Magazine van het ICT-onderzoek Platform Nederland (IPN)
Jaargang 8 / nummer 1 / februari 2011

I/O ICT-onderzoek

Technische Universiteit
Eindhoven
University of Technology

Computations and Interaction

Jos Baeten

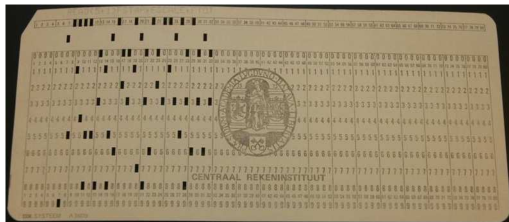
Systems Engineering (Dept. of Mech. Eng.) and
Theory of Computing (Dept. of Math. & Comp. Sci.)
–
Joint work with Bas Luttik and Paul van Tilburg

March 4, 2011

TU/e Technische Universiteit Eindhoven University of Technology

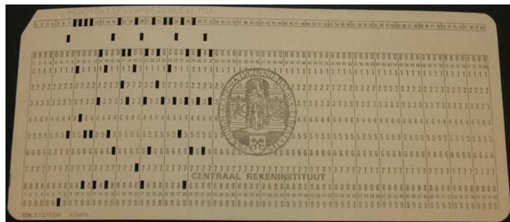
What is a computation?

My first computer program, 1973.
Hand in stack of punched cards at counter
Wait 2 hours
Find dump in pigeon hole



What is a computation?

My first computer program, 1973.
Hand in stack of punched cards at counter
Wait 2 hours
Find dump in pigeon hole



Modeled by a Turing machine.
Input separated from output
Fixed input string

Nowadays: interaction

Input: one click.
Immediate reaction from computer: reactive systems.
Not an input *string*

Nowadays: interaction

Input: one click.
Immediate reaction from computer: reactive systems.
Not an input *string*
One Google query has different answers all the time: non-determinism

Nowadays: interaction

Input: one click.
Immediate reaction from computer: reactive systems.
Not an input *string*
One Google query has different answers all the time: non-determinism
A Turing machine cannot fly an airplane, but a computer can.

Foundations of computing is automata theory and formal languages.

Foundations of computing is automata theory and formal languages.
Finite automaton, pushdown automaton, Turing machine: languages.

Foundations of computing is automata theory and formal languages.
Finite automaton, pushdown automaton, Turing machine: languages.
Interaction: from concurrency theory, process theory.

Foundations of computing is automata theory and formal languages.
Finite automaton, pushdown automaton, Turing machine: languages.
Interaction: from concurrency theory, process theory.
Parallel composition (Milner). Transition systems.

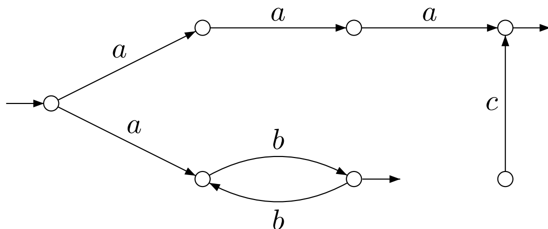
Related Work

Gurevich – Abstract State Machines (from early 90s)
Wegner and Goldin – Persistent Turing Machine (from 97)
Van Leeuwen and Wiedermann – interactive transducers (from 00)

Differences

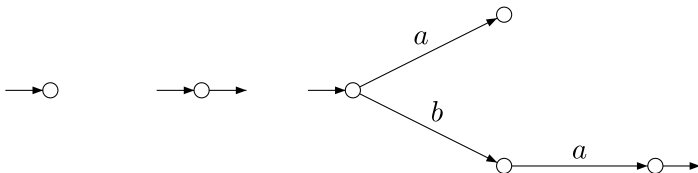
- Final state, termination, in concurrency
- Transition systems not necessarily finite
- Language equivalence cannot capture interaction.
  $a.b + a.c \neq a.(b + c)$ (lady and the tiger)
  Branching bisimulation.

Finite automaton



- Reachable nodes, unnamed nodes, layout
- Language equivalence throws away a lot of information.

