Type homogeneity is not a restriction for safe recursion schemes

William Blum

June 23, 2009

Abstract

We show that for generating trees, order-n pushdown automata are equivalent to order-n *incrementally-bound* recursion schemes. This generalizes Knapik *et al.*'s result about equi-expressivity of pushdown automata and *safe* (*i.e.*, incrementally-bound and homogeneously-typed) recursion schemes [5].

NB: A proof attempt was presented in a previous version of this note. Broadbent has then obtained a working proof based on the same idea but using a different notion of stack-safety [2]. This note fixes the hole in my original proof by introducing yet another definition of stack safety inspired by Broadbent's one.

1 Introduction

Hague, Murawski, Ong and Serre [4] introduced higher-order collapsible push-down automata (CPDA). They prove that this type of device is equivalent to higher-order recursion schemes: given an order-n recursive schemes G we can construct an equivalent order-n CPDA (in the sense that they generate the same tree), and conversely. Here we show that if the recursion scheme is incrementally-bound then the generated automaton can easily be changed into an equivalent order-n (non-collapsible) push-down automaton (PDA). Conversely, given an order-n PDA, the recursion scheme generated by Hague *et al.*'s transformation induces an equivalent incrementally-bound and homogeneously-typed recursion scheme.

Consequently, pushdown automata are equivalent to incrementally-bound recursion schemes. Further, by applying the above two transformations, every incrementally-bound recursion scheme can be converted to an equivalent *safe* one (*i.e.*, incrementally-bound and type-homogeneous). Type-homogeneity is therefore not a proper restriction for safe recursion schemes. This generalizes Knapik *et al.*'s results about equi-expressivity of pushdown automata and *safe* recursion schemes [5].

2 Notations

Stack An element of the CPDA stack is written $a^{(j,k)}$ where $a \in \Gamma$ and the exponent (j, k) encodes the pointer associated to the stack symbol. The value

j is called the order of the link, and k is called the height of the link. We use the following abbreviations:

- "$\mathit{push}_1^{a,j}$" for the operation $\mathit{push}_1\, a^{(j,1)}$;
- "push_1^{a}" for the operation $\mathit{push}_1\, a^{(j,k)}$ where the components j and k are undetermined.

Given an order-n CPDA, we call ***configuration*** an order-n stack.

> Add definition of stack prefix "$s_{\leqslant m}$" and "$s_{<m}$" [4].

Definition 2.1. Let s be a higher-order stack. We define $s^{\langle j\rangle}$ as the operation that replaces every link occurring in s of the form (j,k) by $(j,k+1)$. Formally,

$$\begin{aligned} {a^{(j,k)}}^{\langle j\rangle} &= a^{(j,k+1)} \\ {a^{(j,k)}}^{\langle j'\rangle} &= a^{(j,k)} && \text{when } j \neq j', \\ [s_1 \dots s_p]^{\langle j\rangle} &= [s_1^{\langle j\rangle} \dots s_p^{\langle j\rangle}]\ . \end{aligned}$$

This operation clearly commutes with prefixing. We will therefore write $s_{\leqslant m}^{\langle j\rangle}$ as an abbreviation for $(s^{\langle j\rangle})_{\leqslant m} = (s_{\leqslant m})^{\langle j\rangle}$, for all stack s and stack-symbol m occurring in s.

We reproduce here the definition of the CPDA operation push_j [4]:

$$\mathit{push}_j \underbrace{[s_1 \dots s_{l+1}]}_{s} = \begin{cases} [s_1\ \dots\ s_{l+1}\ s_{l+1}^{\langle j\rangle}] & \text{if } j = \operatorname{ord} s; \\ [s_1\ \dots\ s_{l+1}\ \mathit{push}_j\ s_{l+1}] & \text{if } j < \operatorname{ord} s. \end{cases}$$

Convention 2.1 (Top stack convention). In the original definition, the operation top_i, that returns the top $(i-1)$-stack of a higher-order stack, removes any dangling pointer resulting from the operation. Here, we suppose that top_i is defined in such a way that all pointers are preserved. From an implementation viewpoint, since links are encoded as pairs of integers, this means that top_i just returns an unmodified copy of the top $(i-1)$-stack.

Recursion schemes

> Add definition of recursion scheme, ***type-homogeneity***, and ***incremental bound computation tree/graph***[1].

A recursion scheme is ***incrementally-bound*** just if its computation graph is incrementally-bound. A recursion scheme is ***safe*** if it is incrementally-bound and homogeneously-typed.

3 From recursion scheme to collapsible push-down automaton

We fix a higher-order recursion scheme G of order n.

Presentation Hague *et al.* define an order n collapsible pushdown automaton, denoted $CPDA(G)$, constructed from the recursion scheme G [3, Definition 5.2]. They show that this automaton is equivalent to G in the sense that it generates the same tree. The stack-alphabet Γ is given by the set of nodes of the computation graph of G (derived from the long eta-expansion of its rules). The automaton proceeds by computing the set of traversals of the computation tree of G. One can easily transform $CPDA(G)$ into an automaton that "prints out" the traversal that is being computed. This can be done by changing the behaviour of the $push_1$ operation to make it print out the input element before pushing it on the stack. The justification pointers can then be recovered inductively using the node labels: For a variable node, it is the only node-occurrence that binds it in the P-view at that point (which is computable by the induction hypothesis); prime lambda nodes always point to their immediate predecessor; and a non-prime lambda node is always justified by the predecessor of the justifier of the variable node preceding it.

