Formal Languages, Automata and Processes

Finished and Future Work

Paul van Tilburg

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

Formal Methods Group
Department of Mathematics and Computer Science
Eindhoven University of Technology

IPA Fall Days, 2008

Project MoCAP

- Models of Computation: Automata and Processes

  Automata + Interaction = Concurrency

Project MoCAP

- Models of Computation: Automata and Processes

  Automata + Interaction = Concurrency

- Separate development
- Integration
  - Attempt reveals differences and similarities
  - Use *analogies* to make the integration explicit

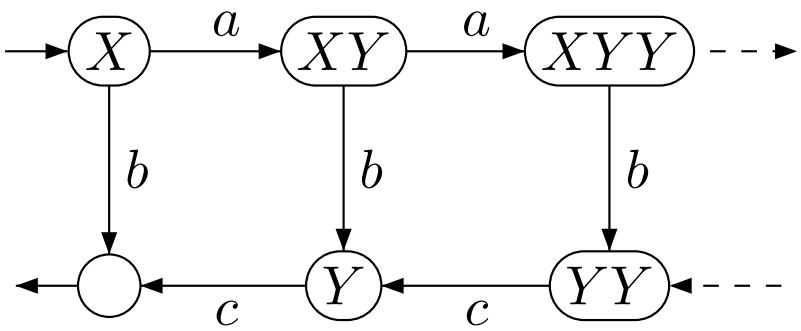
Example: Context-Free Language

Right-linear grammar

Generates a context-free language

$$X \longrightarrow aXY \mid b$$
$$Y \longrightarrow c$$

Transition system



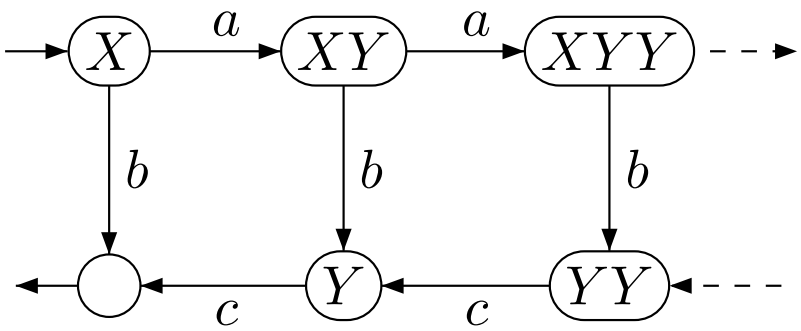
Example: Context-Free Language

Right-linear grammar

Generates a context-free language

$$X \longrightarrow aXY \mid b$$
$$Y \longrightarrow c$$

Transition system



Theorem

For every context-free language there exists a pushdown automaton that accepts it.

