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

TCS seminar
Vrije Universiteit, Amsterdam / June 20, 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
- Study similarities and differences

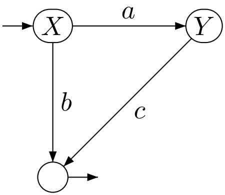
Right-linear grammar

Generates a regular language

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

(Finite) transition system

Accepts a regular language



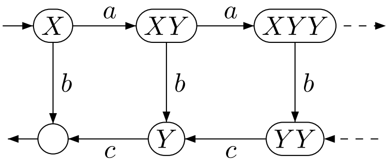
Context-Free Language

Right-linear grammar

Generates a context-free language

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

Transition system



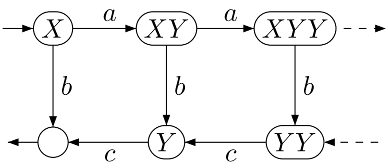
Context-Free Language

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.

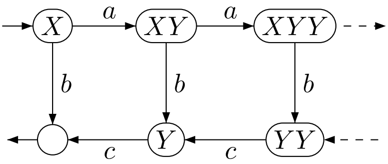
Context-Free Language

Right-linear grammar

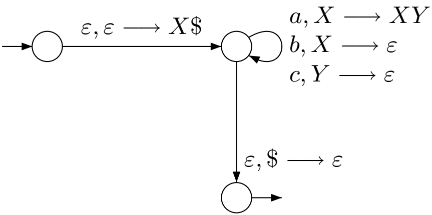
Generates a context-free language

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

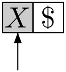
Transition system



Pushdown automaton



Stack



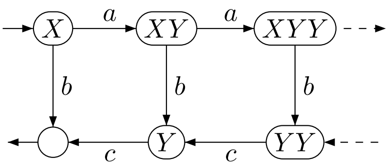
Context-Free Language

Right-linear grammar

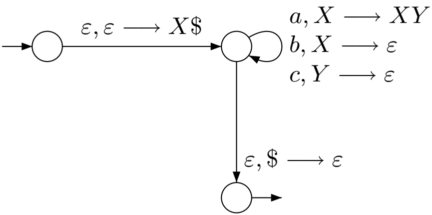
Generates a context-free language

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

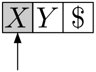
Transition system



Pushdown automaton



Stack



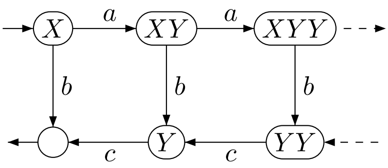
Context-Free Language

Right-linear grammar

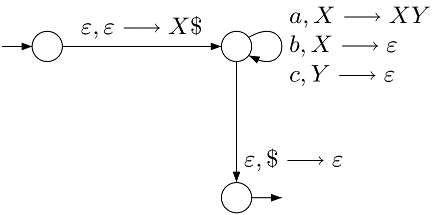
Generates a context-free language

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

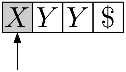
Transition system



Pushdown automaton



Stack



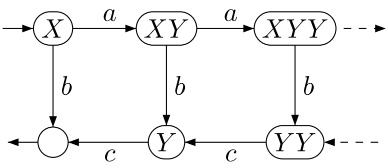
Context-Free Language

Right-linear grammar

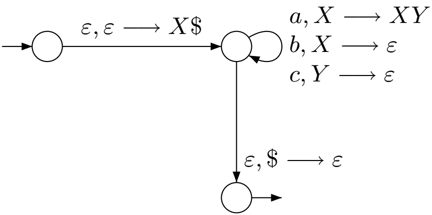
Generates a context-free language

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

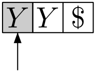
Transition system



Pushdown automaton



Stack



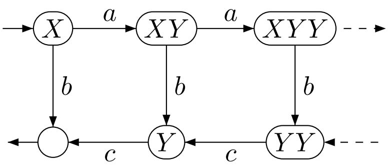
Context-Free Language

Right-linear grammar

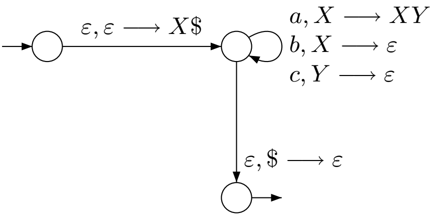
Generates a context-free language

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

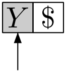
Transition system



Pushdown automaton



Stack



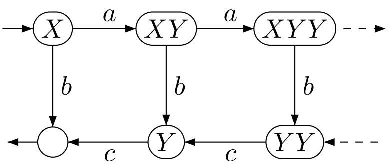
Context-Free Language

Right-linear grammar

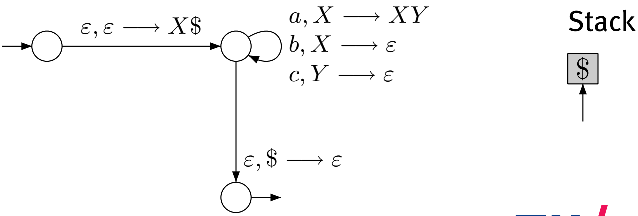
Generates a context-free language

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

