Models of Computation: Automata and Processes

An Overview

Paul van Tilburg

(joint work with Jos Baeten, Bas Luttik and Pieter Cuijpers)

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

July 14, 2009

Introduction

Automata & Formal Language theory

- *Back in the days:* different model and real-world computers
- Fixed input string
- Input separated from output
- Batch process

Introduction

Automata & Formal Language theory

- *Back in the days:* different model and real-world computers
- Fixed input string
- Input separated from output
- Batch process

- *Nowadays:* one click as input
- Computers are reactive systems
- Interaction much more important

Introduction

Automata & Formal Language theory

- *Back in the days:* different model and real-world computers
- Fixed input string
- Input separated from output
- Batch process

- *Nowadays:* one click as input
- Computers are reactive systems
- Interaction much more important

- *Note:* Provides very useful models of computation

Technische Universiteit
Eindhoven
University of Technology

Introduction (2)

Process theory

- Split off, separate development
- Focuses on interaction
- Deals with concurrent setting

Integration

- Attempt reveals differences and similarities
- Use *analogies* to make the integration explicit
- Increase understanding of both theories

Introduction (2)

Process theory

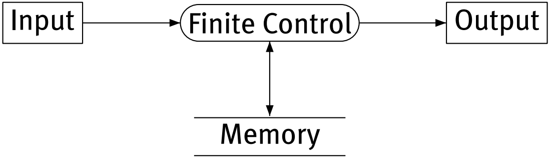
- Split off, separate development
- Focuses on interaction
- Deals with concurrent setting

Integration

- Attempt reveals differences and similarities
- Use *analogies* to make the integration explicit
- Increase understanding of both theories

- *Practical side:* merge in undergraduate curriculum course

Turing machine



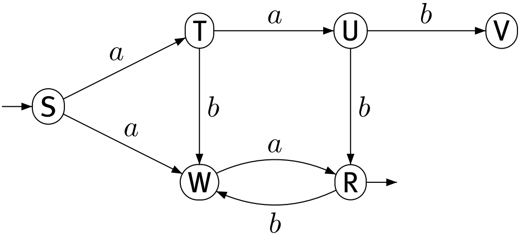
- Control is discrete: states and transitions: automaton
- Input, output: string or word over alphabet
- Alphabet: action, instruction, information

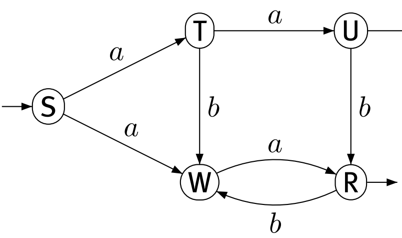
Finite Automaton

- Corresponds to regular language
- No memory!
- Two equivalences: language equivalence and isomorphism

TU/e Technische Universiteit Eindhoven University of Technology

Grammars and Recursive Specifications





$$
\begin{aligned}
S &= aT + aW \\
T &= aU + bW \\
U &= bV + bR \\
V &= \mathbf{0} \\
W &= aR \\
R &= bW + \mathbf{1}
\end{aligned}
$$

- From Finite Automaton to recursive specification

From Recursive Specification to Automaton

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

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

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

- Structural Operational Semantics [Plotkin, *JLAP*, 2004]

Similarities with Process Algebra

- Finite Automaton = finite labelled transition system
- Grammar = recursive specification over $\mathbf{0}, \mathbf{1}, +, \cdot, a$
- Regular expression = closed term over $\mathbf{0}, \mathbf{1}, +, \cdot, a, {}^*$

Technische Universiteit Eindhoven University of Technology

Similarities with Process Algebra

- Finite Automaton = finite labelled transition system
- Grammar = recursive specification over $\mathbf{0}, \mathbf{1}, +, \cdot, a$
- Regular expression = closed term over $\mathbf{0}, \mathbf{1}, +, \cdot, a, {}^*$

Basic Process Algebra

- $\mathbf{0}$ inaction, unsuccessful termination, deadlock
- $\mathbf{1}$ empty process, skip, successful termination
- $a\_$ action prefix
- $\_ + \_$ alternative composition, choice
- $\_ \cdot \_$ sequential composition