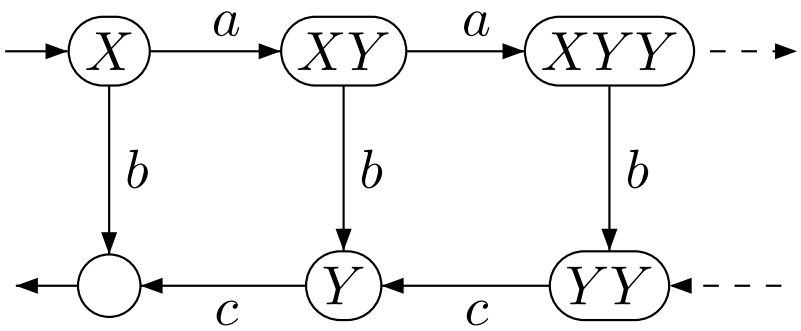
Example: Context-Free Language

Right-linear grammar

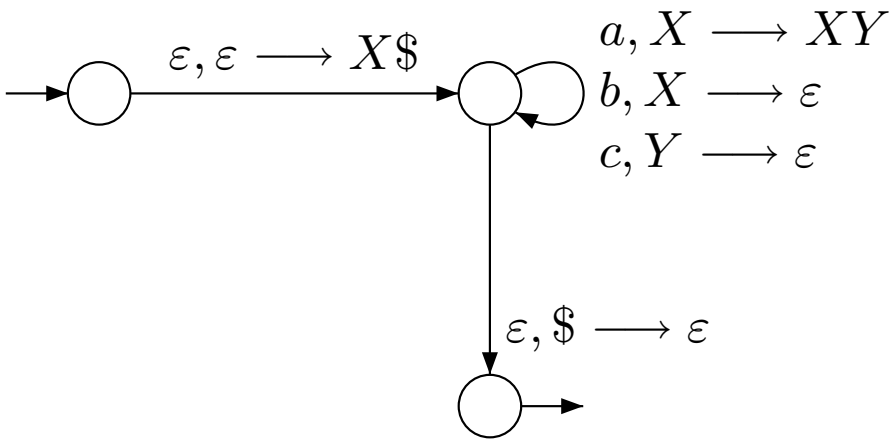
Generates a context-free language

$$X \longrightarrow aXY \mid b$$
$$Y \longrightarrow c$$

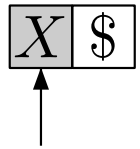
Transition system



Pushdown automaton



Stack



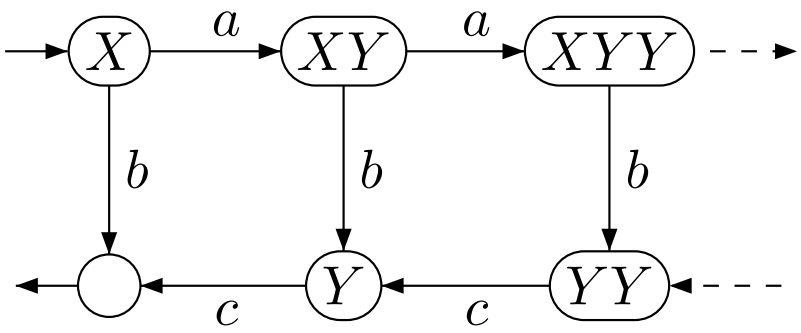
Example: Context-Free Language

Right-linear grammar

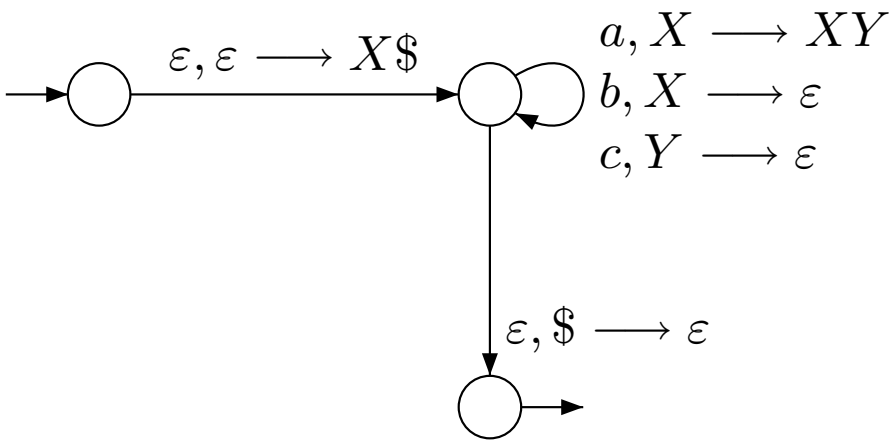
Generates a context-free language

$$X \longrightarrow aXY \mid b$$
$$Y \longrightarrow c$$

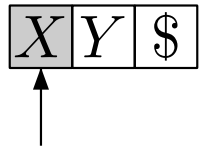
Transition system



Pushdown automaton



Stack



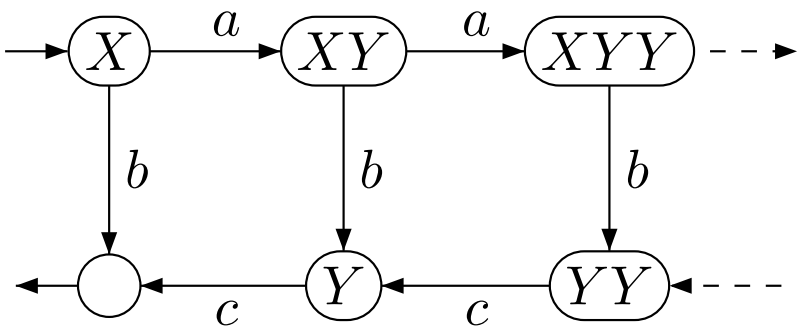
Example: Context-Free Language

Right-linear grammar

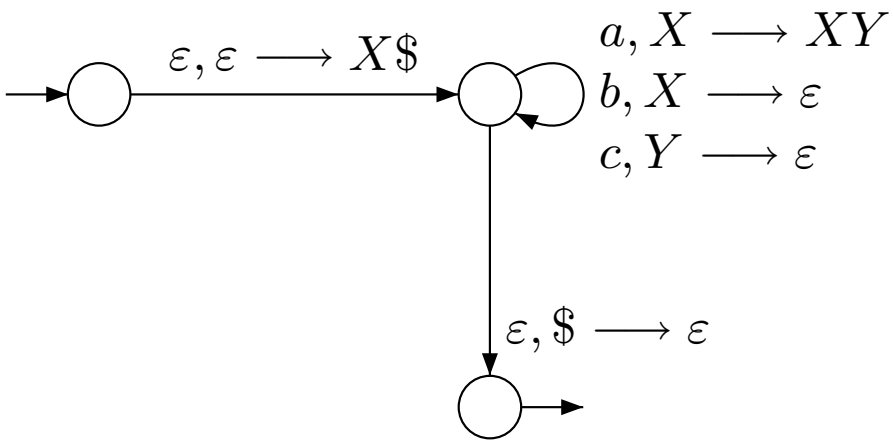
Generates a context-free language

$$X \longrightarrow aXY \mid b$$
$$Y \longrightarrow c$$

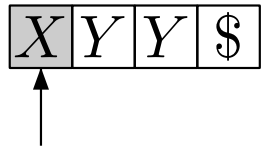
Transition system



