Pushdown Hierarchies and the Safe Lambda-Calculus

Luke Ong

Oxford University Computing Laboratory

www.comlab.ox.ac.uk/oucl/work/luke.ong/

(Joint work with Jolie de Miranda)

Background: Verification of HOPL-computation

Higher-Order Procedural Languages. E.g. ML, C, Reynold's Idealized Algol (IA).

Recent results obtained using fully abstract game semantics:

Fragments of finitary IA	**Is observational equivalence decidable?**
2nd-order	Yes. (Ghica+McCusker 00)
2nd-order + iteration	Yes (GM 00); PSPACE-complete (Murawski 03)
2nd-order + recursion	No. (Ong LICS 02)
3rd-order	Yes: reduction to DPDA Equivalence. (Ong 02)
4th-order or higher	No. (Murawski LICS 03)
3rd-order + iteration	Yes. Rationally innocent strategies.

Computaton: E.g. Hierarchy of purely functional programs defined by recursion (i.e. essentially type-levels of PCF)?

Properties: Other (or weaker) than observational equivalence? E.g. decidable fragments of MSO logic.

Four Hierarchies of Finitely-Presentable Structures

Class of Structures	Hierarchy
string languages	Chomsky (1960's)
string languages	Maslov (1974) and others
(term) trees	Knapik-Niwiński-Urzyczyn (2002)
labelled graphs	Caucal (2002)

Review: Chomsky Hierarchy

A hierarchy of (string) languages. Four classes:

Type	Language Classes	Models of Computation
Type-0	Regular	Finite automaton
Type-1	Context-free	Pushdown automaton
Type-2	Context-sensitive	Linearly bounded automaton
Type-3	R. e.	Turing machines

Outline of the Talk

0. Background

I. **Maslov Hierarchy: Higher-Order Pushdown Automata**

II. OI Hierarchy: Safe Lambda Calculus and Higher-Order Grammars

III. Knapik-Niwinski-Urzyczyn Hierarchy of Pushdown Trees

IV. Problems, a Result and an Example

V. Explanation and Proof Idea

VI. Further Directions

A refinement: Maslov Hierarchy (= OI Hierarchy)

An infinite hierarchy of (string) languages, arguably "more natural" (systematic, "unifying") than Chomsky's.

Levels	Language Classes
0	Regular
1	Context-free
2	Indexed languages [Aho68]
...	...

Three equivalent devices for defining the level-$(n+1)$ languages inductively:

(1) Level-n generalized indexed languages (Maslov '74, '76)

Raising a language to a power (given by a language).

(2) Level-n pushdown automata (Maslov '74, '76, Fisher '68, Greibach '70)

Level-$(n+1)$ store is a stack of level-n stores.

(3) Level-n grammars definable in a system of derived types (Damm '82, Damm-Goerdt '86)

Level-n Stores

Fix a stack alphabet Γ with distinguished $\bot$. Define $\bot_1 = [\,\bot\,]$, $\bot_{k+1} = [\,\bot_k\,]$.

A 1-store is a non-empty sequence $[a_1, \cdots, a_l]$ of elements of Γ.
An $(n+1)$-store is a non-empty sequence $[s_1, \cdots, s_l]$ of n-stores.

For $n \geq 2$, level-n operations, Op_n: defined over n-stores

$$\begin{cases} \mathsf{push}_n\,[s_1,\cdots,s_l] &= [s_1,s_1,\cdots,s_l] \\ \mathsf{push}_k\,[s_1,\cdots,s_l] &= [\mathsf{push}_k\,s_1,s_2,\cdots,s_l], \quad 2 \leq k < n \\ \mathsf{push}_1^a\,[s_1,\cdots,s_l] &= [\mathsf{push}_1^a\,s_1,s_2,\cdots,s_l] \\ \mathsf{pop}_n\,[s_1,\cdots,s_l] &= [s_2,\cdots,s_l] \\ \mathsf{pop}_k\,[s_1,\cdots,s_l] &= [\mathsf{pop}_k\,s_1,s_2,\cdots,s_l], \quad 1 \leq k < n \\ \mathsf{id}\,[s_1,\cdots,s_l] &= [s_1,\cdots,s_l] \end{cases}$$

$$\begin{aligned} \mathsf{top}_n\,[s_1,\cdots,s_l] &= s_1 \\ \mathsf{top}_k\,[s_1,\cdots,s_l] &= \mathsf{top}_k\,s_1, \quad 1 \leq k < n \end{aligned}$$

($\mathsf{push}_k\ s$ undefined if $\mathsf{top}_k\ s$ has only 1 element.)

nPDA: Level-n Pushdown Automaton $A = \langle\, Q, \Sigma, \Gamma, q_0, \Delta \,\rangle$

Maslov 76 (Greibach 70). We follow definition in [KNU02].

