PARALLEL TIME $O(\log N)$ ACCEPTANCE OF DETERMINISTIC CFLs

Philip N. Klein
John Reif

TR-05-84
PARALLEL TIME $O(\log N)$ ACCEPTANCE OF DETERMINISTIC CFLs

Philip N. Klein
John Reif

TR-05-84

March, 1984
# PARALLEL TIME $O(\log N)$ ACCEPTANCE OF DETERMINISTIC CFLs

**AUTHOR(s)**

John H. Reif  
Philip N. Klein

**PERFORMING ORGANIZATION NAME AND ADDRESS**

Harvard University  
Cambridge, MA 02138

**REPORT DATE**

March 1984

**NUMBER OF PAGES**

33

**DISTRIBUTION STATEMENT (of this Report)**

unlimited

**DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)**

unlimited

**SUPPLEMENTARY NOTES**


**KEY WORDS (Continue on reverse side if necessary and identify by block number)**

parallel algorithms, deterministic CFLs, language recognition

**ABSTRACT (Continue on reverse side if necessary and identify by block number)**

see reverse side.
Abstract

We give a parallel RAM algorithm for simulating a deterministic pushdown automaton. On an input of length n, a DPDA will use time $O(n)$; our simulation requires time $O(\log n)$ and only polynomially many processors. The algorithm can be easily adapted to accept the $1.R(k)$ languages as well. An easy generalization of the algorithm will simulate a deterministic auxiliary pushdown automaton that uses space $s(n) \geq \log n$ and time $2^{O(s(n))}$; the simulation then requires time $O(s(n))$ and $2^{O(s(n))}$ processors. This simulation is nearly optimal for parallel RAMs, since we show that the language accepted in time $T(n)$ by a parallel RAM is accepted by a deterministic auxiliary pushdown automaton with space $T(n)$ and time $2^{O(T(n)^2)}$. 
PARALLEL TIME $O(\log N)$ ACCEPTANCE OF DETERMINISTIC CFLs*

Philip N. Klein
John H. Reif

Aiken Computation Laboratory
Division of Applied Sciences
Harvard University
Cambridge, Massachusetts 02138

* This work was supported in part by National Science Foundation grant MCS-82-00269 and the Office of Naval Research contract N00014-80-C-0647.

A Preliminary version of this paper appeared in the IEEE Symposium on the Foundations of Computer Science, Chicago, IL, October 1982.
Abstract

We give a parallel RAM algorithm for simulating a deterministic pushdown automaton. On an input of length \( n \), a DPDA will use time \( O(n) \); our simulation requires time \( O(\log n) \) and only polynomially many processors. The algorithm can be easily adapted to accept the LR(1) languages as well. An easy generalization of the algorithm will simulate a deterministic auxiliary pushdown automaton that uses space \( s(n) \geq \log n \) and time \( 2^O(s(n)) \). The simulation then requires time \( O(s(n)) \) and \( 2^O(s(n)) \) processors. This simulation is nearly optimal for parallel RAMs, since we show that the language accepted in time \( T(n) \) by a parallel RAM is accepted by a deterministic auxiliary pushdown automaton with space \( T(n) \) and time \( 2^O(T(n)) \).

Introduction

This paper assumes the parallel random access machine model \( P-RAM \) as defined in [Fortune and Wyllie, 78] which consists of a collection of synchronous deterministic unit-cost RAMs with shared memory locations indexed by the natural numbers. Simultaneous reads are allowed: on each step, any given memory location may be read simultaneously by any number of processors. However, no two distinct processors can attempt to simultaneously write into the same memory location on the same step.

A fundamental question, which we address in this paper, is the time complexity of the P-RAM: that is, what class of languages can be accepted by the P-RAM within a given time bound?

Previously [Fortune and Wyllie, 78] showed that any language accepted by a deterministic Turing machine with space bound \( s(n) \geq \log n \) is accepted in time \( O(s(n)) \) by a P-RAM. Also [Ruzzo, 80] showed that any language accepted by an auxiliary pushdown machine with space \( s(n) \geq \log n \) and time \( 2^O(s(n)) \) is accepted in time \( O(s(n))^2 \) by various parallel machine models including the P-RAM.

The form of the paper is as follows: in Section 1, we will describe some assumptions we make of the simulated deterministic PDA in order to simplify the presentation of the algorithm. In Section 2, we introduce the notion of a surface configuration, and rephrase some of the assumptions of the first section as propositions about surface configurations. In Section 3, we introduce some notation useful in
the proof of the algorithm, and state some lemmas concerning this notation. In Section 4, we describe the algorithm, and, in Section 5, give an inductive proof of its correctness. In Section 6, we point out that the algorithm may be used to simulate a space-bounded auxiliary pushdown automaton. In Section 7, we give a complementary result concerning simulation of PRAMs by deterministic auxiliary PDAs. In Section 7, we mention some related work, and in Section 8, we mention alternative models of parallel computation.

Section 1: Basic Assumptions

We denote the cardinality of a set A by |A|, and the length of a string s by |s|. We denote the empty string by ε. We denote the concatenation of strings s₁ and s₂ by s₁s₂.

We will simulate a deterministic PDA M with input alphabet Σ, state set Q, and stack alphabet S. We will for technical reasons assume that M's transition function is a mapping from (q, w, σ, γ) to (q', s), where q is the current state and q' is the next state, w is the input symbol currently scanned, σ is the top stack symbol (if it exists), s is a string of stack symbols of length at most 2 that will replace σ in the stack, and γ is true iff the stack consists only of σ. Thus M's next move may depend on whether the stack has only one symbol. Note that standard techniques allow this assumption and those described below, and furthermore that any DPDA in the form defined in [Hopcroft and Ullman, pp. 253-255] may be transformed into one in the form used here without changing the language accepted.

For a fixed input ω₁...ωₙ ∈ Σ⁺, we can write each configuration of M as (q,i,s) ∈ Q × {0,...,n} × S*, where q is the state of the finite control, i is the position of the input head (i.e. the number of symbols that have been read), and s is the stack (with the rightmost symbol being the top of the stack). Let =⇒ be the next move relation, and =⇒⁺ and =⇒⁻ be its transitive and reflexive-transitive closures, respectively. For any non-negative integer a, let =⇒ᵃ be the a-fold composition of =⇒; i.e. reachability in exactly a steps.

We assume of the PDA that each push and pop changes the stack height by only one, that each push is accompanied by an advance of the input head, and that each advance of the input head is accompanied by a push. Any DCFL has such a PDA, which is essentially a shift-reduce parser for the DCFL (see Hopcroft and Ullman, pp. 253-255). It follows that the height of the stack never grows by more than n, the length of the input.
We also assume that $M$ never changes more than the top symbol of the stack; i.e., for $s,s'\in \Sigma^*, \sigma \in \Sigma$.

if $(q,i,s) \vdash (q',i',s')$, then $s' = s \sigma'$ for some $\sigma'$.