[Baeten, Basten, Reniers, *Process Algebra*, Cambridge UP, 2009]

Bisimulation

- In process theory a difference equivalent is used
- Expose interaction and preserve choices

Definition

We call the largest symmetric relation R such that

- if $p \xrightarrow{a} p'$ then there exists q' such that $q \xrightarrow{a} q'$ and $p' \; R \; q'$
- if $q \xrightarrow{a} q'$ then there exists p' such that $p \xrightarrow{a} p'$ and $p' \; R \; q'$
- if $p\downarrow$ implies $q\downarrow$ and vice versa

a *bisimulation* relation

- If $(p, q) \in R$, then p and q are *bisimilar* (notation: $p \leftrightarrows q$)

Technische Universiteit Eindhoven University of Technology

Definition

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

Technische Universiteit Eindhoven University of Technology

Regular Process

Definition

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

Definition

A *regular process* is a bisimulation equivalence class of a finite, non-deterministic automaton

Definition

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

Definition

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 $\mathbf{0}, \mathbf{1}, a, +$

Regular Process

Definition

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

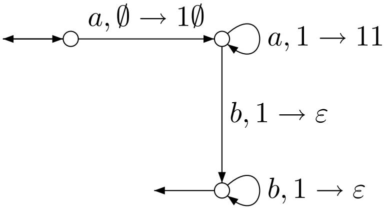
Definition

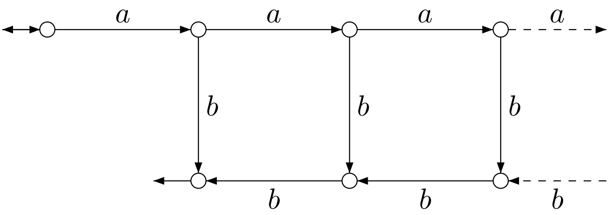
A *regular process* is a bisimulation equivalence class of a finite, non-deterministic automaton

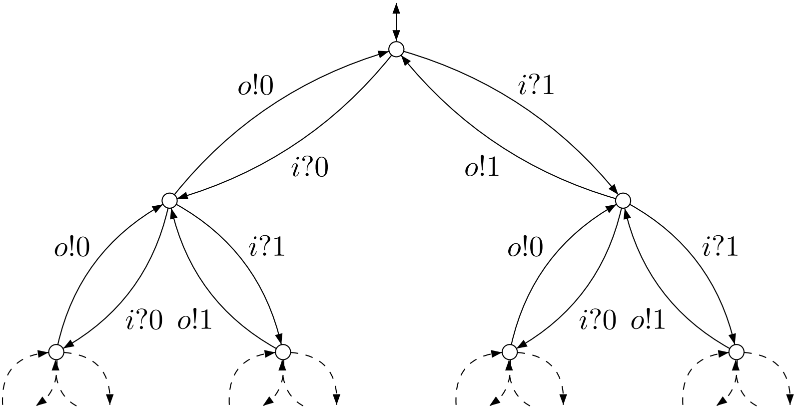
- A regular process is given by a recursive specification over the signature $\mathbf{0}, \mathbf{1}, a, +$
- Processes given by deterministic automata, and by regular expressions, form a subclass
  [Baeten, Corradini, Grabmayer, *JACM* 2007]

Technische Universiteit Eindhoven University of Technology

Pushdown Automaton







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

Pushdown Process

Theorem

A process p is a pushdown process iff there is a regular process q with

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

- Proof in [Baeten, Cuijpers, Luttik, van Tilburg, *FSEN*, 2009]

Pushdown Process

Theorem

A process p is a pushdown process iff there is a regular process q with

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

- Proof in [Baeten, Cuijpers, Luttik, van Tilburg, *FSEN*, 2009]