By definition, 0PDAs are DFAs, and 1PDAs are PDAs. For $n \geq 2$, we have:

- Input alphabet Σ, Stack alphabet Γ.
- Control states Q, initial state q_0
- Transition relation: $\Delta \subseteq Q \times (\Sigma \cup \{\,\epsilon\,\}) \times (\Gamma \cup \{\,\bot\,\}) \times Q \times Op_n$

Configuration: (q, s) where s is an n-store. Initial configuration: $(q_0, \bot_n)$.

Define: $(q,s) \xrightarrow{a} (q',s')$ just if $(q, a, \mathsf{top}_1(s), q', \theta) \in \Delta$ where $\theta s = s'$.
Inductively $(q,s) \xrightarrow{wa} (q',s')$ just if $(q,s) \xrightarrow{w} (q'',s'')$ and $(q'',s'') \xrightarrow{a} (q,s)$, for some q'', s''.

A accepts $w \in \Sigma^*$ just if $(q_0, \bot_n) \xrightarrow{w} (q, \bot_n)$, some q.

Examples of 2PDA- or Indexed Languages

$\{ a^n b^n c^n : n \geq 1 \}$ (not context-free)

Idea: Check $a^n b^n$ using the top 1-store, then check c^n against length of 2-store.

$$\begin{array}{ll}
 & q_0, [[\bot]] \\
\xrightarrow{a} & q_0, [[Z,\bot],[\bot]] \\
\xrightarrow{a} & q_0, [[Z,Z,\bot],[Z,\bot],[\bot]] \\
\xrightarrow{a} & q_0, [[Z,Z,Z,\bot],[Z,Z,\bot],[Z,\bot],[\bot]] \\
\xrightarrow{b} & q_1, [[Z,Z,\bot],[Z,Z,\bot],[Z,\bot],[\bot]] \\
\xrightarrow{b} & q_1, [[Z,\bot],[Z,Z,\bot],[Z,\bot],[\bot]] \\
\xrightarrow{b} & q_1, [[\bot],[Z,Z,\bot],[Z,\bot],[\bot]] \\
\xrightarrow{c} & q_2, [[Z,Z,\bot],[Z,\bot],[\bot]] \\
\xrightarrow{c} & q_2, [[Z,\bot],[\bot]] \\
\xrightarrow{c} & q_2, [[\bot]]
\end{array}$$

Outline of the Talk

0. Background: Four Hierarchies
I. Maslov Hierarchy: Higher-Order Pushdown Automata
II. **OI Hierarchy: Safe Lambda Calculus and Higher-Order Grammars**
III. Knapik-Niwinski-Urzyczyn Hierarchy of Pushdown Trees
IV. Problems, a Result and an Example
V. Explanation and Proof Idea
VI. Further Directions

OI Hierarchy: Safe Types

Derived types: Damm '82; equivalently Safety (syntactic constraint): Knapik *et al*.

Let A range over simple types i.e. $A ::= o \mid A \to A$. Each A can be uniquely written $(A_1, \cdots, A_n, o)$, meaning $A_1 \to \cdots \to A_n \to o$.

Define: order$(o) = 0$; order$(A \to B) = \max($order$(A) + 1,$ order$(B))$.

Definition o is safe. For $n \geq 1$, $A = (A_1, \cdots, A_n, o)$ is safe just if order$(A_1) \geq$ order$(A_2) \geq \cdots \geq$ order(A_n), and each A_i is safe.

Assume $A = (\underbrace{A_{11}, \cdots, A_{1l_1}}_{\overline{A_1}}, \cdots, \underbrace{A_{r1}, \cdots, A_{rl_r}}_{\overline{A_r}}, o)$ is safe; write

$$A \;=\; (\overline{A_1} \mid \cdots \mid \overline{A_r} \mid o)$$

to mean: all types in each sequence $\overline{A_i} = A_{i1}, \cdots, A_{il_i}$ have the same order n_i (say), and $i > j \iff n_i > n_j$, making explicit the type paritions.

