Expressive Power of Collapsible Pushdown Automata

Paweł Parys

University of Warsaw

Higher order pushdown automata (H-O PDA)

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

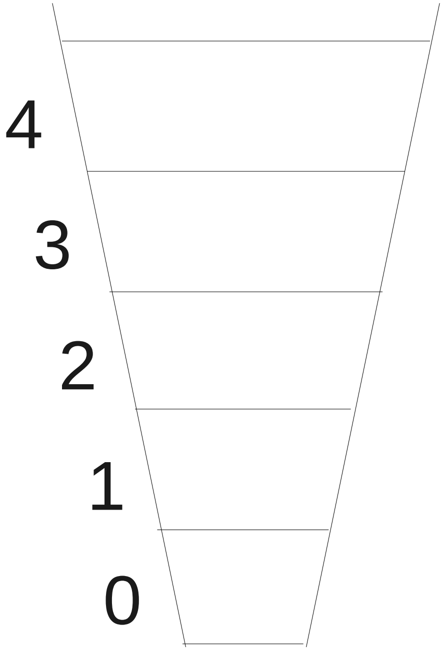
Operations on 2-stacks: s_i are 1-stacks. Top of stack is on right.

$push_2$	:	$[s_1 \ldots s_{i-1} s_i]$	->	$[s_1 \ldots s_{i-1} s_i\, s_i]$
pop_2	:	$[s_1 \ldots s_{i-1} s_i]$	->	$[s_1 \ldots s_{i-1}]$
$push_1 x$	:	$[s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1} a_j]]$	->	$[s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1} a_j\, x]]$
pop_1	:	$[s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1} a_j]]$	->	$[s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1}]]$

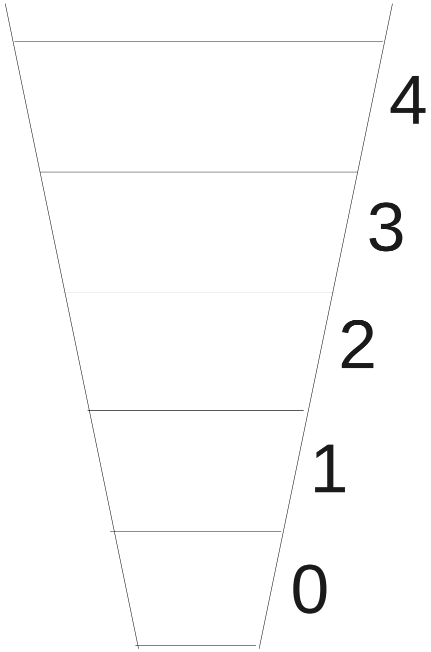
An **order-n PDA** has an order-n stack, and has $push_i$ and pop_i for each $1 \leq i \leq n$.

Two hierarchies (of trees):

trees generated by
H-O pushdown systems



trees generated by
H-O recursion schemes



Are these two hierarchies equal?
- orders 0 and 1 – yes

Two hierarchies (of trees):

Are these two hierarchies equal?

- Knapik, Niwiński, Urzyczyn 2002

trees generated by H-O pushdown systems $\stackrel{?}{=}$ trees generated by H-O schemes

$=$ trees generated by safe H-O schemes

Two hierarchies (of trees):

Are these two hierarchies equal?

- Knapik, Niwiński, Urzyczyn 2002
- Caucal 2002

trees generated by H-O pushdown systems $\stackrel{?}{=}$ trees generated by H-O schemes

=

trees generated by safe H-O schemes

=

Caucal hierarchy

Two hierarchies (of trees):

Are these two hierarchies equal?

- Hague, Murawski, Ong, Serre 2008

trees generated by collapsible H-O pushdown systems

=

trees generated by H-O schemes

?=

trees generated by H-O pushdown systems

=

trees generated by safe H-O schemes

=

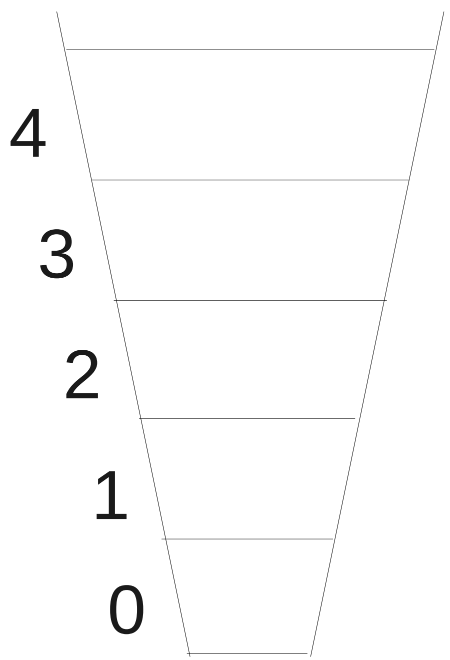
Caucal hierarchy

Two hierarchies (of trees):

