First-Order Logic on CPDA Graphs

Paweł Parys
University of Warsaw

Higher order pushdown systems (automata) - HOPDS

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

Operations on 2-stacks: (s_i are 1-stacks, top of stack is on right)

$push_1 x$: $[s_1 ... s_{i-1} [a_1 ... a_{j-1} a_j]] \rightarrow [s_1 ... s_{i-1} [a_1 ... a_{j-1} a_j\, x]]$

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

$push_2$: $[s_1 ... s_{i-1} s_i] \rightarrow [s_1 ... s_{i-1} s_i\, s_i]$

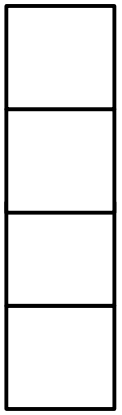
pop_2 : $[s_1 ... s_{i-1} s_i] \rightarrow [s_1 ... s_{i-1}]$

An **order-n PDS** has an order-n stack, and has $push_i$ and pop_i for each $i \in \{1,...,n\}$.

Higher order pushdown systems (automata) - HOPDS

Example: language $\{a^n b^n c^n\}$

- on each “a” put a symbol on the stack

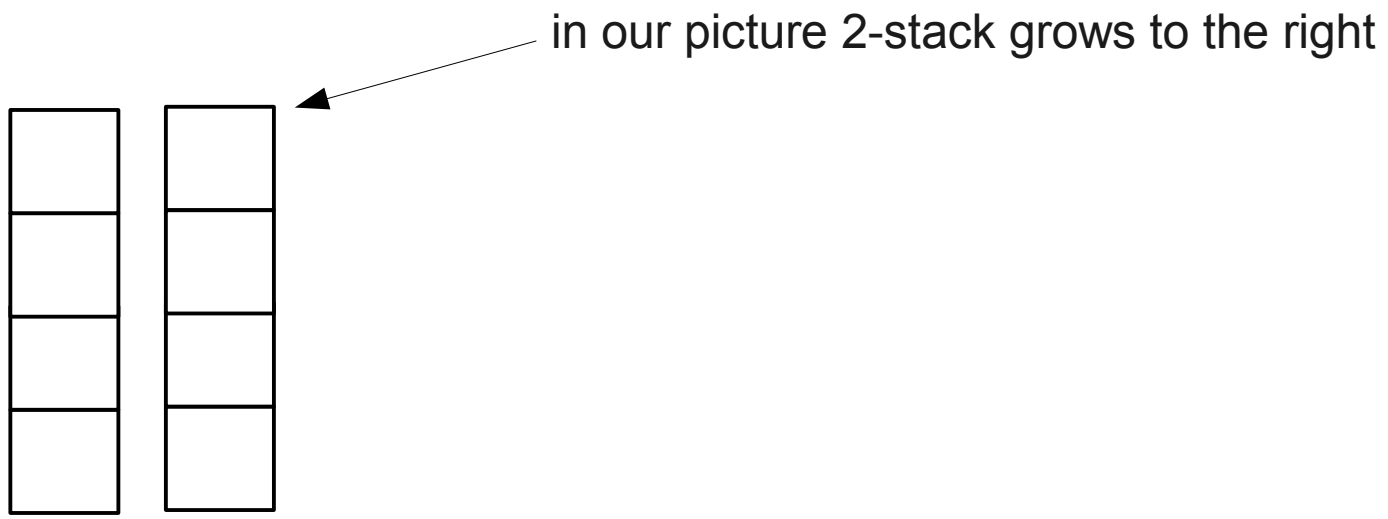


a a a a

Higher order pushdown systems (automata) - HOPDS

Example: language $\{a^n b^n c^n\}$

- on each “a” put a symbol on the stack
- copy the 1-stack (the $push_2$ operation)

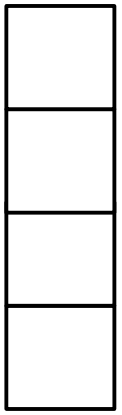


a a a a b

Higher order pushdown systems (automata) - HOPDS

Example: language $\{a^n b^n c^n\}$

- on each “a” put a symbol on the stack
- copy the 1-stack (the $push_2$ operation)
- on each “b” on input remove one symbol from the stack



a a a a b b b b

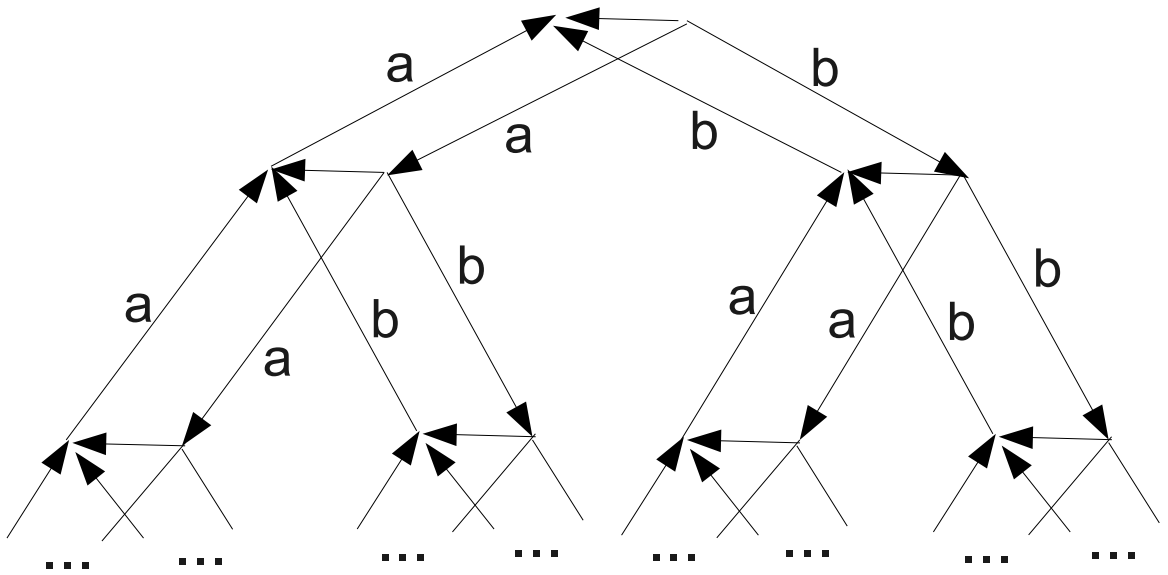
Higher order pushdown systems (automata) - HOPDS