Transition system



Pushdown automaton



TU/e Technische Universiteit Eindhoven University of Technology

Right-linear grammar

Generates a context-free language

$$
\begin{aligned}
X &\longrightarrow aXY \mid b \\
Y &\longrightarrow c
\end{aligned}
$$

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

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

Restrict to:
finite and guarded specifications

Technische Universiteit **Eindhoven** University of Technology

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

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

Technische Universiteit Eindhoven University of Technology

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

Technische Universiteit
Eindhoven
University of Technology

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.

Specifications

Infinite recursive specification (infinite data set)

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

TU/e Technische Universiteit Eindhoven University of Technology

Specifications

Infinite recursive specification (infinite data set)

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

Finite recursive specification over BPA

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

TU/e Technische Universiteit Eindhoven University of Technology

Specifications

Infinite recursive specification (infinite data set)

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

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 $\mathrm{BPA}_{\mathbf{0},\mathbf{1}}$)

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

Specifications

Infinite recursive specification (infinite data set)

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

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 BPA$_{\mathbf{0},\mathbf{1}}$)

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

Shorthand

$$\partial_{?,!}(P \parallel S \cdot !d_1.S \cdot \ldots \cdot !d_n.S) = P\langle d_1 \ldots d_n \rangle$$

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

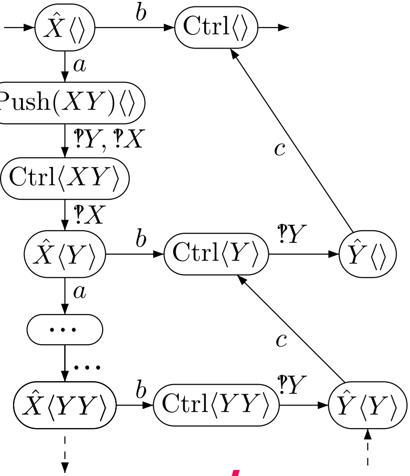
$$X = a.(X \cdot Y) + b.\mathbf{1}$$
$$Y = c.\mathbf{1}$$

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



Pushdown Automaton

Context-free process

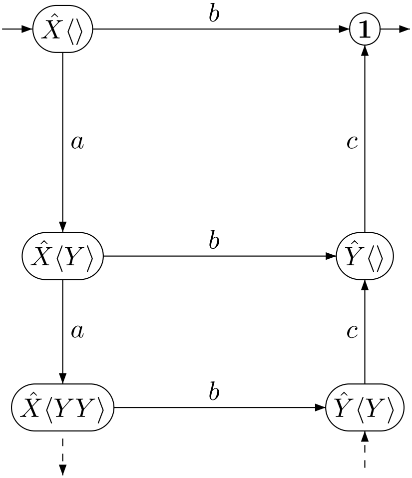
$$X = a.(X \cdot Y) + b.\mathbf{1}$$
$$Y = c.\mathbf{1}$$

Translated

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

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

…modulo rooted br. bisim.