H-O pushdown systems

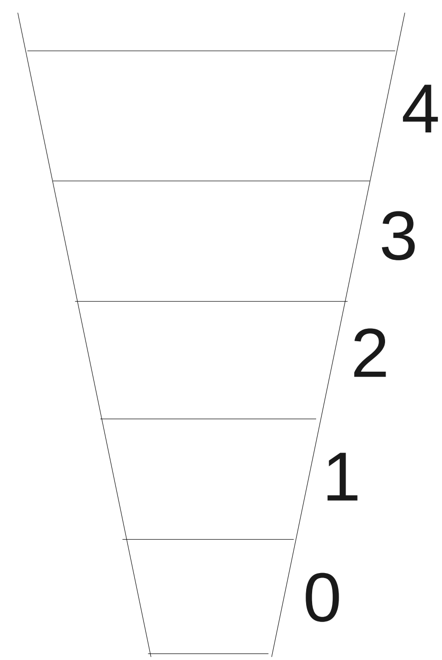
safe H-O schemes

Caucal hierarchy

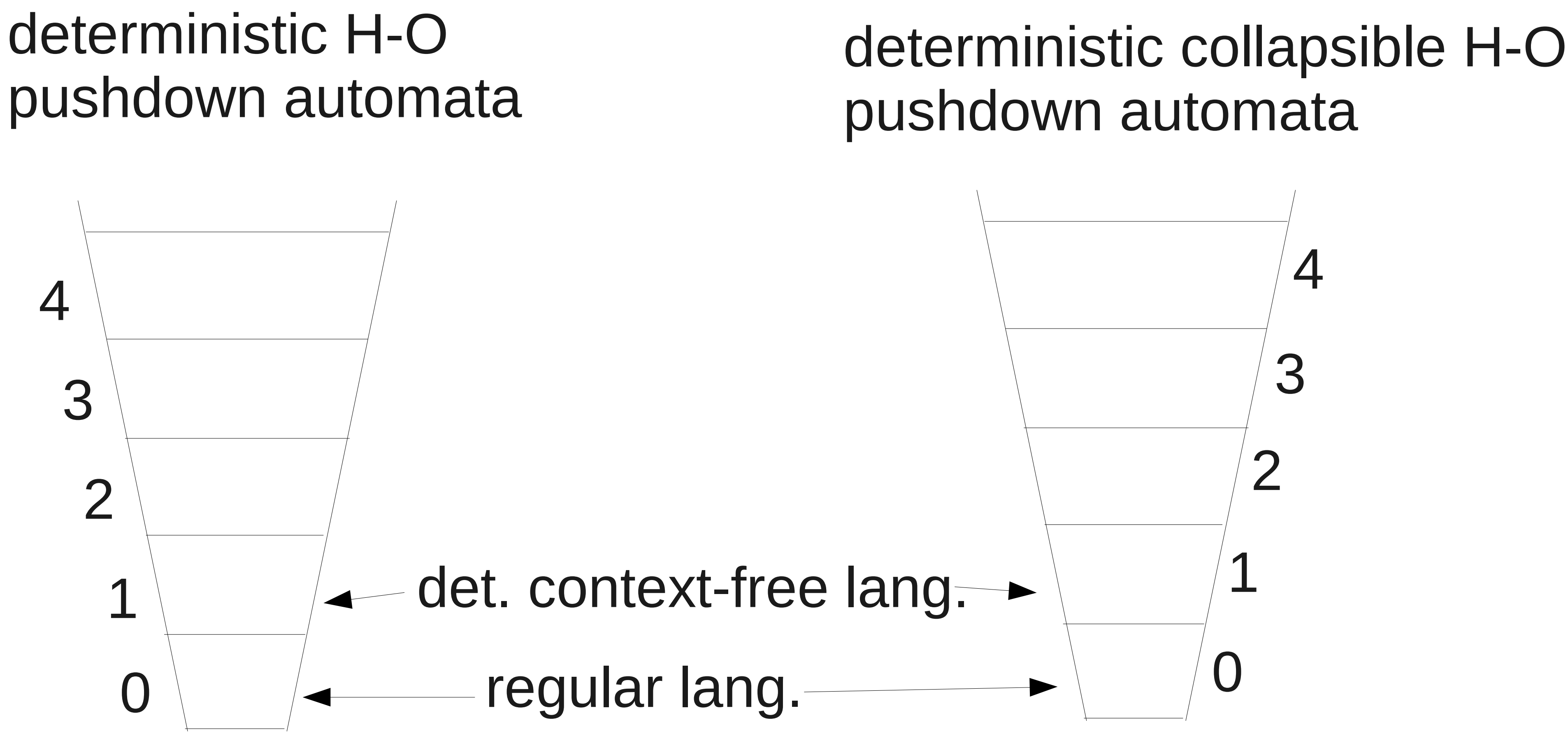


collapsible H-O pushdown systems

all H-O schemes