Pushdown automaton



Stack



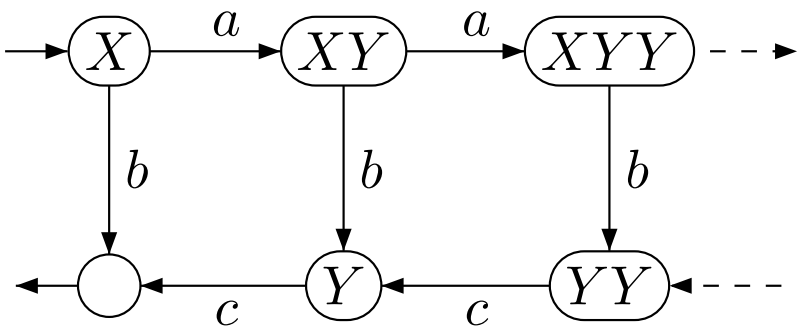
Example: Context-Free Language

Right-linear grammar

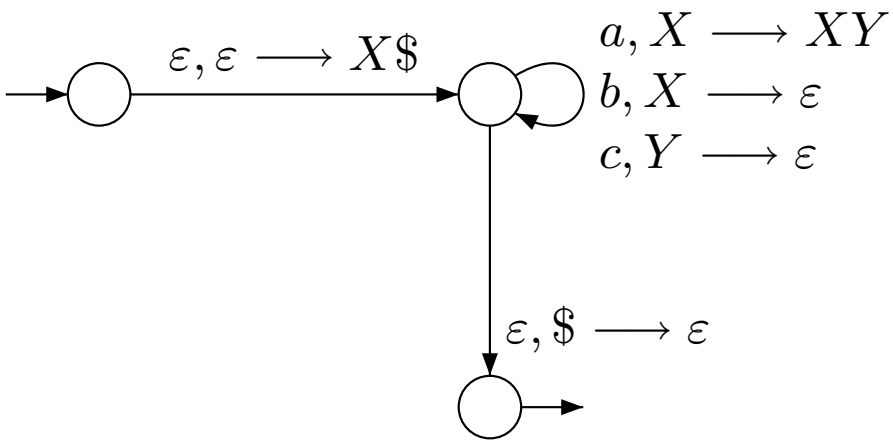
Generates a context-free language

$$X \longrightarrow aXY \mid b$$
$$Y \longrightarrow c$$

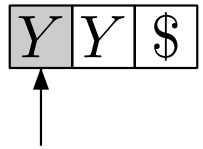
Transition system



Pushdown automaton



Stack



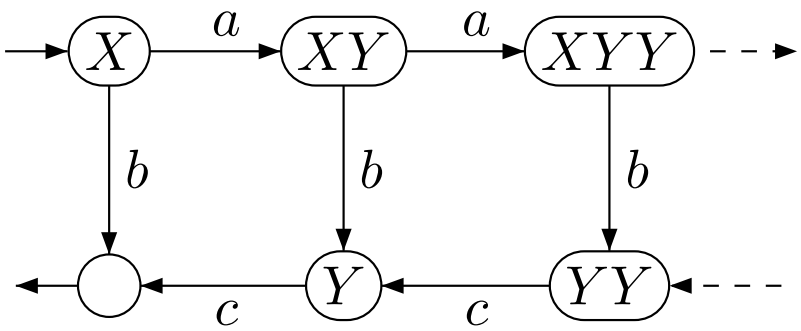
Example: Context-Free Language

Right-linear grammar

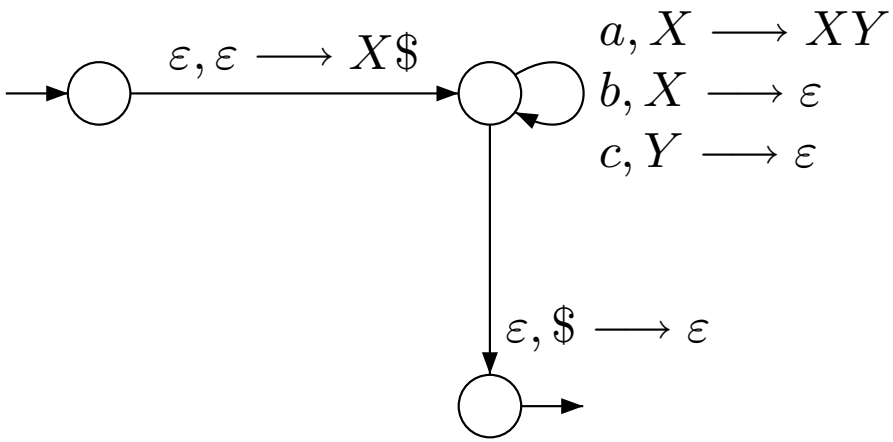
Generates a context-free language

$$X \longrightarrow aXY \mid b$$
$$Y \longrightarrow c$$

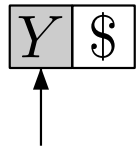
Transition system



Pushdown automaton



Stack



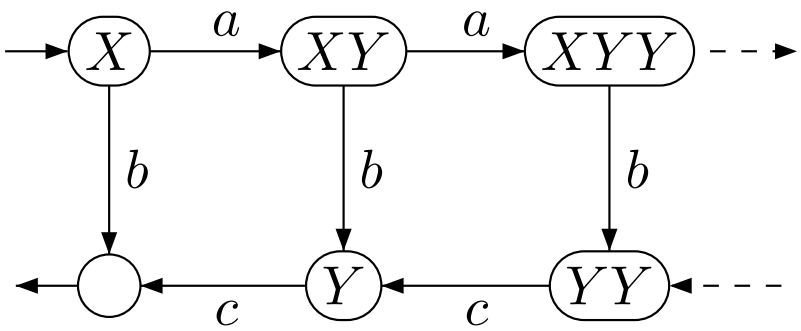
Example: Context-Free Language

Right-linear grammar

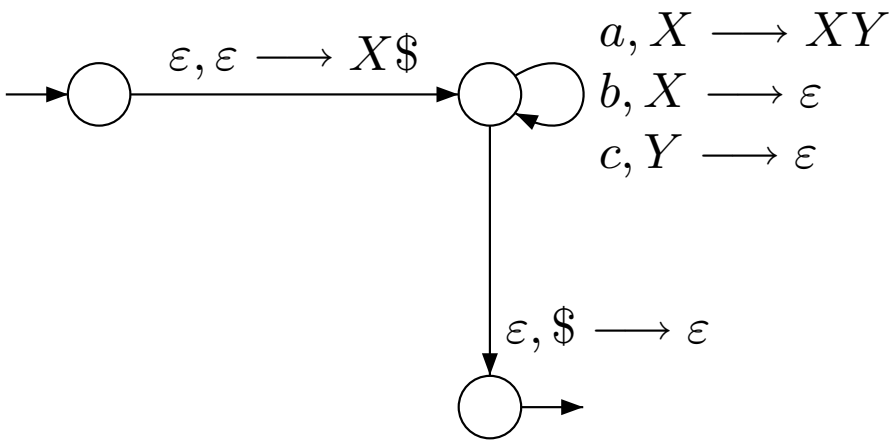
Generates a context-free language

$$X \longrightarrow aXY \mid b$$
$$Y \longrightarrow c$$

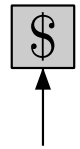