The stack of the current configuration does not suffice in itself to reconstruct the traversal that is being computed due to the use of some "destructive" operations in the CPDA such as *collapse*. Nevertheless, two very useful pieces of information are recoverable from the configuration-stack: the O-view and the P-view of the traversal.

Let c be a configuration. The ***long O-view***, ***O-view*** and ***P-view*** of the traversal that is currently computed by the configuration c, written respectively $\lfloor c \rfloor$, $\llcorner c \lrcorner$ and $\ulcorner c \urcorner$, can be recovered as follows:

- Long O-view:

$$\lfloor s \rfloor = \begin{cases} \epsilon & \text{if } top_1 s \text{ is undefined;} \\ \lfloor pop_1 s \rfloor \cdot top_1 s & \text{if } top_1\, s \in IN_{\mathsf{var}},\ top_1 s \text{ pointing to its immediate predecesor;} \\ \lfloor pop_1 s \rfloor \cdot top_1 s & \text{if } top_1\, s \in IN_{@},\ @ \text{ having no pointer;} \\ \lfloor collapse\, s \rfloor \cdot top_1 s & \text{if } top_1\, s \in IN_{\lambda}^{\mathsf{prime}},\ top_1 s \text{ pointing to its immediate predecesor;} \\ \lfloor collapse\, s \rfloor \cdot top_1 s & \text{if } top_1\, s \in IN_{\lambda} \setminus IN_{\lambda}^{\mathsf{prime}},\ top_1 s \text{ pointing to } \mathsf{ip}(\mathsf{jp}(collapse\, s)). \end{cases}$$

- The O-view is defined similarly to the long-O-view except that the calculation stops when an @-node is reached:

$$\llcorner s \lrcorner = top_1 s \qquad \text{if } top_1 s \in IN_{@},\ @ \text{ having no pointer;}$$

- As shown in the original paper, the P-view $\ulcorner s \urcorner$ is given by $top_2\, c$.

Definition We now give the formal definition of the collapsible pushdown automaton from Hague *et al.* [4]. The presentation here differs slightly from the original one: In the case (A), when pushing the prime child of an application node @ on the stack, we assign it a link pointing to the preceding stack symbol in the top 1-stack (*i.e.*, the @-node itself). This modification avoids the case analysis on the value of j—the child-index of u's binder—in the cases (V_0) and (V_1), and the sequence of instructions pop_1^{p+1} can just be replaced by $pop_1^p; collapse$. The stack-symbols are the nodes of the computation graph of

If u's label is not a variables, the action is just a $push_1^v$, where v is an appropriate child of the node u. Precisely:

- (A) If the label is an @ then $\delta(u) = push_1^{E_0(u),\mathbf{1}}$.
- (S) If the label is a Σ-symbol f then $\delta(u) = push_1^{E_i(u)}$, where $1 \leq i \leq ar(f)$ is the direction requested by the Environment, or Opponent.
- (L) If the label is a lambda then $\delta(u) = push_1^{E_1(u)}$.

Suppose u is a variable which is the i-parameter of its binder and let p be the span of u.

- (V_1) If the variable has order $l \geq 1$, then

$$\delta(u) = push_{n-l+1}; pop_1^p; collapse; push_1^{E_i(top_1), n-l+1}$$

- (V_0) If the variable is of ground type then

$$\delta(u) = pop_1^p; collapse; push_1^{E_i(top_1)}$$

Figure 1: The transition rules of $CPDA(G)$.

G and the transition rules are given in Figure 1. The initial configuration is defined as $c_0 = push_1 \lambda \bot_n$ where λ refers to the root (dummy) lambda node of the computation graph of G.

The automaton is well-defined in the sense that no collapse can occur at an element whose link is undefined. (In particular it never occurs at non-lambda nodes.)

Reachable configurations We use the notion of reachability with respect to the $\rightarrow$-step relation introduced in the original paper: $c \rightarrow c'$ just if $c' = \delta(top_1 c)(c)$ where δ is the transition function defined Hague et al.'s paper [3, Figure 2]. In other words, a configuration is ***$\rightarrow$-reachable*** if it can be attained starting from the initial configuration c_0 by performing one or more applications of the steps (A), (S), (L), (V_1), (V_0) from the algorithm defining $CPDA(G)$. The intermediate configurations visited by the internal transitions within a step are therefore not $\rightarrow$-reachable. A configuration is said to be ***reachable*** if it is $\rightarrow$-reachable or if it an intermediate configuration computed during a $\rightarrow$-step (*i.e.*, if it can be written $(op_1; \ldots; op_k)(c)$ where c is $\rightarrow$-reachable and $op_1, \ldots, op_k$ are the first k instructions of some $\rightarrow$-step).