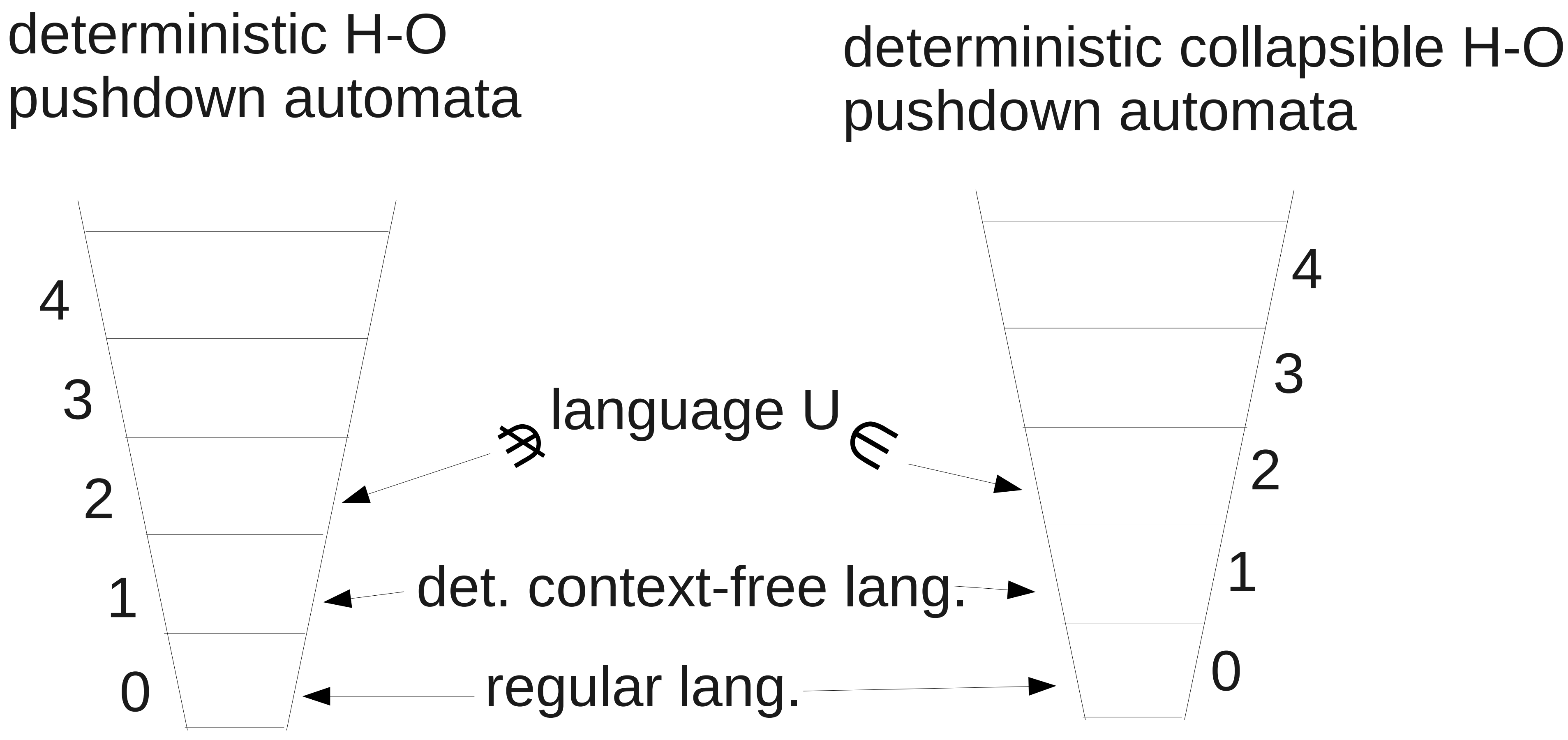


Equivalently: two hierarchies of word languages



First result (STACS 2011):

- order 2 is different

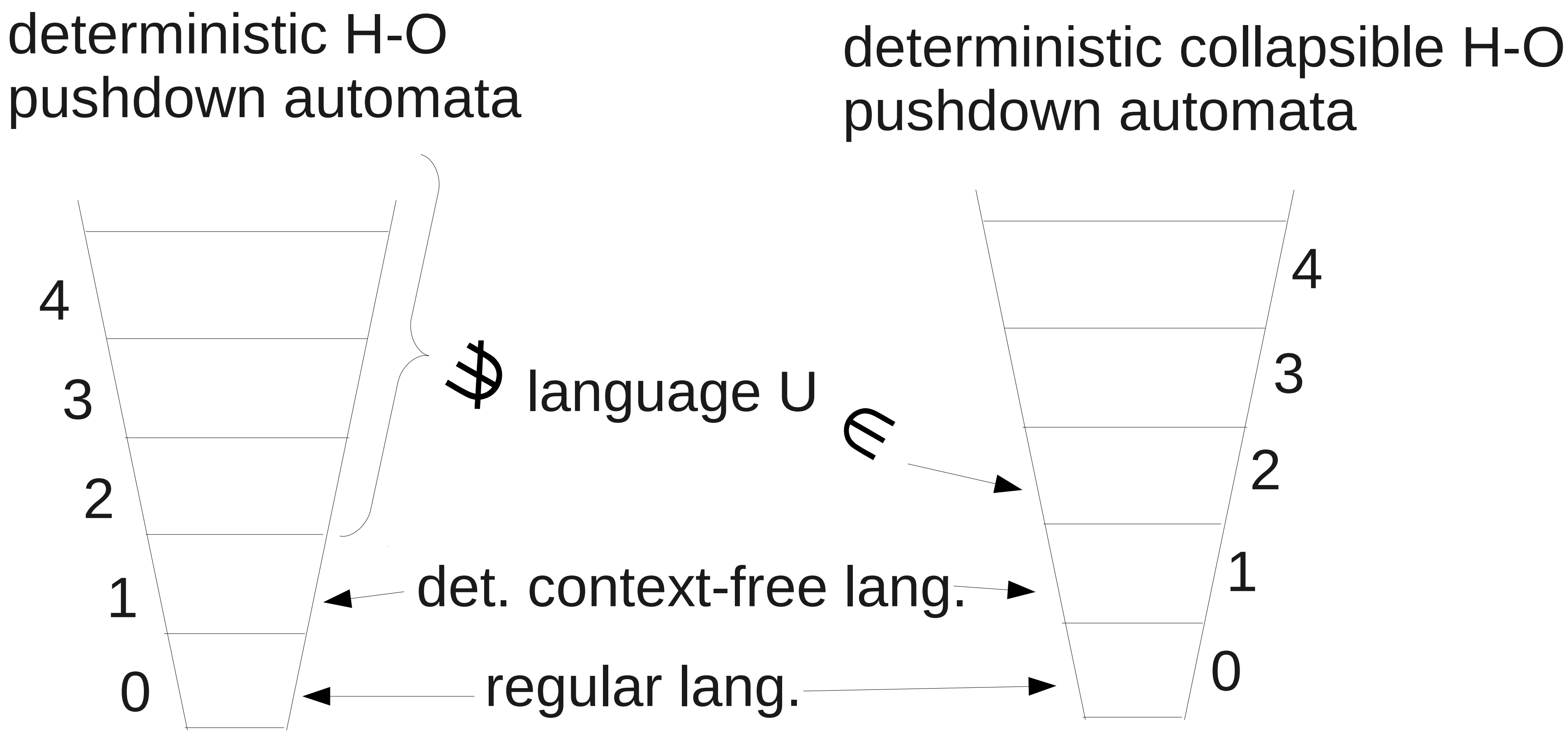


First result (STACS 2011):