Safe λ-Calculus: System $\mathcal{S}$ Typing Rules

$$\frac{(\overline{A_1} \mid \cdots \mid \overline{A_n} \mid B) \text{ safe} \qquad b \text{ is a type-}B \text{ constant}}{\overline{x_1} : \overline{A_1} \mid \cdots \mid \overline{x_n} : \overline{A_n} \vdash b : B}$$

$$\frac{(\overline{A_1} \mid \cdots \mid \overline{A_n} \mid A_{ni}) \text{ safe}}{\overline{x_1} : \overline{A_1} \mid \cdots \mid \overline{x_n} : \overline{A_n} \vdash x_{ni} : A_{ni}}$$

$$\frac{\overline{x_1} : \overline{A_1} \mid \cdots \mid \overline{x_{n+1}} : \overline{A_{n+1}} \vdash M : B}{\overline{x_1} : \overline{A_1} \mid \cdots \mid \overline{x_n} : \overline{A_n} \vdash \boldsymbol{\lambda} \overline{x_{n+1}}.M : (\overline{A_{n+1}} \mid B)}$$

$$\frac{\Gamma \vdash M : (\overline{B_1} \mid \cdots \mid \overline{B_m} \mid o) \qquad \Gamma \vdash N_1 : B_{11} \;\cdots\; \Gamma \vdash N_{l_1} : B_{1l_1}}{\Gamma \vdash M N_1 \cdots N_{l_1} : (\overline{B_2} \mid \cdots \mid \overline{B_m} \mid o)}$$

When forming abstraction, all variables of the (right-most) type-partition must be abstracted. When forming application, the operator-term must be applied to all operand-terms (one for each type) of the left-most type-partition.

Safe λ-Calculus: Observations and Examples

Observations. Suppose $\overline{x_1} : \overline{A_1} \mid \cdots \mid \overline{x_n} : \overline{A_n} \vdash M : B$ is $\mathcal{S}$-valid, where $B = (\overline{B_1} \mid \cdots \mid \overline{B_m} \mid o)$. Then

(i) $(\overline{A_1} \mid \cdots \mid \overline{A_n} \mid \overline{B_1} \mid \cdots \mid \overline{B_m} \mid o)$ is safe.

(ii) Any free variable of M has order $\geq$ order(M).

(iii) For any subterm $\boldsymbol{\lambda}\phi.N$ of M, if variable x occurs in N and order$(x) <$ order(ϕ) then x is bound in N.

Examples

1. $F : ((o,o),o,o,o), \phi : (o,o), x : o, y : o \vdash F(F\phi x)xy : o$ is not safe:
   Reason: $F : ((o,o),o,o,o), \phi : (o,o), x : o \vdash F\phi x : (o,o)$ is not safe.
2. But $F : ((o,o),o,o,o), \phi : (o,o) \vdash F\phi a : (o,o)$ is safe for constant a.
3. $F : ((o,o),o,o,o), \phi : (o,o), x : o, y : o \vdash F\phi xy : o$ is safe.

What does "safe" mean?

Capture-avoiding substitution is commonly achieved using "Barendregt's Variable Convention".

The key clause in definition of capture-avoiding substitution:

$$(\boldsymbol{\lambda}x.M)[N/y] \stackrel{\text{def}}{=} \boldsymbol{\lambda}z.((M[z/x])[N/y]) \quad \text{where "}z\text{ is fresh"}$$

Suppose one is restricted to only n fresh names, for fixed n. There exists a λ-term such that variable-capture occurs in some reduction sequence from it.

Happily in safe λ-calculus, it is safe to use capture-permitting substitution when contracting β-redexes.

Proviso: We only perform $M[N_1/x_1, \cdots, N_n/x_n]$ provided:

(i) $x_1, \cdots, x_n$ are all the free variables of same order in M, and

(ii) The n replacement actions take place simultaneously.

Lemma. "In safe λ-calculus, it is safe not to rename bound variables afresh when performing substitution."

Proof idea. Take order$(\Phi) = 2$, order$(\phi) =$ order$(\psi) = 1$, order$(x) = 0$. Suppose we do not rename bound variables in:

$$\underbrace{(\cdots \boldsymbol{\lambda}\Phi.(\cdots \boldsymbol{\lambda}x.\cdots \phi \cdots) \cdots \boldsymbol{\lambda}\psi.(\cdots \phi \cdots) \cdots)}_{M}[G\,\Phi\psi x/\phi]$$

Three types of variable capture may occur.

Type-1 capture: variable bound has order $>$ order(ϕ)

$$(\cdots \boldsymbol{\lambda}\Phi.\underbrace{(\cdots \phi \cdots)}_{L} \cdots)[G\Phi\psi x/\phi] \quad \text{becomes} \quad \cdots \boldsymbol{\lambda}\Phi.(\cdots (G\Phi\psi x) \cdots) \cdots.$$

Impossible because $\boldsymbol{\lambda}\Phi.L$ safe implies L has no free variables of order $<$ order(Φ).

Type-2 capture: variable bound has order $<$ order(ϕ)

$$(\cdots \boldsymbol{\lambda}x.(\cdots \phi \cdots) \cdots)[G\Phi\psi x/\phi] \quad \text{becomes} \quad \cdots \boldsymbol{\lambda}x.(\cdots (G\Phi\psi x) \cdots) \cdots.$$

Impossible because $\underbrace{G\Phi\psi x}_{N}$ (of order 1) safe implies N has no free variables of order < 1.

