[X/\chi]} n \). Moreover, in case \( X \) is initial, a step \( n \xrightarrow{a[\epsilon/\chi]} n \).

- For each summand \( a \cdot \chi \) of a variable \( X \) with \( X \not\downarrow \) and the first variable of \( \chi \) a variable with \( \downarrow \), add a step \( n \xrightarrow{a[X/\chi]} t \). Moreover, in case \( X \) is initial, a step \( n \xrightarrow{a[\epsilon/\chi]} n \).

- For each summand \( a \cdot \chi \) of a variable \( X \) with \( X \not\downarrow \) and the first variable of \( \chi \) a variable with \( \not\downarrow \), add a step \( n \xrightarrow{a[X/\chi]} n \). Moreover, in case \( X \) is initial, a step \( n \xrightarrow{a[\epsilon/\chi]} n \).

6 Signals and conditions

See [16].

**Definition 6.** First of all, we look at the data domain \( \mathcal{B} \) of the Booleans, with two constants true and false, operators \( \neg \) for negation and \( \lor, \land \) for or, and. We use a set \( P_1, \ldots, P_n \) as propositional variables. In this data type, we can make propositional formulas.

**Definition 7.** Next, we can define a conditional statement or guarded command. Given a propositional formula \( \phi \), we write \( \phi \rightarrow x \), with the intuitive meaning 'if \( \phi \) then \( x \)'. In order to give an operational semantics, it is important to note that it is needed to know the values of the propositional variables in order to decide on possible transitions. Moreover, values of propositional variables can change during the execution of a process. Thus, we need a valuation that in each state of a process assigns true or false to each propositional variable. Upon executing an action \( a \) in a state with valuation \( v \), a state with a possibly different valuation \( v' \) results. The resulting valuation \( v' \) is called the effect of the execution of \( a \) in a state with valuation \( v \).

We present operational rules for guarded command in Fig. 5. We define when a term in a certain valuation can take a step or be in a final state.

On the basis of these rules, we can define a notion of bisimulation. We use stateless bisimulation, which means that two transition systems \( T, T' \) are bisimilar iff there is a bisimulation relation that relates two states iff they are related under every possible valuation. Notice that \( \langle \text{false} : \rightarrow 1, v \rangle \) is not final for any valuation.

Next, we introduce an operator that allows to observe aspects of the current state of a process. The central assumption is that the visible part of the state of a process is a proposition, an expression over the booleans. We introduce the root-signal emission operator \( \bowtie \). A term \( \phi \bowtie x \) represents the process \( x \) that shows the signal \( \phi \) in its initial state. Special care must be taken in the sequel that no state can emit a signal that evaluates to false, we can never enter such a state. Such a state is inconsistent, so the
term \( a.(\text{false} \cdot x) \) can under no valuation execute action \( a \). See the operational rules in Fig. 6. To emphasise again the difference between guarded commands and root signal emission, term \( \text{false} \cdot (1 + a.1) \) is consistent and has, under any valuation, the same transition system as \( 0 \), whereas \( \text{false} \cdot (1 + a.1) \) is inconsistent, this state cannot be reached.

We again have a stateless bisimulation, where two terms are related iff they are related for any valuation that makes the root signal of the terms \( \text{true} \).

\[
\begin{array}{c}
\langle x, v \rangle \xrightarrow{a} \langle x', v' \rangle \quad \langle x, v \rangle \xrightarrow{\phi : \rightarrow x, v} \langle x', v' \rangle \\
\langle x + y, v \rangle \xrightarrow{a} \langle x', v' \rangle \quad \langle y + x, v \rangle \xrightarrow{a} \langle x', v' \rangle \\
\langle x, v \rangle \xrightarrow{a} \langle x', v' \rangle \quad \langle x, v \rangle \xrightarrow{\phi : \rightarrow x, v} \langle x', v' \rangle \\
\langle x, v \rangle \xrightarrow{\phi : \rightarrow x, v} \langle x', v' \rangle \\
\end{array}
\]

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

7 The full correspondence

**Theorem 4.** For every pushdown automaton there is a guarded recursive specification over TSP with signals and conditions that is bisimilar to the process of the automaton.

