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

CONCUR ’08
Toronto, Canada / August 19, 2008

Technische Universiteit Eindhoven University of Technology

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
- Study similarities and differences

Regular Language

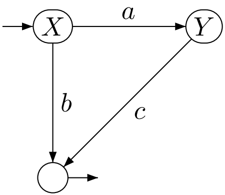
Right-linear grammar

Generates a regular language

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

Non-deterministic Finite Automaton

Accepts a regular language



Also: (finite) transition system

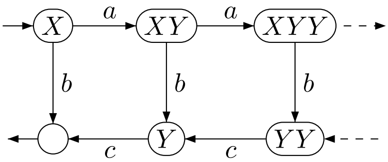
Context-Free Language

Context-free grammar

Generates a context-free language

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

Transition system



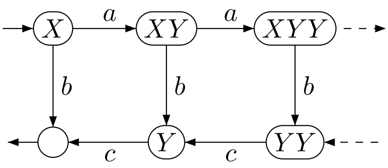
Context-Free Language

Context-free grammar

Generates a context-free language

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

Transition system



Famous theorem from automata theory

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

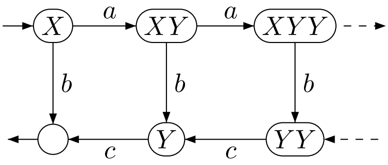
Context-Free Language

Context-free grammar

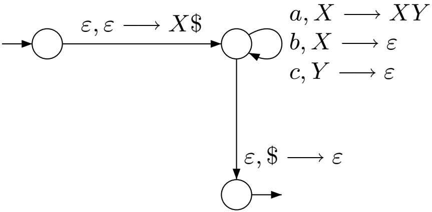
Generates a context-free language

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

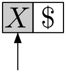
Transition system



Pushdown automaton



Stack



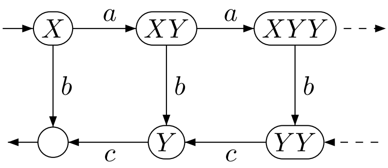
Context-Free Language

Context-free grammar

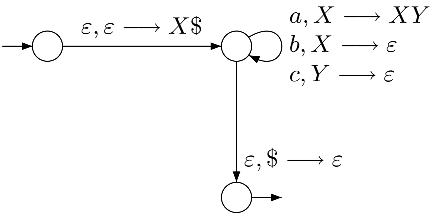
Generates a context-free language

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

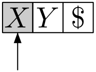
Transition system



Pushdown automaton



Stack



TU/e Technische Universiteit Eindhoven University of Technology

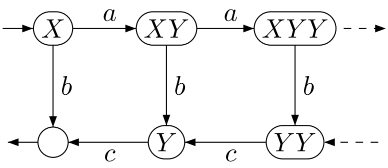
Context-Free Language

Context-free grammar

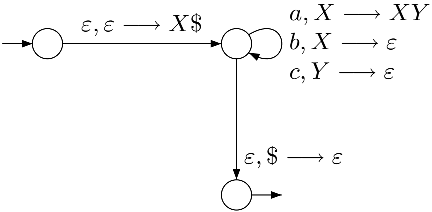
Generates a context-free language

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

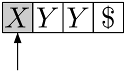
Transition system



Pushdown automaton



Stack



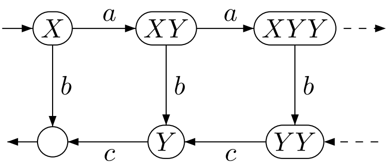
Context-Free Language

Context-free grammar

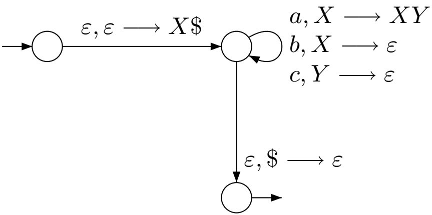
Generates a context-free language

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

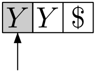
Transition system



Pushdown automaton



Stack



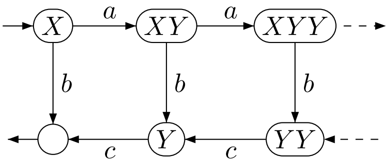
Context-Free Language

Context-free grammar

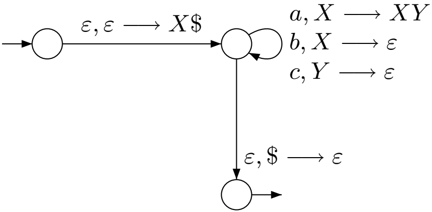
Generates a context-free language

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

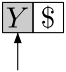
Transition system



Pushdown automaton



Stack



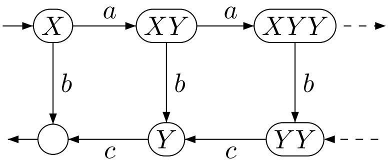
Context-Free Language

Context-free grammar

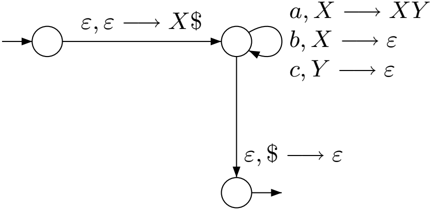
Generates a context-free language

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

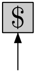
Transition system



Pushdown automaton



Stack