We assume that there are no cycles $(q,i,s) \vdash^+ (q,i,s)$ of length more than one. Every such cycle can be replaced by a cycle of length one: $(q,i,s) \vdash (q,i,s)$. We let $\text{LOOP}(q,i,s)$ be the predicate that is true just in case $(q,i,s) \vdash (q,i,s)$.

We assume that $M$'s stack initially contains a special symbol, and this symbol is never popped ($M$ accepts by final state rather than by empty stack). We may therefore require of $M$ that when started in a configuration with a stack of size one, $M$ will never pop that one symbol, but will loop instead. We have therefore

if $(q,i,s) \vdash (q',i',s')$ then $|s'| \geq 1$.

Moreover, we assume that $M$ never loops unless the stack has at most one symbol (i.e. $M$ pops all but one of the stack symbols before looping):

if $\text{LOOP}(q,i,s)$ then $|s| = 1$.

We assume that the accepting configuration $(x_{\text{acc}}.\epsilon)$ is a looping configuration.

Finally, we assume that if not $\text{LOOP}(q,i,s)$ then the next move from $(q,i,s)$ depends only on the top symbol of $s$:

for any $s_1, s_2 \in \Sigma^*, \sigma \in \Sigma$, if not $\text{LOOP}(q,i,s_1,\sigma)$ and $(q,i,s_1,\sigma) \vdash (q',i',s_2\sigma')$, then $(q,i,s_2) \vdash (q',i',s_2\sigma')$.

Section 2: Surface Configurations
Instead of manipulating complete configurations of \( M \), the algorithm manipulates surface configurations. A surface configuration does not specify the entire stack, only the top symbol. We also include in our surface configurations a stack height parameter, giving relative stack height information. In particular, for a fixed input \( \omega_1, \ldots, \omega_n \in \Sigma^* \), we write each surface configuration as \((q, i, \sigma, h) \in Q \times \{0, \ldots, n\} \times S \times \{0, \ldots, n\}\). Here \( q \) is the state of the finite control, \( i \) is the position of the input head, \( \sigma \) is the top symbol of the stack, and \( h \) is a stack height parameter. Let \( S \) be the set of all such surface configurations for input \( \omega_1, \ldots, \omega_n \), and note that \(|S| = O(n^2)\). We extend the definition of \( \rightarrow \) to apply to surface configurations as follows: if \((q, i, \sigma, h) \rightarrow (q', i', \sigma', h')\), then for any \( h \) we may write \(((q, i, \sigma, h), s) \rightarrow ((q', i', \sigma', h'), s')\), where \( h' = h + |s'| - |s| \). Because this is true for any \( h \), we have the following proposition:

\[
\text{h-Independence Proposition: if } (q, i, \sigma, h), s) \rightarrow (q', i', \sigma', h'), s') \text{, then for any } \delta, ((q, i, \sigma, h + \delta), s) \rightarrow ((q', i', \sigma', h' + \delta), s').
\]

We define \( h : \{0, \ldots, n\} \rightarrow h(q, i, \sigma, h) = h \). By a simple induction, we obtain:

\[
\text{Stack Height Proposition: For any } x, x' \in X \text{ and } s, s' \in S^*, \text{ if } (x, s) \rightarrow^* (x', s') \text{ then } |s'| - |s| = h(x') - h(x).
\]

Define \( p : X \rightarrow \{0, \ldots, n\} \) by \( p(q, i, \sigma, h) = i \). That is, \( p(x) \) gives the position of the input head in surface configuration \( x \). Because the input head is one-way, we have:

\[
\text{Head Position Proposition I: If } (x, s) \rightarrow^* (x', s') \text{ then } p(x') \geq p(x).
\]

Because we assumed each push is accompanied by an advance of the input head, we have:

\[
\text{Head Position Proposition II: if } (x, s) \rightarrow^* (x', s') \text{ and } |s'| = |s|, \text{ then } p(x') > p(x).
\]

We extend the definition of the \( \text{LOOP} \) predicate so that for any \( h \), \( \text{LOOP}(q, i, \sigma, h), s) \) iff \( \text{LOOP}(q, i, \sigma, \alpha) \).
Suppose \( M \) is in the configuration \((x,s)\), where \( x \) is a surface configuration. Because \( M \) never changes more than the top symbol of the stack, if \( M \) does not pop, then \( s \) remains intact. Because \( M \)'s next move depends only on the top symbol of the stack, \( x \) determines the next state. If \( M \) does not pop, then \( x \) also determines the new top stack symbol, and hence the next surface configuration. We have therefore:

**Stack Proposition:** if \((x,s) \vdash (x',s')\) and \(|s'| \geq |s|\) then

1. \( s' = sos' \) for some \( s' \in S^* \), and
2. if not LOOP\((x,s)\) then for any \( s_1 \in S^* \), \((x,s_1) \vdash (x',s_1os')\).

Recalling that \( M \) only loops if its stack contains exactly one symbol:

**LOOP Proposition:** if \( \text{LOOP}(x,s) \) then \( s = \epsilon \).

By induction, we obtain:

**Preservation of Time Proposition:** if \((x,\epsilon) \vdash^a (x',s')\), but there is no \( b \geq a \) such that \((x,\epsilon) \vdash^b (x',s')\), then for any \( s \), \((x,s) \vdash^a (x',sos')\).

**Combining Computations Propositions:**

1. if \((x,\epsilon) \vdash^* (x',s') \) and \((x',\epsilon) \vdash^* (x'',s'')\), then \((x,\epsilon) \vdash^* (x'',s's'')\).
2. if \((x,\epsilon) \vdash^* (x',s) \) and \((x',s) \vdash^* (x'',s'')\), then \((x,s) \vdash^* (x'',s'')\).

**Section 3: Notation**

We now introduce some notation that will be useful in proving the algorithm. We first need to prove a claim that will ensure the well-definedness of the notation:

**Claim:** if \((u,\epsilon) \vdash^* (x,s) \) and \((u,\epsilon) \vdash^* (x,s')\) then \( s = s' \) (i.e., \( s \) is uniquely determined by \( u \) and \( x \)).

**Proof:** Either \((x,s) \vdash^* (x,s')\) or \((x,s') \vdash^* (x,s)\). Suppose without loss of generality that \((x,s) \vdash^* (x,s')\)
Suppose $\mathbf{f}_{xy}, \mathbf{f}_{yz} \in \mathcal{U}$ and $\mathbf{f}_{yz}, \mathbf{f}_{xz} \notin \mathcal{U}$.

Let $y'' =
\begin{cases} y & \text{if } y \neq \bot \\
y' & \text{else} \end{cases}$

Then $\mathbf{f}_{xy''} \notin \mathcal{U}$.