Example: language $\{a^n b^n c^n\}$

- on each “a” put a symbol on the stack
- copy the 1-stack (the $push_2$ operation)
- on each “b” on input remove one symbol from the stack
- on each “c” on input remove one symbol from the stack

a a a a b b b b c c c c

Configuration graph



$(?, q_1) \xrightarrow{a} (push_1 a, q_1)$

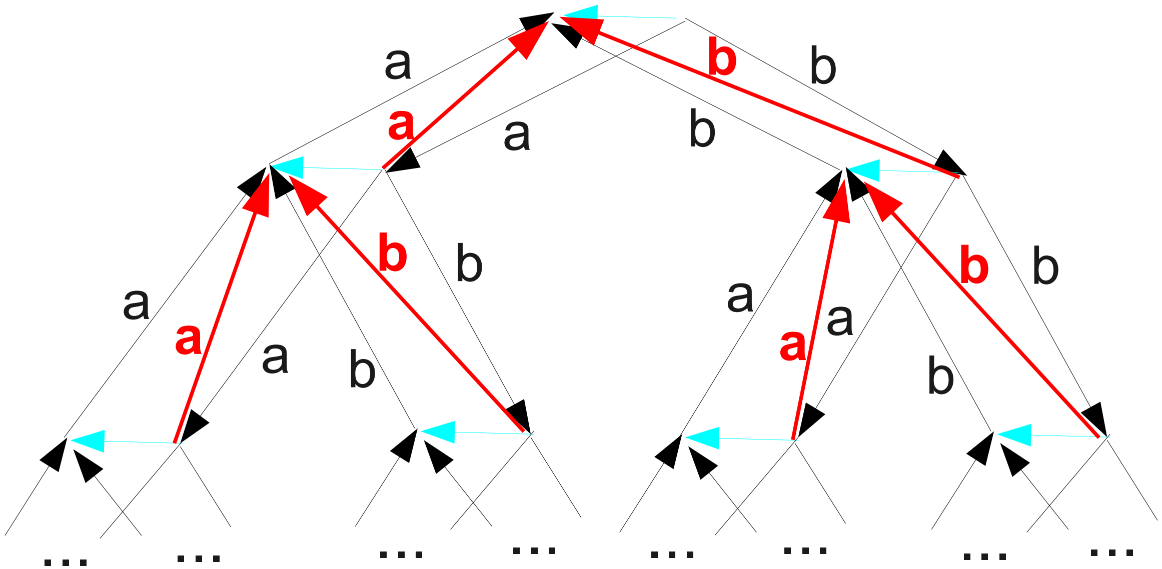
$(?, q_1) \xrightarrow{b} (push_1 b, q_1)$

$(?, q_1) \xrightarrow{\varepsilon} (no\text{-}op, q_2)$

$(a, q_2) \xrightarrow{a} (pop_1, q_2)$

$(b, q_2) \xrightarrow{b} (pop_1, q_2)$

ε-closure of the configuration graph

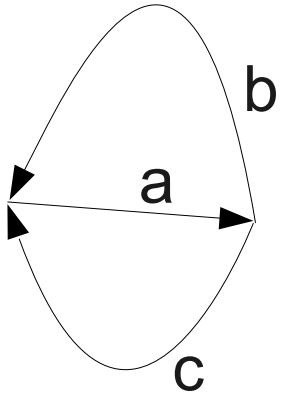


$$(?, q_1) \xrightarrow{a} (\text{push}_1 a, q_1)$$
$$(?, q_1) \xrightarrow{b} (\text{push}_1 b, q_1)$$
$$(?, q_1) \xrightarrow{\varepsilon} (\text{no-op}, q_2)$$
$$(a, q_2) \xrightarrow{a} (\text{pop}_1, q_2)$$
$$(b, q_2) \xrightarrow{b} (\text{pop}_1, q_2)$$

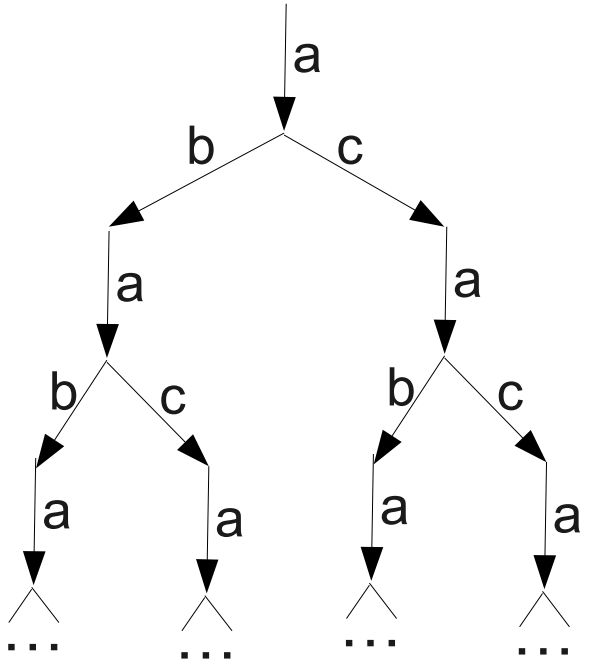
– remove epsilon-edges (blue)
– add edges for sequences of epsilons ended by a letter (red)

configuration tree
(unfolding of the configuration graph)

a graph:



its unfolding:



(a single configuration is represented by several nodes of the tree)

Logics

We are interested in decidability of logics (FO, MSO) on configuration graphs/trees:
INPUT: a pushdown system S and a formula φ
QUESTION: is φ satisfied in the configuration graph/tree of S?