- order 2 is different

Stronger result (LICS 2012):

- the union of the hierarchies is different



Collapsible Pushdown Automata

Collapsible PDA are an extension of H-O PDA

Each 0-stack (stack symbol) is created with a fresh identifier.

For $2 \leq i \leq n$ we have a new operation collapse_i

It removes all (i-1)-stacks which contain the topmost symbol.

Notice: $\text{collapse}_1 = \text{pop}_1$

Example: Urzyczyn's language U (improved)

alphabet: [,], *, #

U contains words of the form:

$$\underbrace{[\ast\ast[\ast\ast\ast]\ast[}_{A}\underbrace{\ast[\ast\ast]\ast[\,[\ast\ast]\ast]}_{B}\;\underbrace{\#\#\#\#\#\#}_{C}$$

- brackets in segment A form a prefix of a well-bracketed word that ends in [which is not matched in the entire word
- brackets in segment B form a well-bracketed word
- the number of sharps in C equals to the number of stars in A

* = [ ]

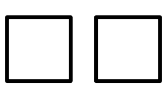
How to recognize U by an automaton with collapse?

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

[*]*[*[* *]* * # # # #

How to recognize U by an automaton with collapse?

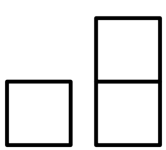
- one stack symbol
- first order stack counts the number of currently open brackets
- a copy ($push_2$) is done after each star



[*]*[*[* *]* * # # # #

How to recognize U by an automaton with collapse?

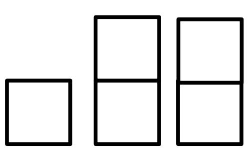
- one stack symbol
- first order stack counts the number of currently open brackets
- a copy ($push_2$) is done after each star



[*]*[*[* *]* * # # # #

How to recognize U by an automaton with collapse?

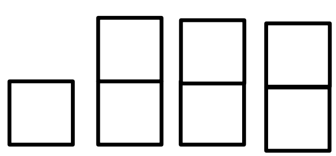
- one stack symbol
- first order stack counts the number of currently open brackets
- a copy ($push_2$) is done after each star



*[* *]*[*[* *] * * # # # #

How to recognize U by an automaton with collapse?

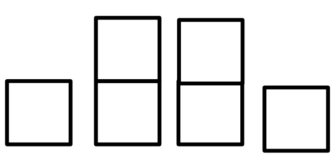
- one stack symbol
- first order stack counts the number of currently open brackets
- a copy ($push_2$) is done after each star



[*]*[*[* *]* * # # # #

How to recognize U by an automaton with collapse?

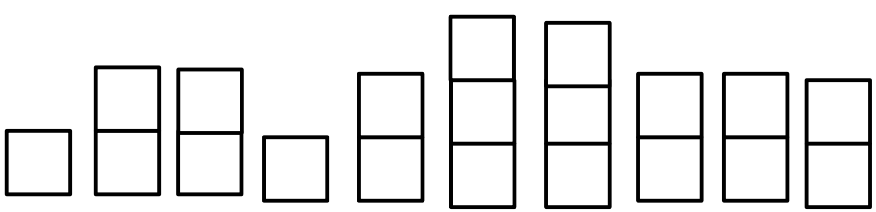
- one stack symbol
- first order stack counts the number of currently open brackets
- a copy ($push_2$) is done after each star



[*]*[*[* *]* * # # # #

How to recognize U by an automaton with collapse?

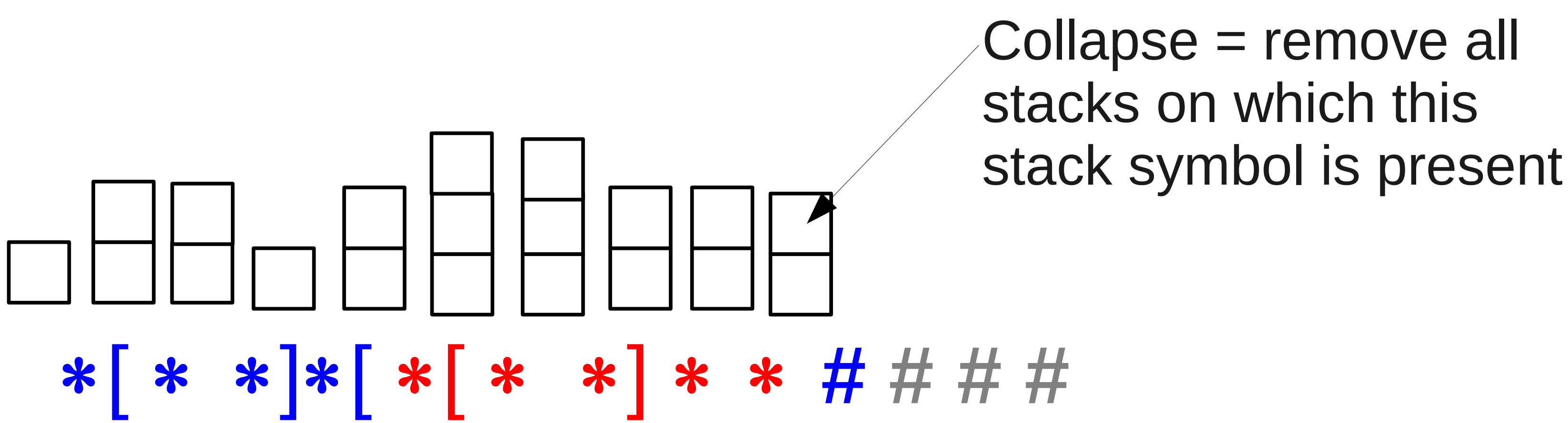
- one stack symbol
- first order stack counts the number of currently open brackets
- a copy ($push_2$) is done after each star



