Executability Theory

Jos Baeten

(joint work with Bas Luttik and Paul van Tilburg)

Centrum Wiskunde & Informatica, Amsterdam, and
Computer Science, Eindhoven University of Technology

OPCT, Bertinoro June 18, 2014

What is computer science?

Computer science is no more about computers than astronomy is about telescopes.

Informatics = Information + Computation

Information

Shannon information theory (1948)

Kolmogorov information theory (1963)

Quantum information theory (1998)

Computation

Church-Turing computation theory (1936)

Executability theory (2007)

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 computer of the ’70s (program, CPU, RAM)
- Criticism possible on suitability as a theoretical model of a modern-day computer

CWI

Reactive Systems

"A Turing machine cannot fly a plane, but a real computer can!"

Interaction

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

Make interaction between control and memory explicit.

> *. . . a theory of concurrency and interaction requires a new conceptual framework, not just a refinement of what we find natural for sequential computing.*

Robin Milner, Turing Award Lecture, 1993

Transition Systems vs. Automata

- Non-termination and termination both important
- Infinitely many states and transitions possible
- Language equivalence too coarse for interaction
- Divergence-preserving branching bisimulation

Executability Theory

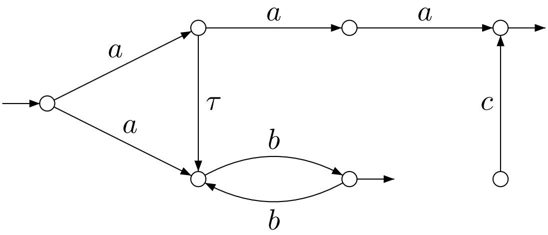
Computability + Concurrency

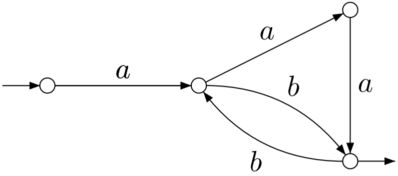
Real integration, aim is not to increase the computational power of the traditional model nor to investigate the extra expressivity of interaction

Regular languages, processes

- A *regular language* is a language equivalence class of transition systems containing a finite one
- A *regular process* is a divergence-preserving branching bisimilarity class of transition systems containing a finite one

Finite automata





Regular Expressions

- Milner 1984: not every regular process given by a regular expression

- Use SOS to give automata for all regular expressions $(\mathbf{0}, \mathbf{1}, a., \tau., \cdot, *, +)$

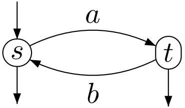
Structural Operational Semantics

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

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

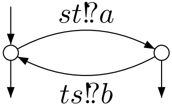
$$\frac{p \xrightarrow{a} p'}{p \cdot q \xrightarrow{a} p' \cdot q} \qquad \frac{p\downarrow \quad q \xrightarrow{a} q'}{p \cdot q \xrightarrow{a} q'} \qquad \frac{p\downarrow \quad q\downarrow}{p \cdot q\downarrow} \qquad \frac{p \xrightarrow{a} p'}{p^* \xrightarrow{a} p' \cdot p^*}$$

Regular Expressions



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

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



Structural Operational Semantics

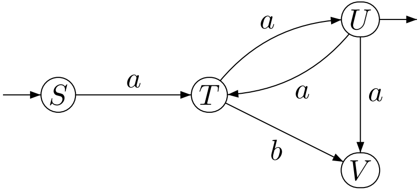
$$\frac{p \xrightarrow{a} p'}{p \parallel q \xrightarrow{a} p' \parallel q} \qquad \frac{q \xrightarrow{a} q'}{p \parallel q \xrightarrow{a} p \parallel q'} \qquad \frac{p \downarrow \quad q \downarrow}{p \parallel q \downarrow}$$

$$\frac{p \xrightarrow{c!d} p' \quad q \xrightarrow{c?d} q'}{p \parallel q \xrightarrow{c!?d} p' \parallel q'} \qquad \frac{p \xrightarrow{c?d} p' \quad q \xrightarrow{c!d} q'}{p \parallel q \xrightarrow{c!?d} p' \parallel q'}$$

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

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

Regular Grammar



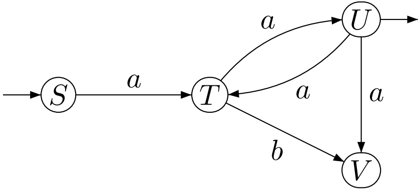
$$S = a.T$$

$$T = a.U + b.V$$

$$U = a.T + a.V + \mathbf{1}$$

$$V = \mathbf{0}$$

Regular Grammar



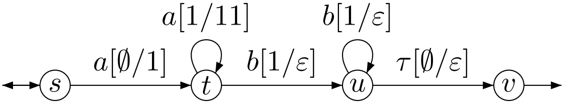
$$S = a.T$$
$$T = a.U + b.V$$
$$U = a.T + a.V + \mathbf{1}$$
$$V = \mathbf{0}$$