An algebra, for calculation



$\mathbf{0}$ $\mathbf{1}$ $a.\mathbf{0} + b.a.\mathbf{1}$

Structural Operational Semantics

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

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

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

Structural Operational Semantics

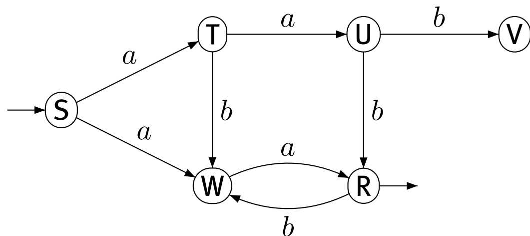
$$\frac{}{x^* \downarrow} \qquad \frac{x \xrightarrow{a} x'}{x^* \xrightarrow{a} x' \cdot x^*} \qquad \frac{x \downarrow \quad y \downarrow}{x \parallel y \downarrow}$$

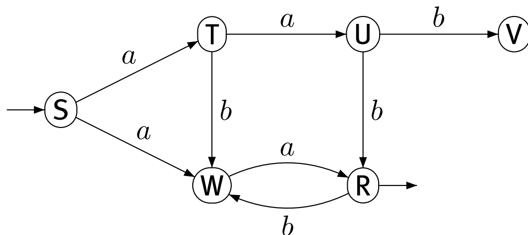
$$\frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y \quad \text{and v.v.}} \qquad \frac{x \xrightarrow{c!d} x' \quad y \xrightarrow{c?d} y'}{x \parallel y \xrightarrow{c!?d} x' \parallel y' \quad \text{and v.v.}}$$

$$\frac{x \xrightarrow{a} x' \quad a \neq c!d, c?d}{\partial_c(x) \xrightarrow{a} \partial_c(x')} \qquad \frac{x \downarrow}{\partial_c(x) \downarrow}$$

$$\frac{x \xrightarrow{c!?d} x'}{\tau_c(x) \xrightarrow{\tau} \tau_c(x')} \qquad \frac{x \xrightarrow{a} x' \quad a \neq c!?d}{\tau_c(x) \xrightarrow{a} \tau_c(x')} \qquad \frac{x \downarrow}{\tau_c(x) \downarrow}$$

Presentation of a Grammar





$$
\begin{aligned}
S &= a.T + a.W \\
T &= a.U + b.W \\
U &= b.V + b.R \\
V &= \mathbf{0} \\
W &= a.R \\
R &= b.W + \mathbf{1}
\end{aligned}
$$

- From automaton to recursive specification

From Recursive Specification to Automaton

$$\frac{t \xrightarrow{a} x \quad P = t}{P \xrightarrow{a} x} \qquad \frac{t\downarrow \quad P = t}{P\downarrow}$$

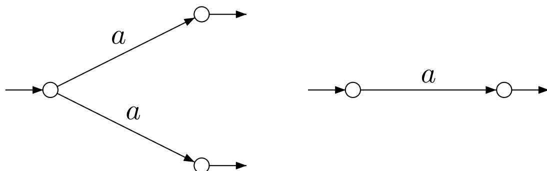
Algebraic Laws

$$\begin{aligned} x + y &\cong y + x \\ (x + y) + z &\cong x + (y + z) \\ x + x &\cong x \\ x + \mathbf{0} &\cong x \end{aligned}$$

$$\begin{aligned} ax + ay &\approx a(x + y) \\ a\mathbf{0} &\approx \mathbf{0} \end{aligned}$$

- First four laws: isomorphic automata. But not a congruence!
- Distributive law: language preserving, removing non-determinism
- $\mathbf{0}$ is right-zero: only successful termination counts

Definition of Bisimulation



$$a(\mathbf{1} + \mathbf{0}) + a\mathbf{1} \not\cong a\mathbf{1} + a\mathbf{1}$$

Bisimulation is the strongest congruence containing isomorphism.

Every congruence on automata containing isomorphism must contain bisimilarity.

Process Algebra

- $\mathbf{0}$ inaction, unsuccessful termination, deadlock
- $\mathbf{1}$ empty process, skip, successful termination
- $a._$ action prefix
- $_ + _$ alternative composition, choice
- $_ \cdot _$ sequential composition
- $_^*$ iteration, Kleene star
- $_ \parallel _$ parallel composition, merge, with communication
- $\partial_c(_)$ encapsulation
- $\tau_c(_)$ abstraction
- Recursive specifications, guarded
- Axiomatizations

B, Twan Basten, Michel Reniers, *Process Algebra*, CUP 2009/2010.

