ITERATED PUSHDOWN AUTOMATA AND COMPLEXITY CLASSES

Joost Engelfriet

Twente University of Technology
P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract. An iterated pushdown is a pushdown of pushdowns of ... of pushdowns. An iterated exponential function is 2 to the 2 to the ... to the 2 to some polynomial. The main result is that nondeterministic 2-way and multi-head iterated pushdown automata characterize deterministic iterated exponential time complexity classes. This is proved by investigating both nondeterministic and alternating auxiliary iterated pushdown automata, for which similar characterization results are given. In particular it is shown that alternation corresponds to one more iteration of pushdowns. These results are applied to the 1-way iterated pushdown automata: (1) they form a proper hierarchy with respect to the number of iterations, (2) their emptiness problem is complete in deterministic iterated exponential time.

1.Introduction

It is well known that several types of 2-way and multi-head pushdown automata and stack automata have the same power as certain time or space bounded Turing machines (see, e.g., [HU2]). For the deterministic and nondeterministic case such characterizations were given by Hopcroft and Ullman [HU1] for nonerasing stack automata, by Cook for auxiliary pushdown and 2-way stack automata [Coo], by Ibarra for auxiliary (nonerasing and erasing) stack automata, and by Beeri for 2-way and auxiliary nested stack automata [Bee]. The alternating case was considered by Chandra, Kozen, and Stockmeyer [CKS], and was extensively studied by Ladner, Lipton, and Stockmeyer [LLS]. The highest complexity class reached in this way by 2-way or multi-head automata was $\cup\{ DTIME(2^{2^{p(n)}}) \mid p(n) \text{ is a polynomial} \}$: the class of languages recognized by alternating multi-head stack automata [LLS].

We generalize these results by an automaton-theoretic characterization of $\cup\{ DTIME(exp_k(p(n))) \mid p(n) \text{ is a polynomial}\}$, where $exp_k(n)$ is the k-iterated exponential function (k is the number of 2's): $exp_0(n) = n$ and $exp_{k+1}(n) = 2^{exp_k(n)}$. The corresponding automaton is the nondeterministic multi-head (k+1)-iterated pushdown automaton (a 1-iterated pushdown is an ordinary pushdown, a 2-iterated pushdown is a pushdown of pushdowns, etc). According to [Gre], iterated pushdown automata were first considered by Aho and Ullman : they showed that the nondeterministic 1-way 2-iterated pushdown automata recognize the indexed languages; see [PDS] for essentially the same result. Greibach shows in [Gre] how pushdowns can be iterated by the use of "nested AFA", but only the special case of "well-nested AFA" is studied there. Maslov [Ma 1,2] defines the nondeterministic 1-way k-iterated pushdown automata and shows that they correspond to the k-level indexed grammars. Damm and Goerdt [DGo] prove that they correspond to the k-level OI macro grammars. In both papers these automata are called multi-level pushdown automata. The classes of k-level OI languages (and hence the classes of nondeterministic 1-way k-iterated pushdown languages) are often viewed as an alternative, more natural, infinite,

Chomsky hierarchy, called the OI-hierarchy (see, e.g.,[Dam]). Taking a 0-iterated pushdown automaton to mean a finite automaton, the first three classes in this hierarchy consist of the regular, the context-free, and the indexed languages.

Our main results on k-iterated pushdown automata (P^k-automata) are given in the Table of Fig.1, where we consider nondeterministic or alternating P^k automata (we did not study the deterministic case) which may be 1-way, (2-way) r-head for $r \geq 1$, (2-way) multi-head, or SPACE(s(n)) auxiliary P^k automata. Note that the multi-head case follows from both the r-head and the SPACE(log n) case; it is added for

P^k	nondeterministic	alternating
1-way	———	$DTIME(exp_k(cn))$ $k \geq 1$
r-head	$DTIME(exp_{k-1}(cn^{2r}))$ $k \geq 2$	$DTIME(exp_k(cn^r))$ $k \geq 1$
multi-head	$DTIME(exp_{k-1}(poly))$ $k \geq 1$	$DTIME(exp_k(poly))$ $k \geq 0$
$SPACE(s(n))$ $s(n) \geq \log n$	$DTIME(exp_k(cs(n)))$ $k \geq 1$	$DTIME(exp_{k+1}(cs(n)))$ $k \geq 0$