MSO logic = FO logic + quantification over sets of nodes

It is an expressive logic, we can e.g. write that:
– a node with state q is reachable
– there exists a loop of odd length
– there exists a path containing infinitely many “a”

MSO logic on HOPDS graphs/trees

We are interested in decidability of logics (FO, MSO) on configuration graphs/trees:

INPUT: a pushdown system S and a formula φ

QUESTION: is φ satisfied in the configuration graph/tree of S?

[Caucal 2002]

ε-closures of n-PDS graphs = graphs that can be MSO-interpreted in configuration trees of (n-1)-PDS

MSO logic on HOPDS graphs/trees

We are interested in decidability of logics (FO, MSO) on configuration graphs/trees:

INPUT: a pushdown system S and a formula φ

QUESTION: is φ satisfied in the configuration graph/tree of S?

[Caucal 2002]

ε-closures of n-PDS graphs = graphs that can be MSO-interpreted in configuration trees of (n-1)-PDS

Corollary: an MSO-formula over an n-PDS graph can be translated to a formula over an (n-1)-PDS tree.

Fact (nontrivial): an MSO-formula over the unfolding of a graph G (over an n-PDS tree) can be translated to a formula over G (over an n-PDS graph).

Thus MSO logic over HOPDA graphs and trees is decidable.

Moreover, these graphs have a nice, logical, machine-independent characterization.

MSO logic on HOPDS graphs/trees

We are interested in decidability of logics (FO, MSO) on configuration graphs/trees:

INPUT: a p

QUESTION graph/tree of S?

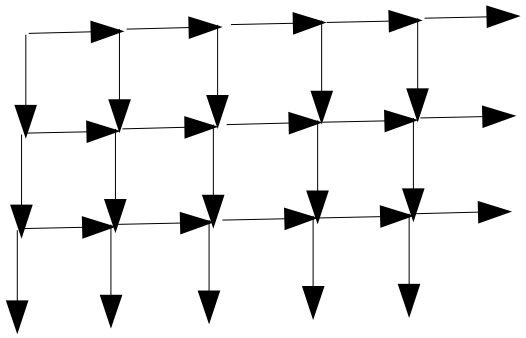
[Caucal 20

ε-closures terpreted

n-PDS gra 1)-PDS

Corollary: a can be translated t

Of course there exist graphs having undecidable MSO logic, e.g. a grid:



Fact (nontrivial): an MSO formula over the unfolding of a graph G (over an n-PDS tree) can be translated to a formula over G (over an n-PDS graph).

Thus MSO logic over HOPDA graphs and trees is decidable.

Moreover, these graphs have a nice, logical, machine-independent characterization.

Stack vs. recursion

programs with stack = recursive programs

What if we allow higher-order recursion (functions taking functions as parameters)?

Stack vs. recursion

programs with stack = recursive programs

What if we allow higher-order recursion (functions taking functions as parameters)?

[Knapik, Niwiński, Urzyczyn 2002]

n-HOPDS trees = trees generated by **safe** recursion schemes of order n

programs without variables ranging over infinite domains

a syntactic restriction (functions of order k cannot contain values of order <k)

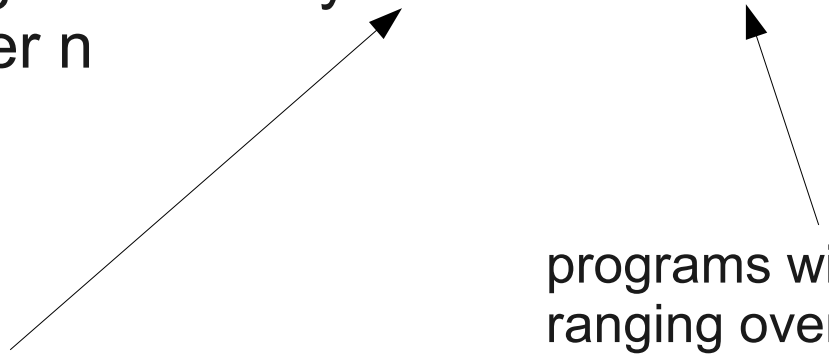
Stack vs. recursion

programs with stack = recursive programs

What if we allow higher-order recursion (functions taking functions as parameters)?

[Knapik, Niwiński, Urzyczyn 2002]