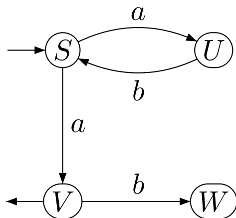
Process Algebra

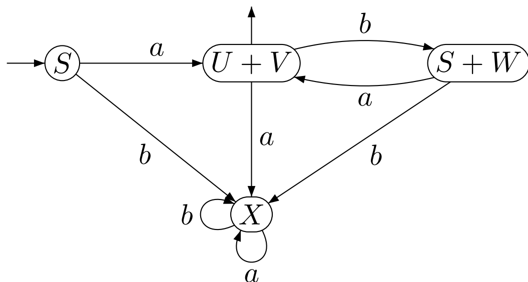
- $\mathbf{0}$ inaction, unsuccessful termination, deadlock
- $\mathbf{1}$ empty process, skip, successful termination
- $a._$ action prefix
- $_ + _$ alternative composition, choice
- $_ \cdot _$ sequential composition
- $_^*$ iteration, Kleene star
- $_ \parallel _$ parallel composition, merge, with communication
- $\partial_c(_)$ encapsulation
- $\tau_c(_)$ abstraction
- Recursive specifications, guarded
- Axiomatizations

B, Twan Basten, Michel Reniers, *Process Algebra*, CUP 2009/2010.
Supersedes CCS, CSP, ACP.

Removing non-determinism and totalizing

LNCS 5065, Montanari festschrift.



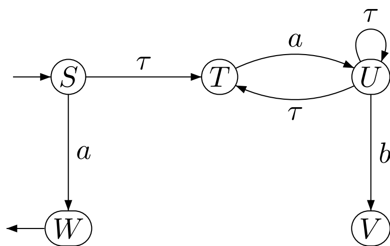


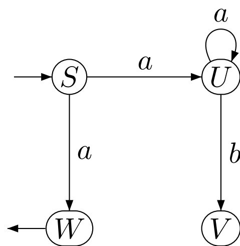
Removing non-determinism and totalizing

$$
\begin{array}{rcl}
S & = & aU + aV \\
U & = & bS \\
V & = & bW + \mathbf{1} \\
W & = & \mathbf{0}.
\end{array}
$$

$$
\begin{array}{rcl}
S & \approx & aU + aV \approx a(U + V) + bX \\
U + V & \approx & bS + bW + \mathbf{1} \approx b(S + W) + \mathbf{1} + aX \\
S + W & \approx & aU + aV + \mathbf{0} \approx a(U + V) + bX \\
X & = & aX + bX
\end{array}
$$

Removing silent steps



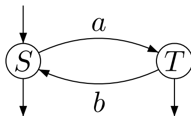


Removing silent steps

$$
\begin{aligned}
S &= aW + \tau T \\
T &= aU \\
U &= \tau U + bV + \tau T \\
V &= \mathbf{0} \\
W &= \mathbf{1}
\end{aligned}
$$

$$
\begin{aligned}
S &\approx aW + \tau T \approx aW + T \approx aW + aU \\
U &\approx \tau U + bV + \tau T \approx U + bV + T \approx bV + aU \\
V &= \mathbf{0} \\
W &= \mathbf{1}
\end{aligned}
$$

Regular expressions



$$S \approx aT + \mathbf{1} \approx a(bS + \mathbf{1}) + \mathbf{1} \approx abS + a\mathbf{1} + \mathbf{1} \approx (ab\mathbf{1})^* \cdot (a\mathbf{1} + \mathbf{1})$$

$$T \approx bS + \mathbf{1} \approx b(aT + \mathbf{1}) + \mathbf{1} \approx baT + b\mathbf{1} + \mathbf{1} \approx (ba\mathbf{1})^* \cdot (b\mathbf{1} + \mathbf{1})$$

Regular Process

A regular language is a language equivalence class of a finite (non-deterministic) automaton.

Regular Process

A regular language is a language equivalence class of a finite (non-deterministic) automaton.
A regular process is a bisimulation equivalence class of a finite, non-deterministic automaton.

