Collapse Operation Increases Expressive Power of Deterministic Higher Order Pushdown Automata

Paweł Parys

University of Warsaw

Higher order pushdown automata (HOPDA) [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.

$push_2$: $[s_1 ... s_{i-1} s_i]$ -> $[s_1 ... s_{i-1} s_i s_i]$

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

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

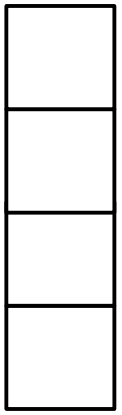
pop_1 : $[s_1 ... s_{i-1} [a_1 ... a_{j-1} a_j]]$ -> $[s_1 ... s_{i-1} [a_1 ... a_{j-1}]]$

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

Higher order pushdown automata (HOPDA)

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

- push a symbol with every “a” on input

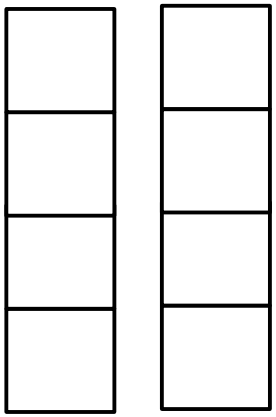


a a a a

Higher order pushdown automata (HOPDA)

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

- push a symbol with every “a” on input
- make $push_2$

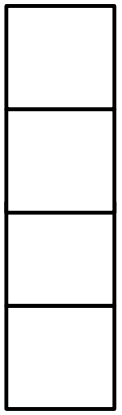


a a a a b

Higher order pushdown automata (HOPDA)

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

- push a symbol with every “a” on input
- make $push_2$
- pop a symbol with each “b” on input



a a a a b b b b

Higher order pushdown automata (HOPDA)

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

- push a symbol with every “a” on input
- make $push_2$
- pop a symbol with each “b” on input
- pop a symbol with each “c” on input

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

Collapsible HOPDA

Collapsible HOPDA is an extension of a HOPDA

Elements of 1-stack are tuples $(a,n_1,...,n_k)$, where $a \in \Sigma$, $n_i \in \mathbb{N}$.

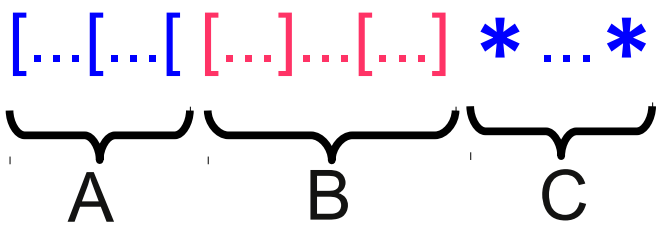
$push_1 a$ - push $(a,n_1,...,n_k)$ on the top of the topmost level-1 stack,
where n_i is the size of the topmost level-i stack

$collapse_i$ - if the topmost stack symbol is $(a,n_1,...,n_k)$
leave only first n_i-1 elements of the topmost level-i stack
(from the topmost level-i stack remove all level i-1 stacks on which this symbol is present)

Notice: $collapse_1 = pop_1$

Example: Urzyczyn's language U

alphabet: [,], ∗
U contains words of the form:



- segment A is a prefix of a well-bracketed word that ends in [which is not matched in the entire word
- segment B is a well-bracketed word
- segments A and C have the same length

for example:

[[ ] [[ ] [[ ]] ∗∗∗∗ ∈U

How to recognize U by an automaton with collapse?

- one stack symbol
- first order stack counts the number of currently open brackets
- a copy ($push_2$) is done after each bracket

How to recognize U by an automaton with collapse?

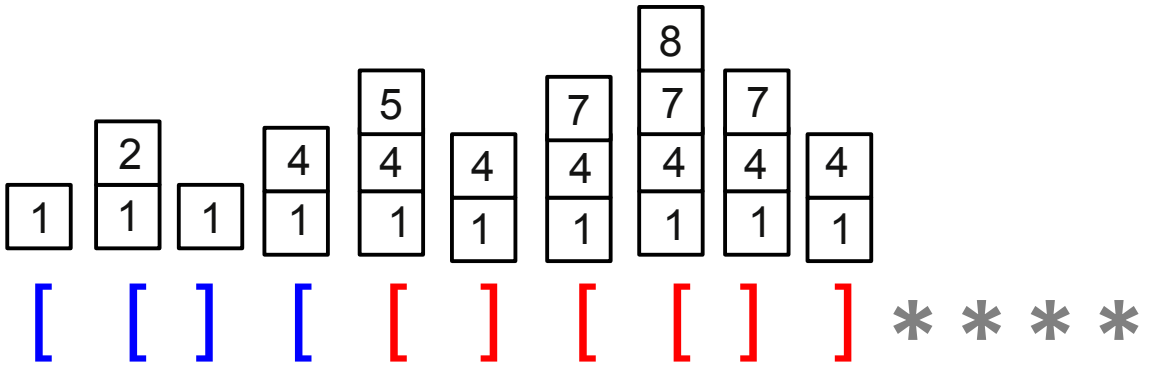
- one stack symbol
- first order stack counts the number of currently open brackets
- a copy ($push_2$) is done after each bracket