**Proof.** Let \( M = (S, A, D, \rightarrow, \uparrow, \downarrow) \) be a pushdown automaton. We can assume \( \rightarrow \) only has push and pop transitions. For every state \( s \in S \) we have a propositional variable \( \text{state}(s) \). We will arrange the recursive specification with an effect function that will ensure that this variable only evaluates to \( \text{true} \) in a situation corresponding to state \( s \).

We proceed to define the recursive specification with initial variable \( X \) and additional variables \( \{X_d \mid d \in D\} \).

- If \( M \) does not contain any transition of the form \( \uparrow^{a[e/d]} \), and the initial state is final, we can take \( X \overset{\text{def}}{=} 1 \) as the specification, and we do not need the additional variables.
– If $M$ does not contain any transition of the form $\uparrow^{[\epsilon/d]} s$, and the initial state is not final, we can take $X \overset{\text{def}}{=} 0$ as the specification, and we do not need the additional variables.

– Otherwise, there is some transition $\uparrow^{[\epsilon/d]} s$ in $M$. For each such transition, add a summand $a.(\text{state}(s) \ast X_d)$ to the equation of $X$. The effect of $a$ in any valuation results in a valuation $v$ with $v(\text{state}(s)) = \text{true}$ and $v(\text{state}(t)) = \text{false}$ for any $t \neq s$. Besides these summands, add a summand 1 iff the initial state of $M$ is final.

– Next, the equation for the added variable $X_d$ has a summand $\text{state}(s) : \rightarrow a.$ for each transition $s \overset{a[d/c]}{\rightarrow} t$, for every $s, t \in S$. The effect of $a$ in any valuation results in a valuation $v$ with $v(\text{state}(t)) = \text{true}$ and $v(\text{state}(u)) = \text{false}$ for any $u \neq t$.

In addition, the equation for the added variable $X_d$ has a summand $\text{state}(s) : \rightarrow a.$ for each transition $s \overset{a[d/c]}{\rightarrow} t$, for every $s, t \in S$. The effect of $a$ in any valuation results in a valuation $v$ with $v(\text{state}(t)) = \text{true}$ and $v(\text{state}(u)) = \text{false}$ for any $u \neq t$.

Finally, the equation for the added variable $X_d$ has summands $\text{state}(s) : \rightarrow 1$ (whenever $s \in \downset$) or $\text{state}(s) : \rightarrow 0$ (otherwise).

Example 3. For the pushdown automaton in Fig. 1, we can find the following guarded recursive specification:

$$S = a.(\text{state} \uparrow \ast A) + c.(\neg \text{state} \uparrow \ast 1)$$

$$A = \neg \text{state} : \rightarrow b.(\neg \text{state} \uparrow \ast 1) +$$

$$+ \text{state} : \rightarrow (a.(\text{state} \uparrow \ast A; A) + b.(\text{state} \uparrow \ast 1) + c.(\neg \text{state} \uparrow \ast A)).$$

The boolean $\text{state} \uparrow$ is true iff we are in the initial state.

Theorem 5. For every guarded recursive specification over TSP$^\ast (A)$ with signals and conditions there is a pushdown automaton with a bisimilar process.

Proof. Suppose a finite guarded recursive specification over TSP with signals and conditions is given. Consider the set of propositional variables $P_1, \ldots, P_n$ occurring in this specification. For each possible valuation $v : \{P_1, \ldots, P_n\} \rightarrow \{\text{true}, \text{false}\}$, we create two states in the pushdown automaton to be constructed, one that is final and one that is not. Then, we can follow the procedure in Theorem 3.

If we have a valuation, then all signals and conditions can be resolved, and we reduce the specification to one in TSP$^\ast (A)$.

8 Conclusion

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

We see that signals and conditions add expressive power to TSP, since a signal can be passed along the sequencing operator. If we go to the theory BCP, so without sequencing but with parallel composition, then we know from [16] 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 expressing power of BCP.

References