Variants of Collapsible Pushdown Systems

Paweł Parys

University of Warsaw

Higher order pushdown systems/automata [Maslov 74, 76]

A 1-stack is an ordinary stack. A 2-stack
(resp. (n + 1)-stack) is a stack of 1-stacks (resp. n-stack).

Operations on 2-stacks: s_i are 1-stacks. Top of stack is on right.

push_2 : $[s_1 \ldots s_{i-1} s_i]$ -> $[s_1 \ldots s_{i-1} s_i\, s_i]$

pop_2 : $[s_1 \ldots s_{i-1} s_i]$ -> $[s_1 \ldots s_{i-1}]$

$\mathrm{push}_1 x$: $[s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1} a_j]]$ -> $[s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1} a_j\, x]]$

pop_1 : $[s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1} a_j]]$ -> $[s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1}]]$

An **order-n PDA** has an order-n stack, and
has push_i and pop_i for each $1 \leq i \leq n$.

Collapsible pushdown systems/automata [Hague, Murawski, Ong, Serre 08]

Collapsible PDS are an extension of a higher-order PDS

$push_1$(x) pushes not only the x symbol, but also a fresh marker

new operation: $collapse_k$ – removes all those (k-1)-stacks from the topmost k-stack, which contain the marker present in the topmost symbol

Collapsible pushdown systems/automata [Hague, Murawski, Ong, Serre 08]

Collapsible PDS are an extension of a higher-order PDS

$push_1(x)$ pushes not only the x symbol, but also a fresh marker

new operation: $collapse_k$ – removes all those (k-1)-stacks from the topmost k-stack, which contain the marker present in the topmost symbol

Collapsible PDS are equiexpressive with higher-order recursion schemes!

Trees generated by collapsible PDS have decidable MSO theory!

First contribution:

We compare three possible ways (definitions)
how a collapsible pushdown system can generate a tree
- we show that these three ways are equivalent.

How collapsible pushdown systems generate trees?

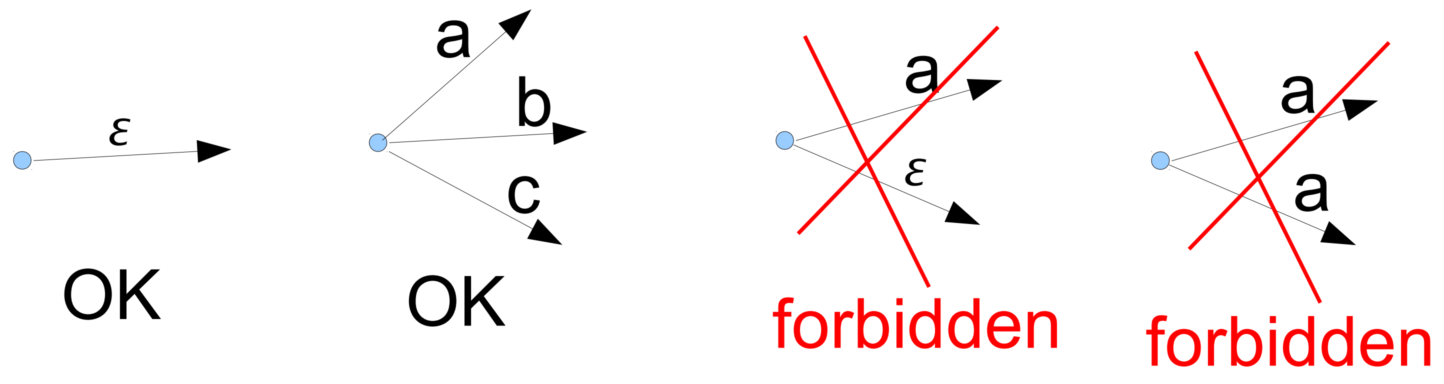
(we consider potentially infinite trees with labels on edges)

Classical definition:

- every transition reads a label, or ε (nothing)
- we consider only **deterministic** systems

From every configuration we have:

- one ε-transition, or
- only non-ε-transitions, every labeled by a different letter