How to recognize U by an automaton with collapse?

- one stack symbol
- first order stack counts the number of currently open brackets
- a copy ($push_2$) is done after each bracket

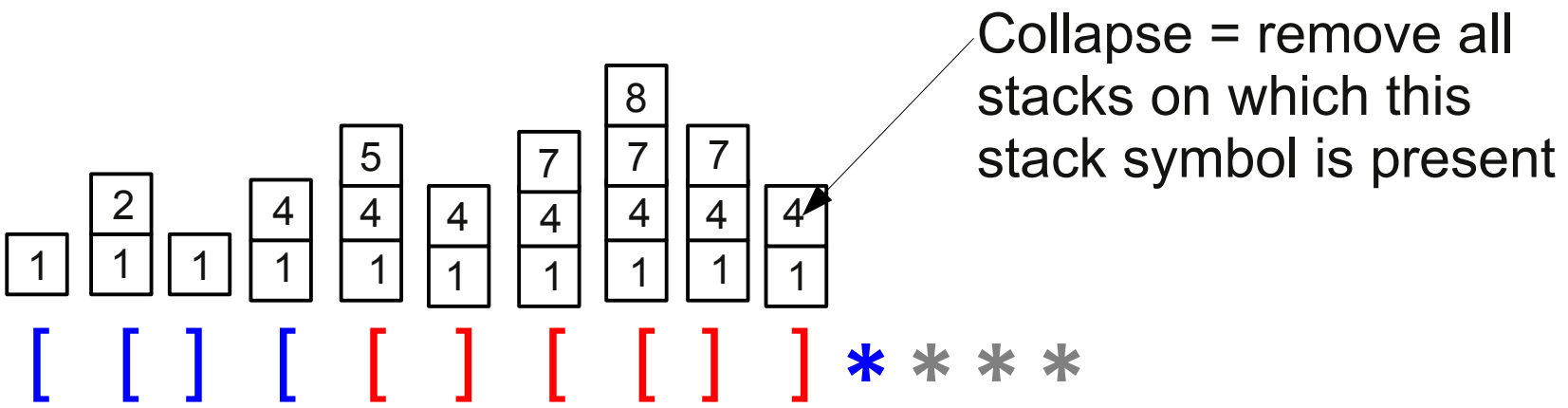
How to recognize U by an automaton with collapse?

- one stack symbol
- first order stack counts the number of currently open brackets
- a copy ($push_2$) is done after each bracket



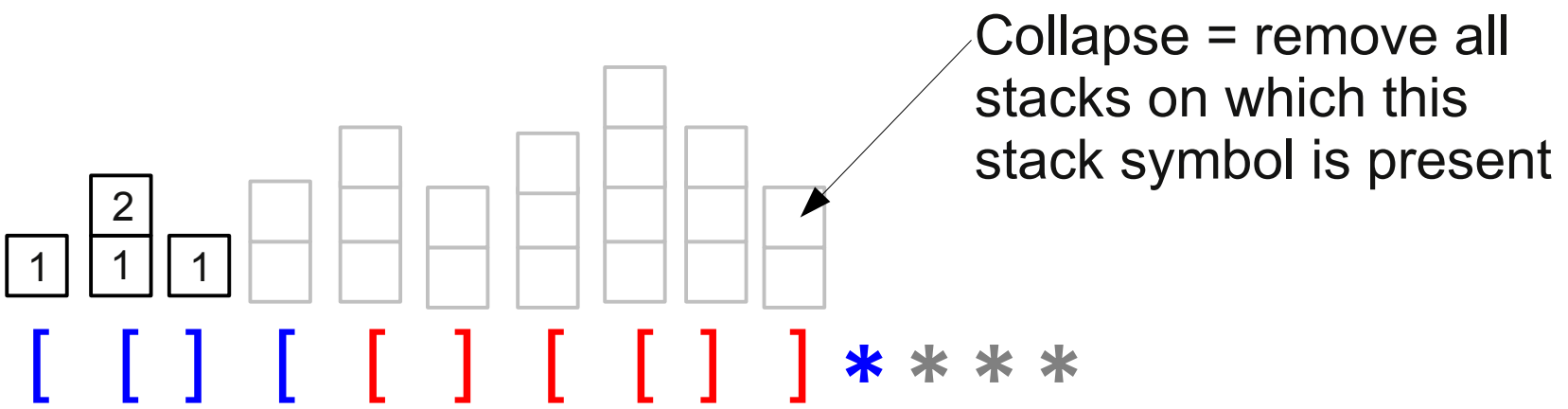
How to recognize U by an automaton with collapse?