Link convention Observe that in $CPDA(G)$, when we push a lambda node $\lambda\overline{\xi}$ on the stack, the associated link has order 1 if it is a prime lambda node (case (A)), and $n - \operatorname{ord} \lambda\overline{\xi} + 1$ otherwise (case (V_1)). Hence, since no CPDA instruction can change the link order of an element pushed on the stack, at every stage during the execution of the CPDA the link order can be recovered from the (order of the) node itself.

From now on we will only work with stacks occurring as sub-stack of reachable configurations of $CPDA(G)$ therefore we will omit the order-component altogether when representing stack symbols: we write $\lambda\overline{\xi}^k$ to mean $\lambda\overline{\xi}^{(1,k)}$ if $\lambda\overline{\xi}$ is a prime lambda node and $\lambda\overline{\xi}^{(n-\operatorname{ord}\lambda\overline{\xi}+1,k)}$ otherwise.

4 From incrementally-bound RS to PDA

We now fix an **incrementally-bound** higher-order recursion scheme G of order n. We give a detailed analysis of $CPDA(G)$ that will be used in the next section to derive an equivalent (non-collapsible) order-n pushdown automaton.

4.1 Incremental order-decomposition

Observation Let s be a 1-stack. For any $l \in \mathbb{N}$, s can then be written

$$s = u_{r+1} \cdot \lambda\overline{\eta}_r^{k_r} \cdot u_r \cdot \ldots \cdot \lambda\overline{\eta}_1^{k_1} \cdot u_1$$

where

- $\lambda\overline{\eta}_1^{k_1}$ is the last λ-node in s with order strictly greater than l;
- for $1 < l \leq r$, $\lambda\overline{\eta}_l^{k_l}$ is the last λ-node in $s_{\leqslant\lambda\overline{\eta}_{l-1}^{k_{l-1}}}$ with order strictly greater than $\operatorname{ord}\lambda\overline{\eta}_{l-1}^{k_{l-1}}$,
- r is defined as the smallest number such that $s_{\leqslant\lambda\overline{\eta}_r^{k_r}}$ does not contain any lambda node of order strictly greater than $\lambda\overline{\eta}_r^{k_r}$.

In other words:

- for $1 \leq k \leq r$, all the lambda nodes occurring in u_l have order strictly smaller than $\operatorname{ord}\lambda\overline{\eta}_l$;
- for $1 \leq l < l' \leq r$ we have $\operatorname{ord}\lambda\overline{\eta}_l^{k_l} < \operatorname{ord}\lambda\overline{\eta}_{l'}^{k_{l'}}$;
- $r = 0$ if and only if all the lambda node in s have order $\geq l$.

The subsequence $\lambda\overline{\eta}_r^{k_r} \ldots \lambda\overline{\eta}_1^{k_1}$ of s consisting of the lambda nodes $\lambda\overline{\eta}_l^{k_l}$ defined above is called the ***incremental order-decomposition of the* 1*-stack* s with respect to** $l \in \mathbb{N}$. This sequence is uniquely determined for any given $l \in \mathbb{N}$.

We now generalize this notion to higher-order stacks.

Definition 4.1. The ***incremental order-decomposition of a higher-order stack* s with respect to** $l \in \mathbb{N}$ (or order-decomposition for short), written $\mathsf{orddec}_l(s)$, is defined as the order-decomposition of the top order-1 stack (defined using convention 2.1). Equivalently it can be defined as follows:

$$\mathsf{orddec}_l(\epsilon) = \epsilon$$

$$\text{for } s \neq \epsilon \quad \mathsf{orddec}_l(s) = \mathsf{orddec}_{\operatorname{ord}\lambda\overline{\eta}}(s_{<\lambda\overline{\eta}}) \cdot \lambda\overline{\eta}^k$$

$$\text{where } \lambda\overline{\eta}^k \text{ is the last node in } top_2\, s \text{ with order} > l\ .$$

The ***incremental order-decomposition*** of s, written $\mathsf{orddec}(s)$, is defined as:

$$\mathsf{orddec}(s) = \mathsf{orddec}_0(s) \ .$$

It follows immediately from the definition that:

$$l < l' \implies \mathsf{orddec}_{l'}(s) \leqslant \mathsf{orddec}_l(s) \tag{1}$$

where $\leqslant$ denotes the sequence-prefix ordering.

Lemma 4.1. *Let s be a (possibly higher-order) stack such that $top_1\, s \in IN_@ \cup IN_{var}$ and $l \geq 0$.*

(i) Suppose that $\mathsf{orddec}_l(s) = \langle \lambda\overline{\eta}_r^{k_r}, \ldots, \lambda\overline{\eta}_1^{k_1} \rangle$ then for any lambda node $a \in \Gamma$ and link (j,k) we have

$$\mathsf{orddec}_l(push_1 a^{(j,k)}\ s) =$$
$$\begin{cases} \mathsf{orddec}_l(s), & \textit{if } \operatorname{ord} a \leq l; \\ \langle a^k \rangle, & \textit{if } \operatorname{ord} a \geq \operatorname{ord} \lambda\overline{\eta}_r; \\ \langle \lambda\overline{\eta}_r^{k_r}, \ldots, \lambda\overline{\eta}_i^{k_i}, a^k \rangle, & \textit{otherwise,} \\ & i = \min\{ i \ \in \{1..r\} |\ \operatorname{ord} a < \operatorname{ord} \lambda\overline{\eta}_i \} \ . \end{cases}$$