Context-free grammar

Generates a context-free language

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

Processes and Recursive Specifications

Context-free 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

TU/e Technische Universiteit Eindhoven University of Technology

Processes and Recursive Specifications

Context-free 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

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

- Regular expressions use $\mathbf{0}$ (deadlock) and $\mathbf{1}$ (final state)
- Capture deadlocked states and (intermediate) final states
- The $\mathbf{1}$ is also present as λ in grammars
  - Removable using language equivalence, not modulo bisimulation

Processes and Recursive Specifications

Context-free grammar

Generates a context-free language

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

Recursive specification over $BPA_{\mathbf{0},\mathbf{1}}$

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)
- Capture deadlocked states and (intermediate) final states
- The $\mathbf{1}$ is also present as λ in grammars
  - Removable using language equivalence, not modulo bisimulation

Technische Universiteit Eindhoven University of Technology

Main Theorem

Process theory enables us to introduce *interaction* by...

- Modeling the data (a stack) as a process
- Making communication with the stack explicit
- Using bisimulation equivalences to preserve branching structure

Main Theorem

Process theory enables us to introduce *interaction* by...

- Modeling the data (a stack) as a process
- Making communication with the stack explicit
- Using bisimulation equivalences to preserve branching structure

Theorem

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

Specifications

Infinite recursive specification (infinite data set)

$$S_\varepsilon = \mathbf{1} + \sum_{d \in D} ?d.S_d \qquad S_{d\sigma} = !d.S_\sigma + \sum_{e \in D} ?e.S_{ed\sigma}$$

Technische Universiteit Eindhoven University of Technology

Specifications

Infinite recursive specification (infinite data set)

$$S_\varepsilon = \mathbf{1} + \sum_{d \in D} ?d.S_d \qquad S_{d\sigma} = !d.S_\sigma + \sum_{e \in D} ?e.S_{ed\sigma}$$

Finite recursive specification over BPA

$$S = T \cdot S \qquad T = \sum_{d \in D} ?d.T_d \qquad T_d = !d + T \cdot T_d$$

Specifications

Infinite recursive specification (infinite data set)

$$S_\varepsilon = \mathbf{1} + \sum_{d \in D} ?d.S_d \qquad S_{d\sigma} = !d.S_\sigma + \sum_{e \in D} ?e.S_{ed\sigma}$$

Finite recursive specification over BPA

$$S = T \cdot S \qquad T = \sum_{d \in D} ?d.T_d \qquad T_d = !d + T \cdot T_d$$

Even smaller specification (over $\text{BPA}_{\mathbf{0},\mathbf{1}}$)

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

The Stack

Specifications

Infinite recursive specification (infinite data set)

$$S_\varepsilon = \mathbf{1} + \sum_{d \in D} ?d.S_d \qquad S_{d\sigma} = !d.S_\sigma + \sum_{e \in D} ?e.S_{ed\sigma}$$