Type-3 capture: variable bound has order $=$ order(ϕ)

$$\underbrace{(\cdots \boldsymbol{\lambda}\psi.(\cdots \phi \cdots) \cdots)}_{M}[G\Phi\psi x/\phi] \quad \text{becomes} \quad (\cdots \boldsymbol{\lambda}\psi.(\cdots (G\Phi\psi x) \cdots) \cdots).$$

Impossible because abstraction formation rule would force M to be $(\cdots \boldsymbol{\lambda}\psi\phi.(\cdots \phi \cdots) \cdots)$, making ϕ a bound variable of M.

Higher-Order Grammar

Fix a typed alphabet Σ of symbols. Two versions:

- For generating string languages: all Σ-symbols of type (o, o) with distinguished end-of-word marker $e : o$. E.g. for $a, b : (o, o)$, $a(be)$ corresponds to word $a\,b$.
- For generating term-trees: all Σ-symbols of order at most 1 (as "function symbols").

Level-n Grammar: $G = \langle N, V, \Sigma, \mathcal{R}, S \rangle$

- N set of typed non-terminals, with start $S \in N$; and V set of typed variables
- $\mathcal{R}$ is finite set of rules

$$F x_1 \cdots x_n \rightarrow E$$

where $F : (A_1, \cdots, A_m, o) \in N, x_i \in V, x_1 : A_1, \cdots, x_m : A_m \vdash E : o$ an applicative term-in-context with "constants" from $N \cup \Sigma$.

Say G is n-level grammar if highest order of $F \in \mathcal{R}$ is n, and safe if all types and terms occurring in the definition are safe.

Example: A safe 2-grammar that generates $\{ a^n b^n c^n : n \geq 1 \}$

$$\left\{ \begin{array}{rcl} S & \rightarrow & a(A\,C\,e) \\ A\,\phi\,x & \rightarrow & a\,(A\,(B\phi)\,x) \\ A\,\phi\,x & \rightarrow & b\,(\phi\,x) \end{array} \quad \begin{array}{rcl} B\,\phi\,x & \rightarrow & b\,(\phi\,(C\,x)) \\ B\,\phi\,x & \rightarrow & c\,x \\ C\,x & \rightarrow & c\,x \end{array} \right.$$

$$\begin{array}{rcl} S & \rightarrow & a\,(A\,C\,e) \\ & \rightarrow & a^2\,(A\,(B\,C)\,e) \\ & \rightarrow & a^3\,(A\,(B\,(B\,C))\,e) \\ & \rightarrow & a^3 b\,(B\,(B\,C)\,e) \\ & \rightarrow & a^3 b^2\,(B\,C\,(C\,e)) \\ & \rightarrow & a^3 b^3\,(C\,(C\,(C\,e))) \\ & \rightarrow & a^3 b^3 c\,(C\,(C\,e)) \\ & \rightarrow & a^3 b^3 c^2\,(C\,e) \\ & \rightarrow & a^3 b^3 c^3\,e \end{array}$$

(An example of an unsafe 2-grammar later on.)

OI Hierarchy = Maslov Hierarchy

Theorem. (Damm-Goerdt). A string language is accepted by an nPDA iff it is generated by some safe n-grammar.

Open problems concerning Maslov Hierarchy

Not much is known about level-3 and above.

1. Pumping Lemma (or Myhill-Nerode-type results)

   There are "pumping lemmas" for levels 0, 1 and 2 ([Hay73,Gil96]).

   Pace Blumensath '04 for whole Maslov Hierarchy – runs are pumpable, conditions given as lengths of runs and configuration size.

2. Logical Characterization.

   Regular languages are exactly those that are MSO definable (Buchi '60).

   There is a characterization of context-free languages using quantification over matchings [LST94].

3. Complexity-Theoretic Characterization.

   Engelfriet '83, '91: characterizations of languages accepted by alternating / two-way / multi-head / space-auxiliary nPDAs in terms of time-complexity classes (but no result for Maslov Hierarchy itself).

4. Relationship with Chomsky Hierarchy. E.g. Is nPDA context-sensitive, for $n \geq 3$?

Outline of the Talk

0. Background: Four Hierarchies
I. Maslov Hierarchy: Higher-Order Pushdown Automata
II. OI Hierarchy: Safe Lambda Calculus and Higher-Order Grammars
III. **Knapik-Niwinski-Urzyczyn Hierarchy of Pushdown Trees**
IV. Problems, a Result and an Example
V. Explanation and Proof Idea
VI. Further Directions

Σ-Trees (or Trees given by Σ-terms)

Fix a typed alphabet Σ of symbols of order at most 1.