(ii) For any non-lambda node $a \in \Gamma$ and link (j,k) we have

$$\mathsf{orddec}_l(push_1\, a^{(j,k)}\ s) = \mathsf{orddec}_l(s) \ .$$

(iii) If $top_1 s$ is not a lambda node then

$$\mathsf{orddec}_l(pop_1\ s) = \mathsf{orddec}_l(s) \ .$$

Proof. Follows immediately from the definition of $\mathsf{orddec}_l(s)$. □

Lemma 4.2 (Incremental binders are in the order-decomposition)**.** *Let c be a $\rightarrow$-reachable configuration of $CPDA(G)$ such that $top_1\, c$ is a variable x. Then*

(i) $\mathsf{orddec}(c)$ contains at least a node with order strictly greater than $\operatorname{ord}(x)$;

(ii) the last lambda node in $\mathsf{orddec}(c)$ satisfying the first condition is precisely x's binder.

In other words, x's binder is the last lambda node in $\mathsf{orddec}_{\operatorname{ord} x}(c)$.

Proof. (i) The top 1-stack of a $\rightarrow$-reachable configuration contains the P-view of some traversal whose last visited node is the top_1 symbol [3, Corollary 8]; and the P-view of a traversal is exactly the path (in the unfolding of) the computation graph from the last visited node to the root [6, Proposition 6]. Hence the binder of x, whose order is strictly greater than $\operatorname{ord} x$, occurs in the top 1-stack. Consequently $\mathsf{orddec}(c)$ must contain at least one node of order strictly greater than l.

(ii) Since the recursion scheme is incrementally-bound, x's binder is precisely the first λ-node in the path to the root in the computation tree with order strictly greater than x. □

4.2 Stack safety

Definition 4.2 (Safe stack)**.** Let s be an order-j non-empty stack for $j \geq 1$. The stack s is ***l-safe*** iff

1. $\mathsf{orddec}_l(s) = \langle \lambda\overline{\eta}_r^1, \ldots, \lambda\overline{\eta}_1^1 \rangle$ for some $r \geq 0$ *i.e.*, the height component of the links in $\mathsf{orddec}_l(s)$ are all equal to 1;
2. for all $1 \leq q \leq r$ such that $n - \operatorname{ord} \lambda\overline{\eta}_q + 1 \leq \operatorname{ord} s$:
   - for $q = 1$ we have *collapse* $s_{\leqslant \lambda\overline{\eta}_1}$ is l-safe;
   - for $q > 1$ we have *collapse* $s_{\leqslant \lambda\overline{\eta}_q}$ is $\operatorname{ord}(\lambda\overline{\eta}_{q-1})$-safe.

We say that s is ***safe*** if it is 0-safe.

Since s is a stack, and not necessarily a configuration, it may have dangling pointers. The condition $n - \operatorname{ord} \lambda\overline{\eta}_q + 1 \leq \operatorname{ord} s$ in the definition ensures that $\lambda\overline{\eta}_j$'s link is not dangling so that we can indeed perform a collapse at $\lambda\overline{\eta}_j$.

Lemma 4.3. *Let s be a stack such that $\mathsf{orddec}_l(s) = \langle \lambda\overline{\eta}_r^{k_r}, \ldots, \lambda\overline{\eta}_1^{k_1} \rangle$ and $l \geq 0$. If s is l-safe and $l \leq k \leq n$ then s is k-safe.*

Proof. Immediate consequence of the definition. □

Lemma 4.4 (Collapse simulation)**.** *Let s be a sub-stack of a reachable configuration of $CPDA(G)$ and $l \geq 0$. If* $\operatorname{ord} s \geq 2$ *and $top_2\, s$ is l-safe or if* $\operatorname{ord} s = 1$ *and s is l-safe then for any lambda node $\lambda\overline{\eta}$ in $\mathsf{orddec}_l(s)$ we have:*

$$collapse\, s_{\leqslant \lambda\overline{\eta}} = \begin{cases} pop_1\, s_{\leqslant \lambda\overline{\eta}} & \textit{if } \lambda\overline{\eta} \textit{ is prime,} \\ pop_{n - \operatorname{ord} \lambda\overline{\eta} + 1}\, s_{\leqslant \lambda\overline{\eta}} & \textit{otherwise.} \end{cases}$$

Proof. The *collapse* operation is defined as $collapse\, s = pop_j^k\, s$ where $(j, k) \in \mathbb{N} \times \mathbb{N}$ is the link attached to $top_1\, s$. Since s is a sub-stack of a reachable configuration we have $j = 1$ if $\lambda\overline{\eta}$ is prime and $j = n - \operatorname{ord} \lambda\overline{\eta} + 1$ otherwise. Furthermore, since $top_2\, s$ is safe and $\lambda\overline{\eta}$ belongs to the order-decomposition, the height component necessarily equals 1. □

4.2.1 Operations preserving stack safety

Lemma 4.5. *Let s be a higher-order stack. Suppose that s is l-safe, $l \geq 0$. Then:*

(i) $top_{\operatorname{ord} s}\, s$ *is l-safe;*

(ii) if $top_1\, s$ is not a lambda node then $pop_1\, s$ is l-safe;

(iii) for every non-lambda node a, $1 \leq j \leq n$, $k \geq 1$, $push_1\, a^{(j,k)}\, s$ is l-safe;

(iv) for every lambda node a, $push_1\, a^{(1,1)}\, s$ is l-safe if $\operatorname{ord} a < l$*, and safe if* $\operatorname{ord} a \geq l$.