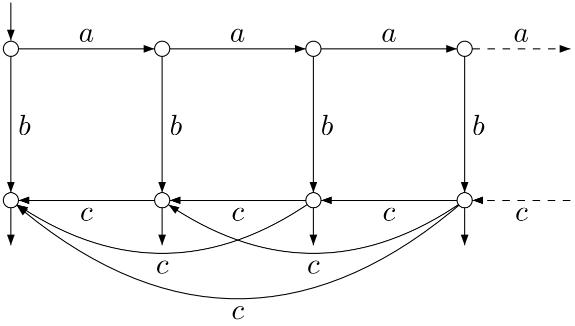
Recursive specification

Every recursive specification over $\mathrm{BPA}_{\mathbf{0},\mathbf{1}}$ with bounded branching denotes a pushdown process

- Example: $X = \mathbf{1} + aX \cdot b\mathbf{1}$

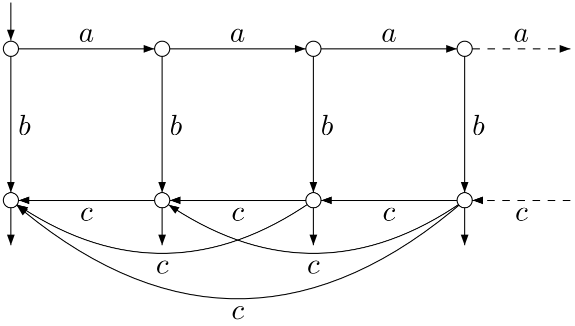
$$X \xrightarrow{a} X \cdot b\mathbf{1} \xrightarrow{a} X \cdot b\mathbf{1} \cdot b\mathbf{1} \ldots$$

Problem with 1-summands



$$X = aX \cdot Y + b\mathbf{1}$$
$$Y = \mathbf{1} + c\mathbf{1}$$

Problem with 1-summands



$$X = aX \cdot Y + b\mathbf{1}$$
$$Y = \mathbf{1} + c\mathbf{1}$$

- Recursive specifications over $\text{BPA}_{\mathbf{0},\mathbf{1}}$ can lead to unboundedness
- Cannot be done by our pushdown process due to stack and finite control
- Can be solved using a *forgetful stack* [Baeten, Cuijpers, van Tilburg, *CONCUR*, 2008]

What About Context-Free Processes?

- Context-free languages correspond to language accepted by PDAs
- Not the case with bisimulation! [Moller, 1996]
- *Fix:* do not allow for *pop choice* (to ensure existence specification)

What About Context-Free Processes?

- Context-free languages correspond to language accepted by PDAs
- Not the case with bisimulation! [Moller, 1996]
- *Fix:* do not allow for *pop choice* (to ensure existence specification)

- Recursive specification over $\text{BPA}_{\mathbf{0},\mathbf{1}}$ can lead to unbounded branching
- *Fix:* transparency-restricted Greibach normal form

What About Context-Free Processes?

- Context-free languages correspond to language accepted by PDAs
- Not the case with bisimulation! [Moller, 1996]
- *Fix:* do not allow for *pop choice* (to ensure existence specification)

- Recursive specification over $\mathrm{BPA}_{\mathbf{0},\mathbf{1}}$ can lead to unbounded branching
- *Fix:* transparency-restricted Greibach normal form

Theorem

A process is a pop choice-free pushdown process iff it is definable by a transparency-restricted recursive specification [FSEN, 2009]

Technische Universiteit Eindhoven University of Technology

What About Context-Free Processes?

- Context-free languages correspond to language accepted by PDAs
- Not the case with bisimulation! [Moller, 1996]
- *Fix:* do not allow for *pop choice* (to ensure existence specification)

- Recursive specification over $\mathrm{BPA}_{\mathbf{0},\mathbf{1}}$ can lead to unbounded branching
- *Fix:* transparency-restricted Greibach normal form

Theorem

A process is a pop choice-free pushdown process iff it is definable by a transparency-restricted recursive specification [FSEN, 2009]

- Not every pushdown process is context-free
- Decidability of bisimulation shown for this class!