- one stack symbol
- first order stack counts the number of currently open brackets
- a copy ($push_2$) is done after each bracket
- on the first star we make the collapse
- we count the number of stacks



How to recognize U by an automaton with collapse?

- one stack symbol
- first order stack counts the number of currently open brackets
- a copy ($push_2$) is done after each bracket
- on the first star we make the collapse
- we count the number of stacks



Main theorem

Theorem
No level-2 deterministic PDA can recognize the language U.

Corollary
There is a language recognized by a level-2 deterministic collapsible PDA which is not recognized by any level-2 deterministic PDA without collapse.

Corollary
There is a tree generated by a level-2 recursion scheme which is not generated by any safe level-2 recursion scheme.

Motivation: from program verification to higher order pushdowns

Example

```
open(x, “foo”)
a := 0
while a<100 do
    read(x)
    a := a+1
close(x)
```

is the file “foo” accessed according to open,read*,close?

Motivation: from program verification to higher order pushdowns

Example

Step 1: information about infinite data domains is approximated.

```
open(x, “foo”)
a := 0
while a<100 do
    read(x)
    a := a+1
close(x)
```

```
open(x, “foo”)

while * do
    read(x)

close(x)
```

is the file “foo” accessed according to open,read*,close?

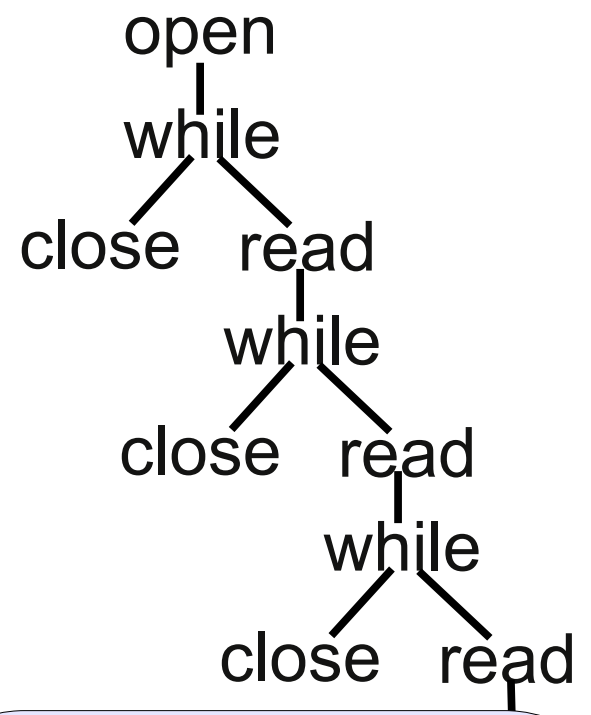
is the file “foo” accessed according to open,read*,close?