[*]*[*[* *]* * # # # #

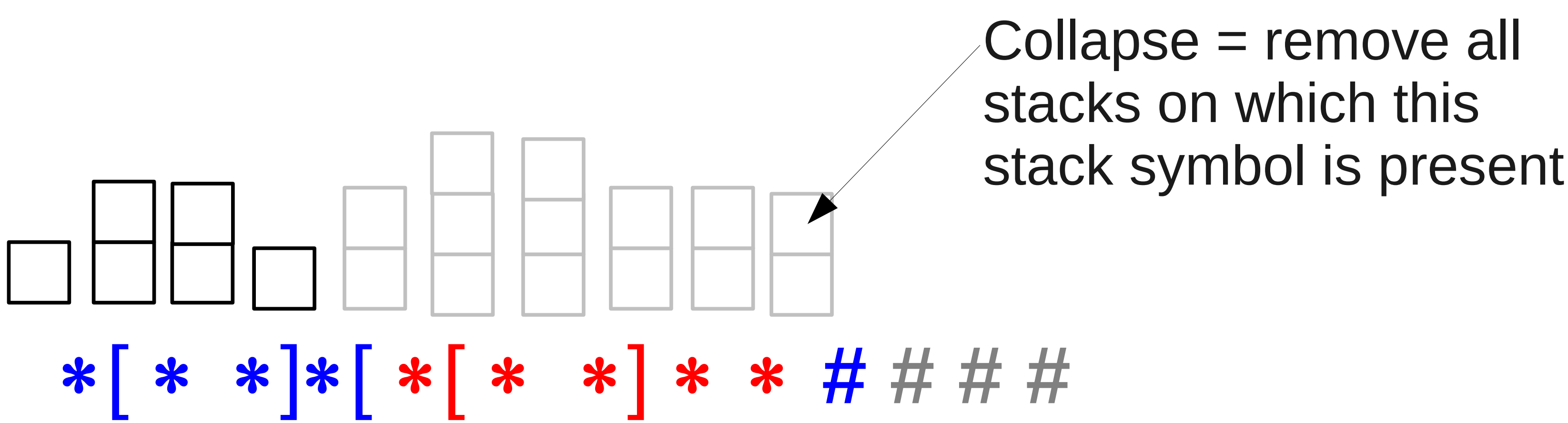
How to recognize U by an automaton with collapse?

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



Language U cannot be recognized – order 1 case

We prove now that deterministic order-1 PDA cannot recognize U.

We read the word:

$$\underbrace{**\ldots*}_{k_1}[\underbrace{**\ldots*}_{k_2}[\underbrace{**\ldots*}_{k_3}[\underbrace{**\ldots*}_{k_4}]\underbrace{**\ldots*}_{k_5}]\underbrace{**\ldots*}_{k_6}]\underbrace{**\ldots*}_{k_7}[$$

At each of the red points in the word, the stack has to be:

$$\underbrace{\text{aaaa...aa}}_{k_1}\ \$\ \underbrace{\text{aaaa...aa}}_{k_2}\ \$\ \underbrace{\text{aaaa...aa}}_{k_3}\ \$\ \text{something}$$

(erasing something is wrong, because the number of sharps after the last [should be $k_1+k_2+k_3+k_4+k_5+k_6+k_7$)

Language U cannot be recognized – order 1 case

We prove now that deterministic order-1 PDA cannot recognize U.

We read the word:

$$\underbrace{\ast\ast\ldots\ast}_{k_1}[\underbrace{\ast\ast\ldots\ast}_{k_2}[\underbrace{\ast\ast\ldots\ast}_{k_3}[\underbrace{\ast\ast\ldots\ast}_{k_4}]\underbrace{\ast\ast\ldots\ast}_{k_5}]\underbrace{\ast\ast\ldots\ast}_{k_6}]\underbrace{\ast\ast\ldots\ast}_{k_7}[$$

At each of the red points in the word, the stack has to be:

$$\underbrace{aaaa...aa}_{k_1}\ \$\ \underbrace{aaaa...aa}_{k_2}\ \$\ \underbrace{aaaa...aa}_{k_3}\ \$\ \text{something}$$

What if we give **#** at a red point?

We should accept after k_1 or k_1+k_2 or $k_1+k_2+k_3$ sharps.

If |Q|<3, this is impossible (see: state while crossing blue line).

Language U cannot be recognized – higher order case

Consider an CPDA of order 2 (or higher).