Proof. This is a direct consequence of Lemma 4.1. For (vi), the cases $\operatorname{ord} a > l$ and $\operatorname{ord} a < l$ follow immediately from Lemma 4.1(i); for $\operatorname{ord} a = l$ it follows from the fact that $\mathsf{orddec}_0(push_1\, a^{(1,1)}\, s) = \mathsf{orddec}_l(s) \cdot a^1$. □

Lemma 4.6. *Let $0 \leq l < n$, $q \geq 0$ and s be a stack of level $1 \leq \operatorname{ord} s < n$. If s is q-safe then $s^{\langle n-l+1\rangle}$ is $\max(l,q)$-safe.*

Proof. Let s be a safe stack with $1 \leq \operatorname{ord} s < n$. We prove the result by induction on the size of s. The base case is the trivial: s is the empty stack. Step case: Since s is q-safe we have $\mathsf{orddec}_q(s) = \langle \lambda\overline{\eta}^1_r, \ldots, \lambda\overline{\eta}^1_1\rangle$. By (1), $\mathsf{orddec}_{\max(l,q)}(s)$ is a prefix of $\mathsf{orddec}_q(s)$. Let b be the index in $\mathsf{orddec}_q(s)$ of the last node of $\mathsf{orddec}_{\max(l,q)}(s)$: thus $\lambda\overline{\eta}_b$ is the last lambda node in $\mathit{top}_2\, s$ with order $> \max(l,q)$.

The stack-operation $(\cdot)^{\langle n-l+1\rangle}$ updates the pointers as follows: the height component of each link is incremented if the order of the stack symbol is l and is kept unchanged otherwise. Hence we have:

$$\mathsf{orddec}_q(s^{\langle n-l+1\rangle}) = \langle \lambda\overline{\eta}^1_r, \ldots, \lambda\overline{\eta}^1_b, \lambda\overline{\eta}^k_{b-1}, \lambda\overline{\eta}^1_{b-2} \ldots, \lambda\overline{\eta}^1_1\rangle$$

for some $1 \leq k \leq 2$. And therefore:

$$\mathsf{orddec}_{\max(l,q)}(s^{\langle n-l+1\rangle}) = \langle \lambda\overline{\eta}^1_r, \ldots, \lambda\overline{\eta}^1_b\rangle\ . \tag{2}$$

Now consider an index j such that $b \leq j \leq r$ and $n - \operatorname{ord} \lambda\overline{\eta}_j + 1 \leq \operatorname{ord} s$. Since the height component of $\lambda\overline{\eta}_j$'s link is not affected by the operation $(\cdot)^{\langle n-l+1\rangle}$, this operation commutes with *collapse* and we have:

$$\mathit{collapse}\, s^{\langle n-l+1\rangle}_{\leqslant\lambda\overline{\eta}_j} = (\mathit{collapse}\, s_{\leqslant\lambda\overline{\eta}_j})^{\langle n-l+1\rangle}\ . \tag{3}$$

By assumption s is q-safe therefore $\mathit{collapse}\, s_{\leqslant\lambda\overline{\eta}_j}$ is q-safe if $j = b = 1$ and $\operatorname{ord}(\lambda\overline{\eta}_{j-1})$-safe if $j > b \geq 1$.

Since $\mathit{collapse}\, s_{\leqslant\lambda\overline{\eta}_j}$ is strictly smaller than s, by the induction hypothesis we have that $(\mathit{collapse}\, s_{\leqslant\lambda\overline{\eta}_j})^{\langle n-l+1\rangle}$ is $\max(q,l)$-safe if $j = b = 1$, and $\max(\operatorname{ord}(\lambda\overline{\eta}_{j-1}), l)$-safe if $j \geq b > 1$.

For $j > b$, $\lambda\overline{\eta}_{j-1}$ occurs in $\mathsf{orddec}_l\, s$ therefore $\operatorname{ord}(\lambda\overline{\eta}_{j-1}) > l$, similarly for $j = b$ we have $\operatorname{ord}(\lambda\overline{\eta}_{j-1}) \leq l$. Hence $(\mathit{collapse}\, s_{\leqslant\lambda\overline{\eta}_j})^{\langle n-l+1\rangle}$ is

(i) $\max(q,l)$-safe for $j = b = 1$,

(ii) l-safe for $j = b > 1$, and therefore $\max(q,l)$-safe by Lemma 4.3,

(iii) $\lambda\overline{\eta}_{j-1}$-safe for $j > b$.