Motivation: from program verification to higher order pushdowns

Example

Step 2: consider the tree of possible control flows.

```
open(x, “foo”)
while * do
    read(x)
close(x)
```



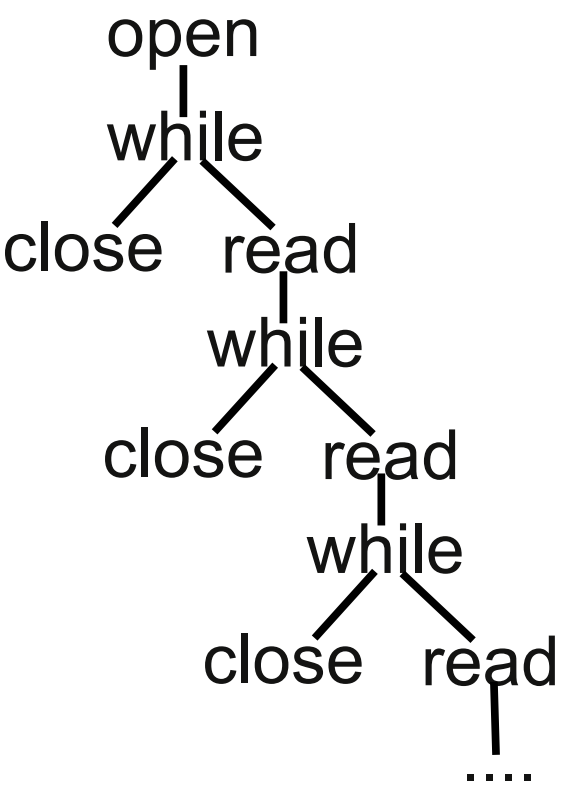
is the file “foo” accessed according to open,read*,close?

is each path labelled by open,read*,close?

Motivation: from program verification to higher order pushdowns

Example

```
open(x, “foo”)
while * do
    read(x)
close(x)
```



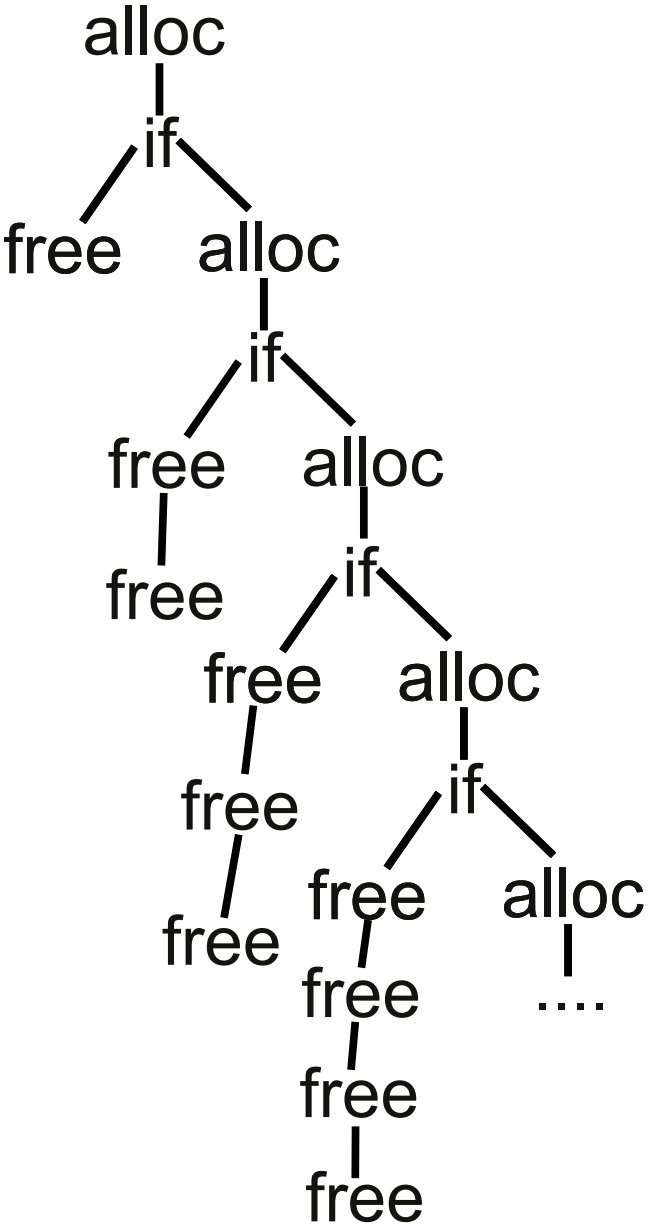
Observation: for programs without recursion, each path of the tree is a regular language.
(the program is a deterministic finite automaton)

Rabin 1969: Regular trees have decidable MSO theory.

