A Context-Free Process as a Pushdown Automaton

Paul van Tilburg

(joint work with Jos Baeten and Pieter Cuijpers)

Department of Mathematics and Computer Science
Eindhoven University of Technology

ProSe / June 12, 2008

Introduction

Project MoCAP:

- Models of Computation: Automata and Processes
- Integration automata theory and process theory
- Particular setting:
  - Investigate automata theory in a process theory setting
  - Find similarities and differences

Introduction

Project MoCAP:

- Models of Computation: Automata and Processes
- Integration automata theory and process theory
- Particular setting:
  - Investigate automata theory in a process theory setting
  - Find similarities and differences

Focus:

Theorem

For every context-free grammar, there exists a pushdown automaton such that the language generated by the grammar is identical with the language generated by the automaton.

Syntax and Semantics

Syntax

$\mathbf{0}$	deadlocked process	$\_ + \_$	choice
$\mathbf{1}$	terminating process	$\_ \cdot \_$	sequential composition
$a.\_$	prefix ($a \in \mathcal{A}$)	$\_ \parallel \_$	parallel composition

Semantics

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

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

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

Syntax and Semantics

Syntax

$\mathbf{0}$	deadlocked process	$\_ + \_$	choice
$\mathbf{1}$	terminating process	$\_ \cdot \_$	sequential composition
$a.\_$	prefix ($a \in \mathcal{A}$)	$\_ \parallel \_$	parallel composition

Semantics

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

$$\frac{x \xrightarrow{?d} x' \quad y \xrightarrow{!d} y'}{x \parallel y \xrightarrow{\mathbb{?}d} x' \parallel y'} \qquad \frac{x \xrightarrow{!d} x' \quad y \xrightarrow{?d} y'}{x \parallel y \xrightarrow{\mathbb{?}d} x' \parallel y'}$$

Process Classes

Chomsky hierarchy

regular

Regular process

Grammar

$$X \xrightarrow{a} Y, \qquad X \xrightarrow{b} \varepsilon,$$

$$Y \xrightarrow{c} \varepsilon$$

Process Classes

Chomsky hierarchy

regular

Regular process

Grammar

$$X \xrightarrow{a} Y, \qquad X \xrightarrow{b} \varepsilon,$$
$$Y \xrightarrow{c} \varepsilon$$

Recursive specification

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

Process Classes

Chomsky hierarchy

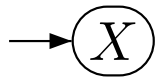
regular

Regular process

Recursive specification

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

Finite state trans. system (NFA)



Process Classes

Chomsky hierarchy

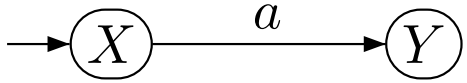
regular

Regular process

Recursive specification

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

Finite state trans. system (NFA)



Process Classes

Chomsky hierarchy

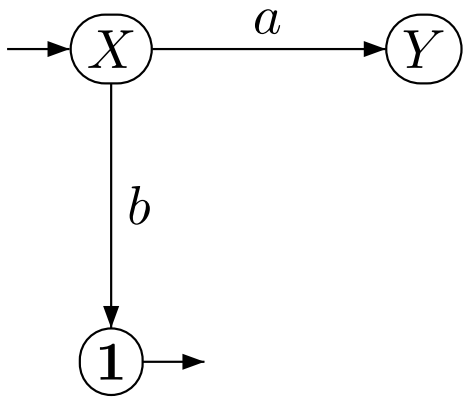
regular

Regular process

Recursive specification

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

Finite state trans. system (NFA)



Process Classes

Chomsky hierarchy

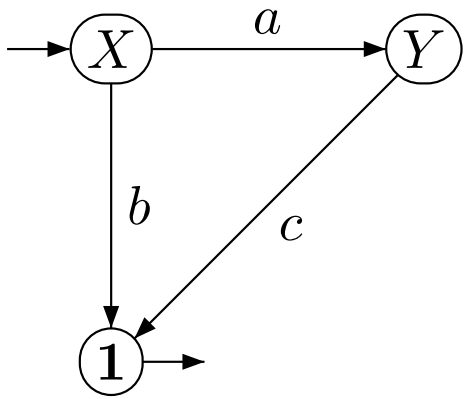
regular

Regular process

Recursive specification

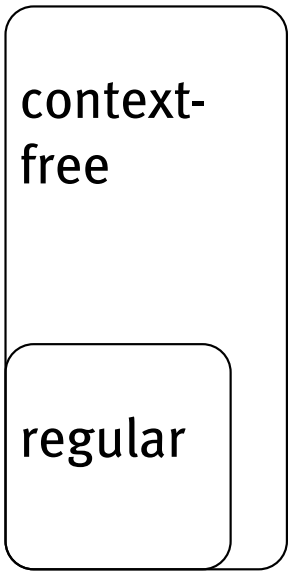
$$X = a.Y + b.\mathbf{1},$$
$$Y = c.\mathbf{1}$$

Finite state trans. system (NFA)



Process Classes

Chomsky hierarchy

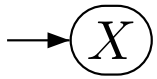


Context-free process

Recursive specification

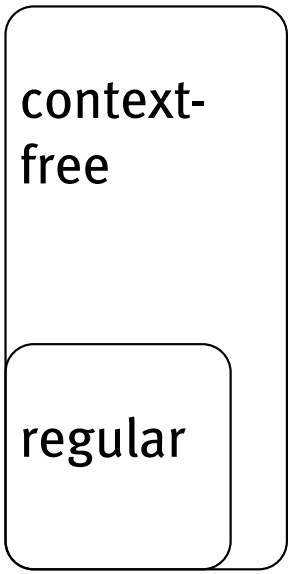
$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1}$$

