Pushdown Automata and Context-Free Grammars in Bisimulation Semantics

Jos Baeten

fellow CWI

emeritus ILLC UvA

LIRa, 16 February 2023

Pushdown Automata and Context-Free Grammars in Bisimulation Semantics

- with Cesare Carissimo (Master of Logic, UvA, now ETH) and Bas Luttik (Eindhoven University of Technology)
- arXiv:2203.01713
- preliminary version, CALCO 2021
- extended version, to appear in special issue in LMCS

What is a computation?

- Church-Turing Thesis
- Given by a Turing machine: input at begin, deterministic steps, output at end
- A computation is a function
- Models a deaf, dumb and blind computer (before advent of terminal)
- Far removed from a modern-day computer as students see all around them

CWI

Reactive Systems

"A Turing machine cannot drive a car, but a real computer can!"

Interaction

User interaction: not just initial, final word on the tape.

Make interaction between control and memory explicit.

Integration of automata theory and process theory.

Aim to develop course on foundations of computer science for all first-year computer science students.
Course in Master of Logic: Computability and Interaction

Reactive Turing Machine

Defined in I&C 2013 with Bas Luttik and Paul van Tilburg

Executability instead of computability, but not more expressive

Robustness: π-calculus (Robin Milner) is the λ-calculus with interaction (with Bas Luttik and Fei Yang)

Well-known theorem

A language can be defined by a pushdown automaton iff it can be defined by a context-free grammar.

CWI

Well-known theorem

A language can be defined by a pushdown automaton iff it can be defined by a context-free grammar.

A process can be defined by a pushdown automaton iff it can be defined by a finite guarded sequential recursive specification, with a notion of state awareness added.

Definition

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

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

Definition

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

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

A *process graph* is a non-deterministic automaton, possibly infinite.
A *process graph* is a labelled transition system with an initial state.

Bisimulation

$p \leftrightarroweq q$, p is *bisimilar* to q if there is a symmetric binary relation R with $p\ R\ q$ satisfying the following conditions:

1. whenever $s\ R\ t$ and $s \xrightarrow{a} s'$, there is t' such that $t \xrightarrow{a} t'$ and $s'\ R\ t'$; and
2. whenever $s\ R\ t$ and $s{\downarrow}$, then $t{\downarrow}$.

Pushdown Automaton

$$\begin{tikzcd}
 & {} \arrow[d] & & \\
 & \uparrow \arrow[loop left, "\begin{array}{c} a[\epsilon/1] \\ a[1/11] \\ b[1/\epsilon] \end{array}"] \arrow[rr, "\begin{array}{c} c[\epsilon/\epsilon] \\ c[1/1] \end{array}"] & & \downarrow \arrow[loop right, "{b[1/\epsilon]}"]
\end{tikzcd}$$

$$\begin{tikzcd}
{} \arrow[r] & (\uparrow, \epsilon) \arrow[r, bend left, "a"] \arrow[d, "c"] & (\uparrow, 1) \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] \arrow[d, "c"] & (\uparrow, 11) \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] \arrow[d, "c"] & \cdots \arrow[l, bend left, "b"] \\
 & (\downarrow, \epsilon) & (\downarrow, 1) \arrow[l, "b"] & (\downarrow, 11) \arrow[l, "b"] & \cdots \arrow[l, "b"]
\end{tikzcd}$$

Bounded branching.

Context-Free Processes

- Use SOS to give automata for syntax $\mathbf{0}, \mathbf{1}, a.,;,+$
- (Used this to tackle the theorem since CONCUR 2008)

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

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

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

Context-Free Processes

- Use SOS to give automata for syntax $\mathbf{0}, \mathbf{1}, a., ;, +$
- (with Bas Luttik and MSc student Astrid Belder)

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

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

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

The difference

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

$$\begin{tikzcd}
{} \arrow[r] & X \arrow[r, "a"] \arrow[d, "b"] & XY \arrow[r, "a"] \arrow[d, "b"] & XY^2 \arrow[r, dashed] \arrow[d, "b"] & XY^{n-1} \arrow[r, "a"] \arrow[d, "b"] & XY^n \arrow[r, dashed] \arrow[d, "b"] & {} \\
 & \mathbf{1} & Y \arrow[l, "c"'] & Y^2 \arrow[l, "c"'] \arrow[ll, bend left, "c"] & Y^{n-1} \arrow[l, dashed] \arrow[lll, bend left, "c"] \arrow[ll, bend left, "c"] & Y^n \arrow[l, "c"'] \arrow[ll, bend left, "c"] \arrow[lll, bend left, "c"] \arrow[llll, bend left, "c"] & {} \arrow[l, dashed]
\end{tikzcd}$$

Unbounded branching.