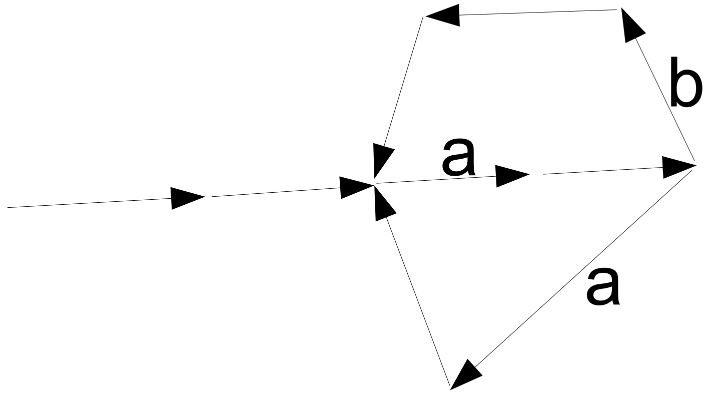


How collapsible pushdown systems generate trees?

(we consider potentially infinite trees with labels on edges)

Classical definition:

- we consider only **deterministic** systems
- we unfold the configuration graph into a tree
- we contract ε-edges



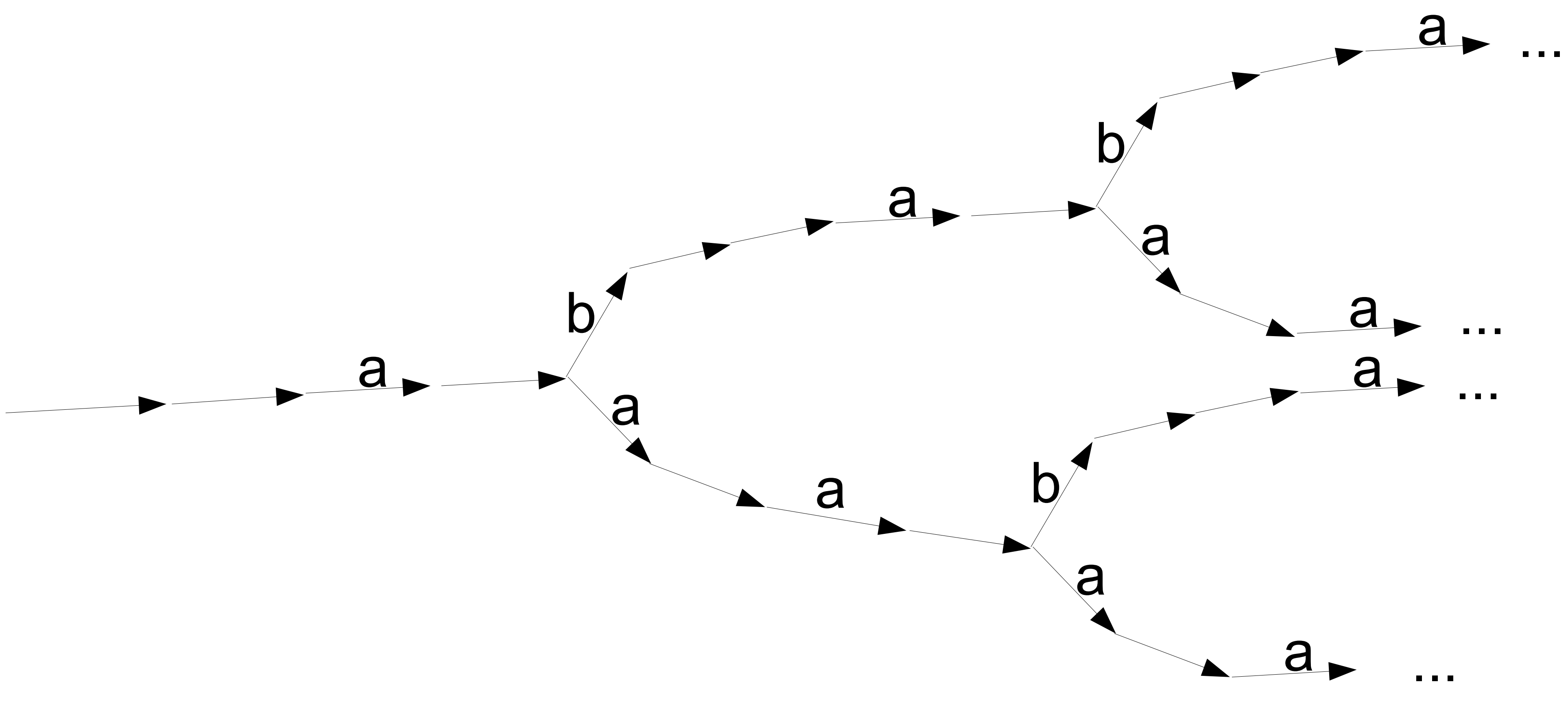
(typically the configuration graph is infinite)

How collapsible pushdown systems generate trees?