A Σ-tree is a map $t : T \longrightarrow \Sigma$ where T is a prefix-closed subset of ω^*, and for $k \geq 0$, whenever $t(w) : o^k \to o$ then w has exactly k successors in T which are $w1, \cdots, wk$.

A Σ-tree is just a (possibly infinite) applicative term constructed using symbols from Σ.

Let the maximum of arities of symbols in Σ be m_Σ. A Σ-tree t can be viewed as a logical structure over the relational vocabulary

$$R_\Sigma = \{ p_f : f \in \Sigma \} \cup \{ d_i : 1 \leq i \leq m_\Sigma \}$$

with $\mathrm{ar}(p_f) = 1$ and $\mathrm{ar}(d_i) = 2$:

$$\mathbf{t} \ = \ \langle \mathrm{dom}(t), \{ p_f^{\mathbf{t}} : f \in \Sigma \}, \{ d_i^{\mathbf{t}} : 1 \leq i \leq m_\Sigma \} \rangle$$

where $p_f^{\mathbf{t}} = \{ w \in \mathrm{dom}(t) : t(w) = f \}$ and $d_i^{\mathbf{t}} = \{ (w, wi) : wi \in \mathrm{dom}(t) \}$.

Knapik-Niwinski-Urzyczyn Hierarchy of Pushdown Trees

Let G be a deterministic n-grammar. Define the Σ-tree, $[\![G]\!]$, to be the possibly infinite "term tree" generated by unfolding the rules in G *ad infinitum*.

> **Theorem** [KNU02]. A Σ-tree is generated by a safe (deterministic) n-grammar iff it is generated by an nPDA.

A remarkable decidability result

Monadic Second-Order Logic (MSO).

Extension of first-order logic with monadic second-order variables ($X, Y, Z, \cdots$), ranging over sets of elements. Fix a relational vocabulary $R = \{ r_1, \cdots, r_n \}$ where r_i is a relation symbol with arity $\mathrm{ar}(r_i)$. The atomic formulas over R are

$$X(z), \quad x = y, \quad r_i(x_1, \cdots, x_{ar(r_i)});$$

we have the usual boolean connectives, and quantification over both types of variables.

MSO is expressive: E.g. "a node-set is finite", or "forms a path", but MSO cannot count.

> **Theorem** [KNU02]. For $n \geq 0$, for any safe deterministic n-grammar G, the MSO theory of the Σ-tree $[\![G]\!]$ is decidable.

Timeline of results

- Rabin 1969: "Mother of all decidability results": Second-order theory of two successors ($S2S$) has a decidable MSO theory.
- Muller and Schupp 1985: Pushdown graphs have decidable MSO theories.
- Courcelle 1995: Σ-trees generated by 1-grammars have decidable MSO theories.
- Knapik, Niwiński and Urzyczyn 2001: Σ-trees generated by safe deterministic 2-grammars have decidable MSO theories.
- KNU '02: For all n, the MSO theories of Σ-trees generated by safe deterministic n-grammars are decidable.

How did they prove it?

A reduction argument: They exhibit effective transformations of level-$(n+1)$ pushdown trees t to level-n pushdown trees $\widehat{t}$, and of MSO-formulas ϕ to $\widehat{\phi}$, such that $t \vDash \phi \iff \widehat{t} \vDash \widehat{\phi}$.

Crucial dependence on safety.

The level-n Σ^+-tree $\widehat{t}$ uses additional symbols from

$$\{\, @ : (o,o,o) \,\} \cup \underbrace{\{\, x_1 : o, \cdots, x_l : o \,\}}_{L} \cup \{\, \lambda x : (o,o) : x \in L \,\}$$

and has "back edges" from type-0 x to the binding λx. (So $\widehat{t}$ is really the corresponding Lamping graph.)

Both crucial conditions

(i) L (obtained from G), and hence Σ^+, is finite

(ii) The structure $\widehat{t}$ is MSO-definable in the structure t.

require the generating grammar G to be safe.

Outline of the Talk

0. Background: Four Hierarchies
I. Maslov Hierarchy: Higher-Order Pushdown Automata
II. OI Hierarchy: Safe Lambda Calculus and Higher-Order Grammars
III. Knapik-Niwinski-Urzyczyn Hierarchy of Pushdown Trees
IV. **Problems, a Result and an Example**
V. Explanation and Proof Idea
VI. Further Directions

Problems and a Result

(1) (String languages) Can every string language generated by an unsafe n-grammar be generated by a safe n-grammar?

Expressivenes: Is safety a real or spurious typing constraint for defining languages?