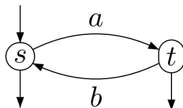
Regular Process

A regular language is a language equivalence class of a finite (non-deterministic) automaton.
A regular process is a bisimulation equivalence class of a finite, non-deterministic automaton.
A regular process is given by a recursive specification over the signature $\mathbf{0}, \mathbf{1}, a._, +$. So right-linear but not left-linear.

Regular Process

A regular language is a language equivalence class of a finite (non-deterministic) automaton.
A regular process is a bisimulation equivalence class of a finite, non-deterministic automaton.
A regular process is given by a recursive specification over the signature $\mathbf{0}, \mathbf{1}, a._, +$. So right-linear but not left-linear.
Processes given by deterministic automata, and by regular expressions, form a subclass. (B, Flavio Corradini, Clemens Grabmayer, JACM 2007.)

Regular expressions revisited



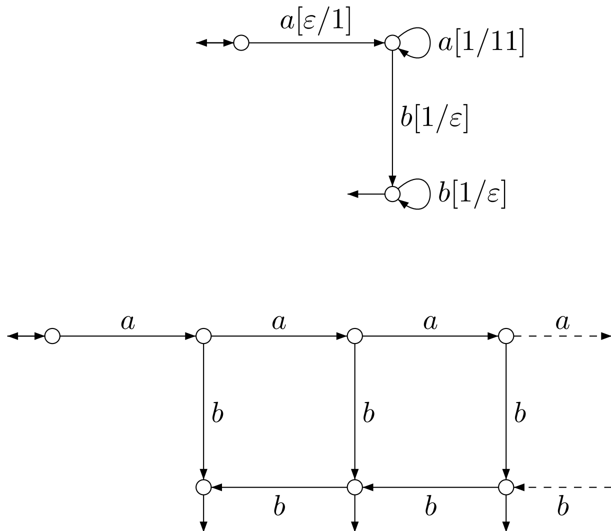
$$s = (ts?b.(st!a.\mathbf{1} + \mathbf{1}))^*$$

$$t = (st?a.(ts!b.\mathbf{1} + \mathbf{1}))^*$$

$$\partial_{st,ts}((st!a.\mathbf{1} + \mathbf{1}) \cdot s \parallel \mathbf{1} \cdot t)$$

New Kleene theorem: all finite automata can be given by reg.exp. plus $\parallel, \partial$.

Pushdown Process



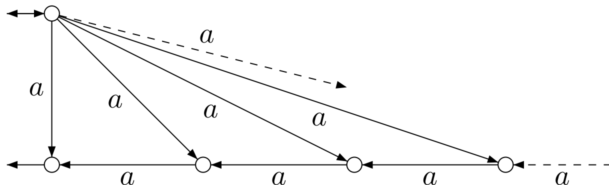
Stack, can only terminate when empty

$o!0$ $i?1$ $i?0$ $o!1$ $o!0$ $i?1$ $o!0$ $i?1$ $i?0$ $o!1$ $i?0$ $o!1$

$$Stack^{io} = \mathbf{1} + \sum_{d \in \mathcal{D}} i?d.Stack^{io} \cdot o!d.Stack^{io}$$

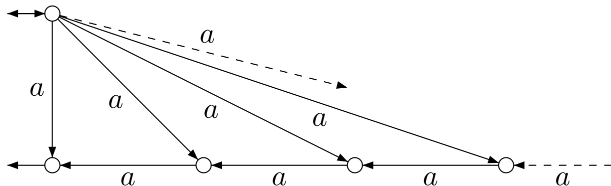
Infinite branching

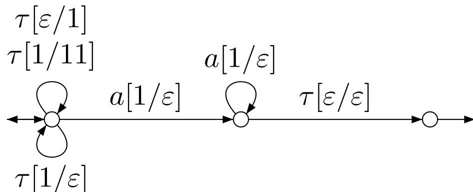
Example: $X = \mathbf{1} + X \cdot a\mathbf{1}$ has infinite branching. Has head recursion.



Infinite branching

Example: $X = \mathbf{1} + X \cdot a\mathbf{1}$ has infinite branching. Has head recursion.