(we consider potentially infinite trees with labels on edges)

Classical definition:

- we consider only **deterministic** systems
- we unfold the configuration graph into a tree
- we contract ε-edges

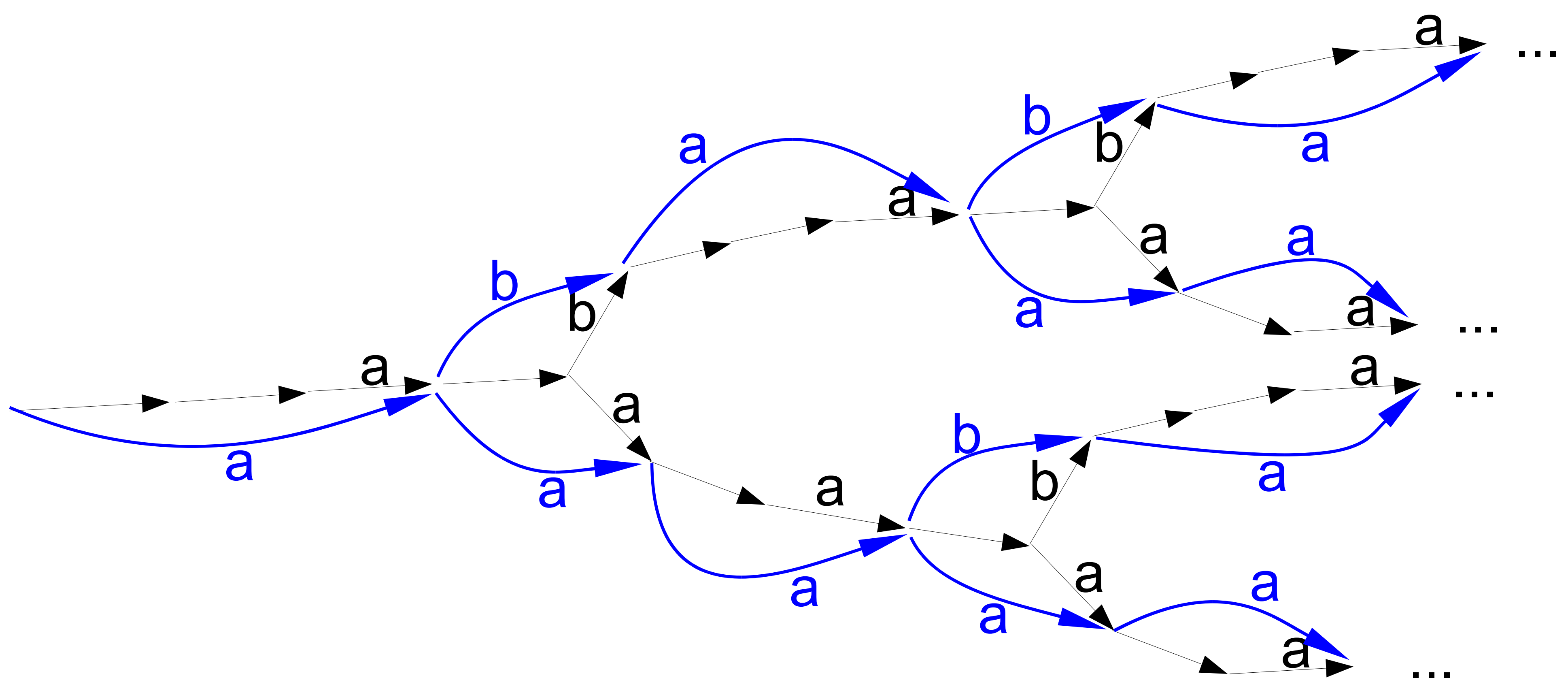


How collapsible pushdown systems generate trees?