Proof: Since $\mathbf{f}_{xy} \in \mathcal{U}$ and $\mathbf{f}_{yz} \in \mathcal{U}$ exist, so does $\mathbf{f}_{xz} \in \mathcal{U}$.

Case 1: $y \neq \bot$. Then for every $w \in \mathcal{U}$ such that $w \neq u$ and $w \neq y$, $h(w) > h(u)$. Since $y \in \mathcal{U}$, clearly $y' \in \mathcal{U}$ exists. We conclude $\mathbf{f}_{xy} \in \mathcal{U}$.

Case 2: $y = \bot$. Then for every $w \in \mathcal{U}$ such that $w \neq u$ and $w \neq y$, $h(w) > h(u)$. Suppose $y' \neq \bot$ (the case in which $y' = \bot$ is analogous). Then for every $w \in \mathcal{U}$ such that $w \neq u$ and $w \neq y'$, $h(w) > h(u)$. In summary, for every $w \in \mathcal{U}$ such that $w \neq u$ and $w \neq y'$, $h(w) > h(u)$. We also have that $y' \in \mathcal{U}$ exists, so $\mathbf{f}_{xy'} \notin \mathcal{U}$.

Lemma 18: If $u \in \mathcal{U}$, $h(u) > h(v)$, and $\mathbf{f}_{xy} \in \mathcal{U}$ exist, then $\mathbf{f}_{x., y} \in \mathcal{U}$.

Proof: Suppose $w \in \mathcal{U}$. Then $u \in \mathcal{U}$, so by the stack height proposition, $h(w) > h(u) > h(v)$. By lemma 1, $\mathbf{f}_{xy} \in \mathcal{U}$ exists, so we may conclude that $\mathbf{f}_{x., y} \in \mathcal{U}$. 
case 2) \( h(y') \leq h(y) \). In this case, \( y = y' \). It follows from the hypothesis that if \( (y',-) \) exists and \( w \in [x,y]_u \), then \( h(w) \in h(y') \). It also follows from the hypothesis that \( [x,y]_u \) and \( [y',-]_u \) exist, so \( [x,y]_u \) exists. Suppose \( w \in [x,y]_u \); then either \( w \in [x,y]_u \) or \( w \in [y',-]_u \). If \( w \in [x,y]_u \), then \( h(w) \geq h(y) \geq h(y') \). If \( w \in [y',-]_u \), then again \( h(w) \geq h(y') \). We may therefore conclude \( \prec x,y,x' \succ_u \).

**Lemma 7** If \( [u]_a \), \( \prec x,y,a[-]_u \), and \( h(u) \in h(u) \), then \( \prec x,y,a[-]_u \), and \( h(u) \in h(u) \).

**Proof.** By the definition of \( \prec x,y,a[-]_u \), \( h(a[-]) \geq h(y) \), so \( h(a[-]) \in h(u) \), so \( a[-]_u = a[-]_u \), by lemma 3. We therefore have \( \prec x,y,a[-]_u \). But then by lemma 4, \( \prec x,y,a[-]_u \).

**Lemma 15** If \( h(w) \in h(x) \) for every \( w \in [x,y]_u \), then \( [y]_x \).

**Proof.** Since \( h(y) \in h(x) \), clearly \( y \neq x \). Since \( [x,y]_u \) exists, \( (u,\varepsilon) \vdash (x,s) \vdash (y,s') \) for some \( s,s' \in S^* \), and some minimal integer \( a \geq 1 \). We prove the lemma by induction on \( a \). Consider the case \( a = 1 \), so \( (x,s) \vdash (y,s') \). Suppose \( (x,s) \vdash (y,s') \). Since \( [x,y]_x \) exists, by lemma 1 \( [x,y]_x = [x,y]_u \subseteq [x,y]_u \), so \( h(s') \in h(x) \), so \( y \neq x \). Then there is no \( b \in a = 1 \) such that \( (x,s) \vdash (y,s') \), \( s' \neq x \). Then there is no \( b \in a \) that \( (x,s) \vdash (y,s') \). By the preservation of time proposition, \( (x,s) \vdash (y,s,s') \), so in fact \( (y,s) = (y,s,s') \). We conclude that \( [y]_x \).

Now assume the lemma holds for \( a \geq 1 \), and suppose \( (x,s) \vdash (x',s_1) \vdash (y,s') \). By the inductive hypothesis, \( [x']_x \), so \( [x,x']_x \) exists, so \( h(x') \in h(x) \) as above. Since \( [x']_x \), we have \( (x,s) \vdash (x',s_1) \vdash (y,s') \) for some \( s_1 \in S^* \). Since \( h(x') \in h(x) \), by the stack height proposition \( s_2 \neq \varepsilon \), so by the LOOP proposition, not \( 1 \)-LOOP \((x',s_2) \). Suppose \( (x',s_2) \vdash (y,s') \); we have that \( (x,s) \vdash (x',s_2) \vdash (y,s') \) but there is no \( b \in a + 1 \) such that \( (x,s) \vdash (y,s') \) (else \( (x',s_2) = (y,s') \), so \( 1 \)-LOOP \((x',s_2) \)). Then by the preservation of time proposition, \( (x,s) \vdash (x',s_2) \vdash (y,s') \), so \( y = y' \), and we conclude that \( [y]_x \).

**Lemma 16** If \( [u]_a \), \( h(u) = h(v) \), and \( [x,y,-]_u \), then \( [x,y,-]_u \).

**Proof.** Suppose \( y = \bot \) (the case in which \( y = \bot \) is analogous). Then \( [x,y,-]_u \) exists, and if \( w \in [x,y]_u \) then \( h(w) \in h(u) \). But by lemma 1, \( [x,y]_u \) exists and \( [x,y]_u = [x,y]_u \), so if \( w \in [x,y]_u \) then
Lemma 4 If \([u]\) and \(<y,z>\) then \(<x,y,z>\).

**Proof** By Lemma 1, \([x,y]\) exists and \([x,z]=\{x,y\}\). Then since \(h(w)\geq h(y)\) for each \(w\in[x,y]\), it follows that \(h(w)\geq h(y)\) for each \(w\in[x,z]\). Similarly, by Lemma 2, if \([y,z]\) exists, then \((y,z)\) exists and \((y,z)=[y,z]\). Hence if \(w\in[y,z]\), then \(w\in[y,z]\), so \(h(w)\geq h(y)\). We conclude that \(<x,y,z>\).

Lemma 5 If \((u,e)\not\rightarrow^* (x,s_1)\not\rightarrow^* (y,s_2)\) and for each \(w\in[x,y]\), \(h(w)\geq h(y)\), then \(s_1=s_2s_s\) for some \(s'\in S^*\).

**Proof** The lemma follows from the following claim:

**Claim** if \((u,e)\not\rightarrow^* (x,s_1)\not\rightarrow^* (y,s_2)\) for some \(a\), then for any \(h\) such that \(h(w)\geq h]\) for every \(w\in[x,y]\), there is a string of length \(h-h(u)\) that is an initial substring of \(s_1\) and \(s_2\).