(1') (Term trees) Can every Σ-tree generated by an unsafe n-grammar be generated by a safe n-grammar?

(2) Is the MSO theory of a Σ-tree generated by an unsafe n-grammar decidable?

Note: Yes to (1') implies yes to (2), thanks to [KNU02].

To date, we have only solved a small part of (1).

Theorem. Every string language generated by an unsafe 2-grammar is generated by a (non-deterministic) safe 2-grammar.

Example: Urzyczyn's(?) Language U

$$w\,\underbrace{*\cdots*}_{n}$$

- w is a proper prefix of a well-bracketed word that ends with a **(**
- each parenthesis in w is implicitly labelled with a number, and n is the label of the last parenthesis (of w).

Labelling rules:

I. Each **(** is labelled with the number of **(**'s read thus far.

II. Each **)** is labelled with the label of the parenthesis that precedes the matching **(**.

Example.

(	(	(	(	)	)	(	(	)	(	(	)	)	)	(	(	)	)	*	*
1	2	3	4	3	2	5	6	5	7	8	7	5	2	9	10	9	2		

An unsafe 2-grammar that generates U

Configuration: $\langle \gamma, y, z \rangle$ where

- γ is a list of future **)**-labels
- y is the number of **(**'s read thus far
- z is the label of the last parenthesis read,

Note: $|\gamma|$ is the number of unmatched **(**'s at that point.

Transition rules:

$$\begin{array}{rcl} \langle x : \phi, y, z \rangle & \xrightarrow{(} & \langle z : x : \phi, y+1, y+1 \rangle \\ \langle x : \phi, y, z \rangle & \xrightarrow{)} & \langle \phi, y, x \rangle \\ \langle x : \phi, y, z \rangle & \xrightarrow{*} & z \\ z+1 & \xrightarrow{*} & z \end{array}$$

An unsafe 2-grammar that generates U

$$\begin{array}{rcl} \Sigma & = & \{\, \mathbf{(} : (o,o), \mathbf{)} : (o,o), * : (o,o), e : o \,\} \\ N & = & \{\, S : o, D : ((o,o,o),o,o,o,o), G : (o,o,o), F : (o,o) \,\} \end{array}$$

with variables $\phi : (o,o,o)$ and $x, y, z : o$. The corresponding rules are:

$$\begin{array}{rcl} D\,\phi\,x\,y\,z & \rightarrow & \mathbf{(}\,(D\,(D\,\phi\,x)\,z\,(F\,y)\,(F\,y)) \\ D\,\phi\,x\,y\,z & \rightarrow & \mathbf{)}\,(\phi\,y\,x) \\ D\,\phi\,x\,y\,z & \rightarrow & *\,z \\ F\,x & \rightarrow & *\,x \\ S & \rightarrow & D\,G\,e\,e\,e \end{array}$$

Question: Is there a safe 2-grammar that generates U?

An earlier conjecture(?) was that the answer is no.

A characterization of U-words

Idea: Each U-word has a unique partition into 3 parts:

$$\underbrace{(\cdots(\cdots(}_{(1)}\;\underbrace{(\cdots)\cdots(\cdots)}_{(2)}\;\underbrace{*\cdots*}_{(3)}$$

where

- (1) is prefix of a well-bracketed word such that no prefix of it is well-bracketed, and has n occurrences of **(**
- (2) is well-bracketed
- (3) has length n.

Example

(	(	(	(	)	)	(	(	)	(	(	)	)	)	(	(	)	)	*	*
1	2	3	4	3	2	5	6	5	7	8	7	5	2	9	10	9	2		

A non-deterministic 2PDA that accepts U

Verify the 3-partition of U-words in three stages:

$$\underbrace{(\cdots(\cdots(}_{(1)}\ \underbrace{(\cdots)\cdots(\cdots)}_{(2)}\ \underbrace{*\cdots*}_{(3)}$$

(1) Use the top 1-store to check word read thus far is a prefix of a well-bracketed word such that no prefix of it is well-bracketed; use push$_2$ to count the number of **(**'s read.

Non-deterministically decide to enter Stage 2 after reading a **(**.

(2) Using only the top 1-store, check word read in Stage 2 is well-bracketed.

Enter Stage 3 on reading $*$.

(3) Check the number of $*$'s read equals the number of 1-stores stacked, using pop$_2$.

Thus U is an indexed language, and thanks to [DG86]

Lemma. There is a safe 2-grammar that generates U.