(we consider potentially infinite trees with labels on edges)

Classical definition:

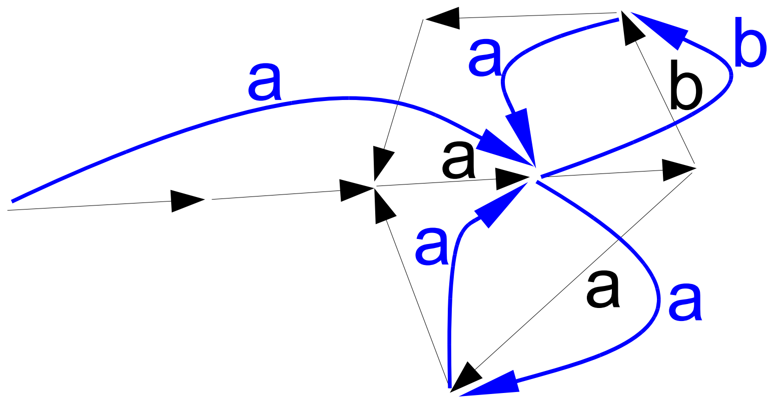
- we consider only **deterministic** systems
- we unfold the configuration graph into a tree
- we contract ε-edges



How collapsible pushdown systems generate trees?

Classical definition - equivalently:

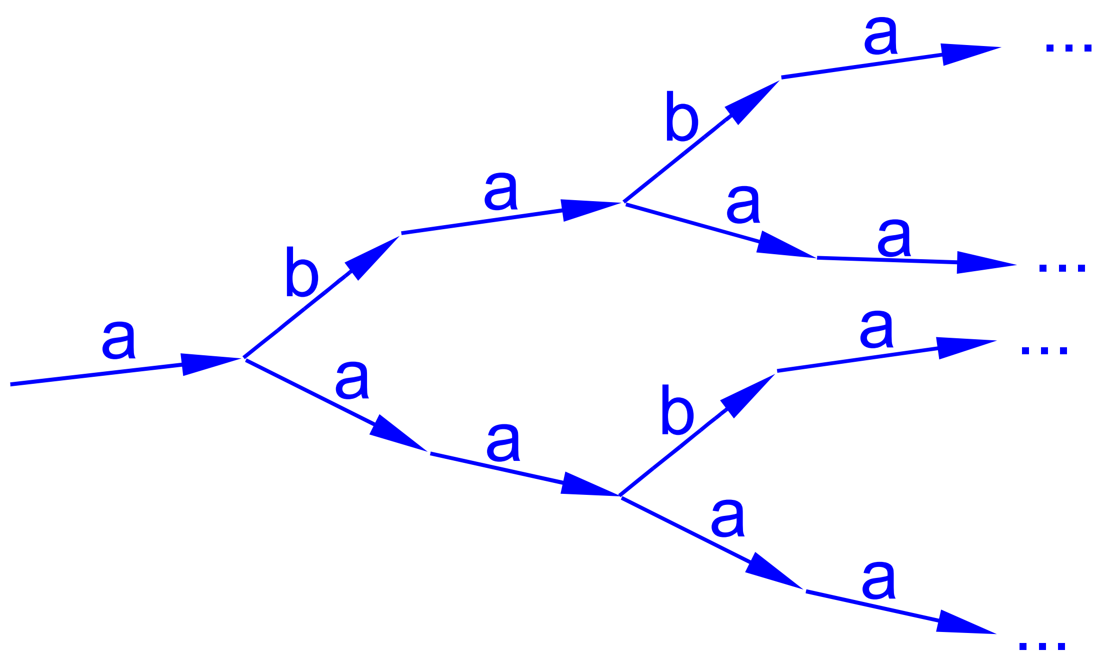
- we consider only **deterministic** systems
- we first make the ε-closure of the configuration graph
- after that we unfold the graph into a tree



How collapsible pushdown systems generate trees?