**Proof of Claim** by induction on \(a\). The case \(a=0\) is trivial, so assume the claim is true for \(a\), and suppose \((u,e)\not\rightarrow^* (x,s_1)\not\rightarrow^* (x',s_1')\not\rightarrow^* (y,s_2)\). By the inductive hypothesis, there is a string \(s\) of length \(h-h(u)\) that is an initial substring of \(s_1'\) and \(s_2\). Consider the move \((x,s_1)\not\rightarrow^* (x',s_1)\).

If the move is a pop, then \(s_1^*\) is an initial substring of \(s_1\), hence so is \(s\). Therefore, assume the move is not a pop, so by the stack proposition, \(s_1^* = s_1s_2\) for some \(s^\prime\in S^*\).

Now by the stack height proposition, \([s_1]=h(x)-h(u)\). Since clearly \(x\in[x,y]\) we have \(h(x)\geq h\). Therefore, \(h-h(u)\leq h(x)-h(u)\), so \(s\) is no longer than \(s_1\). Since \(s\) is an initial substring of \(s_1^* = s_1s_2\), \(s\) is an initial substring of \(s_1\).

Lemma 6 Suppose \(<y,z>\) and \(<z,y'>\).

Let \(y'' = \{y\text{ if } h(y')\leq h(y)\}

\(\{y\text{ else}\}\)

Then \(<x,y'',z'>\).

**Proof**

**Case 1** \(h(y')>h(y)\). In this case, \(y'' = y\). It follows from the hypothesis that \([x,y]\) exists and \(h(w)\geq h(y)\) for every \(w\in[x,y]\). Suppose \([y,z]\) exists, and \(w\in[y,z]\). Either \(w\in[y,z]\) or \(w\in[z,z]\). If \(w\in[y,z]\), then \(h(w)>h(y)\) follows from the hypothesis. If \(w\in[z,z]\), then \(h(w)\geq h(y')>h(y)\). Since in either case \(h(w)>h(y)\), we may conclude \(<x,y,z>\).
Appendix

Lemma 0 Suppose \((x, \varepsilon) \vdash^a (x', s')\), but there is no \(b < a\) such that \((x, \varepsilon) \vdash^b (x', s')\). Then for any \(s \in S^*\) and each \(0 \leq b < a\).

(i) if \((x, \varepsilon) \vdash^b (x'', s'')\) then \((x, s) \vdash^b (x'', s'')\), and

(ii) if \((x, s) \vdash^b (x'', s_2)\) then \((x, \varepsilon) \vdash^b (x'', s'')\) with \(s_2 = s s''\).

Proof By the hypothesis, if \((x, \varepsilon) \vdash^b (x'', s'')\) then not \(\text{LOOP}(x'', s'')\). Then (i) follows by Preservation of Time Proposition. Now suppose \((x, s) \vdash^b (x'', s_2)\). Choose \((x_2, s'')\) so that \((x, \varepsilon) \vdash^b (x_2, s'')\). Again not \(\text{LOOP}(x_2, s'')\), so \((x, s) \vdash^b (x_2, s_2 s'')\), so \((x_2, s_2 s'') = (x'', s_2)\).

Lemma 1 If \([u]_i\), and \([x, y]_u\) exists, then \([x, y]_i\) exists, and \([x, y]_i = [x, y]_u\).

Proof \((x, \varepsilon) \vdash^* (u, s) \vdash^* (x, s_1) \vdash^* (y, s_2)\) for some \(s_1, s_2 \in S^*\). Hence \((x, \varepsilon) \vdash^* (u, s) \vdash^* (x, s_1) \vdash^* (y, s_2)\).

Moreover, if \(w \in [x, y]_i\) then \((x, s_1) \vdash^* (w, s_3) \vdash^* (y, s_2)\) for some \(s_3 \in S^*\). So \((x, s_1) \vdash^* (w, s_3) \vdash^* (y, s_2)\) for \(w \in [x, y]_i\). Now, let \(a, b, c\) minimal such that \((u, \varepsilon) \vdash^a (y, s_2)\), and suppose \(w \in [x, y]_i\). Then \((u, s) \vdash^b (w, s_3) \vdash^c (l, s_3 s_2)\) for some \(s_3\) and some \(b, c\) such that \(b + c \leq a\). If \((w, s_3) = (y, s_3 s_2)\), then \(w = y \in [x, y]_u\). Otherwise, by lemma 0, \((u, \varepsilon) \vdash^b (w, s')\) with \(s' = s s_2\). Because \((u, \varepsilon) \vdash^* (x, s_1)\), either \((x, s_1) \vdash^* (w, s')\) or \((w, s') \vdash^* (x, s_1)\). In the former case, \(w \in [x, y]_u\). In the latter case, \((w, s_3) \vdash^* (x, s_2 s_3)\), contradicting choice of \(w\).

Lemma 2 If \([u]_i\), and \([x, y]_u\) exists, then \([x, y]_i\) exists, and \([x, y]_i = [x, y]_u\).

Proof Immediate from lemma 1.

Lemma 3 If \([u]_i\), and \(h(a[x]_i) \gg h(u)\), then \(a[x]_i = a[x]_u\).

Proof By definition of \(a[x]_u\), \((u, \varepsilon) \vdash^* (x, s_1) \vdash^a (a[x]_u, s_2)\) for some \(s_1, s_2 \in S^*\). Because \(h(a[x]_u) > h(u) \gg 0\), \(s_2 > \varepsilon > s_2 > 0\) by Stack Height Proposition, so \(s_2 \neq \varepsilon\). So not \(\text{LOOP}(a[x]_u, s_2)\) by the LOOP Proposition. Then by Preservation of Time Proposition, \((u, s) \vdash^* (x, s_2 s_1) \vdash^a (a[x]_u, s_2)\) for any \(s \in S^*\). In particular, if \(s\) satisfies \((u, \varepsilon) \vdash^* (u, s)\), then \((u, \varepsilon) \vdash^* (x, s_2 s_1) \vdash^a (a[x]_u, s_2)\). So by definition, \(a[x]_i = a[x]_u\).
Thus only \(\text{O}(n)\) space is required by our simulating deterministic APDA.

Note: [Ruzko, 82] has also proved this result independently by combining some known complexity bounds for various parallel machine models.

Section 8: Further Work

The \(LR(k)\) grammars considered in [Knuth, 65] are frequently used in practice for programming languages. The \(k\)-symbol lookahead required for recognition of an \(LR(k)\) language by a DPDA may be incorporated into stage 0 of our algorithm, using only \(O(k)\) extra time.

