Models of Computation: Automata and Processes

An Overview

Paul van Tilburg

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

July 14, 2009
Automata & Formal Language theory

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

Diagram:

- Start symbol: S
- Productions:
  - S → aT
  - T → aW b
  - W → aR b
  - R → aW b
  - U → aV b
  - V → b

Grammars and Recursive Specifications

From Finite Automaton to recursive specification

\[
S = aT + aW \\
T = aU + bW \\
U = bV + bR \\
V = 0 \\
W = aR \\
R = bW + 1
\]
From Recursive Specification to Automaton

Structural Operational Semantics [Plotkin, *JLAP*, 2004]
Similarities with Process Algebra

- Finite Automaton = finite labelled transition system
- Grammar = recursive specification over $0, 1, +, \cdot, a$
- Regular expression = closed term over $0, 1, +, \cdot, a, *$
Similarities with Process Algebra

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

Basic Process Algebra

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

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: $p \leftrightarrow q$)
Definition

A regular language is a language 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
**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 $0, 1, a, +$.
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 $0, 1, a, +$.
- Processes given by deterministic automata, and by regular expressions, form a subclass [Baeten, Corradini, Grabmayer, JACM 2007].
Pushdown Automaton

\[ a, \emptyset \rightarrow \emptyset \]
\[ a, 1 \rightarrow 11 \]
\[ b, 1 \rightarrow \varepsilon \]

\[ b, 1 \rightarrow \varepsilon \]
\[ S = 1 + \sum_{d \in D} i?d.S \cdot o!d.S \]
Theorem

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

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

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

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

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

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

Recursive specification

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

Example: $X = 1 + aX \cdot b1$

$$X \xrightarrow{a} X \cdot b1 \xrightarrow{a} X \cdot b1 \cdot b1 \ldots$$
Problem with 1-summands

\[ X = a X \cdot Y + b 1 \]
\[ Y = 1 + c 1 \]
Problem with 1-summands

Problem with 1-summands

\[ X = aX \cdot Y + b1 \]
\[ Y = 1 + c1 \]

- Recursive specifications over BPA_{0,1} 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 $\text{BPA}_{0,1}$ 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 BPA\(_{0,1}\) 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]
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 BPA_{0,1} 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
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

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

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

[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

\[
p \iff b \tau_i,o(\partial_i,o(q \parallel B))
\]

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

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

Definition
A basic parallel process is given by a guarded recursive specification over the signature \( 0, 1, +, a_-, \parallel \)

- Any basic parallel process is a parallel pushdown process
Example

\[ X = c.1 + a.(X \parallel b.1) \]

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

\[ X = c.1 + a.(X \parallel b.1) \]

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

\[ X = c.1 + a.(X \ || \ b.1) \]

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
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

Computable

Unbounded

CFP

PDP

BPP

PPDP

Regular
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.
Thank you!

Questions?