Classical definition - equivalently:

- we consider only **deterministic** systems
- we first make the ε-closure of the configuration graph
- after that we unfold the graph into a tree



How CPS generate trees? – second definition

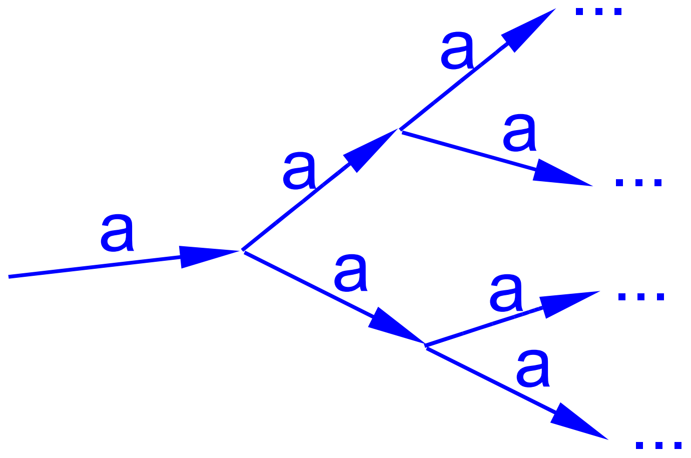
But we can also use nondeterministic systems:
- we consider any system, possibly nondeterministic
- we make the ε-closure of the configuration graph
- after that we unfold the graph into a tree

How CPS generate trees? – second definition

But we can also use nondeterministic systems:

- we consider any system, possibly nondeterministic
- we make the ε-closure of the configuration graph
- after that we unfold the graph into a tree

Now we can obtain some new trees:



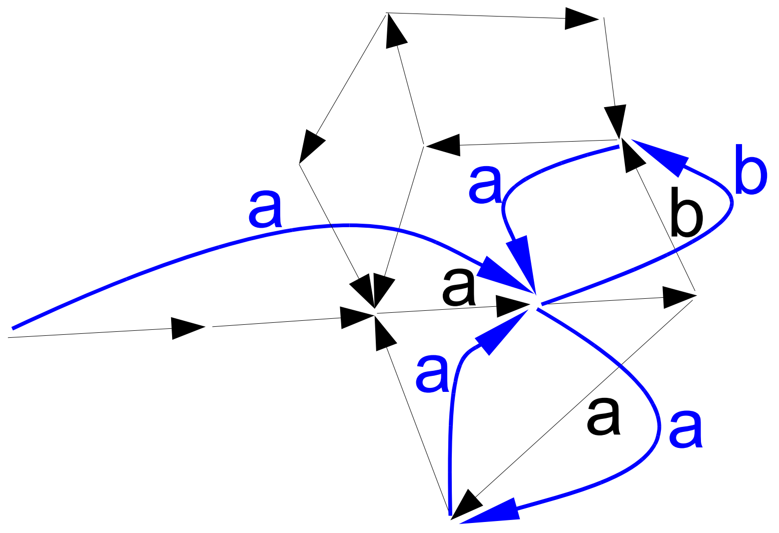
such that from every node there is at most one edge labeled by each letter

This tree is “nondeterministic”.
What if we restrict ourselves to “deterministic” trees?

How CPS generate trees? – second definition

What if we restrict ourselves to “deterministic” trees?

A nondeterministic system can produce a “deterministic” tree.



We can have “big” parts having only ε-transitions.

Determinization

Question: we have a “deterministic” tree
generated by a nondeterministic CPS of some level n.
Can it be be generated by some deterministic CPS of level n?

Determinization

Question: we have a “deterministic” tree
generated by a nondeterministic CPS of some level n.
Can it be be generated by some deterministic CPS of level n?

YES – we have determinization

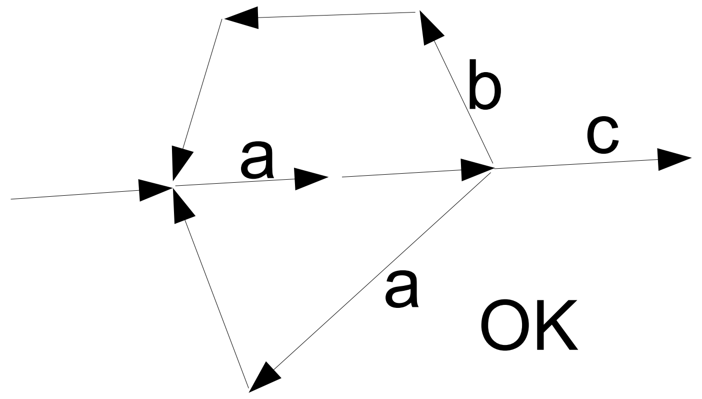
Theorem 1. Every “deterministic” tree generated by a CPS of level n is also generated by a deterministic CPS of level n.