Motivation: from program verification to higher order pushdowns

Example 2 - program with recursion

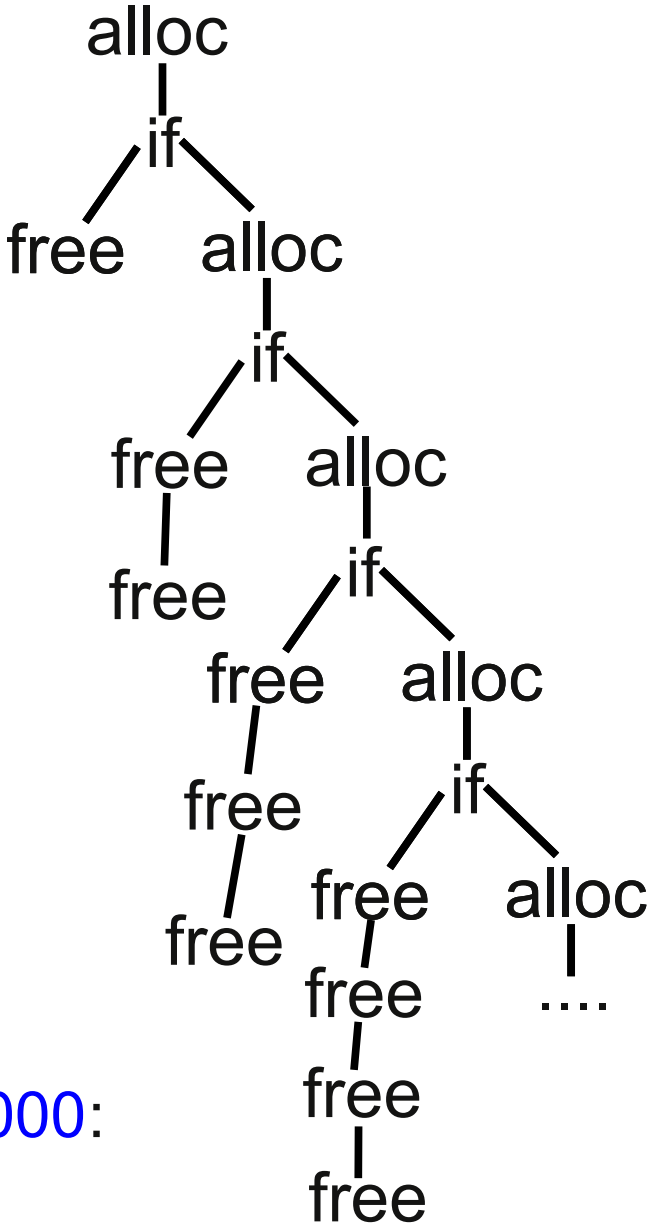
```
let f(x) =
    alloc(x)
    if * then f(x)
    free(x)
f(x)
```



Motivation: from program verification to higher order pushdowns

Example 2 - program with recursion

```
let f(x) =
    alloc(x)
    if * then f(x)
    free(x)
f(x)
```



Now the tree is not regular!!