Transition system



Process Classes

Chomsky hierarchy

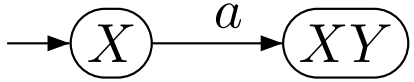


Context-free process

Recursive specification

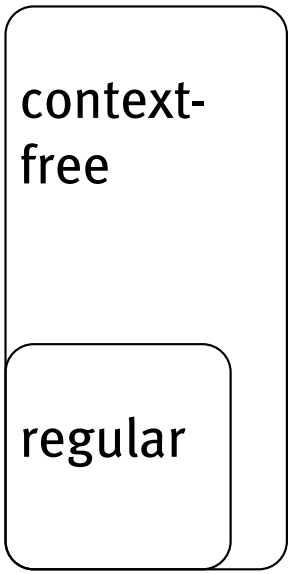
$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1}$$

Transition system



Process Classes

Chomsky hierarchy

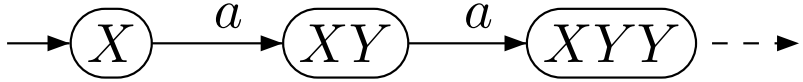


Context-free process

Recursive specification

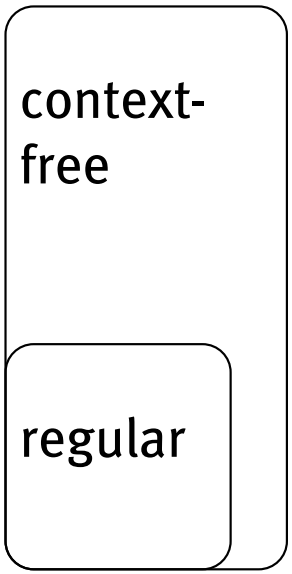
$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1}$$

Transition system



Process Classes

Chomsky hierarchy

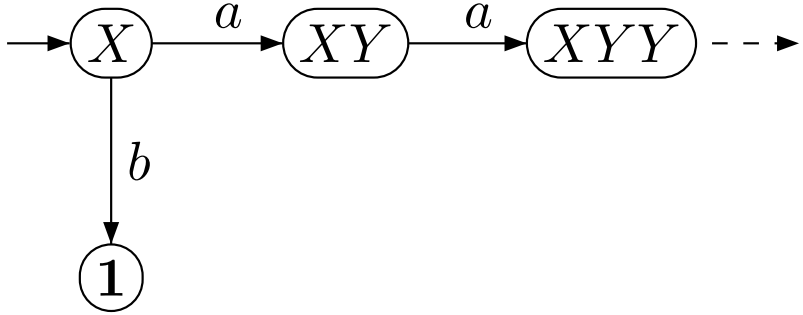


Context-free process

Recursive specification

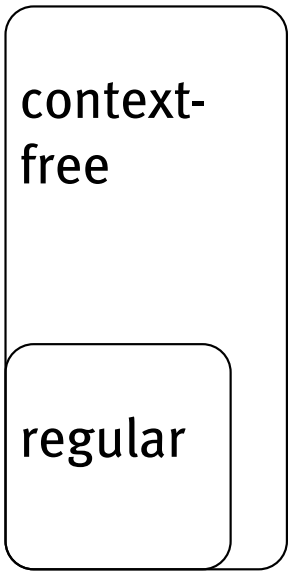
$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1}$$

Transition system



Process Classes

Chomsky hierarchy

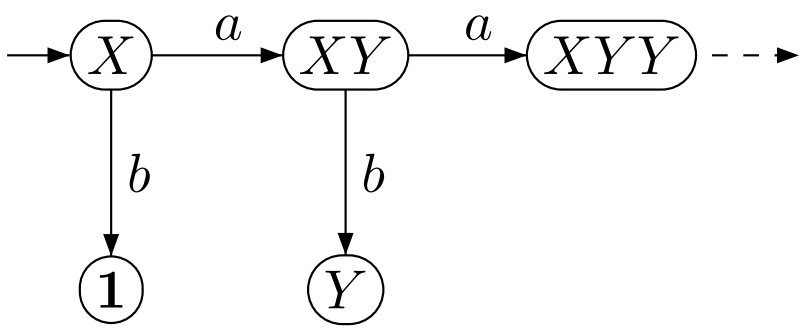


Context-free process

Recursive specification

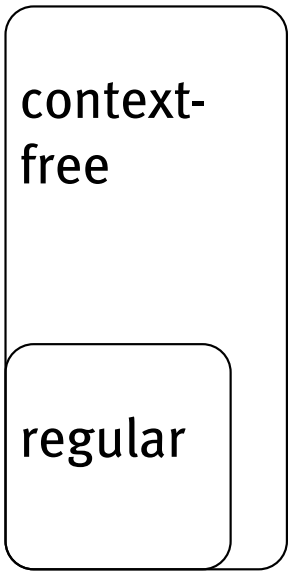
$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1}$$

Transition system



Process Classes

Chomsky hierarchy

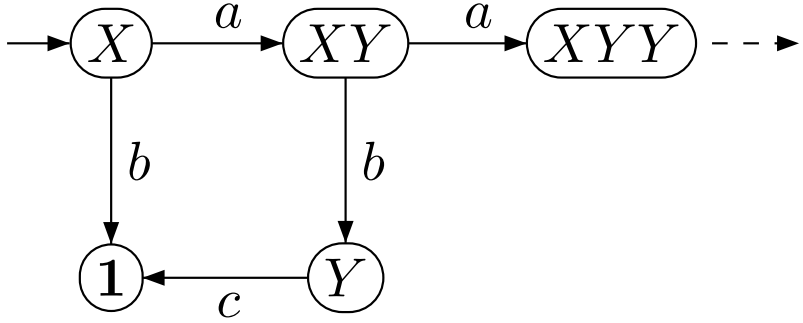


Context-free process

Recursive specification

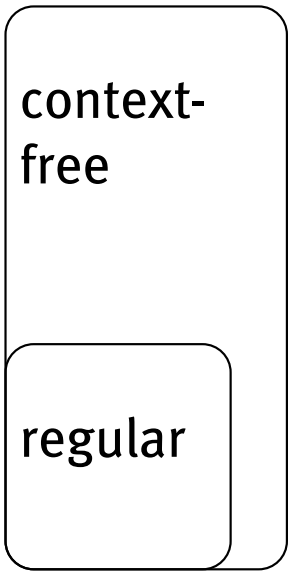
$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1}$$