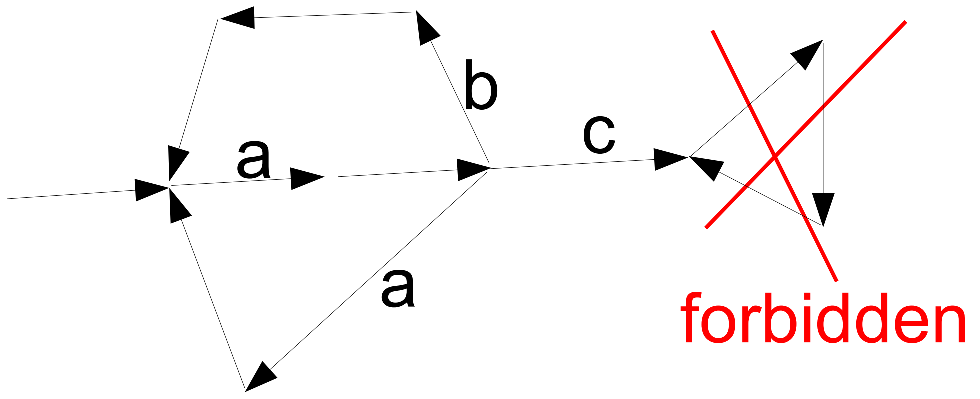
Determinization

Question: we have a “deterministic” tree
generated by a nondeterministic CPS of some level n.
Can it be be generated by some deterministic CPS of level n?

YES – we have determinization

Theorem 1. Every “deterministic” tree generated by a CPS of level n is also generated by a deterministic CPS of level n. Moreover: its configuration graph does not have (finite or infinite) branches which does not read any letter.





Determinization

Question: we have a “deterministic” tree
generated by a nondeterministic CPS of some level n.
Can it be be generated by some deterministic CPS of level n?

YES – we have determinization

Theorem 1. Every “deterministic” tree generated by a CPS of level n is also generated by a deterministic CPS of level n.
Moreover: its configuration graph does not have (finite or infinite) branches which does not read any letter.
Moreover: this deterministic CPS can be effectively constructed (its size grows (n-1)-times exponentially).

On the stack we have to remember some information about the stack below.
Example: the automaton can preform 5 times a pop from a configuration,
and if it sees “x” on the stack, it reads “b”.
We have to remember if there is an “x” 5 positions below the top of the stack

Determinization

<u>Consequences</u> of determinization:

“Deterministic automata are simpler.”

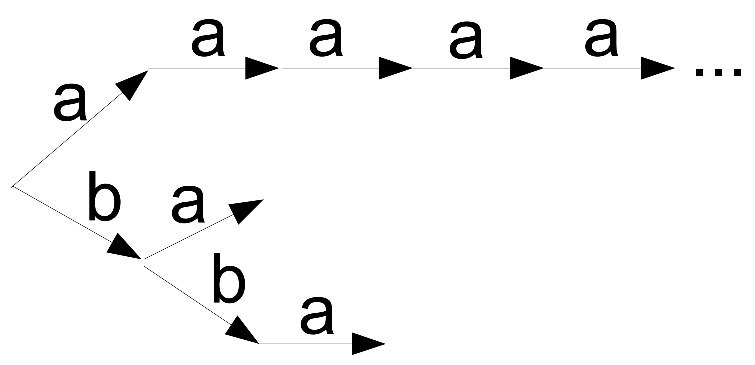
➔ simulation – one can just run the system to see what letter can be read next (impossible for nondeterministic CPS)

➔ easier for proofs – it's easier to prove that a tree is not generated by a deterministic CPS, than that it is not generated by any CPS (e.g. our proof that the CPS graph hierarchy is strict [MFCS'12] simplifies significantly by using this result)

How CPS generate trees? – third definition

Consider a deterministic, word-accepting CPS
(i.e. we have a set of accepting states)
Take a tree consisting of all prefixes of accepted words.

<u>Example</u>
automaton accepts: ba, bba, a, aa, aaa, aaaa, ...



How CPS generate trees? – third definition

Consider a deterministic, word-accepting CPS
(i.e. we have a set of accepting states)
Take a tree consisting of all prefixes of accepted words.

Theorem 2.
Every such tree is also generated by a CPS in a classical sense
(and vice versa, which is obvious).

Difficulty: when we are in a configuration from which we will never accept, we have to stop immediately (without reading more letters).