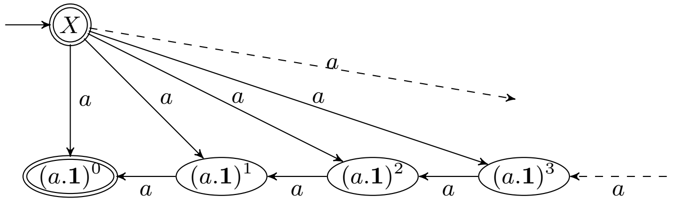
Recursion

$$\frac{p \xrightarrow{a} p' \quad (N = p) \in E}{N \xrightarrow{a} p'} \qquad \frac{p \downarrow \quad (N = p) \in E}{N \downarrow}$$

Limit to finite *guarded* recursive specifications.

Guardedness necessary

$$X \overset{\text{def}}{=} \mathbf{1} + X \,;\, a.\mathbf{1} \quad .$$



Infinitely branching.

Axiomatization: distributivity not valid

$$\begin{aligned}
x + y &= y + x \\
x + (y + z) &= (x + y) + z \\
x + x &= x \\
(x + y) \mathbin{;} z &= x \mathbin{;} z + y \mathbin{;} z \\
(x \mathbin{;} y) \mathbin{;} z &= x \mathbin{;} (y \mathbin{;} z) \\
x + \mathbf{0} &= x \\
\mathbf{0} \mathbin{;} x &= \mathbf{0} \\
x \mathbin{;} \mathbf{1} &= x \\
\mathbf{1} \mathbin{;} x &= x \\
(a.x) \mathbin{;} y &= a.(x \mathbin{;} y)
\end{aligned}$$

Axiomatization requires auxiliary operator

$$
\begin{array}{rcl}
NA(\mathbf{0}) & = & \mathbf{0} \\
NA(\mathbf{1}) & = & \mathbf{0} \\
NA(a.x) & = & a.x \\
NA(x+y) & = & NA(x)+NA(y) \\
\\
NA(x+y)\,;\,z & = & NA(x)\,;\,z+NA(y)\,;\,z \\
(a.x+y+\mathbf{1})\,;\,NA(z) & = & (a.x+y)\,;\,NA(z) \\
(a.x+y+\mathbf{1})\,;\,(z+\mathbf{1}) & = & (a.x+y)\,;\,(z+\mathbf{1})+\mathbf{1}
\end{array}
$$

Sound and Ground-Complete

Theorem: This axiomatization is sound and ground-complete.
Proof: By elimination of operators $;$ and NA.

Sound and Ground-Complete

Theorem: This axiomatization is sound and ground-complete.
Proof: By elimination of operators ; and NA.

Moreover, for guarded recursion we have a Head Normal Form
Theorem: Every process expression can be written in the form
$p = (\mathbf{1}+) \sum_{i=1}^{n} a_i.p_i$.

As a consequence, every guarded specification can be brought into Greibach Normal Form $X = (\mathbf{1}+) \sum_{i=1}^{n} a_i.\alpha_i$ (α_i a sequence of identifiers), and every state in the process graph is given by a sequence of identifiers.

Context-free Grammar

A recursive specification for the process of $\{a^n b^n \mid n \geq 0\}$ is

$$X = \mathbf{1} + a.Y$$

$$Y = b.\mathbf{1} + a.Y; b.\mathbf{1}$$

Context-free Grammar

A recursive specification for the process of $\{a^n b^n \mid n \geq 0\}$ is

$$X = \mathbf{1} + a.Y$$

$$Y = b.\mathbf{1} + a.Y\,;b.\mathbf{1}$$

A recursive specification for the always accepting stack is

$$S = \mathbf{1} + \sum_{d \in D} push(d).T_d\,;S$$

$$T_d = \mathbf{1} + pop(d).\mathbf{1} + \sum_{e \in D} push(e).T_e\,;T_d$$

Theorem 1

For every finite guarded sequential specification there is a pushdown automaton with the same process (with two non-bisimilar states).

Theorem 1

For every finite guarded sequential specification there is a pushdown automaton with the same process (with two non-bisimilar states).

Proof: carefully encode (intermediate) acceptance, remove redundant acceptance ($(a.\mathbf{1} + \mathbf{1})\ ;\ b.\mathbf{1} \leftrightarrows a.\mathbf{1}\ ;\ b.\mathbf{1}$). The stack contains sequences of identifiers of the specification, non-accepting identifiers followed by accepting identifiers.

Theorem 2

For every one-state pushdown automaton there is a finite guarded sequential specification with the same process.

Theorem 2

For every one-state pushdown automaton there is a finite guarded sequential specification with the same process.

Proof: Use an identifier for the initial state and an identifier for each data element. A step of the pushdown automaton corresponds to a summand of an equation in the specification.