Transition system



Pushdown automaton



Stack



Right-linear grammar

Generates a context-free language

$$X \longrightarrow aXY \mid b$$
$$Y \longrightarrow c$$

Right-linear grammar

Generates a context-free language

$$X \longrightarrow aXY \mid b$$
$$Y \longrightarrow c$$

Recursive specification over BPA

Specifies a context-free process

$$X = a \cdot (X \cdot Y) + b$$
$$Y = c$$

Restrict to:
finite and guarded specifications

Right-linear grammar

Generates a context-free language

$$X \longrightarrow aXY \mid b$$
$$Y \longrightarrow c$$

Recursive specification over BPA

Specifies a context-free process

$$X = a \cdot (X \cdot Y) + b$$
$$Y = c$$

Restrict to:
finite and guarded specifications

0 and 1

- Regular expressions use 0 (deadlock) and 1 (final state)
- Not done in grammars in automata theory
- Add it to recursive specifications

Processes and Recursive Specifications

Right-linear grammar

Generates a context-free language

$$X \longrightarrow aXY \mid b$$
$$Y \longrightarrow c$$

Recursive specification over BPA

Specifies a context-free process

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

Restrict to:
finite and guarded specifications

$\mathbf{0}$ and $\mathbf{1}$

- Regular expressions use $\mathbf{0}$ (deadlock) and $\mathbf{1}$ (final state)
- Not done in grammars in automata theory
- Add it to recursive specifications

Main Theorem

Process theory enables us to...

- Model the data (a stack) as a process
- Make communication explicit
- Use bisimulation equivalences to preserve branching structure

Main Theorem

Process theory enables us to...

- Model the data (a stack) as a process
- Make communication explicit
- Use bisimulation equivalences to preserve branching structure

Theorem

Every context-free process is equivalent to a regular process communicating with a stack. [CONCUR08]