Transition system



Process Classes

Chomsky hierarchy

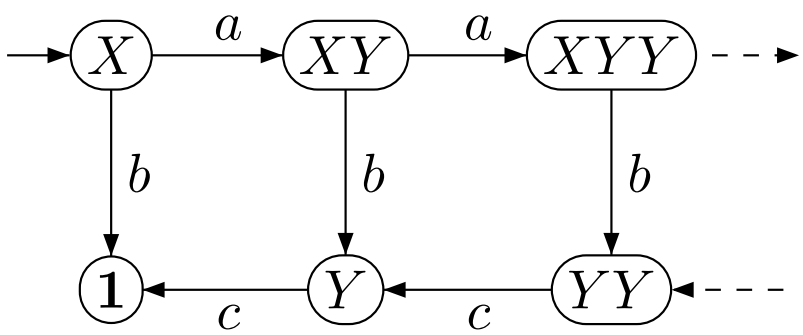


Context-free process

Recursive specification

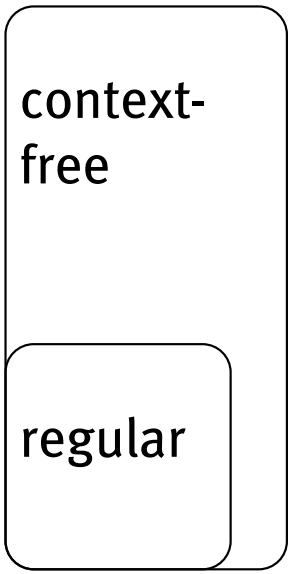
$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1}$$

Transition system



Process Classes

Chomsky hierarchy



The stack process

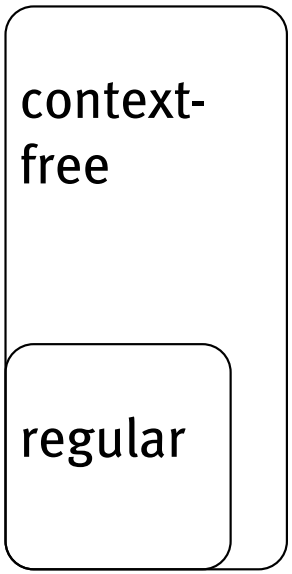
Recursive specification

$$S_\varepsilon = \mathbf{1} + \sum_{d \in D} ?d.S_d,$$

$$S_{d\sigma} = !d.S_\sigma + \sum_{e \in D} ?e.S_{ed\sigma}$$

Process Classes

Chomsky hierarchy



The stack process

Recursive specification

$$S_\varepsilon = \mathbf{1} + \sum_{d \in D} ?d.S_d,$$

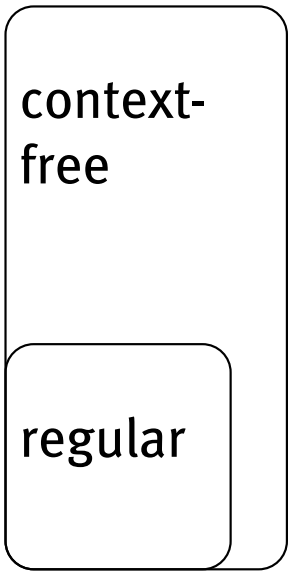
$$S_{d\sigma} = !d.S_\sigma + \sum_{e \in D} ?e.S_{ed\sigma}$$

Better specification

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

Process Classes

Chomsky hierarchy



The stack process

Better specification

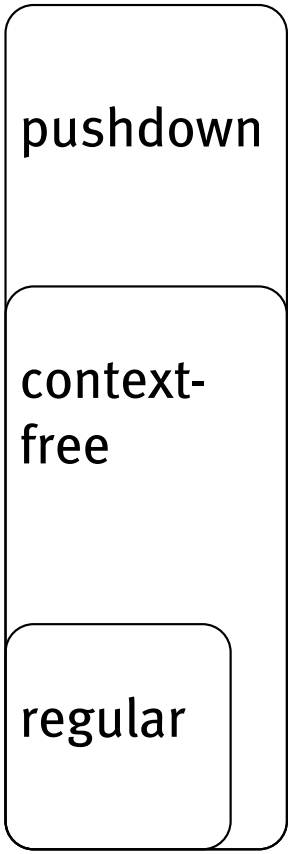
$$S = \mathbf{1} + \sum_{d \in D} ?d.S \cdot !d.S$$