We read the word:

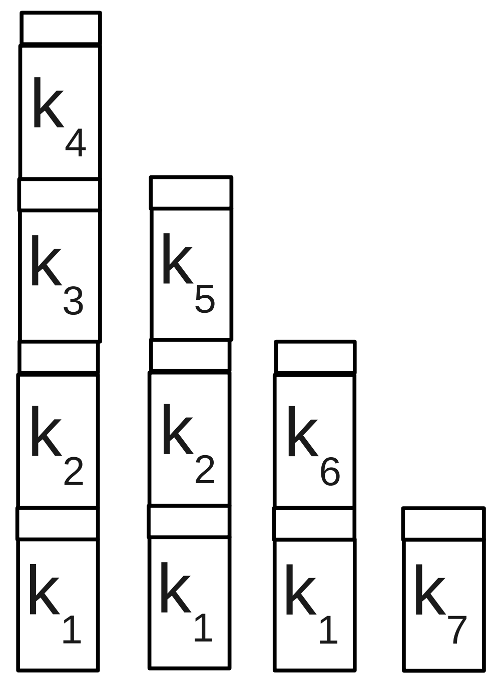
$$w_1 = \underbrace{**\ldots*}_{k_1}[\underbrace{**\ldots*}_{k_2}[\underbrace{**\ldots*}_{k_3}[\underbrace{**\ldots*}_{k_4}]\underbrace{**\ldots*}_{k_5}]\underbrace{**\ldots*}_{k_6}]\underbrace{**\ldots*}_{k_7}[$$

What do we know now?

It was impossible to use only the topmost order-1 stack!

So after reading w_1 some of the numbers k_i (denote it n_1) is not present on the topmost order-1 stack.

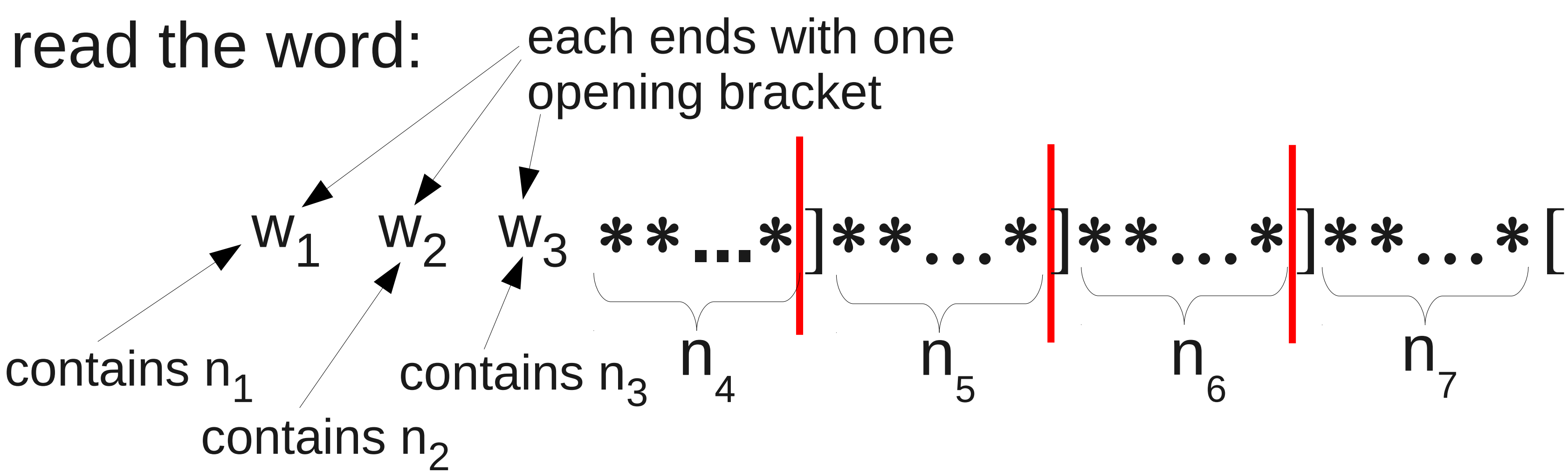
We nest the same argument...



Language U cannot be recognized – higher order case

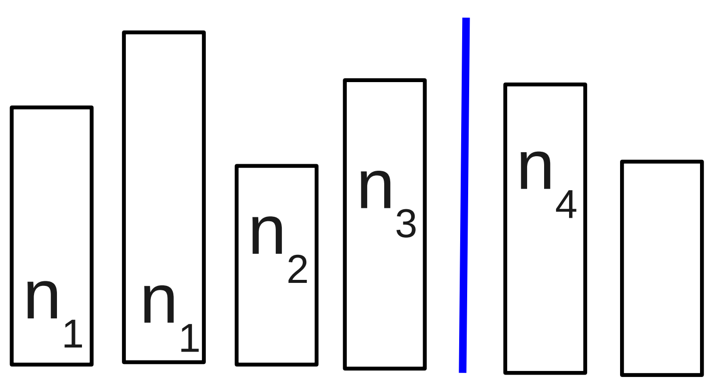
Consider an CPDA of order 2 (or higher).

We read the word:



(the words w_2, w_3 have the same shape as w_1, but they have different numbers inside)

At each of the red points in the word, the part of the stack below the blue line has to be the same (we cannot erase n_4):



but n_1, n_2, n_3 are not present above the blue line (by the order-1 argument)!