Theorem 3

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

Pushdown Automaton

$$\begin{tikzcd}
{} \arrow[d] & \\
\uparrow \arrow[loop left, "{\begin{array}{c} a[\epsilon/1] \\ a[1/11] \\ b[1/\epsilon] \end{array}}"] \arrow[r, "{\begin{array}{c} c[\epsilon/\epsilon] \\ c[1/1] \end{array}}"] & \downarrow \arrow[loop right, "{b[1/\epsilon]}"]
\end{tikzcd}$$

$$\begin{tikzcd}
{} \arrow[r] & (\uparrow,\epsilon) \arrow[r, bend left, "a"] \arrow[d, "c"] & (\uparrow,1) \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] \arrow[d, "c"] & (\uparrow,11) \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] \arrow[d, "c"] & \cdots \arrow[l, bend left, "b"] \\
& (\downarrow,\epsilon) & (\downarrow,1) \arrow[l, "b"'] & (\downarrow,11) \arrow[l, "b"'] & \cdots \arrow[l, "b"']
\end{tikzcd}$$

Proof

Suppose not, there is a specification in Greibach normal form. There is a sequence of identifiers corresponding to each state $(\uparrow, 1^i)$. For k large enough, there is a repetition in the first elements of these sequences up to k, i.e. $X_n = X_m$ for $n < m \leq k$. X_n can execute at most one c and n b's before executing an a, so since the sequence starting with X_m can execute m consecutive b's, it must reach the second identifier, so this identifier can initially execute a c. But the sequence starting with X_m can also execute a c followed by m consecutive b's, so this must also reach the second identifier. Now a second c can be executed, and this is a contradiction.

Proof

Suppose not, there is a specification in Greibach normal form. There is a sequence of identifiers corresponding to each state $(\uparrow, 1^i)$. For k large enough, there is a repetition in the first elements of these sequences up to k, i.e. $X_n = X_m$ for $n < m \leq k$. X_n can execute at most one c and n b's before executing an a, so since the sequence starting with X_m can execute m consecutive b's, it must reach the second identifier, so this identifier can initially execute a c. But the sequence starting with X_m can also execute a c followed by m consecutive b's, so this must also reach the second identifier. Now a second c can be executed, and this is a contradiction.

We see the contradiction is reached, because when we reach the second identifier, we do not know in which state of the pushdown automaton we are, a top state or a bottom state.

Signals and conditions

- The visible part of the state of a process is a proposition, an expression in propositional logic
- Fix a Boolean algebra $\mathcal{B}$, with constants $true, false$, logical connectives, freely generated by generators $P_1, \ldots, P_n$ (propositional variables).
- $\phi :\rightarrow x$ is guarded command
- $\phi \,{}^{\wedge}\!\!\blacktriangle x$ is root signal emission
- Comes with a valuation in every state of the transition system: execution of a in state s with valuation v results in state t with valuation $v' = \mathit{effect}(a, v)$.
- Stateless bisimulation: in each state, consider again all possible valuations

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

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

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

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

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

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

Guarded Command

$$\begin{array}{lcl}
\mathit{true} :\rightarrow x & = & x \\
\mathit{false} :\rightarrow x & = & \mathbf{0} \\
(\phi \vee \psi) :\rightarrow x & = & (\phi :\rightarrow x) + (\psi :\rightarrow x) \\
(\phi \wedge \psi) :\rightarrow x & = & \phi :\rightarrow (\psi :\rightarrow x) \\
\phi :\rightarrow (x + y) & = & (\phi :\rightarrow x) + (\phi :\rightarrow y) \\
\phi :\rightarrow (x \,;\, y) & = & (\phi :\rightarrow x) \,;\, y \\
\phi :\rightarrow NA(x) & = & NA(\phi :\rightarrow x)
\end{array}$$

$$(NA(x) + \phi :\rightarrow \mathbf{1}) \,;\, (NA(y) + \psi :\rightarrow \mathbf{1}) =$$

$$= NA(x) \,;\, (NA(y) + \psi :\rightarrow \mathbf{1}) + (\phi \wedge \psi) :\rightarrow \mathbf{1}$$

No elimination.