Outline of the Talk

0. Background: Four Hierarchies
I. Maslov Hierarchy: Higher-Order Pushdown Automata
II. OI Hierarchy: Safe Lambda Calculus and Higher-Order Grammars
III. Knapik-Niwinski-Urzyczyn Hierarchy of Pushdown Trees
IV. Problems, a Result and an Example
V. **Explanation and Proof Idea**
VI. Further Directions

Transforming a 2-grammar to a 2PDA

Recall [KNU02]: a Σ-tree is generated by a safe 2-grammar iff it is generated by a 2PDA.

Given a 2-grammar G. Define a 2PDA A_G:

- Stack alphabet: subterms of rhs of G-rules
- Control states: $q_0, q_1, \cdots, q_m$ where m is largest arities of Σ-symbols.
- Configurations
  - (q_0, s) meaning "evaluating top$_1(s)$ i.e. working out the head symbol of top$_1(s)$"
  - $(q_i, s), i \geq 1$: meaning "working out (the head symbol of) the i-th argument of top$_1(s)$"

Rules of 2PDA A_G (KNU '02)

Every variable that occurs in a stack item is a formal parameter of the head symbol (which must be a non-terminal) of the item just below in the stack.

$$\begin{array}{rcl}
(q_0, D\bar{t}) & \xrightarrow{\epsilon} & (q_0, \mathsf{push}_1\,\mathsf{rhs}(D)) \\
(q_0, f\bar{t}) & \xrightarrow{[f,i]} & (q_i, \mathsf{id}) \quad \text{if } 0 < i \leq \mathsf{ar}(f) \\
(q_0, a) & \xrightarrow{a} & \text{accept} \quad \text{if } \mathsf{ar}(a) = 0 \\
(q_0, x_j) & \xrightarrow{\epsilon} & (q_j, \mathsf{pop}_1) \\
(q_0, \varphi_j t_1 \cdots t_n) & \xrightarrow{\epsilon} & (q_j, \mathsf{push}_2\,;\,\mathsf{pop}_1) \\
j \geq 1, (q_j, \$t_1 \cdots t_n) & \xrightarrow{\epsilon} & \begin{cases} (q_0, t_j) & \text{if } j \leq n \text{ (argument present)} \\ (q_{j-n}, \mathsf{pop}_2) & \text{if } j > n \text{ (argument missing)} \end{cases}
\end{array}$$

Example: An unsafe grammar

```
   S -> Dgab
D@xz -> h (D(D@x)z(@z)) (H(fz)x) (@z)
 H@x -> @x
```

(To save writing, leave state `q0` and ϵ-transition out.)

```
    [[Dgab,S]]

h1  [[D(D@x)z(@z),Dgab,S]]

h3  [[@z,D(D@x)z(@z),Dgab,S]]

    [[D@x,Dgab,S],[@z,D(D@x)z(@z),Dgab,S]]

h2  [[H(fz)x,D@x,Dgab,S],[@z,D(D@x)z(@z),Dgab,S]]

    [[@x,H(fz)x,D@x,Dgab,S],[@z,D(D@x)z(@z),Dgab,S]]
```



```
D@xz -> h (D(D@x)z(@z)) (H(fz)x) (@z)
 H@x -> @x
---
    [[fz,D@x,Dgab,S],[@x,H(fz)x,D@x,Dgab,S],
     [@z,D(D@x)z(@z),Dgab,S]]

f1  [[z,D@x,Dgab,S],[@x,H(fz)x,D@x,Dgab,S],
     [@z,D(D@x)z(@z),Dgab,S]]

q3  [[D@x,Dgab,S],[@x,H(fz)x,D@x,Dgab,S],
     [@z,D(D@x)z(@z),Dgab,S]]

q1  [[@x,H(fz)x,D@x,Dgab,S],[@z,D(D@x)z(@z),Dgab,S]]
    ...
q2  [[Dgab,S],[@z,D(D@x)z(@z),Dgab,S]]

    [[a,S],[@z,D(D@x)z(@z),Dgab,S]]
```


2PDAL: 2PDA with Links

The word $h1.h3.h2.f1.a$ is not in the branch language of $[\![G]\!]$. Transformation works only for safe n-grammars.

How to remedy it? Idea:

We do a push$_2$ (followed by a pop$_1$) on account of a level-1 head variable ϕ of the top of stack.

Why? So that the missing arguments of ϕ, if needed later, can be accessed from the associated 1-store buried somewhere in the stack.

After the push$_2$, make an explicit (and fresh) link from the subterm pointed to by ϕ, to the 1-store just below. So that subsequently when we do a corresponding pop$_2$, we will do as many pop$_2$ as required to reach the 1-store thus linked.

Indicate start and end of a link by superscripts $\langle n+ \rangle$ and $\langle n- \rangle$ respectively, for $n \geq 0$.

Of course, we will need an unbounded(!) number of such links.