This shows that $(\mathit{collapse}\, s_{\leqslant\lambda\overline{\eta}_j})^{\langle n-l+1\rangle}$ is $\max(l,q)$-safe, and therefore by (3) so is $\mathit{collapse}\, s^{\langle n-l+1\rangle}_{\leqslant\lambda\overline{\eta}_j}$. □

Lemma 4.7. *Let s be a higher-order stack of level ≥ 2 and $l \geq 0$. If*

1. $\mathit{pop}_{\operatorname{ord} s}\, s$ is safe,

2. and $\mathit{top}_{\operatorname{ord} s}\, s$ is l-safe,

then s is l-safe.

Proof. Let $s = [s_1 \ldots s_r\ s_{r+1}]$ for some $l \geq 0$. We proceed by induction on $\mathit{top}_{\operatorname{ord} s}\, s = s_{r+1}$. The base case $s_{r+1} = \bot_{\operatorname{ord} s-1}$ is trivial. Suppose that s_{r+1} is not the empty stack.

(i) Clearly $\mathsf{orddec}_l\, s = \mathsf{orddec}_l\, s_{r+1}$, hence since s_{r+1} is l-safe the lambda nodes in $\mathsf{orddec}_l\, s$ have all a link of height 1.

(ii) Let $\lambda\overline{\eta}$ be a lambda node in $\mathsf{orddec}_l(s) = \mathsf{orddec}_l(s_{r+1})$ such that $n - \operatorname{ord}\lambda\overline{\eta} + 1 \leq \operatorname{ord} s$. Since its link is of height 1 we have $\mathit{collapse}\ s_{\leqslant\lambda\overline{\eta}} = \mathit{pop}_{n-\operatorname{ord}\lambda\overline{\eta}+1}\ s_{\leqslant\lambda\overline{\eta}}$.

If $n - \operatorname{ord}\lambda\overline{\eta} + 1 = \operatorname{ord} s$ then $\mathit{pop}_{n-\operatorname{ord}\lambda\overline{\eta}+1}\ s_{\leqslant\lambda\overline{\eta}} = \mathit{pop}_{\operatorname{ord} s}\ s_{\leqslant\lambda\overline{\eta}} = \mathit{pop}_{\operatorname{ord} s}\ s$ which is l-safe by the first assumption and Lemma 4.3.

Otherwise $n - \operatorname{ord}\lambda\overline{\eta} + 1 < \operatorname{ord} s$ and we have:

$$\begin{aligned}
&\mathit{collapse}\, s_{\leqslant\lambda\overline{\eta}} \\
&\quad = \mathit{pop}_{n-\operatorname{ord}\lambda\overline{\eta}+1}\ s_{\leqslant\lambda\overline{\eta}} && \text{since } \lambda\overline{\eta}\text{'s link has height 1} \\
&\quad = [s_1 \dots s_l\ (\mathit{pop}_{n-\operatorname{ord}\lambda\overline{\eta}+1}\ s_{r+1\leqslant\lambda\overline{\eta}})] && n - \operatorname{ord}\lambda\overline{\eta} + 1 < \operatorname{ord} s \\
&\quad = [s_1 \dots s_p\ (\mathit{collapse}\, s_{r+1\leqslant\lambda\overline{\eta}})] && \text{since } \lambda\overline{\eta}\text{'s link has height 1.}
\end{aligned}$$

By the second assumption, s_{r+1} is l-safe therefore $\mathit{collapse}\ s_{r+1\leqslant\lambda\overline{\eta}}$ is l-safe if $\lambda\overline{\eta}$ is the last node in the l-order decomposition, and k-safe where k is the order of the following node in $\mathsf{orddec}_l(s_{r+1})$ otherwise.

Since $|\mathit{collapse}\ s_{r+1\leqslant\lambda\overline{\eta}}| < |s_{r+1}|$ we can use the induction hypothesis to show that the same hold for $[s_1 \dots s_p\ (\mathit{collapse}\, s_{r+1\leqslant\lambda\overline{\eta}})]$. Therefore it is l-safe and so is $\mathit{collapse}\, s_{\leqslant\lambda\overline{\eta}}$ by the previous equality. □

Lemma 4.8. *Let $n > l \geq 1$ and s be a safe higher-order stack such that $2 \leq n - l + 1 \leq \operatorname{ord} s \leq n$. Then $\mathit{push}_{n-l+1}\ s$ is l-safe.*

Proof. Let $s = [s_1 \dots s_{c+1}]$ be a safe higher-order stack such that $2 \leq n-l+1 \leq \operatorname{ord} s \leq n$. Then by Lemma 4.5(i), s_{c+1} is safe.

We show that $\mathit{push}_{n-l+1}\ s$ is l-safe by finite induction on the order of s.