n-HOPDS trees = trees generated by **safe** recursion schemes of order n



programs without variables ranging over infinite domains

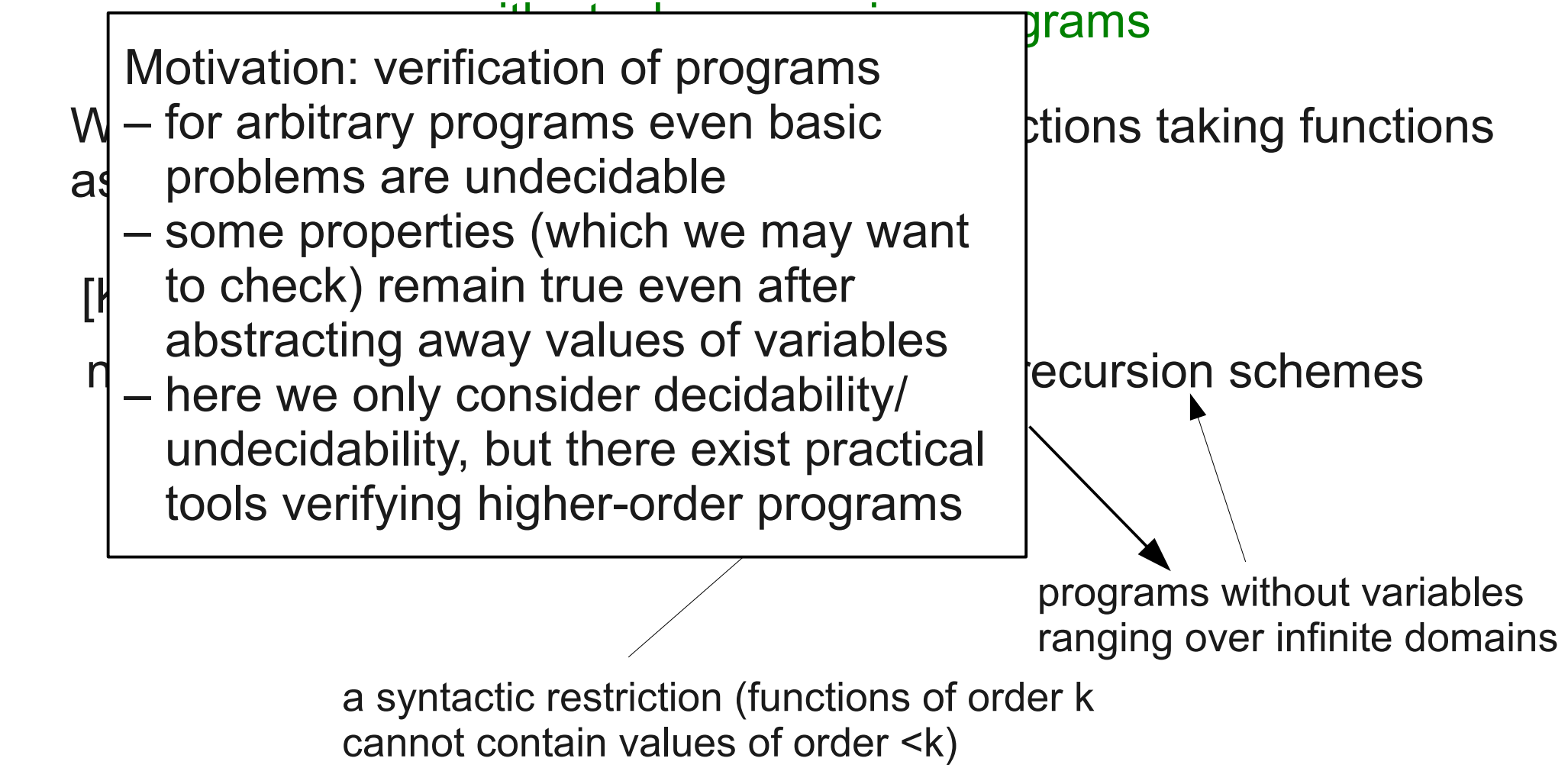
a syntactic restriction (functions of order k cannot contain values of order <k)

Maybe each recursion scheme can be converted into equivalent safe scheme?

NO! - [P. 2012] There exists a tree generated by a recursion scheme of order 2, which is not generated by a safe recursion scheme of any order.

Stack vs. recursion

Motivation: verification of programs
– for arbitrary programs even basic problems are undecidable
– some properties (which we may want to check) remain true even after abstracting away values of variables
– here we only consider decidability/ undecidability, but there exist practical tools verifying higher-order programs



Maybe each recursion scheme can be converted into equivalent safe scheme?

NO! - [P. 2012] There exists a tree generated by a recursion scheme of order 2, which is not generated by a safe recursion scheme of any order.

Stack vs. recursion

trees generated by **safe** recursion schemes of order n = configuration trees of n-HOPDS

[Hague, Murawski, Ong, Serre 2008]

trees generated by **all** recursion schemes of order n = configuration trees of Collapsible Pushdown Systems of order n

Collapsible Pushdown Systems (CPDS)

It is an extension of HOPDS:

– Each element of a 1-stack can contain a link (pointer) to some prefix of its k-stack (for any order k).
– For each k we have operation $push_{1,k}x$ – it pushes x together with a link to the topmost k-stack without its topmost (k-1)-stack.
– Higher order push operations do not modify the pointers.
– A new operation “collapse” replaces the topmost k-stack by the k-stack from the pointer contained in the topmost stack symbol.

Collapsible Pushdown Systems (CPDS)

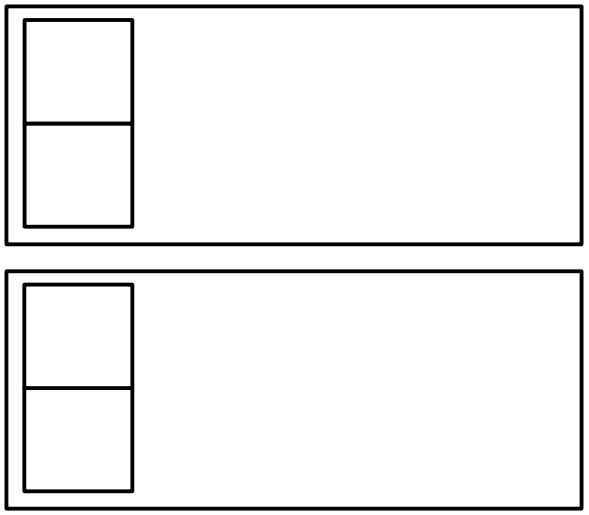
It is an extension of HOPDS:

– Each element of a 1-stack can contain a link (pointer) to some prefix of its k-stack (for any order k).

