Higher-Order Pushdown Systems with Data

Paweł Parys
University of Warsaw

Higher order pushdown 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	:	$[s_1 \ldots s_{i-1} s_i]$	->	$[s_1 \ldots s_{i-1}]$
$push_1 x$	:	$[s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1} a_j]]$	->	$[s_1 \ldots s_{i-1} [a_1 \ldots a_{j-1} a_j\, x]]$
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 $1 \le i \le n$.

Decisions depend on the state and the topmost symbol of the topmost stack.

Collapsible pushdown systems/automata [Knapik, Niwiński, Urzyczyn, Walukiewicz 05] [Hague, Murawski, Ong, Serre 08]

Collapsible PDA are an extension of a higher-order PDA

$push_1(x)$ 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 [Knapik, Niwiński, Urzyczyn, Walukiewicz 05] [Hague, Murawski, Ong, Serre 08]

Collapsible PDA are an extension of a higher-order PDA

$push_1(x)$ 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 (as tree generators) are equiexpressive with higher-order recursion schemes!

Trees generated by collapsible PDS have decidable MSO theory!

Higher-order pushdown automata with data

Input alphabet: $A \times D$
A – finite set – labels
D – infinite set (e.g. $D=\mathbb{N}$) – data values

we ignore the numeric values, the order,
we can only compare $x=y$, $x \neq y$

Example word: $\begin{array}{ccccc} a & b & a & c & a \\ 1 & 2 & 3 & 2 & 2 \end{array}$

(this is a typical setting in the theory of XML documents)

Higher-order pushdown automata with data

Input alphabet: $A \times D$
Stack alphabet: $B \times (D \cup \{\text{empty}\})$

A, B – finite sets
D – infinite set (e.g. $D=\mathbb{N}$) – data values

Higher-order pushdown automata with data

Input alphabet: $A \times D$
Stack alphabet: $B \times (D \cup \{\text{empty}\})$

Operations:

- transitions are chosen basing on the finite part of a symbol
- operations of order >1 work as previously (with or without collapse)
- $\text{push}_1(b)$ in a transition reading (a,x) pushes (b,x)
- $\text{push}_1(b)$ in an ε-transition pushes (b,empty)
- pop_1 in a transition reading (a,x) has two result states $q_=, q_\neq$:
  when the topmost stack symbol is (b,y),
  we go to state $q_=$ if $x=y$, and to $q_\neq$ otherwise
- in an ε-transition pop_1 has one result state

In this talk we consider only deterministic automata.

For nondeterministic automata it is reasonable that $\text{push}_1(b)$ in an ε-transition can guess a data value which is pushed.

Higher-order pushdown automata with data

Input alphabet: $A \times D$
Stack alphabet: $B \times (D \cup \{\text{empty}\})$

Operations:

- transitions are chosen basing on the finite part of a symbol
- operations of order >1 work as previously (with or without collapse)
- $\text{push}_1(b)$ in a transition reading (a,x) pushes (b,x)
- $\text{push}_1(b)$ in an ε-transition pushes (b,empty)
- pop_1 in a transition reading (a,x) has two result states $q_=, q_{\neq}$:
  when the topmost stack symbol is (b,y),
  we go to state $q_=$ if $x=y$, and to $q_{\neq}$ otherwise
- in an ε-transition pop_1 has one result state

In this talk we consider only deterministic automata.

For nondeterministic automata it is reasonable that $\text{push}_1(b)$ in an ε-transition can guess a data value which is pushed.

Higher-order pushdown automata with data

<u>Example</u>
A = {a, b} – input alphabet
Language: words in which the first letter is equal
to the last letter (including the data value)

Automaton of level 2
B = {a, b} – stack alphabet
push the first letter on the stack
then repeat: push_2, pop_1 (reading a letter), pop_2
if the topmost stack symbol is equal to the letter read,
the pop_1 enters to an accepting state in case of equality

Higher-order pushdown automata with data

<u>Observation.</u>
Languages recognized by CPA with data
are "permutation invariant."

This means that for any permutation π of D
and for any word $(a_1,d_1)(a_2,d_2)\ldots(a_n,d_n)$ in a language,
the word $(a_1,\pi(d_1))(a_2,\pi(d_2))\ldots(a_n,\pi(d_n))$ is also in the language.

E.g. if the word $\begin{array}{ccccc} a & b & a & c & a \\ 1 & 2 & 3 & 2 & 2 \end{array}$ is in a language,

then the word $\begin{array}{ccccc} a & b & a & c & a \\ 4 & 3 & 1 & 3 & 3 \end{array}$ also is in the language.