Fig.1. Table: Characterization of k-iterated pushdown automata by time-bounded Turing machines (...c... abbreviates $\cup\{...c... \mid c > 0\}$, and ...poly... abbreviates $\cup\{...p(n)... \mid p(n)$ is a polynomial$\}$).

clearness sake. For k=1 (k=0), the results are of course the known ones for pushdown automata (finite automata, respectively). For k=2, the results are those for stack automata: we show that the mentioned types of 2-iterated pushdown automata are equivalent to the corresponding stack automata. Since, moreover, almost all reasonable types of 2-iterated pushdown automata are equivalent to the corresponding nested stack automata (e.g., the 1-way types both recognize the indexed languages), this also fits with the results of [Bee]. In fact, for iterated stack automata (SA^k) and iterated nested stack automata (NSA^k) we show that the Table of Fig.1 holds with k replaced by 2k everywhere.

As mentioned before, we will point out the consequences of these facts for the deterministic and nondeterministic 1-way P^k automata.

2. Terminology

We first mention some general terminology on automata (cf. [Sco, Gin, En2, Gol]), and then define iterated pushdown automata.

A storage type X is specified by a set C (of storage configurations), an element c_0 of C (the initial storage configuration), a set T of tests which are partial functions $C \to \{$true, false$\}$, and a set F of operations which are partial functions $C \to C$. As an example, the usual pushdown storage type P is defined by taking $C = G^*$ for some, possibly infinite, alphabet G (with at least two elements), $c_0 = g_0$ for some fixed symbol g_0 in G, $T = \{top=g? \mid g \in G\} \cup \{empty?\}$ such that empty?(w) iff w is the empty string λ, and top=g?(w) iff the rightmost symbol of w is g, $F = \{push(g) \mid g \in G\} \cup \{pop\}$ with $push(g) = \{(w, wg) \mid w \in G^*\}$ for all $g \in G$, and $pop = \{(wg, w) \mid w \in G^*, g \in G\}$. An automaton type consists of a storage type X together with a way of handling the input (such as 1-way, 2-way r-head, 2-way multi-head, or

auxiliary SPACE(s(n))) and a class of flowcharts that make use of the tests and operations of X, and of the appropriate tests and operations on the input (such as the classes of all deterministic, nondeterministic, or alternating flowcharts). Each such flowchart M is an X-automaton of that type; M starts its computation with its storage in configuration c_0 and accepts "by final state", i.e., by entering a halt-node; thus M accepts a language over its input alphabet. Note that an auxiliary SPACE(s(n)) X-automaton has, in addition to its X-storage, a 2-way read-only input tape and a Turing machine worktape with space restricted to s(n), where n is the length of the input. For the notion of alternation see [CKS, LLS].

Notation. The classes of languages accepted by alternating 1-way, 2-way r-head ($r \geq 1$), 2-way multihead, and auxiliary SPACE(s(n)) X-automata are denoted by 1A-X, 2ArH-X, 2AMH-X, and ASPACE(s(n))-X, respectively. □

We will use the following terminology on the simulation of one storage type by another. For a storage type $X = (C,c_0,T,F)$, let FC(X) denote the class of all deterministic flowcharts that make use of the tests in T and the operations in F, and of which the halt-nodes are labeled true or false. For $p \in FC(X)$ and $c \in C$, $p(c) \in C$ is the resulting configuration when executing p with start configuration c, and $exit(p,c) \in \{true,false\}$ is the label of the resulting halt-node.