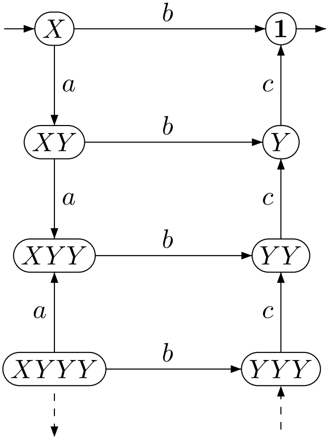
TU/e Technische Universiteit Eindhoven University of Technology

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

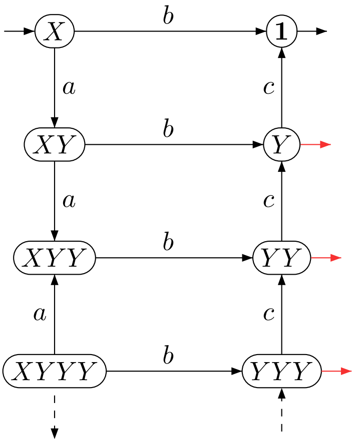


Context-Free Processes with 1-summands

Context-free process

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

Transition system



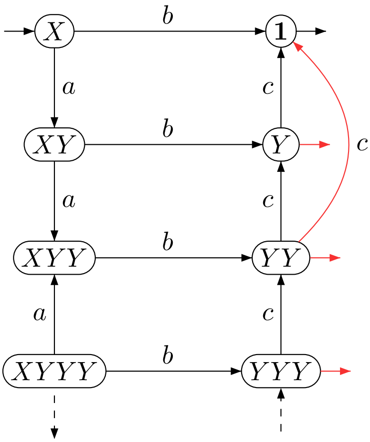
Technische Universiteit Eindhoven University of Technology

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



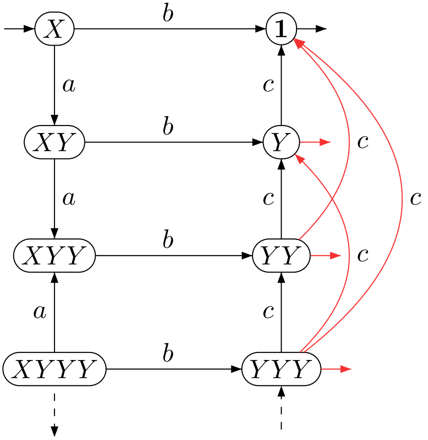
TU/e Technische Universiteit Eindhoven University of Technology

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

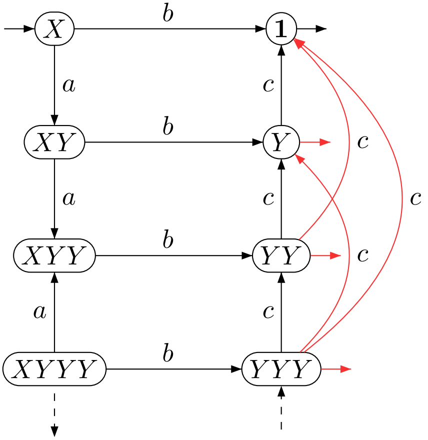
Context-free process

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

Translation adaptation

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

Transition system



Context-free process

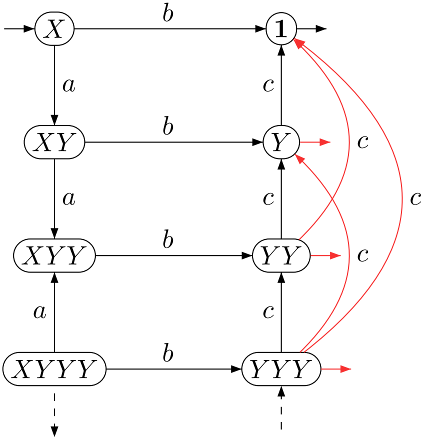
$$X = a.(X \cdot Y) + b.\mathbf{1},$$
$$Y = c.\mathbf{1} + \mathbf{1}$$