As mentioned at the end of Section 4, our algorithm for DCF1 recognition requires time \(O(\log n)\) and \(O(n^3)\) processors. It is easy to show that in fact only \(O(n^3)\) processors are necessary, after a minor modification of the algorithm.

Section 9: Alternative Parallel Machine Models

[Ruzko, 80] gave an alternating machine algorithm for recognition of context-free languages in time \(O(\log^2 n)\) and simultaneously polynomial tree-size. As he points out, this algorithm can easily be simulated in time \(O(\log n)\) and a polynomial number of processors by parallel machine models which allow resolution of both read and write conflicts. It is also easy to show that Ruzzo's algorithm can be simulated by a depth \(O(\log n)\) circuit with a polynomial number of logical elements but unbounded degree. (See [Stockmeyer and Vishkin, 81].)

However, Ruzzo's algorithm requires \(\Omega(\log^2 n)\) time on the usual models of parallel computation that disallow write conflicts. It is an open question whether general context-free recognition can be done in time \(o(\log^2 n)\) on a P-RAM. Also, it is open whether circuits of constant degree and depth \(o(\log^2 n)\) can recognize the class of languages accepted by deterministic TMs with space \(O(\log n)\).
Theorem 2 Let L be accepted by a DPDA. Then L is accepted by a P-RAM with time O(log n) and O(n^4) processors.

proof The theorem follows from Theorem 1 and comments at the end of section 4.

Section 6: Simulation of a Deterministic Auxiliary Pushdown Automaton

Consider now the simulation of an s(n) space-bounded, t(n) time-bounded deterministic auxiliary pushdown automaton M with a stack discipline satisfying the assumptions of Section 1. Each surface configuration for such a machine will contain:

(i) the current state of M (in the finite control).
(ii) the position of the input head.
(iii) the contents of the work tapes and positions of the work tape heads.
(iv) the relative height parameter, which may be bounded by the time bound t(n).

For a fixed machine M, the number of such surface configurations is bounded by \(2^{O(s(n)) \cdot t(n)O(1)}\).

The simulation algorithm is exactly the one given in section 4, except that the number of stages is now \(T \log t(n) + 1\). Thus, if \(t(n) = 2^{O(s(n))}\), the algorithm requires time \(O(s(n))\) and \(2^{O(s(n))}\) processors. Thus we have

Theorem 3 Let L be accepted by a deterministic APDA with space s(n) and time \(2^{O(s(n))}\). Then L is accepted by a P-RAM with time \(O(s(n))\) and \(2^{O(s(n))}\) processors.

Section 7: Simulation of P-RAMs by Deterministic APDAs

We show here that our P-RAM algorithm of Section 4 for simulating deterministic APDAs is nearly optimal, since there is a complementing simulation of P-RAMs by deterministic APDAs.

Theorem 4 Let L be accepted by P-RAM with time T(n). Then L is accepted by a deterministic APDA with space T(n) and time \(2^{O(T(n)^2)}\).

proof [Fortune and Wyllie, 78] prove in their Lemma 1b that L is accepted by a deterministic TM with \(T(n)^2\) space, and time \(2^{O(T(n)^2)}\). We use exactly their algorithm, but implement it on a deterministic APDA. Their algorithm is recursive and requires a pushdown stack of size at most T(n), where each element on the stack can be represented by a bit sequence of length
lemma 13 If \( \langle P_k + 1[x], y \rangle_x \cdot <x, \text{PREDICT}_{k + 1[x], y} \rangle_x \cdot (2k + 1) \cdot y \rangle_x \). (Correctness of \( \text{PREDICT}_{k + 1} \)).

proof Let \( x^* = R_k[x] \).

case 1) \( h(y) \leq h(x^*) \). In this case, \( \text{PREDICT}_{k + 1[x], y} = \text{HOP}_{k + 1[x], y} \). [see figure 12a] By corollary to lemma 10, \( \langle P_k[x], k[x^*], P_k + 1[x] \rangle_x \cdot \), and we assume \( \langle P_k + 1[x], y, y \rangle_x \cdot \).

Since \( h(y) \leq h(x^*) = h(l_k[x^*]) \), by lemma 6, \( \langle P_k[x], y \rangle_x \cdot \). Then by lemma 12,

\( \langle y, \text{HOP}_{k + 1[x], y} \rangle_x \cdot (2k + 1) \cdot y \rangle_x \).

case 2) \( h(y) > h(x^*) \). We are given \( \langle P_k + 1[x], y, y \rangle_x \cdot \). Choose \( b \) so that \( y = b[P_k + 1[x]]_x \).

Recall that \( P_k + 1[x] = P_k[x^*] \cdot \), \( [P_k + 1[x]]_x \cdot \), and let \( y' = b[P_k + 1[x]]_x \cdot \). We have \( h(P_k + 1[x]) \geq h(y) > h(x^*) \), so by lemma 3, \( y' = b[P_k + 1[x]]_x \cdot = b[P_k + 1[x]]_x \cdot = y \). Then \( [P_k + 1[x], y]^* \cdot \) exists, so by lemma 1, \( \langle P_k + 1[x], y, y \rangle_x \cdot \).

Now suppose \( w \in [P_k + 1[x], y]_x \cdot \); then \( w \in [P_k + 1[x], y]_x \cdot \) so \( h(w) \geq h(y) \) (since we have

\( \langle P_k + 1[x], y, y \rangle_x \cdot \). We may then conclude \( \langle P_k[x^*], y, y \rangle_x \cdot \) (recalling that

\( P_k[x^*] = P_k + 1[x] \). Let \( z = \text{HOP}_{k + 1[x], y} \). By lemma 12, \( \langle y, z, (2k + 1) \cdot y \rangle_x \cdot \).

case A) \( h(z) > h(x^*) \). [see figure 12b] In this case, \( \text{PREDICT}_{k + 1[x], y} = z \). and by lemma 7, \( \langle y, z, (2k + 1) \cdot y \rangle_x \cdot \).

case B) \( h(z) = h(x^*) \). In this case, \( \text{PREDICT}_{k + 1[x], y} = \text{HOP}_{k + 1[x], y} \). [see figure 12c] Since \( \langle P_k + 1[x], y, y \rangle_x \cdot \) and \( \langle y, z, y \rangle_x \cdot \), by lemma 6 we have

\( \langle P_k + 1[x], z, z \rangle_x \cdot \). Let \( z' = \text{HOP}_{k + 1[x], z} \). By lemma 12, \( \langle z, z', (2k + 1) \cdot z \rangle_x \cdot \).

Then certainly \( \langle z, z', (2k + 1) \cdot z \rangle_x \cdot \) and so, by lemma 6, \( \langle y, z', (2k + 1) \cdot y \rangle_x \cdot \).

lemma 14 \( \langle P_k + 1[x], R_k + 1[x], (2k + 1) \cdot P_k + 1[x] \rangle_x \cdot (\text{Correctness of } R_k + 1) \).

proof By definition, \( R_k + 1[x] = \text{PREDICT}_{k + 1[x], P_k + 1[x]} \). Trivially,

\( \langle P_k + 1[x], P_k + 1[x], P_k + 1[x] \rangle_x \cdot \), so the lemma follows from lemma 13.