– For each k we have operation $push_{1,k}x$ – it pushes x together with a link to the topmost k-stack without its topmost (k-1)-stack.

– Higher order push operations do not modify the pointers.

– A new operation “collapse” replaces the topmost k-stack by the k-stack from the pointer contained in the topmost stack symbol.



Collapsible Pushdown Systems (CPDS)

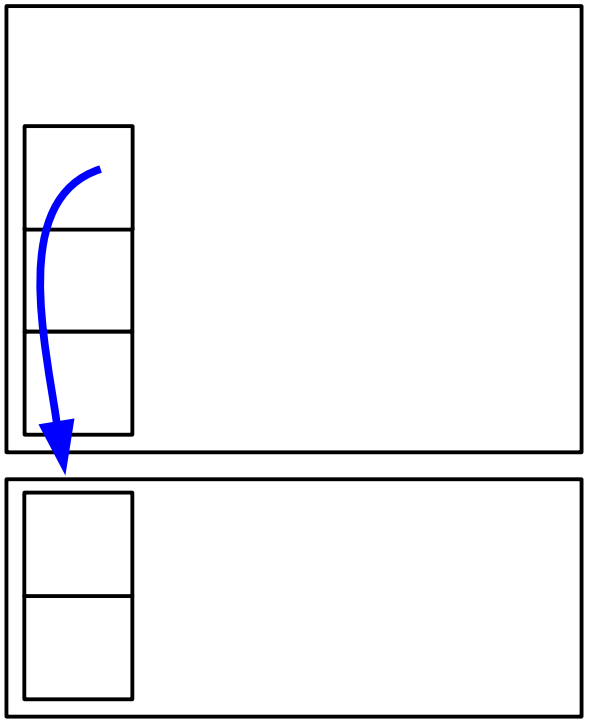
It is an extension of HOPDS:

– Each element of a 1-stack can contain a link (pointer) to some prefix of its k-stack (for any order k).

– For each k we have operation $push_{1,k}x$ – it pushes x together with a link to the topmost k-stack without its topmost (k-1)-stack.

– Higher order push operations do not modify the pointers.

– A new operation “collapse” replaces the topmost k-stack by the k-stack from the pointer contained in the topmost stack symbol.



$push_{1,3}$

Collapsible Pushdown Systems (CPDS)

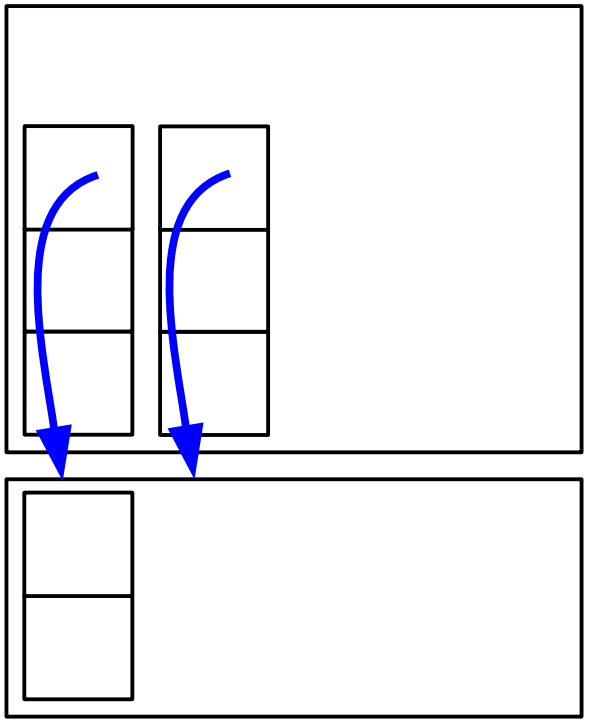
It is an extension of HOPDS:

– Each element of a 1-stack can contain a link (pointer) to some prefix of its k-stack (for any order k).

– For each k we have operation $push_{1,k}x$ – it pushes x together with a link to the topmost k-stack without its topmost (k-1)-stack.

– Higher order push operations do not modify the pointers.

– A new operation “collapse” replaces the topmost k-stack by the k-stack from the pointer contained in the topmost stack symbol.



$push_{1,3}$

$push_2$

Collapsible Pushdown Systems (CPDS)

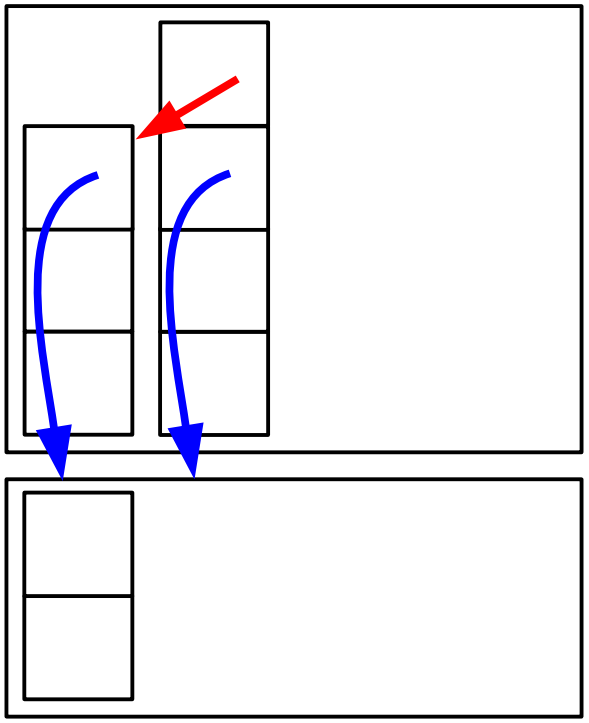
It is an extension of HOPDS:

– Each element of a 1-stack can contain a link (pointer) to some prefix of its k-stack (for any order k).