Language U cannot be recognized – higher order case

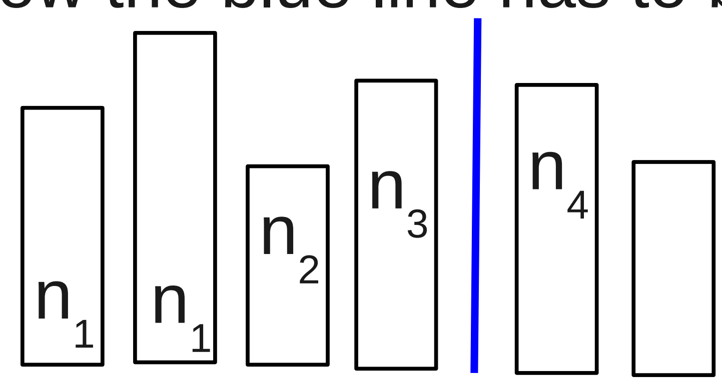
We read the word:

each ends with one opening bracket

$$w_1 \; w_2 \; w_3 \; \underbrace{**\dots*}_{n_4}]\underbrace{**\dots*}_{n_5}]\underbrace{**\dots*}_{n_6}]\underbrace{**\dots*}_{n_7}[$$

contains n_1 contains n_2 contains n_3

At each of the red points in the word, the part of the stack below the blue line has to be the same (we cannot erase n_4):



but n_1, n_2, n_3 are not present above the blue line (by the order-1 argument)!

What if we give **#** at a red point?
The result should include n_1 or n_1+n_2 or $n_1+n_2+n_3$.
If |Q|<3, this is impossible (see: state while crossing blue line).

We nest the same argument again...

Summing up the proof...

The overall idea is simple, but the proof is difficult to formalize.
There are several problems:

Summing up the proof...

The overall idea is simple, but the proof is difficult to formalize.
There are several problems:

1) Where a number is stored on the stack? What does it mean that a number is not present in the topmost order-k stack?

Key observation: deterministic automaton reading always the same symbol $*$ modifies the stack in a “regular” way.

Summing up the proof...

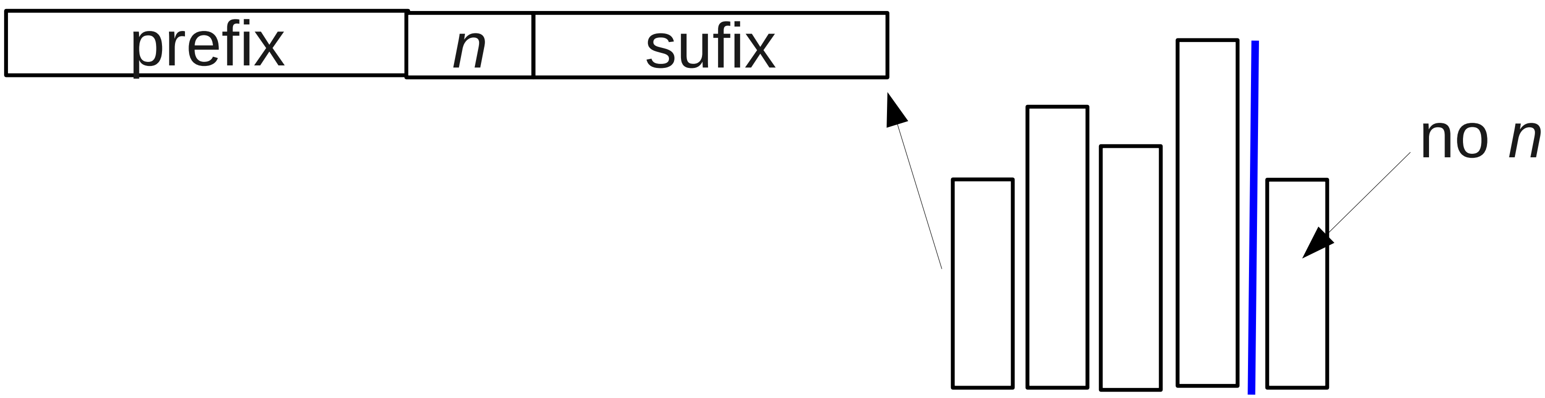
The overall idea is simple, but the proof is difficult to formalize.
There are several problems:

1) Where a number is stored on the stack? What does it mean that a number is not present in the topmost order-k stack?
2) We say that we cannot read n sharps at the end without inspecting a place in the stack where n is written. But maybe “by accident” n is present somewhere else?

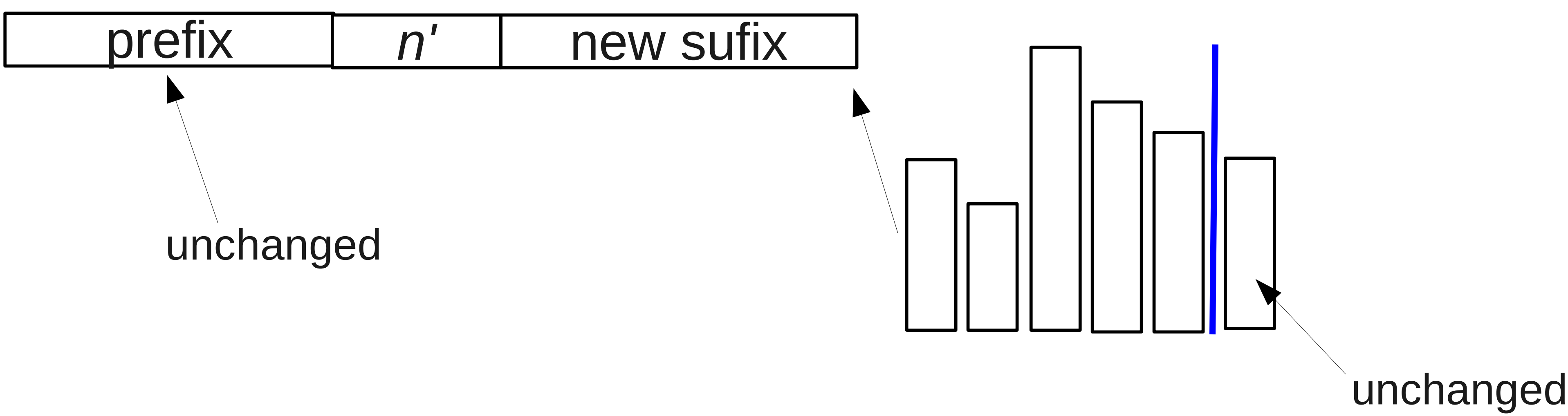
   Solution: a pumping lemma (very purpose-specific)