How CPS generate trees? – third definition

Consider a deterministic, word-accepting CPS
(i.e. we have a set of accepting states)
Take a tree consisting of all prefixes of accepted words.

Theorem 2.
Every such tree is also generated by a CPS in a classical sense
(and vice versa, which is obvious).
Equivalently:
every word-language recognized by a deterministic CPS of level n is also recognized by a deterministic CPS of level n such that from every reachable configuration there is an accepting run.

How CPS generate trees? – third definition

Consider a deterministic, word-accepting CPS
(i.e. we have a set of accepting states)
Take a tree consisting of all prefixes of accepted words.

<u>Theorem 2.</u>
Every such tree is also generated by a CPS in a classical sense
(and vice versa, which is obvious).
<u>Equivalently:</u>
every word-language recognized by a deterministic CPS of level n is also recognized by a deterministic CPS of level n such that from every reachable configuration there is an accepting run.
Moreover: this CPS can be effectively constructed
(its size grows (n-1)-times exponentially).

Note that:
- Theorem 2 is slightly easier than Theorem1 (about determinization),
- Theorems 1 and 2 hold also for (non-collapsible) higher-order pushdown systems
- word-accepting CPS cannot be determinized.

Notice:
Proofs of Theorems 1 and 2 can be quite easily deduced from a recent paper:
A. Carayol, O. Serre. “Collapsible Pushdown Automata and Labeled Recursion Schemes. Equivalence, Safety and Effective Selection” (LICS 2012)

Our proofs are completely different
(and were obtained independently).

The word hierarchy of CPS is infinite

$\mathscr{L}_n$ – languages recognized by (nondeterministic) CPS of level n

<u>Theorem 3.</u>

$\mathscr{L}_{2n+1}$ is strictly greater than $\mathscr{L}_n$.
So the word-languages hierarchy of CPS is infinite.

The separating language is:

$$\{ a^k\, b^{\overbrace{2^{2^{2^{\cdots 2^{k}}}}}^{2n}} : k \in \mathbb{N} \}$$

The word hierarchy of CPS is infinite

$\mathscr{L}_n$ – languages recognized by (nondeterministic) CPS of level n

<u>Theorem 3.</u>

$\mathscr{L}_{2n+1}$ is strictly greater than $\mathscr{L}_n$.
So the word-languages hierarchy of CPS is infinite.

Theorem 3 is a consequence of Lemma 4:

<u>Lemma 4.</u>
Let S be a (nondeterministic) CPS of level n.
Then there exists an accepting run of S of length at most

$$\overbrace{2^{2^{2^{\cdots^{2}}}}}^{2n-1}{}^{8|Q||Q||\Gamma|}$$

The word hierarchy of CPS is infinite

$\mathscr{L}_n$ – languages recognized by (nondeterministic) CPS of level n

Theorem 3.

$\mathscr{L}_{2n+1}$ is strictly greater than $\mathscr{L}_n$.
So the word-languages hierarchy of CPS is infinite.

Note:

- We don't know whether $\mathscr{L}_{n+1}$ is strictly greater than $\mathscr{L}_n$.
- We can deduce that the tree and graph hierarchies are infinite (but it is already known even that each their level is different [MFCS 2012]).

Another contribution – reachability algorithm

Input: (nondeterministic) CPS of level n, a set of states F
Question: is there reachable a configuration with a state in F?
(equivalently: emptiness of the recognized language)

We show a new (rather simple) algorithm solving this problem in (n-1)-EXPTIME.

Another contribution – reachability algorithm

Input: (nondeterministic) CPS of level n, a set of states F
Question: is there reachable a configuration with a state in F?
(equivalently: emptiness of the recognized language)

We show a new (rather simple) algorithm solving this problem in (n-1)-EXPTIME.

Note:
- the same complexity can be achieved by previously known algorithms (for deciding mu-calculus)
- the algorithm is very similar to the one described (independently) in: C. Broadbent, A. Carayol, M. Hague, O. Serre. “A Saturation Method for Collapsible Pushdown Systems” (ICALP 2012)

Summary

1) The three presented methods of generating deterministic trees by CPS are equivalent.

2) The word-languages hierarchy of CPS is infinite.

3) Algorithm for reachability in CPS.

Related open problems

1) Is the word-languages hierarchy of CPS strict (are every two levels different)?
2) Are all these languages context-sensitive?