Forgetful

$$S = \mathbf{1} + \sum_{d \in D} ?d.S \cdot (\mathbf{1} + !d.S)$$

Process Classes

Chomsky hierarchy



Pushdown process

Recursive specification context-free process

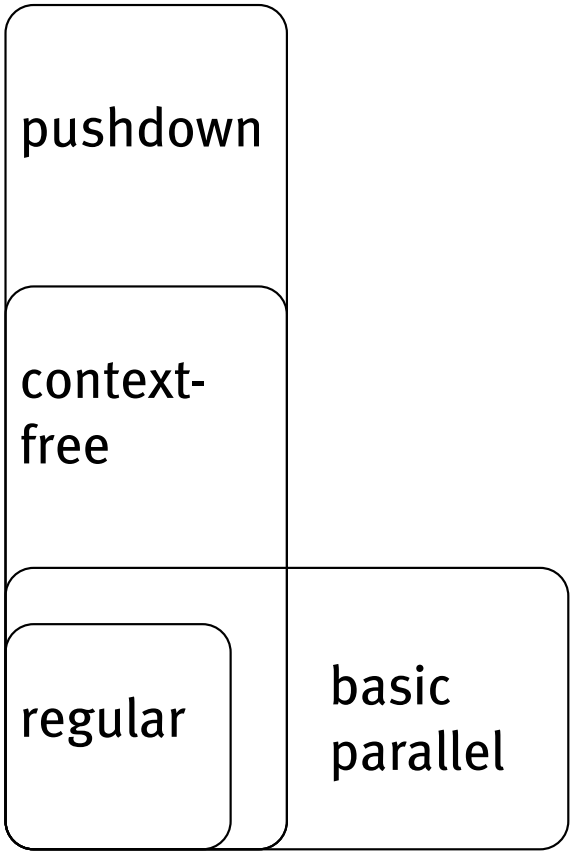
$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1}$$

Recursive specification pushdown process

$$\overline{P}(X\xi) = a.\overline{P}(XY\xi) + b.\overline{P}(\xi)$$
$$\overline{P}(Y\xi) = c.\overline{P}(\xi)$$
$$\overline{P}(\varepsilon) = \mathbf{1}$$

Process Classes

Chomsky hierarchy

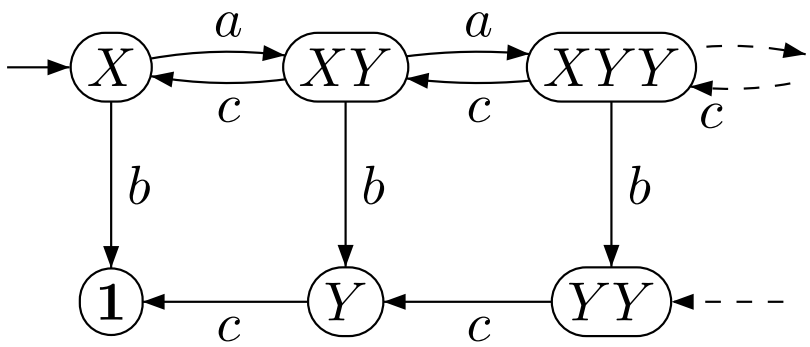


Basic parallel process

Recursive specification

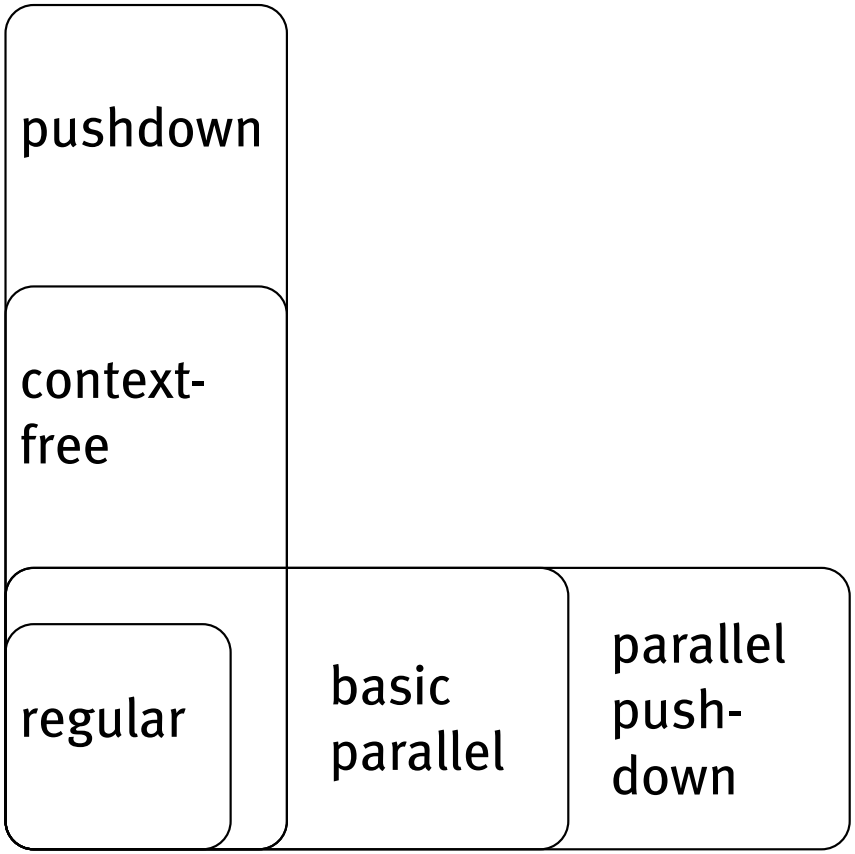
$$X = a.(X \parallel Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1}$$

Transition system



Process Classes

Chomsky hierarchy



Parallel pushdown process

Recursive specification basic parallel process

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

Recursive specification pushdown process

$$\overline{P}(X\xi) = a.\overline{P}(XY\xi) + b.\overline{P}(\xi)$$
$$\overline{P}(Y\xi) = c.\overline{P}(\xi)$$
$$\overline{P}(\varepsilon) = \mathbf{1}$$

Goals

- Take the data out of the pushdown automaton
  $\Rightarrow$ model it as a process
- Make communication explicit:
  (regular) finite control communication with a *stack process*
- Add 1 to our definition of context-free processes:
  in accordance with automata theory, but leads to complications!