Root Signal Emission

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

$$\frac{Cons(\langle p, v\rangle) \;\; Cons(\langle q, v\rangle)}{Cons(\langle p + q, v\rangle)}$$

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

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

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

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

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

$$\frac{\langle p, v \rangle \downarrow \quad \mathit{Cons}(\langle q, v \rangle)}{\langle p + q, v \rangle \downarrow \;\; \langle q + p, v \rangle \downarrow}$$

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

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

Axiomatization

$$
\begin{array}{lcl}
\mathit{true} \,{}^{\wedge\!\blacktriangle}\, x & = & x \\
\mathit{false} \,{}^{\wedge\!\blacktriangle}\, x & = & \mathit{false} \,{}^{\wedge\!\blacktriangle}\, \mathbf{0} \\
a.(\mathit{false} \,{}^{\wedge\!\blacktriangle}\, x) & = & \mathbf{0} \\
(\phi \,{}^{\wedge\!\blacktriangle}\, x) + y & = & \phi \,{}^{\wedge\!\blacktriangle}\, (x + y) \\
\phi \,{}^{\wedge\!\blacktriangle}\, (\psi \,{}^{\wedge\!\blacktriangle}\, x) & = & (\phi \wedge \psi) \,{}^{\wedge\!\blacktriangle}\, x \\
\phi :\rightarrow (\psi \,{}^{\wedge\!\blacktriangle}\, x) & = & (\neg\phi \vee \psi) \,{}^{\wedge\!\blacktriangle}\, (\phi :\rightarrow x) \\
\phi \,{}^{\wedge\!\blacktriangle}\, (\phi :\rightarrow x) & = & \phi \,{}^{\wedge\!\blacktriangle}\, x \\
\phi \,{}^{\wedge\!\blacktriangle}\, (x \, ; y) & = & (\phi \,{}^{\wedge\!\blacktriangle}\, x) \, ; y \\
\phi \,{}^{\wedge\!\blacktriangle}\, NA(x) & = & NA(\phi \,{}^{\wedge\!\blacktriangle}\, x)
\end{array}
$$

Head Normal Form

Can write every process expression in the form

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

(ψ the root signal, χ the acceptance condition)

Head Normal Form

Can write every process expression in the form

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

(ψ the root signal, χ the acceptance condition)
Reduce HNF: for some v, $v(\phi_i \wedge \psi) = \mathit{true}$ and
$\mathit{Cons}(\langle p_i, \mathit{effect}(a_i, v)\rangle))$.
Reset property: the *effect* makes every propositional variable *true*.

$$a.(\neg P_1 \vee \cdots \vee \neg P_k) \mathbin{{}^{\wedge}\!\blacktriangle} x = \mathbf{0}$$

Axiomatization

The axiomatization (including the reset axiom) is sound and ground-complete.

Axiomatization

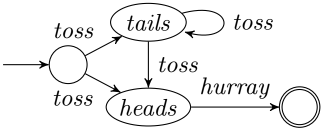
The axiomatization (including the reset axiom) is sound and ground-complete.

Give a semantics in terms of process graphs, not depending on valuations:
Suppose the root signal of t is not *false*. Define

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

Makes all undetermined guarded commands *false*.

Example: coin toss



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

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

Theorem 4

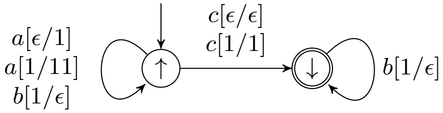
For every pushdown automaton there is a guarded sequential specification with signals and conditions with the same process.

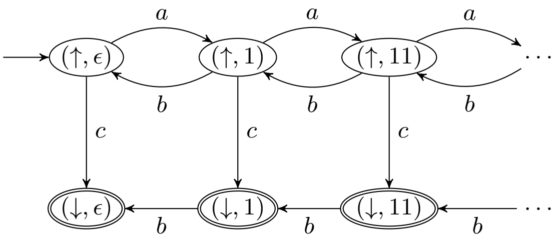
$$S = a.(state{\uparrow}\ {}^{\wedge}\!\!\blacktriangle A\ ;(state{\uparrow}{:}\!\to S + state{\downarrow}{:}\!\to \mathbf{1})) + c.(state{\downarrow}\,{}^{\wedge}\!\!\blacktriangle\mathbf{1})$$

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

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

Pushdown Automaton





Theorem 5

For every guarded sequential specification with signals and conditions there is a pushdown automaton with the same process.

Conclusion

Interaction is a key ingredient of any computer.
A model of computation needs to incorporate interaction.
Aim is a full integration of automata theory and process theory.
Result is a richer and more refined theory.
Turn lecture notes into a text book:
Models of Computation based on Automata: Formal Languages and Communicating Processes.

Current Work

A language can be defined by a parallel pushdown automaton iff it can be defined by a commutative context-free grammar.

Current Work

A language can be defined by a parallel pushdown automaton iff it can be defined by a commutative context-free grammar.

A process can be defined by a parallel pushdown automaton iff it can be defined by a finite guarded parallel recursive specification, with communication (including asymmetric communication) added.

Current Work

A language can be defined by a parallel pushdown automaton iff it can be defined by a commutative context-free grammar.

A process can be defined by a parallel pushdown automaton iff it can be defined by a finite guarded parallel recursive specification, with communication (including asymmetric communication) added.

Data structure of a bag instead of a stack, multisets instead of sequences.