Finite recursive specification over BPA

$$S = T \cdot S \qquad T = \sum_{d \in D} ?d.T_d \qquad T_d = !d + T \cdot T_d$$

Even smaller specification (over $\mathrm{BPA}_{\mathbf{0},\mathbf{1}}$)

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

Technische Universiteit Eindhoven University of Technology

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

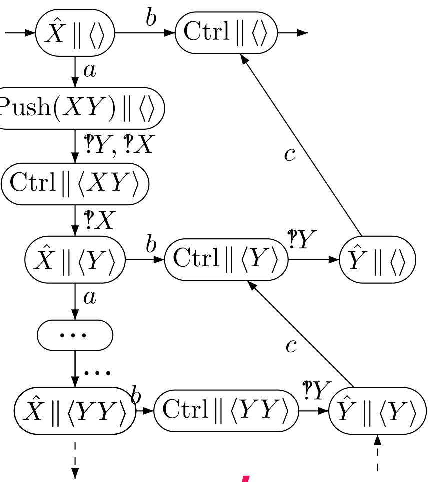
$$X = a.(X \cdot Y) + b.\mathbf{1}$$
$$Y = c.\mathbf{1}$$

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



Context-free process

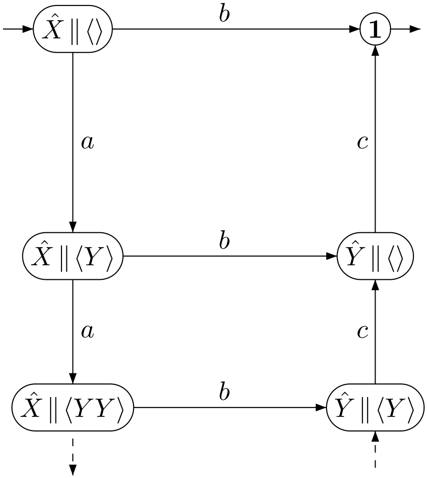
$$X = a.(X \cdot Y) + b.\mathbf{1}$$
$$Y = c.\mathbf{1}$$

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.

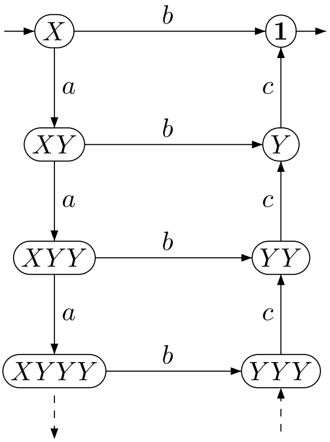


Context-Free Processes with 1-summands

Context-free process

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

Transition system

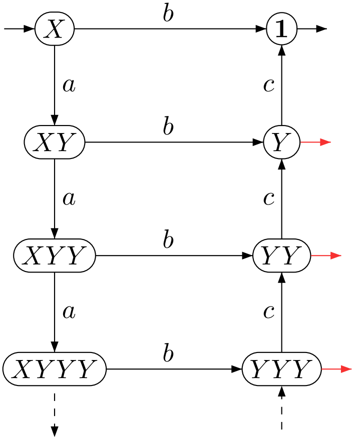


Context-Free Processes with 1-summands

Context-free process

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

Transition system

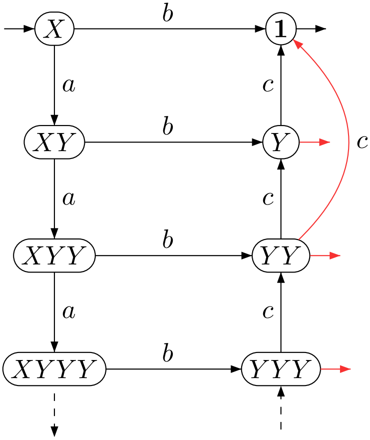


Context-Free Processes with 1-summands

Context-free process

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

Transition system



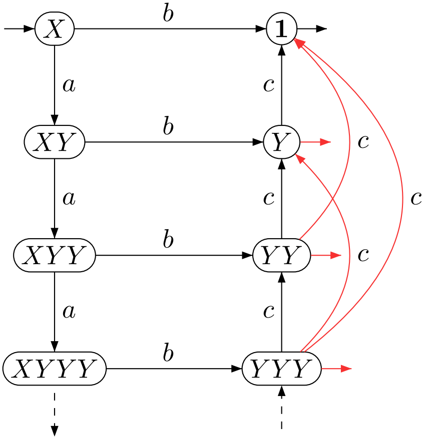
Technische Universiteit Eindhoven University of Technology