Bounded Branching Context-Free Process

Context-free process:

$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1}$$

Translated:

$$\hat{X} = a.\mathrm{Push}(XY) + b.\mathrm{Push}(\mathbf{1}),$$
$$\hat{Y} = c.\mathrm{Push}(\mathbf{1})$$

$$\mathrm{Push}(\mathbf{1}) = \mathrm{Ctrl}$$
$$\mathrm{Push}(\xi Y) = !Y.\mathrm{Push}(\xi)$$
$$\mathrm{Ctrl} = \sum_{V \in \mathcal{V}} ?V.\hat{V} + \mathbf{1}$$
$$S = \mathbf{1} + \sum_{d \in D} ?d.S \cdot !d.S$$

Bounded Branching Context-Free Process

Context-free process:

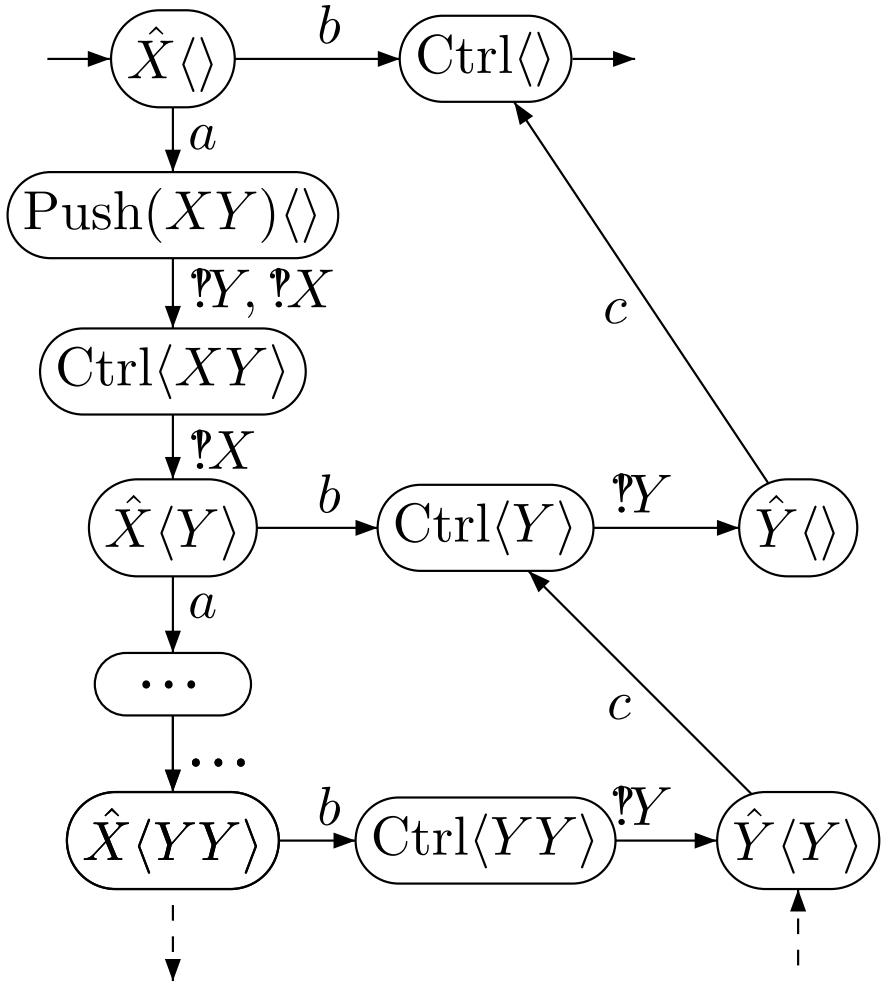
$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1}$$

Translated:

$$\hat{X} = a.\mathrm{Push}(XY) + b.\mathrm{Push}(\mathbf{1}),$$
$$\hat{Y} = c.\mathrm{Push}(\mathbf{1})$$

$$\mathrm{Push}(\mathbf{1}) = \mathrm{Ctrl}$$
$$\mathrm{Push}(\xi Y) = !Y.\mathrm{Push}(\xi)$$
$$\mathrm{Ctrl} = \sum_{V \in \mathcal{V}} ?V.\hat{V} + \mathbf{1}$$
$$S = \mathbf{1} + \sum_{d \in D} ?d.S \cdot !d.S$$

Transition system



Bounded Branching Context-Free Process

Context-free process:

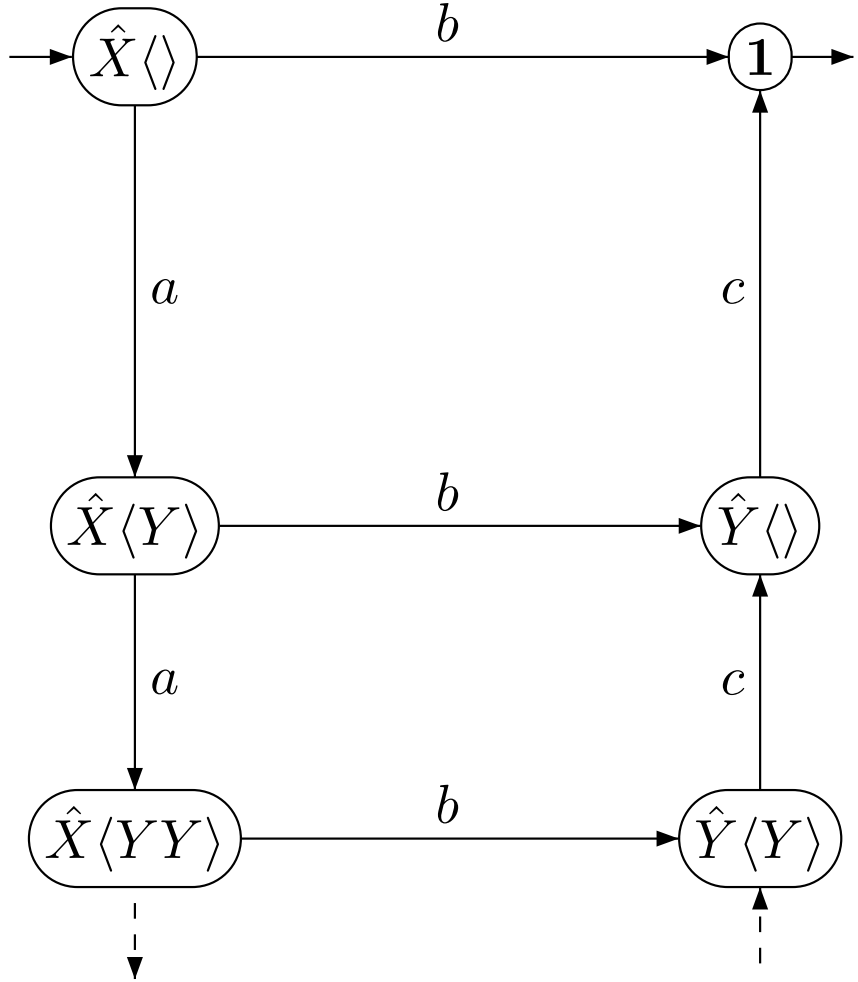
$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1}$$