– For each k we have operation $push_{1,k}x$ – it pushes x together with a link to the topmost k-stack without its topmost (k-1)-stack.

– Higher order push operations do not modify the pointers.

– A new operation “collapse” replaces the topmost k-stack by the k-stack from the pointer contained in the topmost stack symbol.



$push_{1,3}$

$push_2$

$push_{1,2}$

Collapsible Pushdown Systems (CPDS)

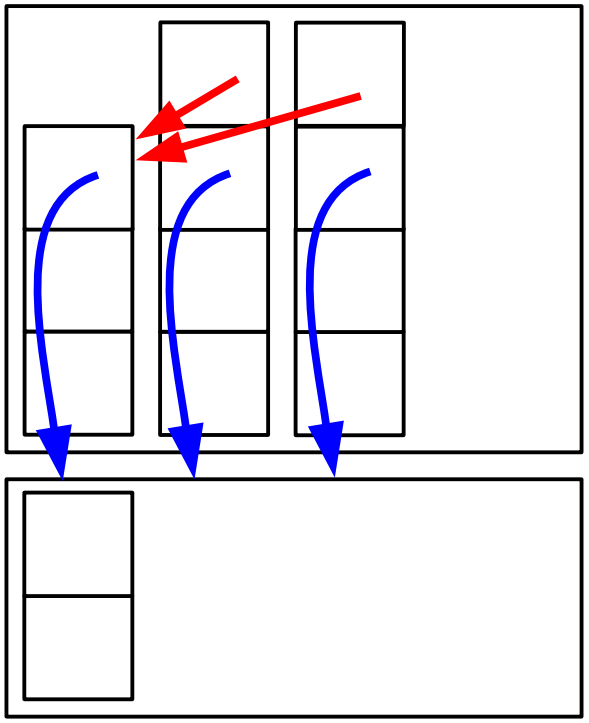
It is an extension of HOPDS:

– Each element of a 1-stack can contain a link (pointer) to some prefix of its k-stack (for any order k).

– For each k we have operation $push_{1,k}x$ – it pushes x together with a link to the topmost k-stack without its topmost (k-1)-stack.

– Higher order push operations do not modify the pointers.

– A new operation “collapse” replaces the topmost k-stack by the k-stack from the pointer contained in the topmost stack symbol.



$push_{1,3}$
$push_2$
$push_{1,2}$
$push_2$

Collapsible Pushdown Systems (CPDS)

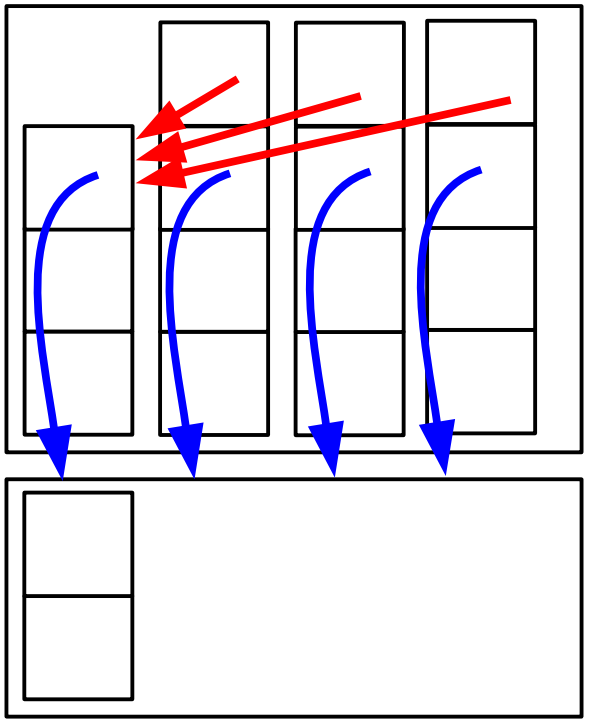
It is an extension of HOPDS:

– Each element of a 1-stack can contain a link (pointer) to some prefix of its k-stack (for any order k).

– For each k we have operation $push_{1,k}x$ – it pushes x together with a link to the topmost k-stack without its topmost (k-1)-stack.

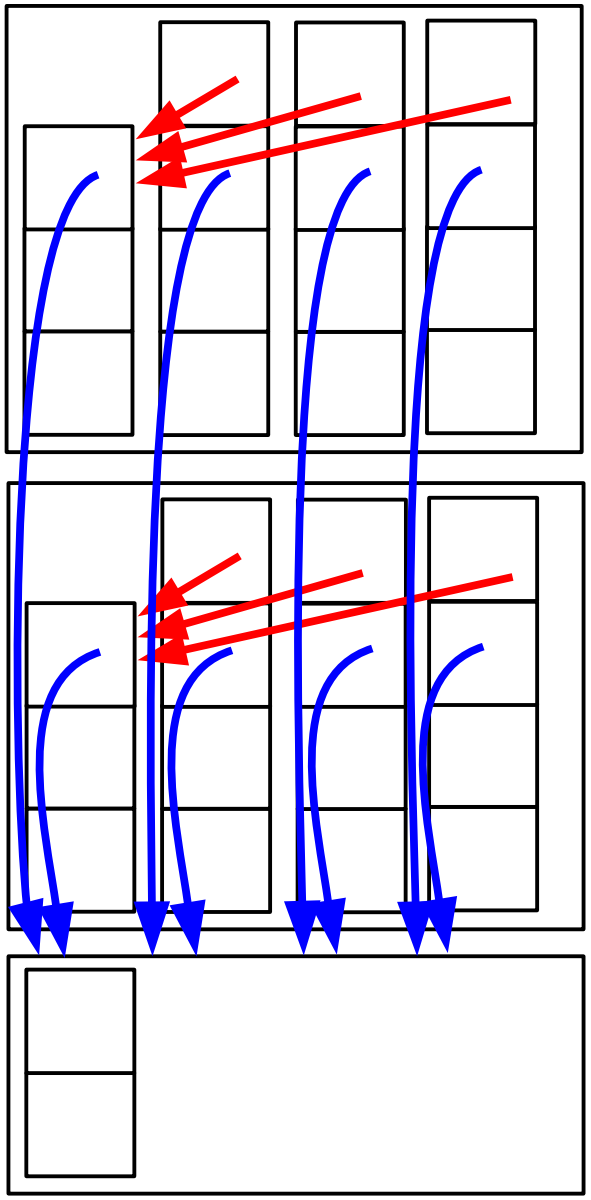
– Higher order push operations do not modify the pointers.