Pushdown Automaton

Context-free process

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

Translated

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

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

Pushdown Automaton

Context-free process

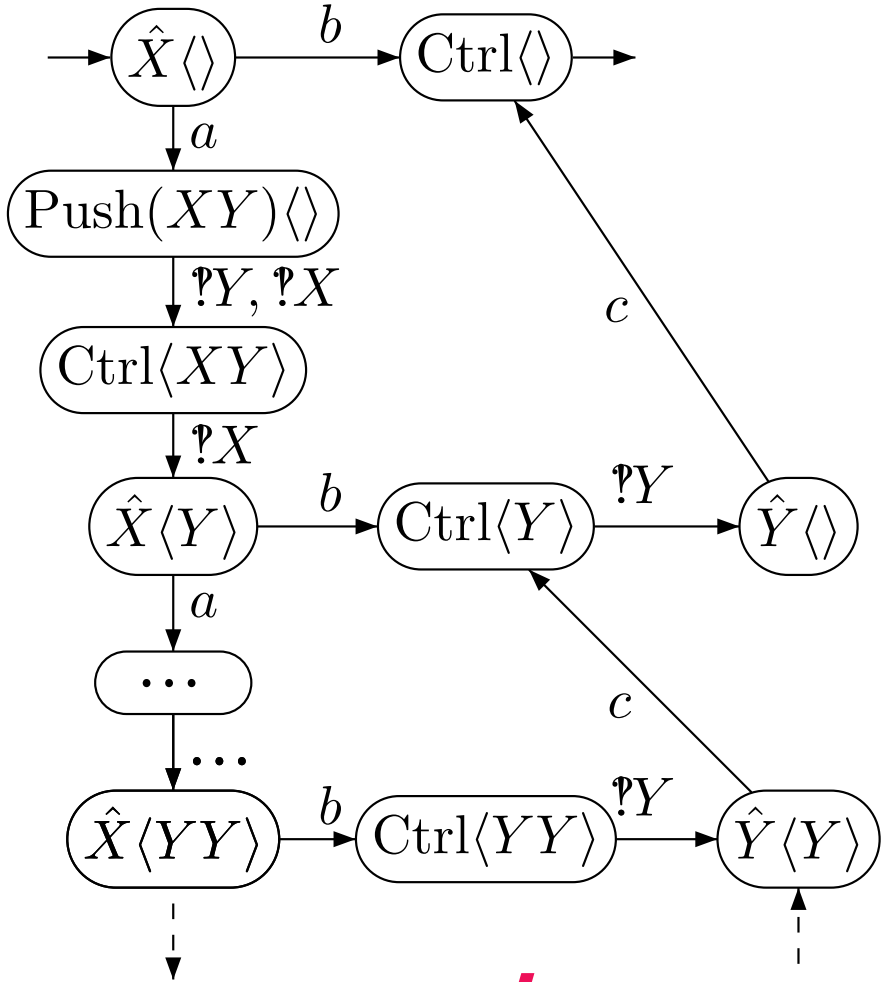
$$X = a.(X \cdot Y) + b.\mathbf{1}$$
$$Y = c.\mathbf{1}$$

Translated

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

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

Transition system



Pushdown Automaton

Context-free process

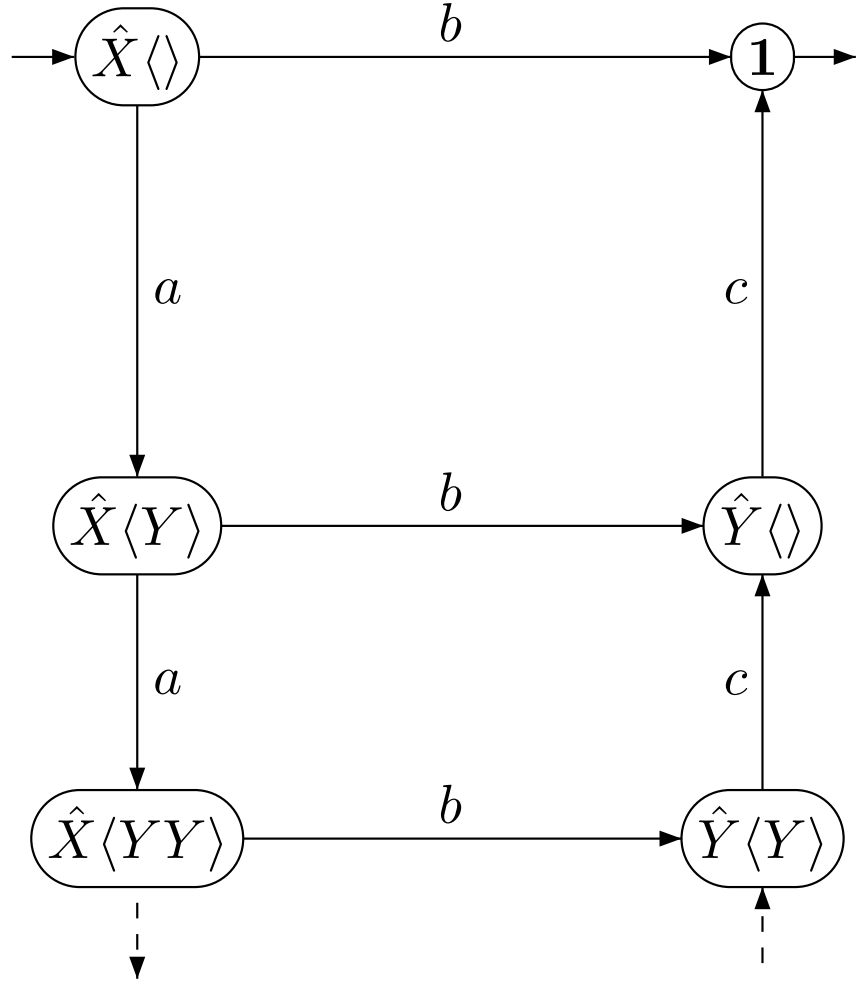
$$X = a.(X \cdot Y) + b.\mathbf{1}$$
$$Y = c.\mathbf{1}$$