Example: Running 2PDAL

$$\begin{cases} S & \rightarrow & Dgab \\ D\phi xz & \rightarrow & h\ (D(D\phi x)z(\phi z))\ (H(fz)x)\ (\phi z) \\ H\phi x & \rightarrow & \phi x \end{cases}$$

$$[\,[\,Dgab, S\,]\,]$$

$$\xrightarrow{h1} [\,[\,D(D\phi x)z(\phi z), Dgab, S\,]\,]$$

$$\xrightarrow{h3} [\,[\,\phi z, D(D\phi x)z(\phi z), Dgab, S\,]\,]$$

$$\longrightarrow [\,[\,D\phi x^{\langle 1- \rangle}, Dgab, S\,], [\,\phi z^{\langle 1+ \rangle}, D(D\phi x)z(\phi z), Dgab, S\,]\,]$$

$$\xrightarrow{h2} [\,[\,H(fz)x, D\phi x^{\langle 1- \rangle}, Dgab, S\,], [\,\phi z^{\langle 1+ \rangle}, D(D\phi x)z(\phi z), Dgab, S\,]\,]$$

$$\begin{cases} D\phi x z & \rightarrow & h\ (D(D\phi x)z(\phi z))\ (H(fz)x)\ (\phi z) \\ H\phi x & \rightarrow & \phi x \end{cases}$$

$\longrightarrow \quad [\,[\,\phi x, H(fz)x, D\phi x^{\langle 1-\rangle}, Dgab, S\,], [\,\phi z^{\langle 1+\rangle}, D(D\phi x)z(\phi z), Dgab, S\,]\,]$

$\longrightarrow \quad [\,[\,fz^{\langle 2-\rangle}, D\phi x^{\langle 1-\rangle}, Dgab, S\,], [\,\phi x^{\langle 2+\rangle}, H(fz)x, D\phi x^{\langle 1-\rangle}, Dgab, S\,],$
$[\,\phi z^{\langle 1+\rangle}, D(D\phi x)z(\phi z), Dgab, S\,]\,]$

$\xrightarrow{f1} \quad [\,[\,z, D\phi x^{\langle 1-\rangle}, Dgab, S\,], [\,\phi x^{\langle 2+\rangle}, H(fz)x, D\phi x^{\langle 1-\rangle}, Dgab, S\,],$
$[\,\phi z^{\langle 1+\rangle}, D(D\phi x)z(\phi z), Dgab, S\,]\,]$

$\longrightarrow q_3 \quad [\,[\,D\phi x^{\langle 1-\rangle}, Dgab, S\,], [\,\phi x^{\langle 2+\rangle}, H(fz)x, D\phi x^{\langle 1-\rangle}, Dgab, S\,],$
$[\,\phi z^{\langle 1+\rangle}, D(D\phi x)z(\phi z), Dgab, S\,]\,]$

$\longrightarrow q_1 \quad [\,[\,\phi z^{\langle 1+\rangle}, D(D\phi x)z(\phi z), Dgab, S\,]\,]$

$\longrightarrow \quad [\,[\,z, D(D\phi x)z(\phi z), Dgab, S\,]\,]$

$\longrightarrow q_3 \quad [\,[\,D(D(\phi z)z(\phi z), Dgab, S\,]\,]$

$\longrightarrow \quad [\,[\,\phi z, Dgab, S\,]\,]$

$\longrightarrow \quad [\,[\,g^{\langle 3-\rangle}\,], [\,\phi z^{\langle 3+\rangle}, Dgab, S\,]\,]$

$\xrightarrow{g1} \quad [\,[\,z, Dgab, S\,]\,]$

$\longrightarrow \quad [\,[\,b\,]\,]$

Simulating 2PDALs by non-deterministic safe 2PDAs

In some cases of push$_2$, we will never ever go back down to the associated 1-store (because no missing argument of that type-1 item is required in that particular run).

If we guess that some missing argument of the type-1 item in queston will be accessed (later in that particular run), we make an explicit link as before, but always using the same pair of start-of-link ($-$) and end-of-link ($+$) markers.

However we would be penalised (i.e. force to abort) if at some point a present (as opposed to missing) argument of a $\langle - \rangle$-marked is accessed instead.

This will turn out to be enough if we maintain an invariant:

Assume the top 1-store contains at least one item marked with a start marker. If the stack item is closest to the top, then the corresponding $+$ will be found in the first 1-store beneath it whose topmost item is marked $+$.

Further directions

Extension to level-3 and beyond.

The case of level-2 pushdown trees.

Kasai's Hierarchy of context-sensitive languages.

$$\{\, a_1^n \cdots a_{l+1}^n : n \geq 1 \,\} \;\in\; K_l \setminus K_{l-1}$$