– A new operation “collapse” replaces the topmost k-stack by the k-stack from the pointer contained in the topmost stack symbol.



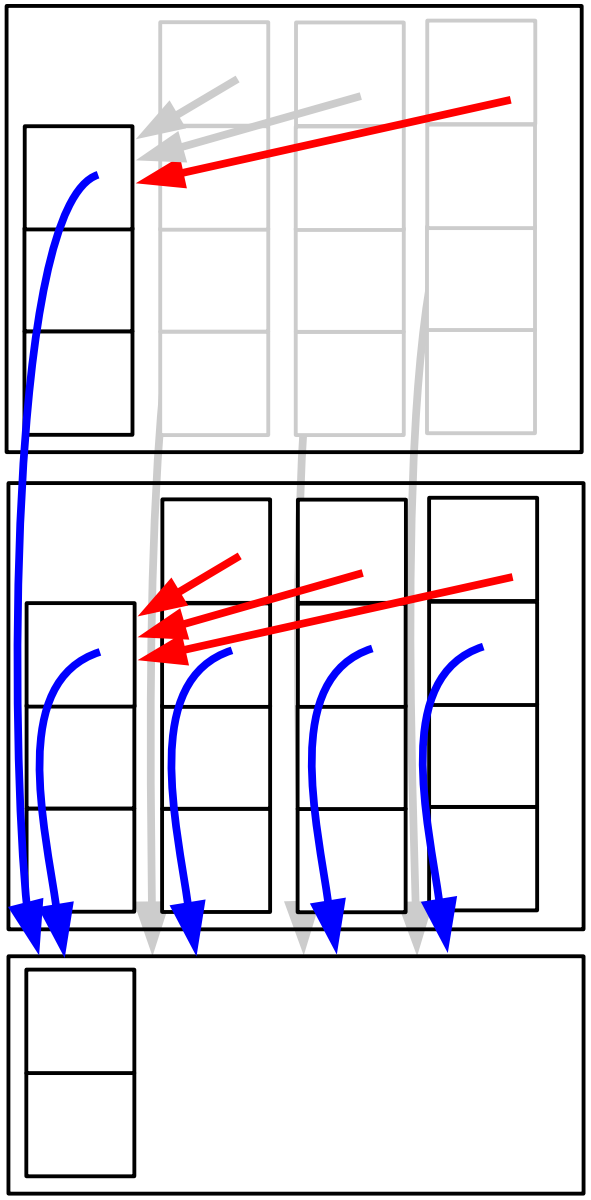
$push_{1,3}$
$push_2$
$push_{1,2}$
$push_2$
$push_2$

Collapsible Pushdown Systems (CPDS)



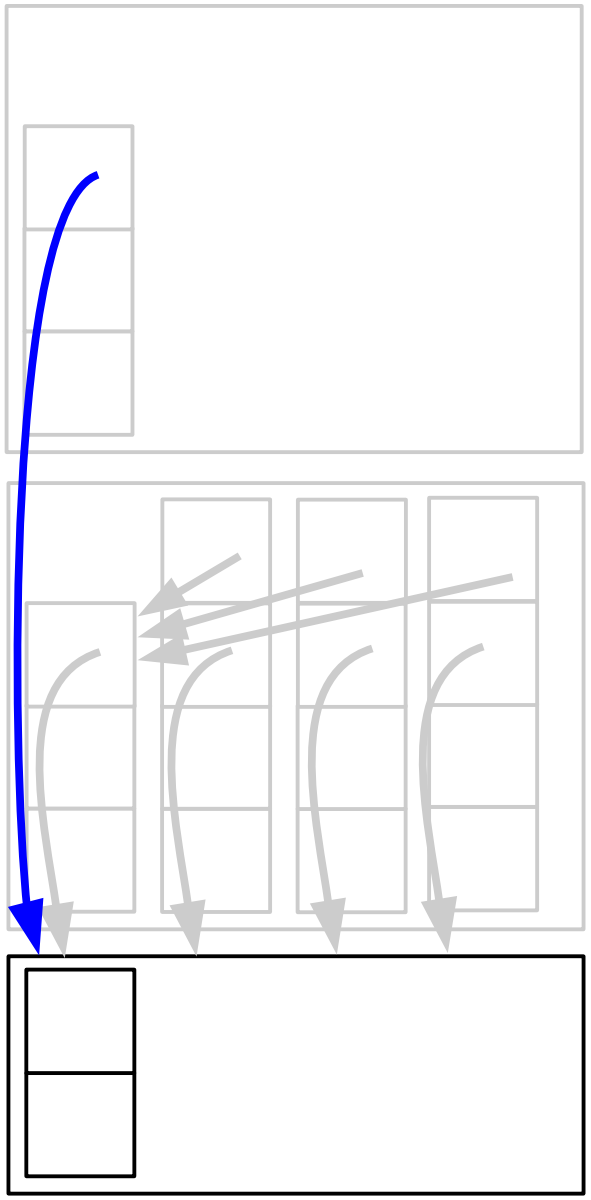
$push_{1,3}$
$push_{2}$
$push_{1,2}$
$push_{2}$
$push_{2}$
$push_{3}$