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)

TU/e 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

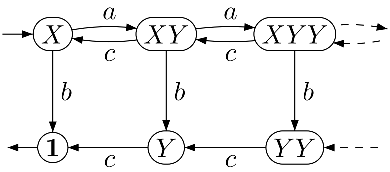
- Solution modulo rooted branching bisimulation!
- Using the partially forgetful stack

TU/e Technische Universiteit Eindhoven University of Technology

Recursive specification over BPP

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

Transition system

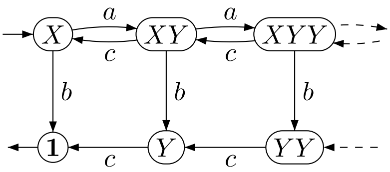


Basic Parallel Processes

Recursive specification over BPP

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

Transition system



The bag

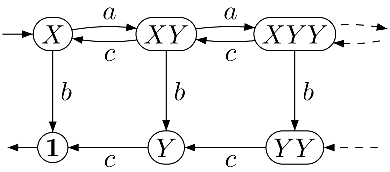
$$B = \mathbf{1} + \sum_{d \in D} ?_i d.(B \parallel !_o d.\mathbf{1})$$

Basic Parallel Processes

Recursive specification over BPP

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

Transition system



The bag

$$B = \mathbf{1} + \sum_{d \in D} ?_i d.(B \parallel !_o d.\mathbf{1})$$

Theorem

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

Parallel Pushdown Automaton

Basic parallel process

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

Translated

$$\hat{X} = a.\text{Push}(XY) + b.\text{Push}(\emptyset)$$
$$\hat{Y} = c.\text{Push}(\emptyset)$$

$$\text{Push}(\emptyset) = \text{Ctrl}$$
$$\text{Push}(X\xi) = !_i X.\text{Push}(\xi)$$
$$\text{Ctrl} = \sum_{V \in \mathcal{V}} ?_o V.(\hat{V} + !_i V.\text{Ctrl})$$

$$B = \mathbf{1} + \sum_{V \in \mathcal{V} - \mathcal{V}^{+1}} ?_i V.(B \parallel !_o V.\mathbf{1}) + \sum_{V \in \mathcal{V}^{+1}} ?_i V.(B \parallel (!_o V.\mathbf{1} + \mathbf{1}))$$

Parallel Pushdown Automaton

Basic parallel process

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

Translated

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

$$\mathrm{Push}(\emptyset) = \mathrm{Ctrl}$$
$$\mathrm{Push}(X\xi) = !_i X.\mathrm{Push}(\xi)$$
$$\mathrm{Ctrl} = \sum_{V \in \mathcal{V}} ?_o V.(\hat{V} + !_i V.\mathrm{Ctrl})$$

$$B = \mathbf{1} + \sum_{V \in \mathcal{V} - \mathcal{V}^{+\mathbf{1}}} ?_i V.(B \parallel !_o V.\mathbf{1}) + \sum_{V \in \mathcal{V}^{+\mathbf{1}}} ?_i V.(B \parallel (!_o V.\mathbf{1} + \mathbf{1})$$

Technische Universiteit Eindhoven University of Technology

Basic Parallel Process Results

All cases

- Solution modulo rooted branching bisimulation
- Using partially forgetful bag, the prototypical basic parallel process

Remark

Using 1 made the solution easier

Concluding Remarks

Proved Theorems

- Made communication explicit
- Introduced 0 and 1, dealt with complications
- The (partially forgetful) stack is the prototypical context-free process
- The (partially forgetful) bag is the prototypical basic parallel process

Concluding Remarks

Proved Theorems

- Made communication explicit
- Introduced 0 and 1, dealt with complications
- The (partially forgetful) stack is the prototypical context-free process
- The (partially forgetful) bag is the prototypical basic parallel process

Future work

- Reverse case, maybe with 1?
- Queues?

TU/e Technische Universiteit Eindhoven University of Technology

Questions?

Technische Universiteit
Eindhoven
University of Technology