Theorem 1 follows by induction on \( k \).
corollary $<P_k[x], 1_k, R_k[x], P_k + 1[x]>_x$.

proof Let $x^* = R_k[x]$. By correctness of $R_k$, $<P_k[x], x^*, x^*>_x$. By lemma 10, $<P_k[x], 1_k, x^*, R_k[x], P_k[x]>_x$.

Recalling that $P_k + 1[x] = P_k[x^*]$, we are done.

lemma 11 $<x, 1_{k + 1}, [x], P_k + 1[x]>_x$ and $h(1_{k + 1}[x]) = h(x)$. (Correctness of $1_{k + 1})$

proof [see figure 10] Clearly $h(1_{k + 1}[x]) = h(x)$ by the definition of $1_{k + 1}[x]$. By correctness of $1_{k - 1}$, $<x, 1_k[x], P_k[x]>_x$. By corollary to lemma 10, $<P_k[x], 1_k, [R_k[x], P_k + 1[x]>_x$. Then by lemma 6, $<x, 1_{k + 1}, [x], P_k + 1[x]>_x$ (see definition of $1_{k + 1}[x]$.)

lemma 12 If $<P_k[x], y, y>_x$ then $<y, HOP_k + 1, [x,y], (2k + 1, [y]>_x$.

proof Let $z = PREDICT_k[x, y]$. Since $<P_k[x], y, y>_x$, by correctness of $PREDICT_k$, $<y, z, (2k)[x]>_x$. It follows that $<y, z, z>_x$, and that $[y, z, \lambda]_x$ exists. Let $z^* = R_k[z]$. By correctness of $R_k$, $<P_k[z], z^*, (2k)[P_k[z]]>_z$.

1) $h(z^*) = h(z)$. In this case, $HOP_k + 1[x, y] = L_k[z]$. [see figure 21a] By lemma 7, $<P_k[z], z^*, (2k)[P_k[z]]>_z$. Since $<y, z, z>_x$, we may apply lemma 10 to obtain $<x, 1_k[z], P_k[z]>_x$. Hence, by lemma 6, since $h(1_k[z]) = h(z) h(z^*)$, we obtain $<y, 1_k[z], (2k)[P_k[z]]>_x$. By correctness of $P_k$, $P_k[z] = a[z]$, for some $a \geq 2k$. Since $h(P_k[z]) = h(z^*)$, by lemma 3 we obtain $a[z] = a[z]_x$, hence $P_k[z] = a[z]_x$. Since $[y, z, \lambda]_x$ exists, $z = b[y]_x$ for some $b \geq 0$, so $(2k)[P_k[z]] = (2k + a + b)[y]_x$. Since $a \geq 2k$, $(2k)[P_k[z]] = (2k + 1 + c)[y]_x$ for some $c \geq 0$, so we may conclude $<x, L_k[z], (2k + 1 + c)[y]_x>.$

2) $h(z^*) = h(z)$. In this case, $HOP_k + 1[x, y] = PREDICT_k[x, z^*]$. [see figure 21b] We assumed $<P_k[x], y, y>_x$. By correctness of $PREDICT_k$, $<y, z, z>_x$. By the stack height proposition, $<z, z^*, z^*>_x$ so by lemma 4, $<z, z^*, z^*>_x$. Then by lemma 6, $<y, z, z^*>_x$, and by a second application of lemma 6, $<P_k[x], z^*, z^*>_x$. Let $z^* = PREDICT_k[x, z^*]$. By correctness of $PREDICT_k$, $<z^*, z^*, (2k)[z^*>_x$. Since $<y, z, z^*>_x$, by lemma 6 we obtain $<y, z, (2k)[z^*>_x$.

If $z^* \in \epsilon(z, (2k)[y], \lambda)_x$, then $z^*$ would contradict $<y, z, (2k)[y]_x$ because $h(z^*) = h(z)$. Thus $z^* = a[y]_x$ for some $a > 2k$, so $(2k)[z^*] = b[y]_x$ for some $b > 2k + 1$, so we may
hold for \( k + 1 \). Then by induction, \( P_{\Gamma_1} \rangle \Gamma \) satisfies its correctness hypothesis. In particular, if \( \ell_{\text{init}} \) is the initial surface configuration for input \( \omega_1 \ldots \omega_n \), \( P_{\Gamma_1} \rangle \Gamma \rangle \ell_{\text{init}} \) will give the final surface configuration, and thus the final state of the PDA. The proof follows:

**Theorem 1** \((x,e) \vdash a (P_{\Gamma_1} \rangle \Gamma \rangle x, s_{\Gamma_1} \rangle \Gamma \rangle [s]) \) for \( a \geq 1 \).

**Lemma 8** Let \( x^* = R_k \). Then \((x,e) \vdash^* (x^*, sk[x,x^*])\).

**Proof** [see figure 17] By correctness of \( P_k \), \((x,e) \vdash^* (P_k \rangle x, sk[x])\). By correctness of \( R_k \), \((P_k \rangle x, sk[x]) \vdash^* (x^*, s) \) for some \( s \) and if \( w \in [P_k \rangle x, x^*, x] \) then \( h(w) \geq h(x^*) \). Hence by lemma 5, \( s \) is an initial substring of \( sk[x] \). Since by stack height proposition, \( |s| = h(x^*) - h(x) \), we conclude \( s = sk[x,x^*] \).

**Lemma 9** \((x,e) \vdash a (P_{k+1} \rangle [x], sk + 1 \rangle [x]) \) for some \( a \geq 2^k + 1 \). (Correctness of \( P_{k+1} \))

**Proof** Let \( x^* = R_k \). By correctness of \( P_k \), \((x,e) \vdash a (P_k \rangle x, sk[x])\) and \((x^*, e) \vdash^b (P_k \rangle x^*, sk[x^*])\) for some \( a \) and \( b \geq 2^k \).

By lemma 8, \((x,e) \vdash^* (x^*, sk[x,x^*])\). By combining computations, \((x,e) \vdash^* (P_k \rangle x^*, sk[x^*])\), and \((P_k \rangle x^*, sk[x^*] + sk[x^*]) = (P_{k+1} \rangle [x], sk + 1 \rangle [x])\) by definition of \( P_{k+1} \) and \( sk + 1 \).

**Case 1** \( h(P_k \rangle x^*) = h(x^*) \). [see figure 18] By correctness of \( R_k \), if \( w \in (x^*, 2^k, [P_k \rangle x], x) \) then \( h(w) \geq h(x^*) \), so we must conclude that \( P_k \rangle x^* \rangle (x^*, (2^k, [P_k \rangle x], x) \rangle x \). So \((P_k \rangle x^*, sk[x]) \vdash^c (P_k \rangle x + sk + 1 \rangle [x])\) for some \( c \geq 2^k \).