Only works with action prefix, not with action postfix

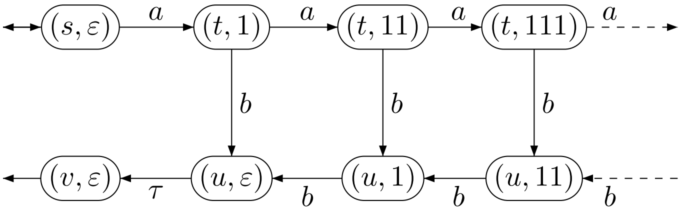
Structural Operational Semantics

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

Pushdown Automaton



Language $\{a^n b^n \mid n \geq 0\}$



Pushdown language, process

- Use acceptance by final state, not by empty stack
- A *pushdown language* is a language equivalence class of transition systems containing one of a pushdown automaton
- A *pushdown process* is a divergence-preserving branching bisimilarity class of transition systems containing one of a pushdown automaton

Context-free Grammar

A recursive specification for the example pushdown process is

$$X = \mathbf{1} + a.X \cdot b.\mathbf{1}$$

Context-free Grammar

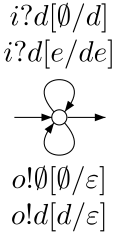
A recursive specification for the example pushdown process is

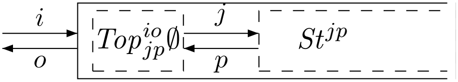
$$X = \mathbf{1} + a.X \cdot b.\mathbf{1}$$

Several problems occur concerning the relation with pushdown processes

Grammar for Pushdown Processes

The Stack St^{io}





$$St^{io} \underline{\leftrightarrow}_{\mathsf{b}}^{\Delta} \tau_{jp}(\partial_{jp}(Top^{io}_{jp}\emptyset \parallel St^{jp}))$$

Specification of the Stack

$$
\begin{aligned}
St^{io} &= \mathbf{1} + o!\emptyset.St^{io} + \sum_{d \in \mathcal{D}} i?d.\tau_{jp}(\partial_{jp}(Top^{io}_{jp}d \parallel St^{jp})) \\
Top^{io}_{jp}\emptyset &= \mathbf{1} + o!\emptyset.Top^{io}_{jp}\emptyset + \sum_{d \in \mathcal{D}} i?d.Top^{io}_{jp}d \\
Top^{io}_{jp}d &= \mathbf{1} + o!d.(p?\emptyset.Top^{io}_{jp}\emptyset + \sum_{e \in \mathcal{D}} p?e.Top^{io}_{jp}e) + \\
&\qquad \sum_{f \in \mathcal{D}} i?f.j!d.Top^{io}_{jp}f
\end{aligned}
$$

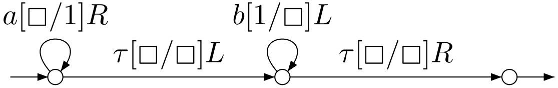
Grammar for Pushdown Processes

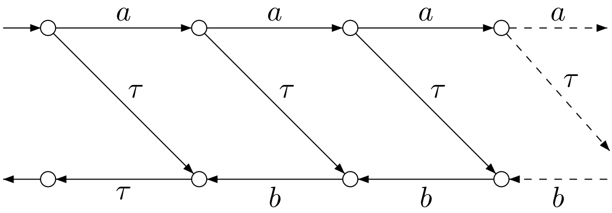
Every pushdown process can be written as a regular process communicating with a stack, so in the form

$$\tau_{io}(\partial_{io}(p \parallel St^{io}))$$

and vice versa

Reactive Turing Machine

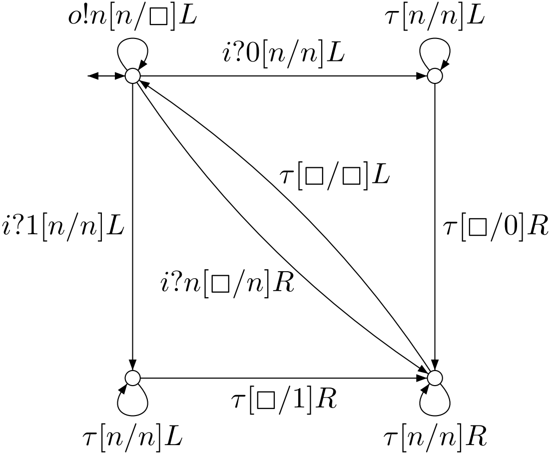




Executable language, process

- A *executable language* is a language equivalence class of transition systems containing one of an RTM
- A *executable process* is a divergence-preserving branching bisimilarity class of transition systems containing one of an RTM
- For language, function: executable = computable

Reactive Turing Machine: Queue



Conclusion

- Executability = Computability + Concurrency
- Unified framework for computation and interaction
- Undergraduate course in computer science:
  Models of Computation: Automata, Formal Languages and Communicating Processes