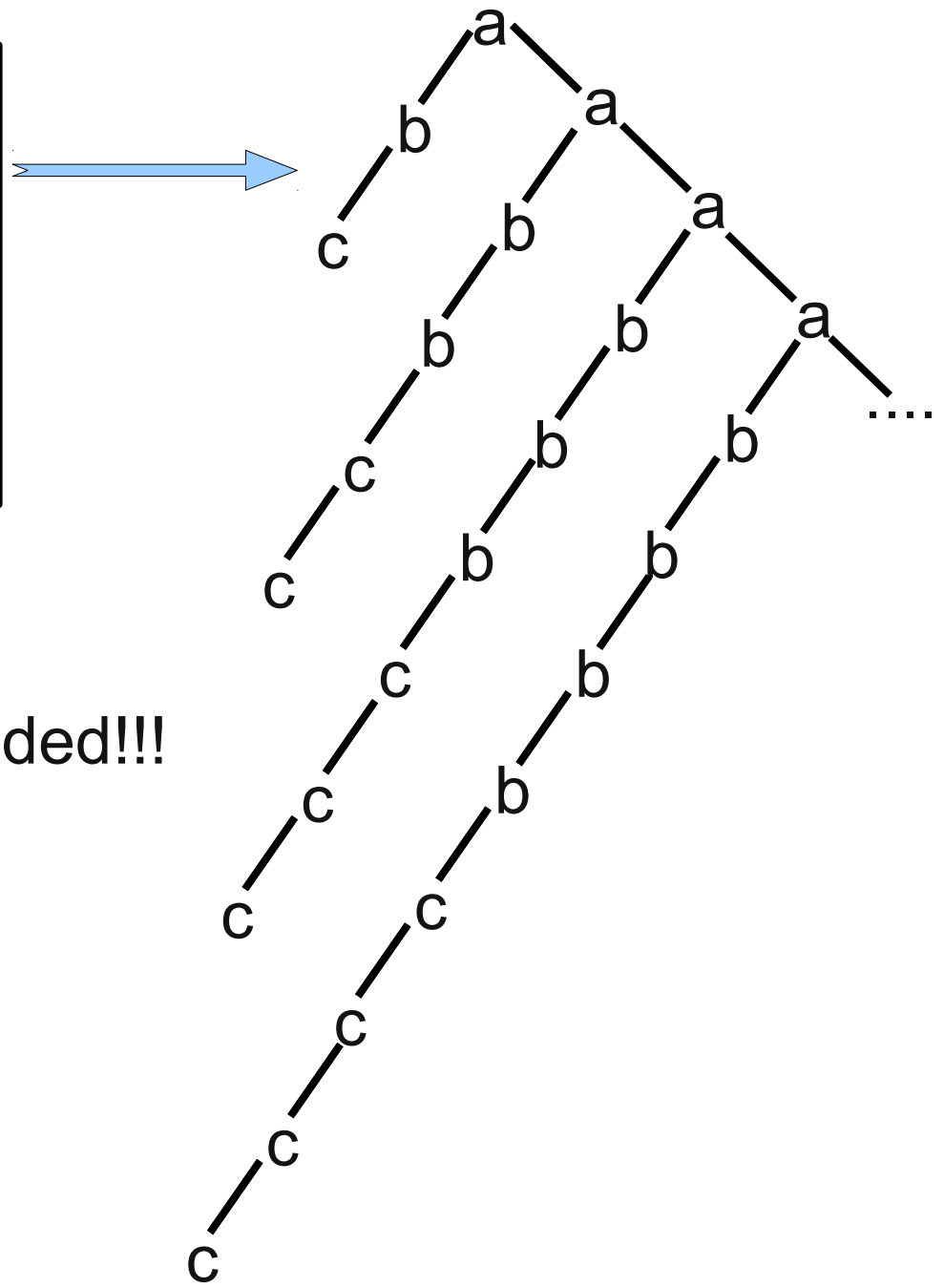
But each path is recognized by
a **deterministic** pushdown automaton.

Muller, Schupp 1985 / Caucal 1986 / Stirling 2000:
such trees have decidable MSO theory.

Motivation: from program verification to higher order pushdowns

What about higher order programs?

```
let f(x, g) =
    a(x)
    if * then g(x)
    else f(x, fun h x -> h(x); b(x))
    c(x)
f(x, b)
```



Higher order automata are needed!!!

Relation between HOPDA and programs

We skip the formal definition

For each level we have introduced two classes of trees:

$\mathbf{ColPdaTree}_n\,\Sigma$ = trees generated by order-n deterministic collapsible PDA

$\mathbf{RecSchTree}_n\,\Sigma$ = trees generated by order-n recursion scheme (program)

- Are these classes equal?

  Hague, Murawski, Ong, Serre 2008:

  yes: $\mathbf{RecSchTree}_n\,\Sigma = \mathbf{ColPdaTree}_n\,\Sigma$

so it is natural to consider collapsible automata

- What about MSO decidability?

  Ong 2006:

  Trees from $\mathbf{RecSchTree}_n\,\Sigma$ have decidable MSO theory.

Relation between HOPDA and programs – earlier results

$\mathbf{PdaTree}_n\,\Sigma$ = trees generated by order-n deterministic HOPDA

$\mathbf{SafeRecSchTree}_n\,\Sigma$ = trees generated by order-n safe recursion scheme

Knapik, Niwiński, Urzyczyn 2002:
For each n, $\mathbf{PdaTree}_n\,\Sigma = \mathbf{SafeRecSchTree}_n\,\Sigma$
and these trees have decidable MSO theory.

what is safety?
It is some syntactic constraint on the recursion schemes.
(the result of passing order-k parameters to a function has to be of order lower than k)
Safety restriction disappears at level 1.