**Case 2** \( h(P_k \rangle x^*) > h(x^*) \). [see figure 19] Then by lemma 3, \((x^*, sk[x,x^*]) \vdash^b (P_k \rangle x^*, sk[x^*])\), so again \((P_k \rangle x^*, sk[x]) \vdash^c (P_k \rangle x^*, sk + 1 \rangle [x])\) for some \( c \geq b \geq 2^k \).

We conclude in either case that \((x,e) \vdash a + c (P_{k+1} \rangle [x], sk + 1 \rangle [x])\) with \( a + c \geq 2^k + 2^k = 2^{k+1} \).

**Lemma 10** If \( \langle x,y,y \rangle \rangle u \) then \( \langle x,1,k \rangle \rangle P_k \rangle y \rangle u \).

**Proof** Let \( y^* = 1, k \). [see figure 20] By correctness of \( L_k \), \( \langle y, y^*, P_k \rangle y \rangle \rangle y \rangle h(y') = h(y) \). Since \( \langle y \rangle \rangle u \), \( \langle y, y^*, P_k \rangle y \rangle \rangle u \) by lemma 4. Then by lemma 6, we conclude \( \langle x,y', P_k \rangle y \rangle \rangle u \) since \( h(y') = h(y) \).
Section 5: Proof of Correctness

We introduce variables denoting the contents of the simulated stack. Their values are never computed; they exist only to make the proof easier. We define $s_k[x]$ and $s_k[x,y]$ inductively as follows:

if $(x, \varepsilon) \vdash (y, \varepsilon)$
then let $s_0[x] = s$

For $k = 0, \ldots, T \log T - 1$,
let $s_{k+1}[x] = s_k[x^*]s_k[x^*]$
where $x^* = R_k[x]$
and for all $y \in X$ with $h(y) \geq h(x)$,
$s_k[x,y] = \text{the leftmost } h(y) - h(x) \text{ symbols of } s_k[x]$.

The values of the arrays $P_k[x]$, $L_k[x]$, $HOP_k[x,y]$, $PREDICT_k[x,y]$, and $R_k[x]$ will be inductively shown to satisfy the following correctness hypotheses, for each $k$:

**Correctness of $P_k$:** $(x, \varepsilon) \vdash ^a (P_k[x], s_k[x])$ for some $a \geq 2^k$. [see figure 13]

**Correctness of $L_k$:** $(x, L_k[x], P_k[x]) \vdash x$ and $h(L_k[x]) = h(x)$. [see figure 14]

**Correctness of $PREDICT_k$:** if $(P_k[x], y, y) \vdash x$ then $(x, PREDICT_k[x], y, 2^k, y) \vdash x$. [see figure 15]

**Correctness of $R_k$:** $(P_k[x], R_k[x], 2^k, P_k[x]) \vdash x$. [see figure 16]

It can easily be checked that the initial values satisfy the above hypotheses, for $k = 0$. Assuming the correctness hypotheses hold for $P_k$, $L_k$, $PREDICT_k$, and $R_k$, we show the corresponding conditions...
For each $x,y \in \mathcal{X}$ such that $h(x) \leq h(y) \leq h(P_k[x])$.

let $\text{HOP}_{k+1}[x,y] := \begin{cases} \text{PREDICT}_k[x,x^*] & \text{if } h(x^*) = h(z) \\ 1_{k}[z] & \text{else} \end{cases}$

where $z = \text{PREDICT}_k[x,y]$ and $x^* = R_k[z]$

[see figure 11]

For each $x,y \in \mathcal{X}$ such that $h(x) \leq h(y) \leq h(P_k[x])$.

let $\text{PRDICT}_{k+1}[x,y] := \begin{cases} \text{HOP}_{k+1}[x,x] & \text{if } h(y) \leq h(x^*) \\ \text{HOP}_{k+1}[x,x] & \text{if } h(y) > h(x^*) \text{ and } h(z) = h(x^*) \\ z & \text{else} \end{cases}$

where $x^* = R_k[x]$ and $z = \text{HOP}_{k+1}[x^*,y]$

[see figure 12]

For each $x \in \mathcal{X}$,

let $R_{k+1}[x] := \text{PREDICT}_{k+1}[x,P_{k+1}[x]]$

We shall show in the next section that $P_k[x]$ gives a surface configuration reachable from $(x,\varepsilon)$ in at least $2^k$ steps. In particular, if $(x_{\text{init}},\varepsilon)$ is the initial configuration and $(x_{\text{acc}},\varepsilon)$ the accepting configuration when the input is $\omega_1...\omega_n \in \Sigma^\ast$, then $P_{\Gamma \log T}[x_{\text{init}}] = x_{\text{acc}}$ iff $M$ accepts the input (recalling that we assumed $(x_{\text{acc}},\varepsilon)$ to be a looping configuration).

Each of the $\Gamma \log T + 1$ stages requires constant time on a P-RAM, if we assign a processor to each pair $x,y$ of surface configurations. Recalling that $T = 2 |Q||S| \cdot (n + 1)$ and that the number of surface configurations is $|\mathcal{X}| = |Q||S|(n + 1)^2$, we see that, for a fixed DPDA, the algorithm requires parallel time $O(\log n)$ and $O(n^4)$ processors.
reader scan the inductive corrective hypotheses and the corresponding figures (found in the next section) in order to better understand the algorithm.

The initialization is as follows.

For each \( x \in X \),

\[ \begin{align*}
\text{if } (x, \varepsilon) &\rightarrow (y, s), \\
\text{let } P_0(x) &\Rightarrow y \\
\text{let } L_0(x) &\begin{cases} P_0(x) & \text{if } h(P_0(x)) = 0 \\
x & \text{else} \end{cases}
\end{align*} \]

For each \( x, y \in X \) such that \( h(x) \leq h(y) \leq h(P_0[x]) \),

\[ \begin{align*}
\text{if } (x, \varepsilon) &\rightarrow (x', s'), s \text{ is the initial substring of } s' \text{ of length } h(y) - h(x), \\
\text{and } (y, s) &\rightarrow (z, s'') \text{ for some } s'' \\
\text{let } \text{PREDICT}_0[x, y] &\Rightarrow \begin{cases} z & \text{if } h(z) \leq h(y) \\
y & \text{else} \end{cases}
\end{align*} \]

[see figure 8]

For each \( x \in X \),

\[ \text{let } R_0(x) : = \text{PREDICT}_0[x, P_0[x]] \]

The algorithm proceeds in stages \( k = 0, \ldots, \lceil \log T \rceil \). At stage \( k + 1 \), we assume that the values of \( P_k, L_k, \text{PREDICT}_k, \) and \( R_k \) have been stored, and compute \( P_{k + 1}, L_{k + 1}, \text{PREDICT}_{k + 1}, \) and \( R_{k + 1} \) as follows:

For each \( x \in X \),

\[ \begin{align*}
\text{let } P_{k + 1}[x] &\Rightarrow P_k[R_k[x]] \\
\text{[see figure 9]} \\
\text{let } L_{k + 1}[x] &\begin{cases} P_k[R_k[x]] & \text{if } h(R_k[x]) = h(x) \\
L_k[x] & \text{else} \end{cases}
\end{align*} \]

[see figure 10]
Lemma 1 If \([u]_x, [x,y]_u\) exist, then \([x,y]_x, [x,z]_u\).