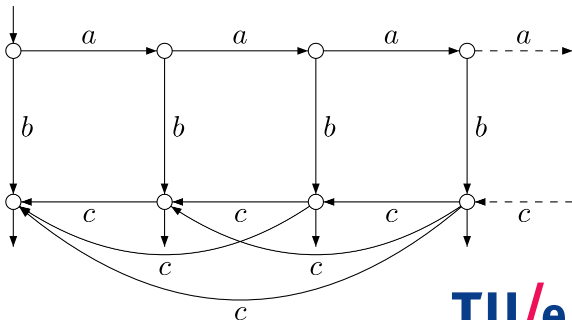




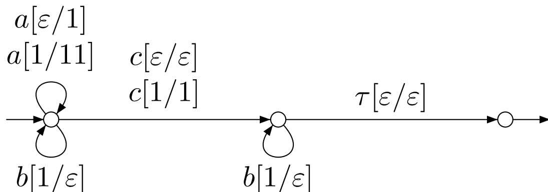
Unbounded branching

$$\begin{aligned} X &= aX \cdot Y + b\mathbf{1} \\ Y &= \mathbf{1} + c\mathbf{1} \end{aligned}$$

Variable Y is transparent.

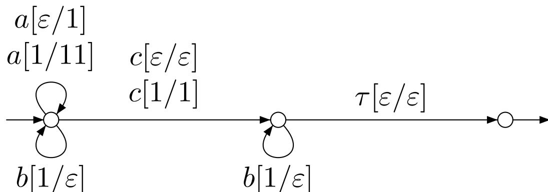


Pop choice



This push-down automaton has no sequential recursive specification.

Pop choice



This push-down automaton has no sequential recursive specification.
Using parallel composition, it has:

$$X = c.\mathbf{1} + a.(X \parallel b.\mathbf{1})$$

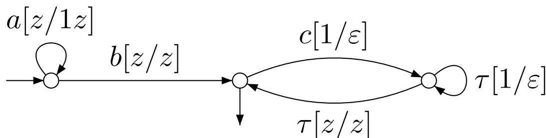
Theorem

A process is a popchoice-free pushdown process terminating only on empty
iff it is definable by a transparency-restricted sequential recursive specification without head recursion.

Removing restrictions

When there is head recursion, can still find a push-down automaton in some cases.
When there is transparency, we cannot.
However, can get rid of both restrictions when we move to *contra-simulation*.

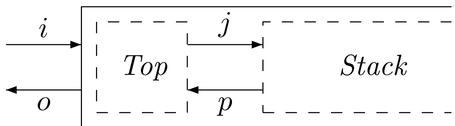
$$ax + ay = a(\tau x + \tau y)$$



B, Pieter Cuijpers, Paul van Tilburg, CONCUR 2008.

Find a different grammar

Stack = Top || Stack



$$Stack^{io} = \tau_{jp}(\partial_{jp}(Top^{io}_{jp}\emptyset \parallel Stack^{jp}))$$

Specification of always terminating stack

$$Stack^{io} = \mathbf{1} + o!\emptyset.Stack^{io} + \sum_{d \in \mathcal{D}} i?d.\tau_{jp}(\partial_{jp}(Top^{io}_{jp}d \parallel Stack^{jp}))$$

$$Top^{io}_{jp}d = \mathbf{1} + o!d. \sum_{f \in \mathcal{D} \cup \{\emptyset\}} p?f.Top^{io}_{jp}f + \sum_{e \in \mathcal{D}} i?e.j!d.Top^{io}_{jp}e$$

$$(d \in \mathcal{D})$$

$$Top^{io}_{jp}\emptyset = \mathbf{1} + o!\emptyset.Top^{io}_{jp}\emptyset + \sum_{e \in \mathcal{D}} i?e.Top^{io}_{jp}e$$

Theorem

For every pushdown automaton M there is a regular process p and for every regular process p there is an pushdown automaton M such that the transition system of M is branching bisimilar with the transition system of

$$\tau_{i,o}(\partial_{i,o}(p \parallel Stack^{io}))$$

Theorem

For every pushdown automaton M there is a regular process p and for every regular process p there is an pushdown automaton M such that the transition system of M is branching bisimilar with the transition system of

$$\tau_{i,o}(\partial_{i,o}(p \parallel Stack^{io}))$$

The (always terminating) stack is the prototypical pushdown process. Constitutes a grammar for pushdown processes.