Translated

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

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

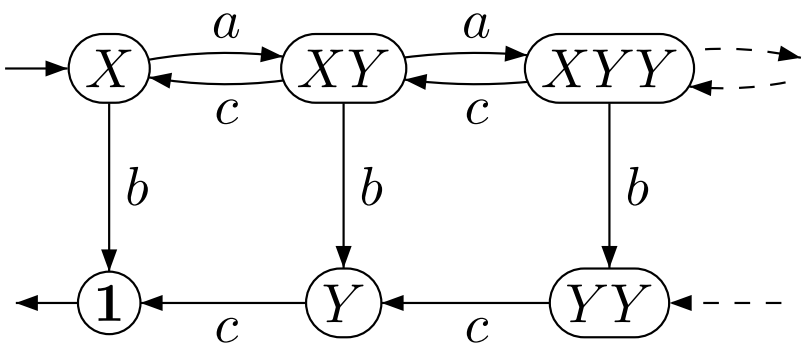
... modulo rooted br. bisim.



Recursive specification over BPP

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

Transition system

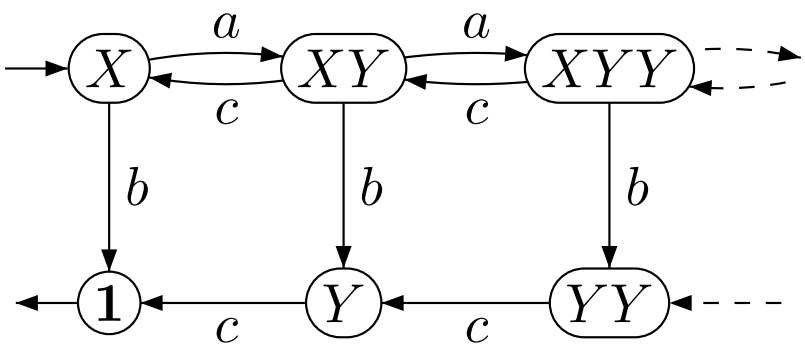


Basic Parallel Processes

Recursive specification over BPP

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

Transition system



Theorem

Every basic parallel process is equivalent to a regular process communicating with a bag. [EXPRESS08]

Future work

Basic Queueing Processes

- Future work with master students
- Prove a similar theorem

Future work

Basic Queueing Processes

- Future work with master students
- Prove a similar theorem

Expressivity

- Relative expressivity of BPA_0, BPP_0 and PA_0 known
- Complete the picture with 1

Future work

Basic Queueing Processes

- Future work with master students
- Prove a similar theorem

Expressivity

- Relative expressivity of BPA_0, BPP_0 and PA_0 known
- Complete the picture with 1

Decidability

- Decidability of bisimulation for BPA known
- Investigate with 0 and 1

Thank you!

Questions?