Motivation

1. (minor) Such automata possibly can be useful in verification of higher-order programs operating on data.

2. Automata with data can be useful in understanding automata without data. They are similar, but automata with data are easier to understand.
   a) We may transfer some ideas.
   b) Probably we have some equivalence (work in progress).

Two questions

Why is it simpler to analyze automata with data?

How are they related to automata without data?

Relations between automata with and without data

L – language over $A \times D$ recognized by a (deterministic or nondeterministic) CPA with data

<u>Trivial observation.</u>
Then the projection of L to the first coordinate (a language over A) is recognized by a classical nondeterministic CPA (without data)

Relations between automata with and without data

How to simulate words over $A \times D$ using words over a finite alphabet?

Let $A' = A \cup \{\star\}$
Assume that $D = \mathbb{N}$.

A word $(a_1,d_1)(a_2,d_2)\ldots(a_n,d_n)$ over $A \times \mathbb{N}$ is represented as

a word $a_1 \star^{d_1} a_2 \star^{d_2} \ldots a_n \star^{d_n}$ over A (after a_k there is a star repeated d_k times).

Relations between automata with and without data

How to simulate words over $A \times D$ using words over a finite alphabet?

Let $A' = A \cup \{\ast\}$
Assume that $D = \mathbb{N}$.

A word $(a_1,d_1)(a_2,d_2)\ldots(a_n,d_n)$ over $A \times \mathbb{N}$ is represented as a word $a_1 \ast^{d_1} a_2 \ast^{d_2} \ldots a_n \ast^{d_n}$ over A (after a_k there is a star repeated d_k times).

Easy observation.
L – language over $A \times D$ recognized by a deterministic CPA with data. Then the language (over A') of representations of words in L is recognized by a deterministic CPA without data.

Proof: push/pop a symbol on the stack as many times as the star is seen on the input.

Relations between automata with and without data

Easy observation.
L – language over $A \times D$ recognized by a deterministic CPA with data. Then the language (over A') of representations of words in L is recognized by a deterministic CPA without data.

Is the opposite true?

Hypothesis – yes
L – permutation-invariant language over $A \times D$ such that the language (over A') of representations of words in L is recognized by a deterministic CPA without data. Then L is recognized by a deterministic CPA with data.
Probably the same is true for nondeterministic CPA.

Notice: the language over A' has to be “permutation invariant” (we can permute the number of stars) and cannot contain words beginning with a star.

Relations between automata with and without data

Equivalence hypothesis

L – permutation-invariant language over A$\times$D such that the language (over A') of representations of words in L is recognized by a deterministic CPA without data. Then L is recognized by a deterministic CPA with data.
Probably the same is true for nondeterministic CPA.

Why the above is true? A general idea:

When a big number of stars is read, the CPA can:

- perform many pushes (of some levels) - this is done according to some “regular” pattern – can be simulated by a push$_1$ pushing the data value
- before pushes, it can perform some pops, which possibly compare this number with some earlier number – this is simulated by appropriate pop$_1$
- it cannot simultaneously compare with two earlier numbers
- it can compare with sum of two numbers, with a number minus two, etc., the rest of the run may depend on the result, but in both cases it is accepting / not accepting, so we can assume any result.

Why automata with data are simper?

Generally:
We can precisely trace where a data value is on the stack;
as soon as it is removed, we are sure it cannot be compared
to some other data value, etc.

On the other hand, the number of repetitions can be written on
the stack in a strange way; we don't know exactly where;
and even when this place is removed, it can be present
somewhere else “by accident”, etc.
(even for a “permutation invariant” language, the run can
completely change, if we apply a permutation)

Why automata with data are simper?

Generally:
We can precisely trace where a data value is on the stack;
as soon as it is removed, we are sure it cannot be compared
to some other data value, etc.

On the other hand, the number of repetitions can be written on
the stack in a strange way; we don't know exactly where;
and even when this place is removed, it can be present
somewhere else “by accident”, etc.
(even for a “permutation invariant” language, the run can
completely change, if we apply a permutation)

In this paper:
We prove that a CPA of level 2 with data can recognize
a language, which is not recognized by a HOPA (without collapse)
with data, of any level. [this means that collapse is important]

Why automata with data are simper?

In this paper:
We prove that a CPA of level 2 with data can recognize a language, which is not recognized by a HOPA (without collapse) with data, of any level.