Definition

A parallel pushdown automaton gives a parallel pushdown process

Technische Universiteit
Eindhoven
University of Technology

Basic Parallel Process

Definition

A parallel pushdown automaton gives a parallel pushdown process

Theorem

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

$$p \leftrightarroweq_b \tau_{i,o}(\partial_{i,o}(q \parallel B))$$

where B is the bag: $B = \mathbf{1} + \sum_{d \in \mathcal{D}} i?d.(B \parallel o!d.\mathbf{1})$

[Baeten, Cuijpers, van Tilburg, EXPRESS, 2008]

Basic Parallel Process

Definition

A parallel pushdown automaton gives a parallel pushdown process

Theorem

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

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

where B is the bag: $B = \mathbf{1} + \sum_{d \in \mathcal{D}} i?d.(B \parallel o!d.\mathbf{1})$

[Baeten, Cuijpers, van Tilburg, EXPRESS, 2008]

Definition

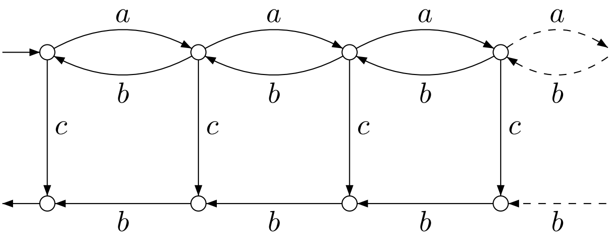
A *basic parallel process* is given by a guarded recursive specification over the signature $\mathbf{0}, \mathbf{1}, +, a\_, \parallel$

- 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 context-free

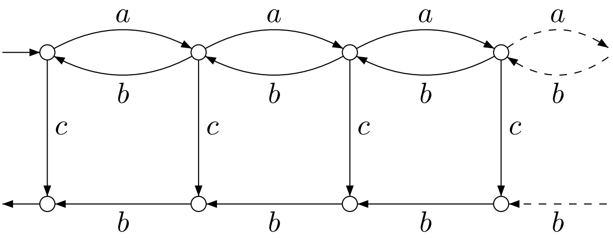


TU/e Technische Universiteit Eindhoven University of Technology

Example

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

is basic parallel, parallel pushdown and pushdown but not context-free

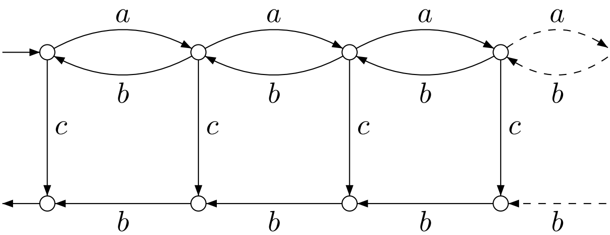


The bag is basic parallel, parallel pushdown but not pushdown, nor context-free

Example

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

is basic parallel, parallel pushdown and pushdown but not context-free



The bag is basic parallel, parallel pushdown but not pushdown, nor context-free

The stack is context-free, pushdown but not basic parallel, nor parallel pushdown

Computable Process

Definition

A computable process is a bisimulation equivalence class of a computable transition system

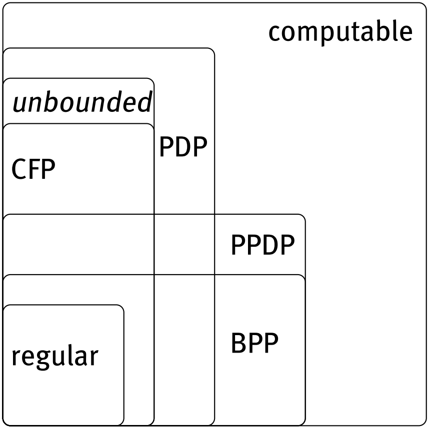
Theorem

A process is computable iff it is an abstraction of a process given by a guarded recursive specification over communication algebra [FSEN, 2009]

Theorem

A process is computable iff it can be written as a regular process communicating with two stacks [FSEN, 2009]

Process Classes



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)
- Draft syllabus available

Technische Universiteit Eindhoven University of Technology

Thank you!

Questions?