Basic Parallel Process

A parallel pushdown automaton gives a parallel pushdown process.
A process p is parallel pushdown iff there is a regular process q with

$$p \leftrightarrows_{\mathsf{b}} \tau_{i,o}(\partial_{i,o}(q \parallel Bag))$$

Bag is given by

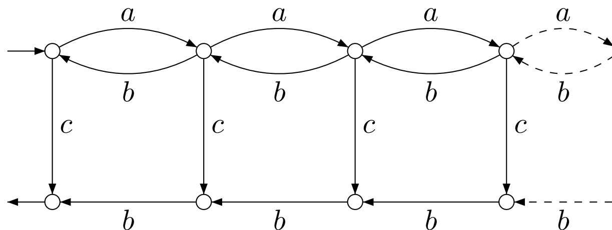
$$Bag = \mathbf{1} + \sum_{d \in \mathcal{D}} i?d.(Bag \parallel o!d.\mathbf{1}).$$

A basic parallel process is given by a rec.spec. over $\mathbf{1}, \mathbf{0}, +, a_, \parallel$ (only interleaving)
Any basic parallel process is a parallel pushdown process.

Example

$$X = c.\mathbf{1} + a.(X \parallel b.\mathbf{1})$$

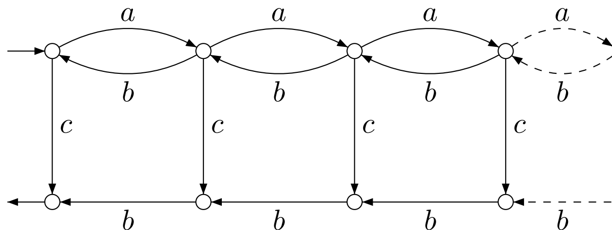
is basic parallel, parallel pushdown and pushdown but not sequential.



Example

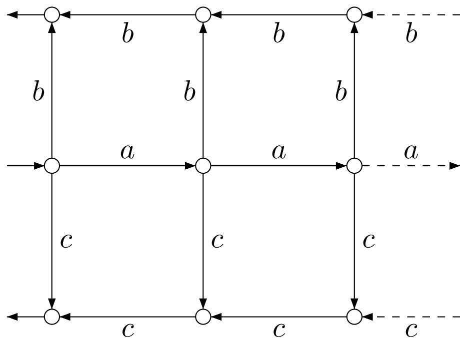
$$X = c.\mathbf{1} + a.(X \parallel b.\mathbf{1})$$

is basic parallel, parallel pushdown and pushdown but not sequential.



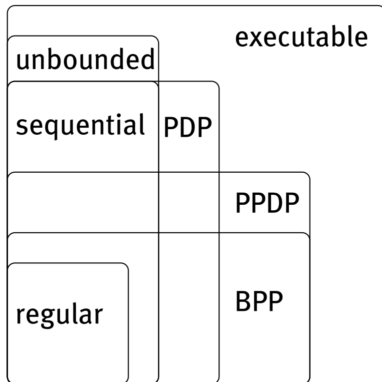
The bag is basic parallel, parallel pushdown but not pushdown, not sequential.
The stack is sequential and pushdown but not basic parallel, parallel pushdown.

Example



Pushdown and parallel pushdown process that is not basic parallel or sequential.
Can be adapted to become pushdown and not parallel pushdown, or parallel pushdown and not pushdown.

Process Classes



Reactive Turing Machine

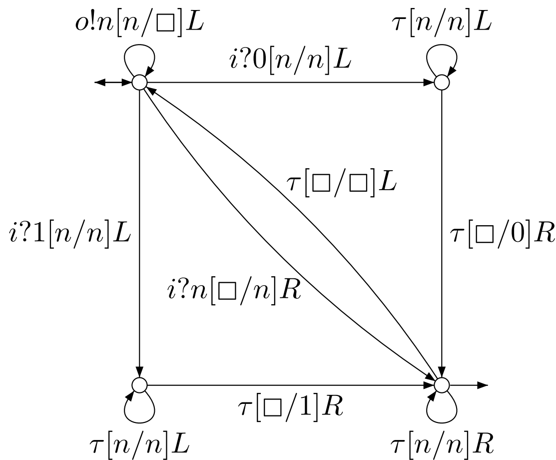
$a[\square/1]R$ $\tau[\square/\square]L$ $b[1/\square]L$ $\tau[\square/\square]R$

a a a a

τ τ τ τ

τ b b b

Queue that can always terminate



Executable Process

An executable process is the branching bisimulation equivalence class of a transition system of a Reactive Turing Machine.