- Base case: $\operatorname{ord} s = n - l + 1$. We have $\mathit{push}_{n-l+1}\ s = [s_1 \dots s_{c+1} s_{c+1}^{\langle n-l+1\rangle}]$. Since s_{c+1} is safe, by Lemma 4.6 $s_{c+1}^{\langle n-l+1\rangle}$ is l-safe, and by Lemma 4.7, $[s_1 \dots s_{c+1} s_{c+1}^{\langle n-l+1\rangle}]$ is l-safe.
- Suppose $\operatorname{ord} s > n-l+1$. Then $\mathit{push}_{n-l+1}\ s = [s_1 \dots s_{c+1} \mathit{push}_{n-l+1}\, s_{c+1}]$. Since s_{c+1} is safe, by the induction hypothesis $\mathit{push}_{n-l+1}\ s_{c+1}$ is l-safe, and by Lemma 4.7 so is $[s_1 \dots s_{c+1} \mathit{push}_{n-l+1}\, s_{c+1}]$. □

4.3 Simulation and proof of correctness

Proposition 4.1. *Let G be an incrementally-bound recursion scheme. The $\rightarrow$-reachable configurations of $CPDA(G)$ are safe.*

Proof. If $n = \operatorname{ord} c = 1$ then the result holds trivially since $CPDA(G)$ does not contain any transition of the form push_j for $j > 1$ and therefore the links in a reachable configuration all have a height component equal to 1.

Take $n \geq 2$. We proceed by induction on the $\rightarrow$-step relation. The initial configuration is clearly safe. Suppose that c is a safe $\rightarrow$-reachable configuration and that $c \rightarrow c'$. We do a case analysis on $\mathit{top}_1\, c$:

- (A): We have $c' = \mathit{push}_1^{E_0(u),1}\, c = \mathit{push}_1\, E_0(u)^{(1,1)}\, c$ where $E_0(u)$ denotes a lambda node. It is safe by Lemma 4.5(iv).

- (S): We have $c' = push_1^a\, c = push_1\, a^{(j,k)}\, c$ for some dummy lambda node a and undetermined link (j,k). It is safe by Lemma 4.5(iv) since $\operatorname{ord} a = 0$.
- (L): We have $c' = push_1^{E_1(u)} = push_1 E_1(u)^{(j,k)}$ where $E_1(u)$ is not a lambda node and $j,k \geq 1$ are undetermined. It is safe by Lemma 4.5(iii).
- (V_1) & (V_0): Suppose u is labelled by a variable x of order l. Since c is safe we have $\mathsf{orddec}(c) = \langle \lambda\overline{\eta}_r^1, \ldots, \lambda\overline{\eta}_1^1 \rangle$, $r \geq 0$. Since the recursion-scheme G is safe, by Lemma 4.2, x's binder is precisely the last node of $\mathsf{orddec}_l(c)$. Let b be its index in $\mathsf{orddec}(c)$, and $i \geq 1$ be the index of x in $\overline{\eta}_b$.
  - (V_1): $l \geq 1$. We have $c' = push_1\, E_i(top_1)^{(n-l+1,1)}\, t$ where t is given by $(push_{n-l+1}; pop_1^p; collapse)(c) = collapse((push_{n-l+1}\, c)_{\leqslant \lambda\overline{\eta}_b})$. By Lemma 4.8 $push_{n-l+1}\, c$ is l-safe therefore, since $\lambda\overline{\eta}_b$ is the last node in $\mathsf{orddec}_l(c)$, by definition of l-safety we have that t is l-safe. Finally the lambda node $E_i(top_1)$ pushed by the last operation has precisely order $l = \operatorname{ord}(x)$ therefore
$$\mathsf{orddec}_0(c') = \langle \lambda\overline{\eta}_r^1, \ldots, \lambda\overline{\eta}_b^1, E_i(top_1(t))^1 \rangle \ .$$
    Thus all the lambda nodes in $\mathsf{orddec}_0(c')$ have a link of height 1.
    We now need to show that safety is preserved when collapsing at nodes of $\mathsf{orddec}_0(c')$. Let $b \leq j \leq r$, we have $c'_{\leqslant \lambda\overline{\eta}_j} = t_{\leqslant \lambda\overline{\eta}_j}$. For $j > b$, the l-safety of t implies that $collaspse\, c'_{\leqslant \lambda\overline{\eta}_j}$ is $\operatorname{ord} \lambda\overline{\eta}_{j-1}$-safe as required. For $j = b$ it gives that $collaspse\, c'_{\leqslant \lambda\overline{\eta}_b}$ is l-safe as required since $l = \operatorname{ord} E_i(top_1(t))$.
    Now it remains to show that $collapse(c'_{\leqslant E_i(top_1(t))}) = collapse\, c'$ is safe. Since we have $i \geq 1$ the node $top_1(c') = E_i(top_1(t))$ is not a prime lambda node, thus by Lemma 4.4 we can simulate the *collapse* by a *pop* of order $n - \operatorname{ord} E_i(top_1(t)) + 1$:
$$\begin{aligned} collapse\, c' &= pop_{n-\operatorname{ord} E_i(top_1(t))+1}\, c' \\ &= pop_{n-l+1}\, c' \\ &= (push_{n-l+1}; pop_1^p; collapse; push_1^{E_i(top_1),n-l+1}; pop_{n-l+1})\, c \end{aligned}$$
    The operation pop_1 and $push_1$ only affects the top 1-stack. Furthermore, since x's binder has order $> l$, its link has order $< n - l + 1$ therefore the collapse operation following the pop_1^p only affects the top $(n-l)$-stack. Consequently, the operation pop_{n-l+1} effectively restores the configuration to its value prior to performing the $push_{n-l+1}$ operation:
$$collapse\, c' = c \ .$$
    Hence c' is safe.
  - (V_0): $l = 0$ (which implies that $b = 1$). The configuration c' is given by $push_1 E_i(top_1)\, collapse(c_{\leqslant \lambda\overline{\eta}_b})$. Since c is safe by definition so is $collapse(c_{\leqslant \lambda\overline{\eta}_b})$. Since the pushed lambda node $E_i(top_1)$ has order $l = 0$, by Lemma 4.5(iv) c' is safe. □