Collapsible Pushdown Systems (CPDS)



$push_{1,3}$
$push_2$
$push_{1,2}$
$push_2$
$push_2$
$push_3$
collapse

Collapsible Pushdown Systems (CPDS)



$push_{1,3}$
$push_2$
$push_{1,2}$
$push_2$
$push_2$
$push_3$
collapse
collapse

Stack vs. recursion

trees generated by **safe** recursion schemes of order n = configuration trees of n-HOPDS

[Hague, Murawski, Ong, Serre 2008]

trees generated by **all** recursion schemes of order n = configuration trees of Collapsible Pushdown Systems of order n

Stack vs. recursion

trees generated by **safe** recursion schemes of order n = configuration trees of n-HOPDS

[Hague, Murawski, Ong, Serre 2008]

trees generated by **all** recursion schemes of order n = configuration trees of Collapsible Pushdown Systems of order n

Moreover, these trees have decidable MSO theory!

However, the configuration graph of some 2-CPDA has undecidable MSO theory!

Stack vs. recursion

trees generated by **safe** recursion schemes of order n = configuration trees of n-HOPDS

[Hague, Murawski, Ong, Serre 2008]

trees generated by **all** recursion schemes of order n = configuration trees of Collapsible Pushdown Systems of order n

Moreover, these trees have decidable MSO theory!

However, the configuration graph of some 2-CPDA has undecidable MSO theory!

What about First-Order logic?

[Kartzow 2010] FO over ε-closures of 2-CPDA graphs is decidable (these graphs are tree-automatic).

Logics on CPDA-graphs

k-CPDA, k≥2	MSO	→	undecidable
2-CPDA	FO	→	decidable

[Broadbent 2012]

k-CPDA, k≥3	FO	→	undecidable

Logics on CPDA-graphs

k-CPDA, $k \geq 2$ MSO → undecidable

2-CPDA FO → decidable

[Broadbent 2012]

k-CPDA, $k \geq 3$ FO → undecidable

a lot of results on the border:

n_m-CPDA, $n \geq 3$, $3 \leq m \leq n$ Σ_2 form. → undecidable

n-CPDA in which only links of order m are allowed

$\exists x ... \exists y \, (\forall z ... \forall t$ (quantifier free))

Logics on CPDA-graphs

k-CPDA, $k \geq 2$ MSO → undecidable

2-CPDA FO → decidable

[Broadbent 2012]

k-CPDA, $k \geq 3$ FO → undecidable

a lot of results on the border:

n_m-CPDA, $n \geq 3$, $3 \leq m \leq n$ Σ_2 form. → undecidable

3_2-CPDA Σ_2 form. → undecidable on ε-closure

n_m-CPDA, $n \geq 4$, $2 \leq m \leq n-2$ Σ_1 form. → undecidable

$\exists x ... \exists y$ (quantifier free)

Logics on CPDA-graphs

k-CPDA, $k \geq 2$ MSO → undecidable

2-CPDA FO → decidable

[Broadbent 2012]

k-CPDA, $k \geq 3$ FO → undecidable

a lot of results on the border:

n_m-CPDA, $n \geq 3$, $3 \leq m \leq n$ Σ_2 form. → undecidable

3_2-CPDA Σ_2 form. → undecidable on ε-closure

n_m-CPDA, $n \geq 4$, $2 \leq m \leq n-2$ Σ_1 form. → undecidable

2-CPDA (FO+transitive closure of quant.free formulas) → decidable

3_2-CPDA FO → decidable without ε-closure

n_n-CPDA and 3_2-CPDA Σ_1 form. → decidable

Only a few cases left, among them Σ_1 formulas on 3-CPDA graphs without ε-closure (the only one for order 3).

Contribution 1: it is decidable!!! (we extend Broadbent's methods)

Logics on CPDA-graphs (with unreachable configurations)

Notice that FO (in graphs without ε-closure) describes only local properties, but we restrict our graph to configurations reachable from the initial one. Reachability is not expressible in FO, it is much more difficult. So maybe this is the main problem for decidability of FO? What if we consider graphs without restricting to reachable confs?

Logics on CPDA-graphs (with unreachable configurations)

Notice that FO (in graphs without ε-closure) describes only local properties, but we restrict our graph to configurations reachable from the initial one. Reachability is not expressible in FO, it is much more difficult. So maybe this is the main problem for decidability of FO? What if we consider graphs without restricting to reachable confs?

But which configurations are allowed in our graph?
Three possibilities:

1) only those which can be constructed from the empty stack using stack operations (constructible stacks)
2) also non-constructible, but links have to point to prefixes of the stack (links as numbers/pointers – classical stacks)
3) links are allowed to contain any stack, not necessarily a prefix of the "external" stack (links containing stacks – annotated stacks)

Logics on CPDA-graphs (with unreachable configurations, without ε-closure)

Which configurations are allowed in our graph?
Three possibilities:

1) only those which can be constructed from the empty stack using stack operations (constructible stacks)
2) also non-constructible, but links have to point to prefixes of the stack (links as numbers/pointers – classical stacks)
3) links are allowed to contain any stack, not necessarily a prefix of the "external" stack (links containing stacks – annotated stacks)

Case 2 – FO undecidable (for 3-CPDA) [Broadbent 2012]

This paper:

Case 1 – FO undecidable (for 4-CPDA) (a similar encoding to Broadbent's)

Case 3 – FO decidable (the graph is very uniform, for each quantifier it is enough to check candidates from a finite set)

Conclusion

Three new results about decidability of FO on CPDA graphs
2 x decidability
1 x undecidability
filling holes left in earlier results.

Thank you.