For CPA/HOPA without data we have proved the same (LICS 2012), but the proof is long and technically complicated.

Notice:
- The theorem for automata with data plus the “equivalence hypothesis” implies the standard theorem.
- The standard theorem implies the theorem for automata with data.

Moreover: the “equivalence hypothesis” can give a useful and relatively easy to use tool for proving that a language is not recognized by a CPA/HOPA.

The separating language

alphabet: [,],$

L contains words such that
- there is exactly one $
- without the dollar we have a well-bracketed word
- the suffix after $ is symmetric to a prefix of the word (including the data values) and begins with “]”

for example:

$$\begin{array}{ccccccccccccccc} [& [&] & [& [&] & [& [&] &] & \$ &] & [&] &] \\ 0 & 1 & 2 & 3 & 4 & 5 & 5 & 5 & 6 & 7 & 8 & 3 & 2 & 1 & 0 \end{array} \in U$$

when we see a dollar, there is exactly one possible suffix

L can be recognized by a deterministic CPA of level 2

The separating language

L can be recognized by a deterministic CPA of level 2

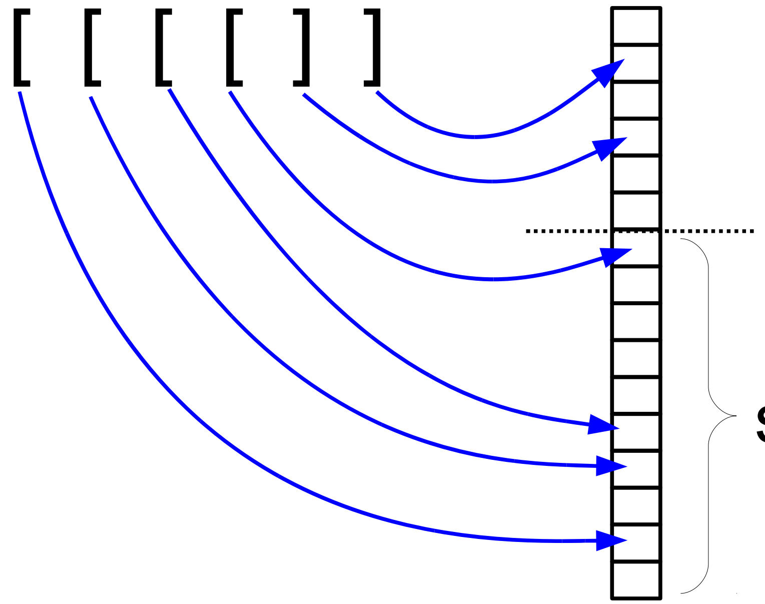
Now we will prove that L is not recognized by a deterministic PDA of level 1 (the proof for higher levels uses similar ideas).

A – deterministic PDA of level 1, which recognizes L
Take the word $[^N]^N$ with all data values different.
for big enough N - greater than the number of states

Observation 1:
At each moment, all data values read till now are present on the stack.
If not, we can continue with $[^N]^N[\$][^N]^N$, and the automaton cannot check if the data values in the second part are equal to those in the first part.

The separating language

At each moment, all data values read till now are present on the stack.



What happens for the word [^{N}]^k\$]$^{N-k}$?

The data value after \$ has to be compared with a data value which appears only inside s.
So after \$, but before], we are in stack s (and some state)

As N>number of states, we are in the same configuration after reading [^{N}]^k\$ and [^{N}]^m\$ for some $k \neq m$.
Thus the word [^{N}]^m\$]$^{N-k}$ will be accepted (for some data values).

The separating language

L can be recognized by a deterministic CPA of level 2

A – deterministic PDA of level n, which recognizes L
We consider word w_n:

$w_0 = [$ $\quad$ $w_{k+1} = w_k{}^N\,]^N[$ $\quad$ (for big enough N)

A trick like previously shows that:
for each fragment w_k, a data value appearing in this fragment is not present in the topmost stack of level k just after reading this fragment.

Main difficulty: a stack (of some level k) can be visited only in finite number of ways.

Nontrivial for automata of level ≥3

Summary

- higher-order pushdown automata with data were introduced
- we prove that in this setting collapse increases the expressive power
- a hypothesis how they are related to normal automata (without data)

Open problems:

- prove this hypothesis
- extend to nondeterministic automata
  - the hypothesis
  - the proof that collapse increases the expressive power

We thank M.Bojańczyk for suggesting the idea of introducing data values to higher-order pushdown automata.