Definition 4.3 (Simulating PDA)**.** Let G be an incrementally-bound recursion scheme. We define $PDA(G)$ as the higher-order PDA obtained from $CPDA(G)$ by replacing every *collapse* operation by pop_1 if $top_1\, s$ is prime, and by $pop_{n-\operatorname{ord} top_1(s)+1}$ otherwise.

Theorem 4.1 (Correctness of the simulation). *$PDA(G)$ and $CPDA(G)$ are equivalent.*

Proof. In $CPDA(G)$, the collapse operation occurs only in the steps (V_1) and (V_0). For (V_1) it is of the form:

$$collapse(pop_1^p(push_{n-l+1}\ c))$$

for some $\rightarrow$-reachable configuration c, where $top_1\, c$ is a variable x of order l and span p. By the previous proposition, c is safe and by Lemma 4.8 $push_{n-l+1}\, c$ is l-safe. Since x has span p, after the operation pop_1^p the top stack symbol is precisely x's binder, which by Lemma 4.2, belongs to $\mathsf{orddec}_l(c)$, therefore by Lemma 4.4 the collapse can be soundly simulated by a *pop* of order 1 if x's binder is a prime node, and a pop of order $n - \mathrm{ord}(top_1(pop_1^p(push_{n-l+1}\, c))) + 1$ otherwise.

The case (V_0) is proved similarly. $\square$

Remark 4.1. The same result holds for the original CPDA from Hague *et al.*: Clearly the two CPDAs have the same set of $\rightarrow$-reachable configurations so Proposition 4.1 clearly still holds. The simulating PDA from 4.3 is obtained from Hague *et al.*'s CPDA by replacing every *collapse* operation by $pop_{n-\mathrm{ord}\, top_1(s)+1}$; and the equality in Lemma 4.4 should be replaced by:

$$collapse\ s_{\leqslant\lambda\overline{\eta}} = pop_{n-\mathrm{ord}\,\lambda\overline{\eta}+1}\ s_{\leqslant\lambda\overline{\eta}}\ .$$

The correctness of the simulation follows similarly.

5 From PDA to incrementally-bound RS

Let $\mathcal{A}$ be an order-n PDA. Following the transformation from Hague *et al.* from order-n CPDA to order-n recursion scheme [4] we can produce an equivalent incrementally-bound homogeneously-typed recursion scheme. Since we do not need to implement the collapse operation we can remove the parameters $\overline{\Phi}$ altogether from the rules $\mathcal{F}_p^{a,e}$ of the recursion scheme. The type of the non-terminal $\mathcal{F}_p^{a,e}$ for each stack symbol a, $1 \le e \le n$, and state $1 \le p \le m$ becomes:

$$\mathcal{F}_p^{a,e} : (n-1)^m \rightarrow \ldots \rightarrow 0^m \rightarrow 0$$

and the production rules are of the general form:

$$\mathcal{F}_p^{a,e}\overline{\Psi_{n-1}} \ldots \overline{\Psi_0} \overset{(q,\theta)}{\longrightarrow} \Xi_{(q,\theta)}$$

where the right-hand side is given by:

Cases of θ	**Corresponding $\Xi_{(q,\theta)}$**
$push_1^{b,k}$	$\mathcal{F}_q^{b,k}\langle\mathcal{F}_i^{a,e}\overline{\Psi_{n-1}}\vert i\rangle\overline{\Psi_{n-2}}\ldots\overline{\Psi_0}$
$push_j$	$\mathcal{F}_q^{a,e}\overline{\Psi_{n-1}}\ldots\overline{\Psi_{n-(j-1)}}\langle\mathcal{F}_i^{a,e}\overline{\Psi_{n-1}}\ldots\overline{\Psi_{n-j}}\vert i\rangle\overline{\Psi_{n-(j+1)}}\ldots\overline{\Psi_0}$
pop_k	$\Psi_{n-k,q}\overline{\Psi_{n-k-1}}\ldots\overline{\Psi_0}$

It is easy to verify that this recursion scheme is incrementally-bound and homogeneously-typed.

References

[1] W. Blum and C.-H. L. Ong. The safe lambda calculus. In S. R. D. Rocca, editor, *TLCA*, volume 4583 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2007.

[2] C. Broadbent. A proof of Blum's conjecture, June 2009.

[3] M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible push-down automata and recursive schemes. extended version (59p), November 2006.

[4] M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible push-down automata and recursive schemes. *LICS*, pages 452–461, 2008.

[5] T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FOSSACS'02*, pages 205–222. Springer, 2002. LNCS Vol. 2303.

[6] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *Proceedings of IEEE Symposium on Logic in Computer Science.*, pages 81–90. Computer Society Press, 2006. Extended abstract.