Executable Process

An executable process is the branching bisimulation equivalence class of a transition system of a Reactive Turing Machine.
The queue is executable but not push-down.

Executable Process

An executable process is the branching bisimulation equivalence class of a transition system of a Reactive Turing Machine.

The queue is executable but not push-down.

A transition system is computable iff it is finitely branching and (with some coding) the set of final states is decidable and for each state, we can determine the set of outgoing transitions.

Executable Process

An executable process is the branching bisimulation equivalence class of a transition system of a Reactive Turing Machine.
The queue is executable but not push-down.
A transition system is computable iff it is finitely branching and (with some coding) the set of final states is decidable and for each state, we can determine the set of outgoing transitions.
A transition system is effective if its set of transitions and set of final states are recursively enumerable.

Results

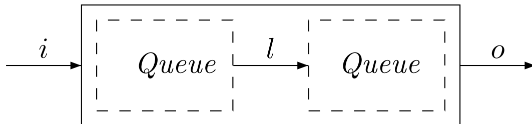
The transition system defined by a Reactive Turing Machine is computable.

Every effective transition system is branching bisimilar with a transition system of a Reactive Turing Machine.

For every RTM M there is a regular process p and for every regular process p there is an RTM M such that the transition system of M is branching bisimilar with the transition system of

$$\tau_{i,o}(\partial_{i,o}(p \parallel Queue^{io}))$$

One queue is two chained queues



Specification of the queue

$$Queue^{io} = \mathbf{1} + \sum_{d \in \mathcal{D}} i?d.\tau_l(\partial_l(Queue^{il} \parallel (\mathbf{1} + o!d.Queue^{lo})))$$

$$Queue^{il} = \mathbf{1} + \sum_{d \in \mathcal{D}} i?d.\tau_o(\partial_o(Queue^{io} \parallel (\mathbf{1} + l!d.Queue^{ol})))$$

$$Queue^{lo} = \mathbf{1} + \sum_{d \in \mathcal{D}} l?d.\tau_i(\partial_i(Queue^{li} \parallel (\mathbf{1} + o!d.Queue^{io})))$$

$$Queue^{ol} = \mathbf{1} + \sum_{d \in \mathcal{D}} o?d.\tau_i(\partial_i(Queue^{oi} \parallel (\mathbf{1} + l!d.Queue^{il})))$$

$$Queue^{li} = \mathbf{1} + \sum_{d \in \mathcal{D}} l?d.\tau_o(\partial_o(Queue^{lo} \parallel (\mathbf{1} + i!d.Queue^{oi})))$$

$$Queue^{oi} = \mathbf{1} + \sum_{d \in \mathcal{D}} o?d.\tau_l(\partial_l(Queue^{ol} \parallel (\mathbf{1} + i!d.Queue^{li})))$$

Conclusion

Integration of automata theory and process theory is beneficial for both theories.

Conclusion

Integration of automata theory and process theory is beneficial for both theories.

This integrated theory can be a first-year course in any academic bachelor program in computer science (or related subjects).

Conclusion

Integration of automata theory and process theory is beneficial for both theories.
This integrated theory can be a first-year course in any academic bachelor program in computer science (or related subjects).
Syllabus available.

Conclusion

Integration of automata theory and process theory is beneficial for both theories.
This integrated theory can be a first-year course in any academic bachelor program in computer science (or related subjects).
Syllabus available.
This provides a basic course in theoretical computer science as always, but at the same time gives a foundation for later courses on formal methods and model checking.

Conclusion

Integration of automata theory and process theory is beneficial for both theories.
This integrated theory can be a first-year course in any academic bachelor program in computer science (or related subjects).
Syllabus available.
This provides a basic course in theoretical computer science as always, but at the same time gives a foundation for later courses on formal methods and model checking.
We defined a grammar for all executable processes.

Definition

Informatics is the science and engineering of
discrete behavior of
interacting information processing agents.