Translated:

$$\hat{X} = a.\mathrm{Push}(XY) + b.\mathrm{Push}(\mathbf{1}),$$
$$\hat{Y} = c.\mathrm{Push}(\mathbf{1})$$

$$\mathrm{Push}(\mathbf{1}) = \mathrm{Ctrl}$$
$$\mathrm{Push}(\xi Y) = !Y.\mathrm{Push}(\xi)$$
$$\mathrm{Ctrl} = \sum_{V \in \mathcal{V}} ?V.\hat{V} + \mathbf{1}$$
$$S = \mathbf{1} + \sum_{d \in D} ?d.S \cdot !d.S$$

Transition system modulo br. bisim.

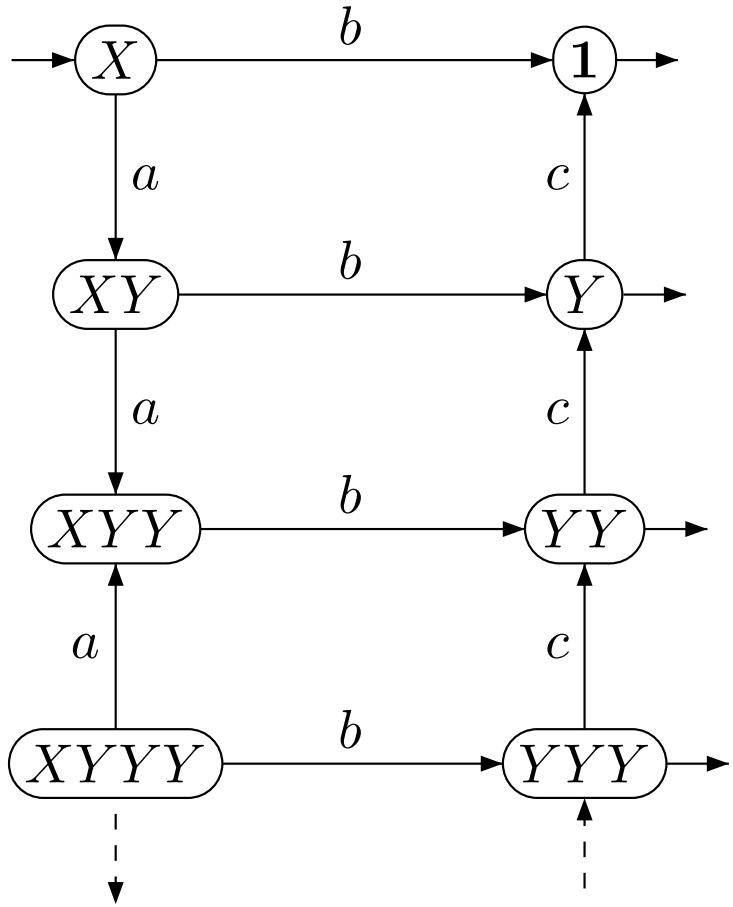


Unbounded Branching Context-Free Processes

Context-free process:

$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1} + \mathbf{1}$$

Transition system

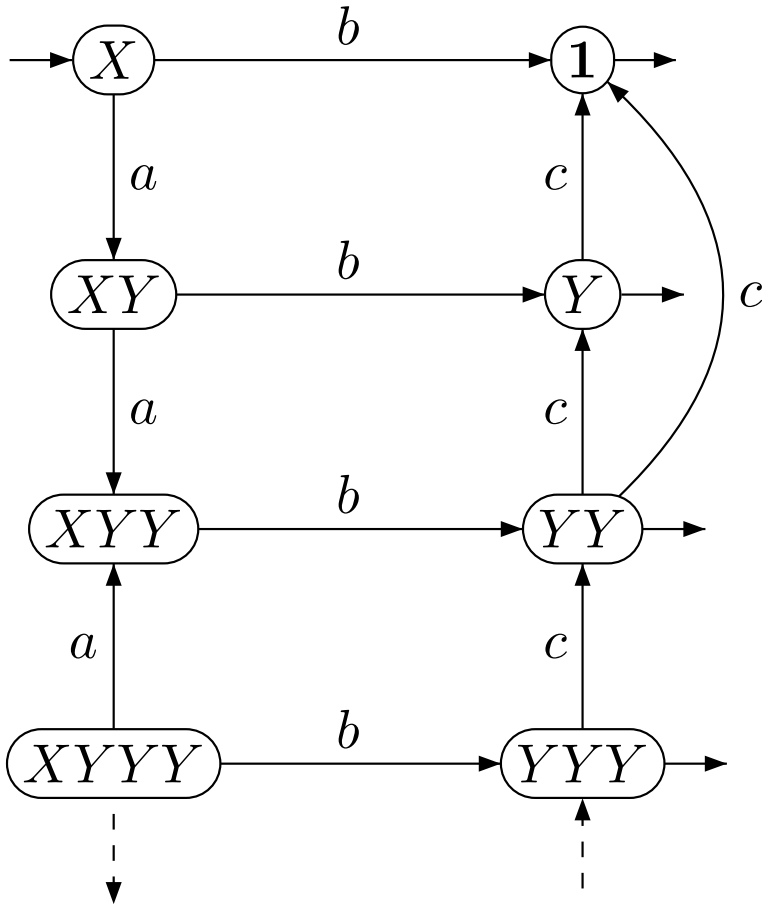


Unbounded Branching Context-Free Processes

Context-free process:

$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1} + \mathbf{1}$$

Transition system

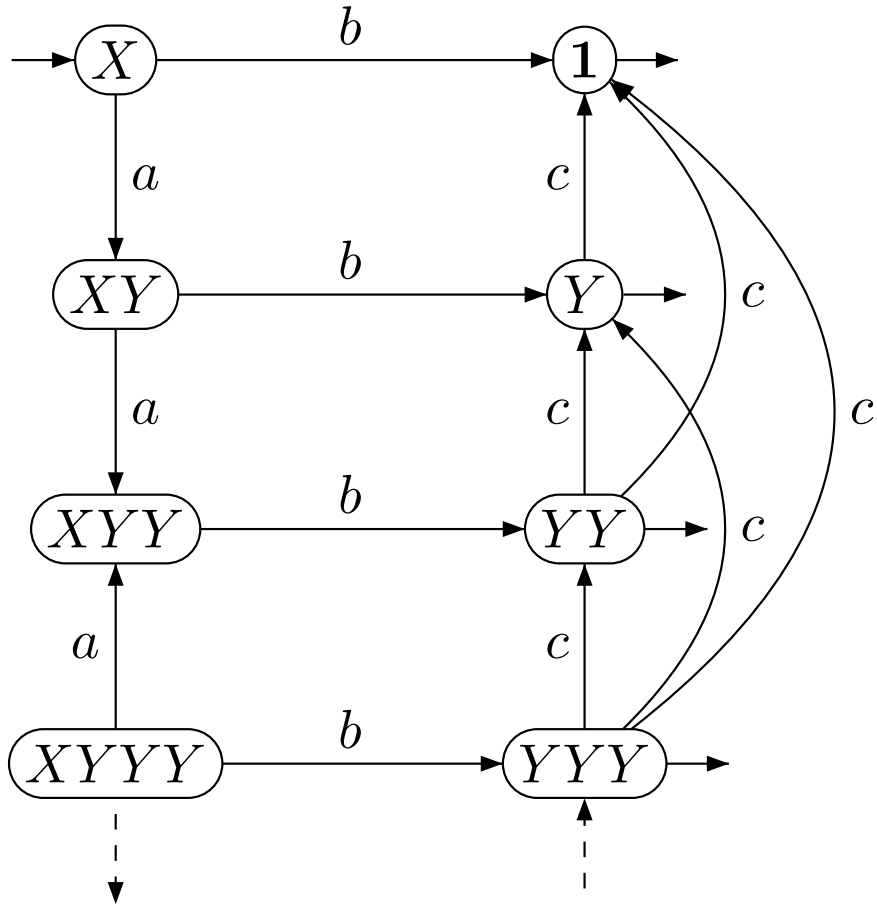


Unbounded Branching Context-Free Processes

Context-free process:

$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1} + \mathbf{1}$$

Transition system



Unbounded Branching Context-Free Processes

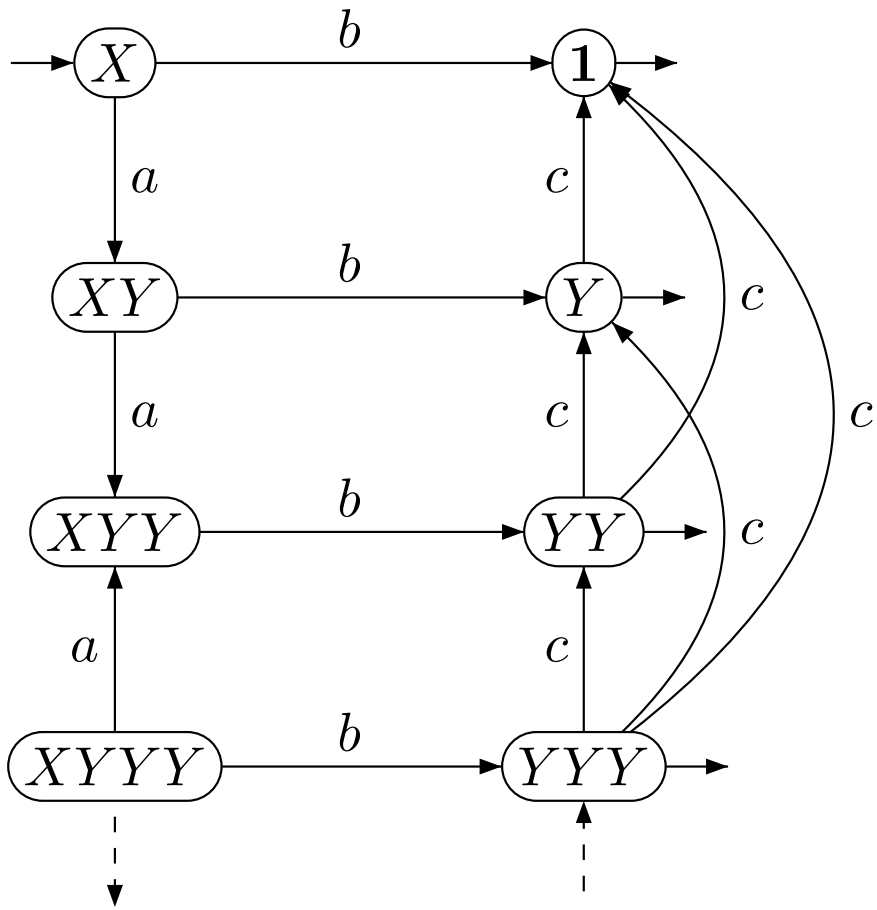
Context-free process:

$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1} + \mathbf{1}$$

Translated:

$$\hat{X} = a.\mathrm{Push}(XY) + b.\mathrm{Push}(\mathbf{1}),$$
$$\hat{Y} = c.\mathrm{Push}(\mathbf{1}) + \mathrm{Push}(\mathbf{1})$$
$$\mathrm{Push}(\mathbf{1}) = \mathrm{Ctrl}$$
$$\mathrm{Push}(\xi Y) = !Y.\mathrm{Push}(\xi)$$
$$\mathrm{Ctrl} = \sum_{V \in \mathcal{V}} ?V.\hat{V} + \mathbf{1}$$
$$S = \mathbf{1} + \sum_{d \in D} ?d.S \cdot !d.S$$

Transition system



Unbounded Branching Context-Free Processes

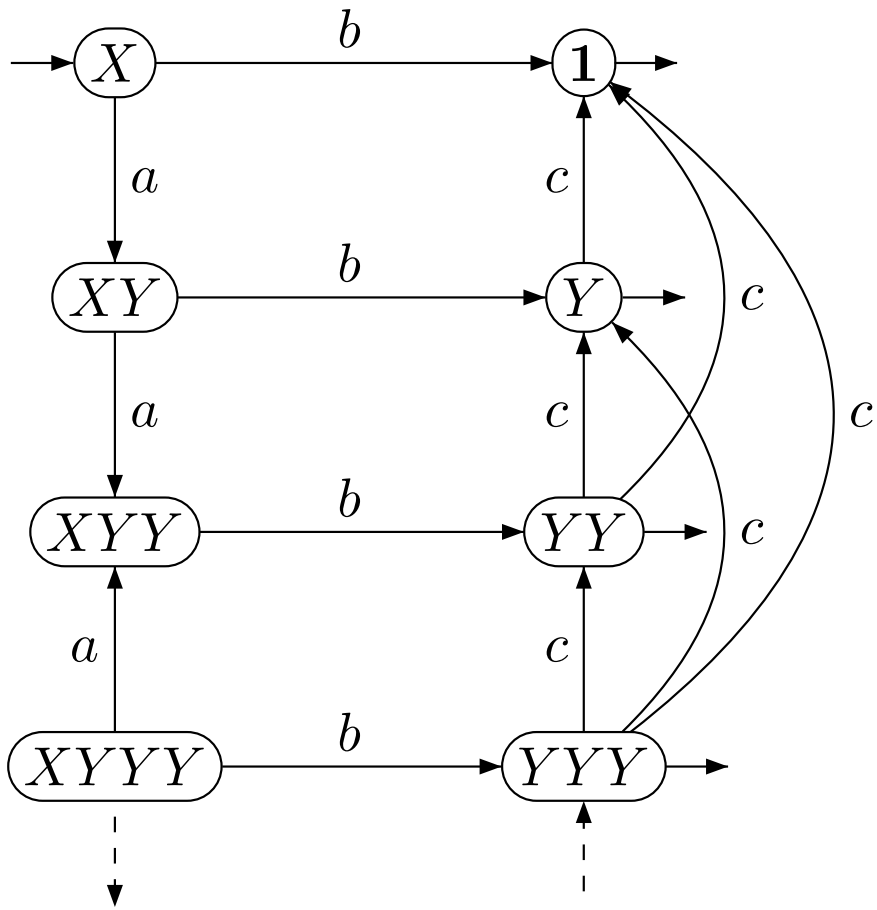
Context-free process:

$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1} + \mathbf{1}$$

Translated:

$$\hat{X} = a.\mathrm{Push}(XY) + b.\mathrm{Push}(\mathbf{1}),$$
$$\hat{Y} = c.\mathrm{Push}(\mathbf{1}) + \mathrm{Push}(\mathbf{1})$$
$$\mathrm{Push}(\mathbf{1}) = \mathrm{Ctrl}$$
$$\mathrm{Push}(\xi Y) = !Y.\mathrm{Push}(\xi)$$
$$\mathrm{Ctrl} = \sum_{V \in \mathcal{V}} ?V.\hat{V} + \mathbf{1}$$
$$S = \mathbf{1} + \sum_{d \in D} ?d.S \cdot (\mathbf{1} + !d.S)$$

Transition system



Concluding Remarks

Found solution

- Made communication explicit
- Introduced 1, dealt with complications
- Solution equivalent to context-free process modulo contrasimulation in general case
- Solution equivalent to context-free process modulo (rooted) branching bisimulation in bounded branching case
- The stack can be seen as the prototypical context-free process!

Concluding Remarks

Found solution

- Made communication explicit
- Introduced 1, dealt with complications
- Solution equivalent to context-free process modulo contrasimulation in general case
- Solution equivalent to context-free process modulo (rooted) branching bisimulation in bounded branching case
- The stack can be seen as the prototypical context-free process!

Future work

- Reverse case, maybe with 1?
- Bags?
- Queues?

Questions?