Pumping lemma

If *n* is not present in the topmost order-k stack after reading a word, then we can change *n* into some other (arbitrarily big) number *n'* without changing the prefix before *n* and without changing the topmost order-k stack at the end.



after pumping:



Pumping lemma

If *n* is not present in the topmost order-k stack after reading a word, then we can change *n* into some other (arbitrarily big) number *n'* without changing the prefix before *n* and without changing the topmost order-k stack at the end.

A similar pumping lemma allows us to prove that the hierarchy of CPDA trees is strict (joint work with A.Kartzow).

Differences:

- generalized to CPDA
- the stack at the end is not important
- instead, we bound the length of the new suffix

Considered words are of the form: $\underbrace{aa...a}_{k}\underbrace{bbbb.....b}_{2^{2^{\cdot^{\cdot^{\cdot^{2^k}}}}}}$

height = order of CPDA-1

but: there is an earlier proof of strictness for HO PDA without collapse by Engelfriet (based on complexity arguments), which works also in the case of CPDA.

Summing up the proof...

The overall idea is simple, but the proof is difficult to formalize.
There are several problems:

1) Where a number is stored on the stack? What does it mean that a number is not present in the topmost order-k stack?
2) We say that we cannot read n sharps at the end without inspecting a place in the stack where n is written. But maybe “by accident” n is present somewhere else?
3) In a k-PDA we have infinitely many ways of inspecting an order-1 stack (not just |Q| as in 1-CPDA): we may have an arbitrary order-2 (order-3, …) stack below.

   Solution: a stack may be described by its “intersection type” coming from a finite set (like the types of N.Kobayashi and like “stack automata” of Broadbent, Carayol, Ong, Serre).

   Stacks of low order correspond to terms of high order; we can say something about a stack of order k, if we know the types of stacks of higher order placed below it (“its arguments”).

   (these “intersection types” are also present in the proof of the pumping lemma)

A connected result

Consider simply-typed λ-terms build over constants **0**, **1+** of arity 0 and 1.
The β-normal form of each term M of type *o* is **1+** (**1+** (… (**1+ 0**)...)),
it represents a number, denoted *val(*M*)*.

A connected result

Consider simply-typed λ-terms build over constants **0**, **1+** of arity 0 and 1.
The β-normal form of each term M of type *o* is **1+** (**1+** (… (**1+ 0**)...)),
it represents a number, denoted *val(*M*)*.

For two terms M, M' of type $\alpha \to o$ we say that M~M' if for each sequence $N_1, N_2, \ldots$ of terms of type α,

$$val(MN_1), val(MN_2), \ldots \text{ bounded} \Leftrightarrow val(MN_1), val(MN_2), \ldots \text{ bounded}$$

e.g. λx.x and λx.(**1+** x) are equivalent.

Thm. For each type α the relation ~ has finitely many equivalence classes.

A connected result

Consider simply-typed λ-terms build over constants **0**, **1+** of arity 0 and 1.
The β-normal form of each term M of type *o* is **1+** (**1+** (… (**1+ 0**)...)),
it represents a number, denoted *val(*M*)*.

For two terms M, M' of type $\alpha\rightarrow o$ we say that M~M' if for each sequence $N_1, N_2, \ldots$ of terms of type α,

$val(MN_1), val(MN_2), \ldots$ bounded $\Leftrightarrow$ $val(MN_1), val(MN_2), \ldots$ bounded

e.g. $\lambda x.x$ and $\lambda x.($**1+** $x)$ are equivalent.

Thm. For each type α the relation ~ has finitely many equivalence classes.

Corollary. We cannot represent arbitrarily long tuples of integers in terms of type α.

Def. We can represent tuples of length k in terms of type α if there exist terms $M_1,\ldots,M_k$ of type $\alpha\rightarrow o$ (extractors) and for each $t\in\mathbb{N}^k$ there exists N of type α such that $(val(M_1N),\ldots,val(M_kN))=t$.

Another idea: CPDA with data

Consider a restricted variant of CPDA: when a symbol on input is repeated k times, the CPDA reads it just once, but it can store the number k on the stack (the stack alphabet is extended by natural numbers), or it can compare k with the number in the topmost stack symbol.

Another idea: CPDA with data

Consider a restricted variant of CPDA: when a symbol on input is repeated *k* times, the CPDA reads it just once, but it can store the number *k* on the stack (the stack alphabet is extended by natural numbers), or it can compare *k* with the number in the topmost stack symbol.

Such automata are much easier to analyze...
(obvious where a number is written on the stack,
no pumping lemma needed)

Hypothesis: Assume that L is “permutation invariant” (we can change numbers in words in L, and they remain in L). Then

L is recognized by a normal CPDA
if and only if
L is recognized by a CPDA with data.

Related open problem

The same question for nondeterministic word languages:

Is there a language
- not recognized by any nondeterministic H-O PDA
- recognized by a nondeterministic Collapsible H-O PDA

(here the second orders are equal,
possibly there is a difference on level 3)

Thank you.