[see figure 2]

Lemma 2 If \([u]_x, [x,y]_u\) exist, then \([x,y]_x, [x,z]_u\).

Lemma 3 If \([u]_x, [x,y]_u\) exist, then \([x,y]_x, [x,z]_u\).

Lemma 4 If \([u]_x, \langle x,y,z \rangle, [x,y]_u\), then \(\langle x,y,z \rangle, [x,y]_u\).

[see figure 3]

Lemma 5 If \((u,x) \leftarrow (x,s_1), (y,s_2)\) and for each \(w \in \{x,y\}_u\), \(h(w) \geq h(y)\), then \(s_1 = s_2 s'\) for some \(s' \in \Sigma^*\).

[see figure 5]

Lemma 6 Suppose \(\langle x,y,z \rangle, [x,y]_u\) and \(\langle x,y',z \rangle, [y]_u\).

Let \(y'' = \begin{cases} y' \text{ if } h(y') \leq h(y) \\ y \text{ else} \end{cases}\)

Then \(\langle x,y'',z \rangle, [y]_u\).

[see figure 6]

Lemma 7 If \([u]_x, \langle x,y,az \rangle, [x,y]_u\), then \(\langle x,y,az \rangle, [x,y]_u\).

[see figure 7]

Section 4: The Algorithm

We now describe the algorithm for simulating \(T = 2|Q||S|(n+1)\) steps of \(M\). (An argument like that found in [Aho and Ullman, p. 396] will establish that if \(M\) accepts an input of length \(n\), it does so in at most \(T\) steps.) The data structures used are arrays indexed by surface configurations and having surface configurations as their elements: \(P_k[x], I_k[x], HOP_k[x,y], PRE_{\Sigma[C]} k[x,y], \) and \(R_k[x]\), for \(k = 0, \ldots, \lceil \log T \rceil\). The algorithm is relatively short, but its proof contains a non-trivial induction. We suggest the
(\sigma, \delta'). By the stack height proposition, \(|s'| = |s|\). If any intermediate configuration \((\sigma, \delta')\) had \(|s'| < |s|\) then \(p(x) \geq p(y) \supset p(x)\), a contradiction. Similarly, if \(|s'| = |s|\), then \(p(x) \supset p(y) \geq p(x)\).

Hence \(|s'| = |s|\), i.e. every intermediate configuration has stack height equal to \(|s|\). But since \(M\) never changes more than the top symbol of the stack, we may conclude \(s' = s\).

Let \([x]_u\) denote the proposition \((u, e) \vdash^* (x, s)\) for some \(s\).

If \((u, e) \vdash^* (x, s_1) \vdash^* (y, s_2)\) for some \(s_1, s_2\), then we shall say \([x, y]_u\) exists, and denotes the set \(\{z \mid (x, s_1) \vdash^* (z, s_3) \vdash^* (y, s_2)\}\). Otherwise \([x, y]_u\) does not exist.

If \([x, y]_u\) exists, we let \((x, y]_u = \{z \mid z \in [x, y]_u\) and \(z \neq x\).\]

Note that \([x, y]_u\) and \((x, y]_u\) are intended to suggest closed and open intervals, respectively.

If \((u, e) \vdash^* (x, s_1) \vdash^* (y, s_2)\) then \([x]_u\) denotes \(y\).

Let \(<x, y, z>_u\) denote the proposition:

1. \([x, y]_u\) and \([x, z]_u\) exist;
2. if \(w \in [x, y]_u\) then \(h(w) \geq h(y)\);
3. if \((y, z]_u\) exists and \(w \in (y, z]_u\) then \(h(w) \supset h(y)\).

Informally, with respect to computations starting at \(u\), either \(y\) occurs between \(x\) and \(z\) and has minimal stack height of all such intermediate configurations (and is otherwise latest), or \(y\) occurs after \(z\) and has minimal stack height of all configurations between \(x\) and \(z\). [see figure 1]

Note that it follows from \(<x, y, z>_u\) that \(<x, y, w>_u\) for any \(w \in [y, z]_u\) (and in particular that \(<x, y, y>_u\) and that \(<w, y, z>_u\) for any \(w \in [x, y]_u\) (and in particular that \(<y, y, z>_u\).

We state eight lemmas whose proofs may be found in the appendix. We suggest the reader try proving one in order to become familiar with the notation.
REFERENCES


Ruzzo, W. L., private communication, (1982).

figure 6a: $\langle x, y', z' \rangle_u$ in case $h(y') \leq h(y)$

figure 6b: $\langle x, y, z \rangle_u$ in case $h(y') > h(y)$

figure 7: $\langle x, y, a[z] \rangle_u$ implies $\langle x, y, a[z] \rangle_v$
Figure 12a: \texttt{PREDICT}_{k+1}[x,y] = HOP_{k+1}[x,y] in case \( h(y) \leq h(x^*) \)

Figure 12b: \texttt{PREDICT}_{k+1}[x,y] = z \text{ in case } h(y) > h(x^*) \text{ and } h(z) > h(x^*)

Figure 12c: \texttt{PREDICT}_{k+1}[x,y] = HOP_{k+1}[x,z] \text{ in case } h(y) > h(x^*) \text{ and } h(z) < h(x^*)
Figure 13: \((\alpha, \epsilon) \rightarrow (P_{k}[x], s_{K}[x])\) for some \(x \in 2^{k}\)

Figure 14: \(\ll L_k(x), P_k(x) \gg \land h(L_k(x)) = h(x)\)
Figure 17: $P_k[x^*] \in \epsilon(x^*, (2^k)P_k[x])_x$ because $h(P_k[x^*]) = h(x^*)$

Figure 18: $(x, \epsilon) \leadsto^* (x^*, s_k[x, x^*])$
figure 19: $h(P_k[x^*]) > h(x^*)$

figure 20: $\langle x, y', P_k[y] \rangle_u$
figure 21a: \( \text{HOP}_{k+1}[x,y] = L_k[z] \) in case \( h(z^*) > h(z) \)

figure 21b: \( \text{HOP}_{k+1}[x,y] = \text{PREDICT}_k[x,z^*] \) in case \( h(z^*) = h(z) \)