Context-Free Processes with 1-summands

Context-free process

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

Transition system



Context-Free Processes with 1-summands

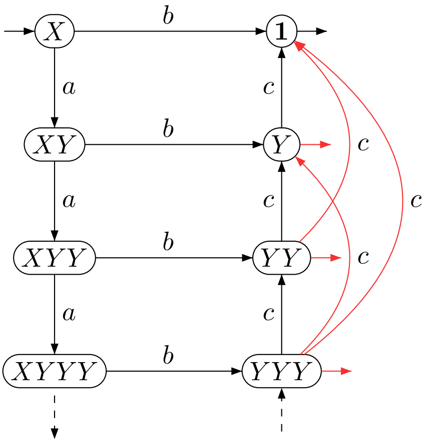
Context-free process

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

Translation adaptation

$$S = \mathbf{1} + \sum_{V \in \mathcal{V}} ?V.S \cdot !V.S$$

Transition system



Context-Free Processes with 1-summands

Context-free process

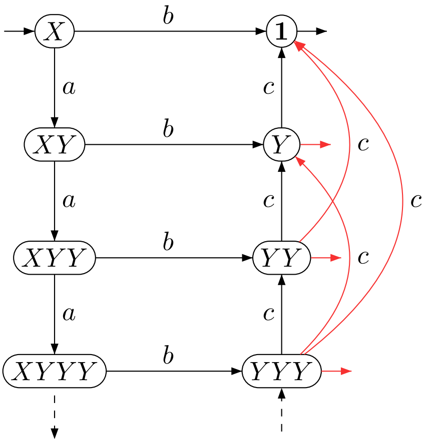
$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1} + \mathbf{1}$$

Translation adaptation

$$S = \mathbf{1} + \sum_{V \in \mathcal{V} - \mathcal{V}^{+1}} ?V.S \cdot !V.S$$
$$+ \sum_{V \in \mathcal{V}^{+1}} ?V.S \cdot (\mathbf{1} + !V.S)$$

for $\mathcal{V}^{+1} \subseteq \mathcal{V}$

Transition system



Context-Free Process Results

Unbounded branching

- Solution modulo contrasimulation
- Using partially forgetful stack, the prototypical context-free process

Without 1-summands

- Solution modulo rooted branching bisimulation
- Using normal stack, the prototypical context-free process (for BPA)

Technische Universiteit Eindhoven University of Technology

Context-Free Process Results

Unbounded branching

- Solution modulo contrasimulation
- Using partially forgetful stack, the prototypical context-free process

Without 1-summands

- Solution modulo rooted branching bisimulation
- Using normal stack, the prototypical context-free process (for BPA)

Bounded branching

- Solution modulo rooted branching bisimulation!
- Using the partially forgetful stack

Concluding Remarks

Proved Theorem

For every context-free process P *there exists a regular process* Q *such that* $P = \tau_*(\partial_*(Q \parallel S))$.

- Formalized *interaction* in the pushdown automaton
- Made communication with the stack explicit
- Introduced $\mathbf{0}$ and 1, dealt with complications
- The (partially forgetful) stack is the prototypical context-free process

Concluding Remarks

Proved Theorem

For every context-free process P there exists a regular process Q such that $P = \tau_*(\partial_*(Q \parallel S))$.

- Formalized *interaction* in the pushdown automaton
- Made communication with the stack explicit
- Introduced $\mathbf{0}$ and $\mathbf{1}$, dealt with complications
- The (partially forgetful) stack is the prototypical context-free process

Future work

- Reverse case, maybe with $\mathbf{1}$?
- Sequential composition replaced by parallel composition [EXPRESS'08]
- Queues?

Thank you!

Questions?