Another characterization of these trees - the Caucal hierarchy (Caucal 2002)
$\mathbf{PdaTree}_n\,\Sigma = \mathbf{SafeRecSchTree}_n\,\Sigma = \mathbf{CaucalTree}_n\,\Sigma$

Two hierarchies (of trees / of word languages):

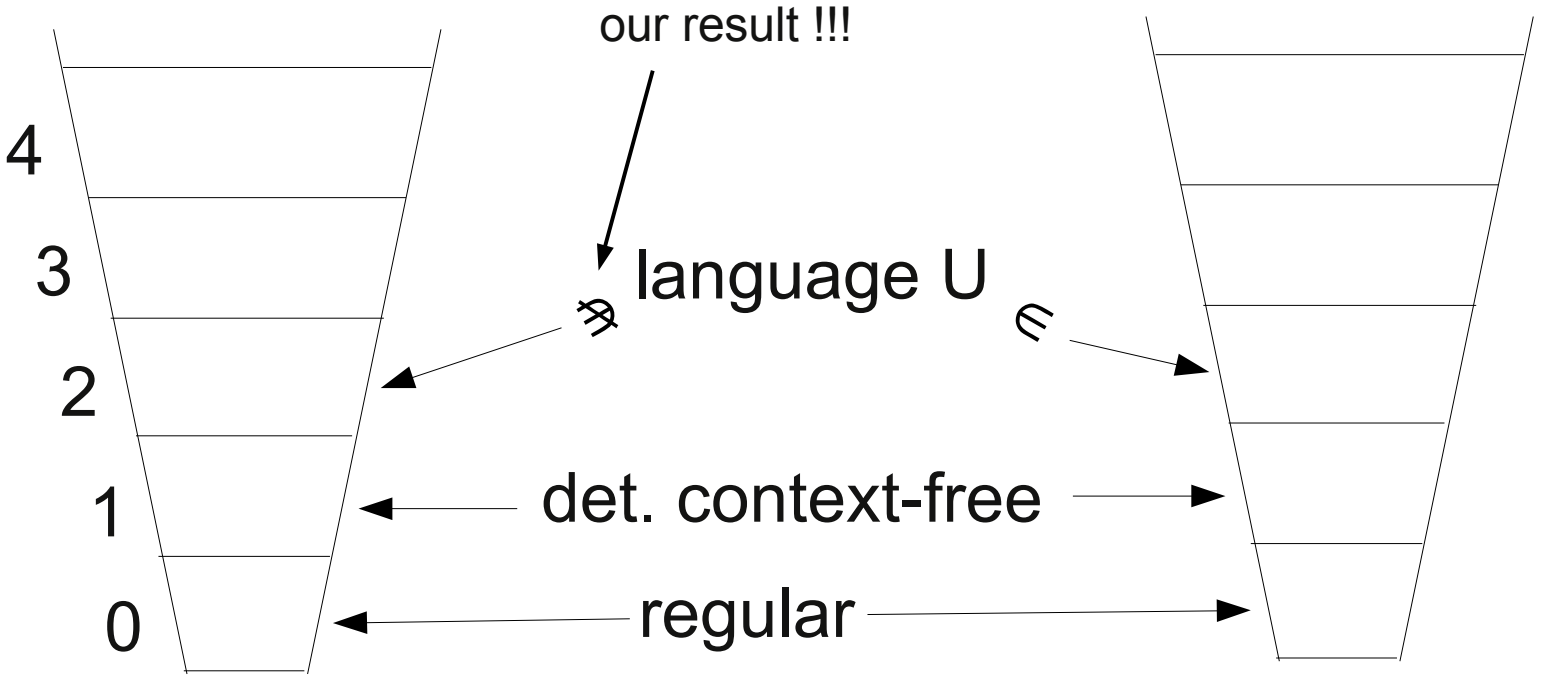
deterministic H-O pushdown automata

safe H-O schemas

Caucal hierarchy

deterministic collapsible H-O pushdown automata

H-O schemas



These are different hierarchies!!!

Open problems

1) Show that U (or some other language) is not accepted by a deterministic HOPDA (without collapse) of an arbitrary level, i.e. that the union of the whole hierarchies are different.

Open problems

1) Show that U (or some other language) is not accepted by a deterministic HOPDA (without collapse) of an arbitrary level, i.e. that the union of the whole hierarchies are different.

2) Does collapse increase recognizing power of **nondeterministic** HOPDA?

Aehlig, Miranda, Ong 2005: for level 2 – NO
(collapse can be simulated by nondeterminism)

but:
- nondeterministic automata does not have a natural connection with verification
- most problems are undecidable, even universality for level-1 PDA (but emptiness is decidable)