For storage types $X_i = (C_i,c_{0i},T_i,F_i)$, $i = 1,2$, we say that X_2 simulates X_1 (denoted $X_1 \leq X_2$) if there exist a partial function $h: C_2 \to C_1$, a program $p_0 \in FC(X_2)$, and for every $a \in T_1 \cup F_1$ a program $p_a \in FC(X_2)$, such that (1) $h(p_0(c_{02})) = c_{01}$, and (2) for every $f \in F_1$, $t \in T_1$, $c_1 \in C_1$, and $c_2 \in C_2$, if $h(c_2) = c_1$, then $h(p_f(c_2)) = f(c_1)$, $h(p_t(c_2)) = c_1$, and $exit(p_t,c_2) = t(c_1)$. Thus each configuration c_1 of X_1 is simulated by any of the configurations of X_2 in $h^{-1}(c_1)$, each instruction of X_1 is simulated by a deterministic flowchart of X_2, and the simulation is initialized by p_0.

It should be clear that for any of the classes Y-X mentioned above (Y = 1A, 2ArH, ..., 1N, 1D, ...), if $X_1 \leq X_2$ then $Y\text{-}X_1 \subseteq Y\text{-}X_2$. If $X_1 \leq X_2$ and $X_2 \leq X_1$, then X_1 and X_2 are equivalent storage types (and then $Y\text{-}X_1 = Y\text{-}X_2$).

We are now ready to define the storage type "pushdown of X-configurations", cf. [Gre]. Let $X = (C,c_0,T,F)$ be a storage type, and let G be a fixed, possibly infinite, alphabet with at least two elements and a fixed designated element g_0. The storage type P(X) is defined to be (C',c_0',T',F') where $C' = (G \times C)^*$, i.e., each element of C' is a pushdown of pairs < symbol, X-configuration >, $c_0' = \langle g_0,c_0\rangle$, $T' = \{test(t) \mid t \in T\} \cup \{top=g? \mid g \in G\} \cup \{empty?\}$, and $F' = \{push(g,f) \mid g \in G, f \in F\} \cup \{push(g) \mid g \in G\} \cup \{pop\}$. The tests and operations of P(X) are defined as follows. Of course, empty?(c') iff $c' = \lambda$. The other tests and operations are undefined on λ. Now let $u\langle g',c\rangle$ be a P(X)-configuration with $u \in C'$, $g' \in G$, $c \in C$; $\langle g',c\rangle$ is the top of the pushdown.

(i) For $t \in T$, the test test(t) applies t to the top X-configuration c; top=g? tests whether $g' = g$.

(ii) Pop is as usual, with result u; when pushing, the top X-configuration c is operated upon, or duplicated: push(g,f) gives $u\langle g',c\rangle\langle g,f(c)\rangle$, and push(g) gives $u\langle g',c\rangle\langle g,c\rangle$.

We observe that one may also allow operations stay(g,f), and stay(g), that transform $u\langle g',c\rangle$ into $u\langle g,f(c)\rangle$, and $u\langle g,c\rangle$, respectively: these can be simulated by a push($g,f), and push($g), respectively, where the $ indicates that, when popping, the pushdown element below this one must also be popped (thus pop should be simulated by the deterministic flowchart while top $\in$ $G do pop od; pop). Formally, if $P_s(X)$ is P(X) with stay-operations added, this shows that $P_s(X) \leq P(X)$; since the other direction is trivial, $P_s(X)$ and P(X) are equivalent storage types. Note that, actually, we have simulated $P_s(X)$ with G by P(X) with $G \cup \$G$; it is left to the reader to show that, as usual, P(X) and $P_s(X)$ do not depend on G (all storage types with different G are equivalent, in particular for $G = \{0,1\}$: if $G = \{g_1,g_2,\ldots\}$, then simulate $\langle g_i,c\rangle$ by $\langle 1,c\rangle\langle 0,c\rangle^i$; e.g., push($g_i,f) is simulated by push(1,f);push(0)^i; and pop by while top = 0 do pop od; pop). We note that $P_s(X)$ is close to the nested AFA of [Gre], the difference being the extra pushdown symbols (in [Gre], $C' = C^*$) which are not needed for well-nested AFA.

The operation P on storage types can be iterated: $P^0(X) = X$, $P^1(X) = P(X)$, $P^{k+1}(X) = P(P^k(X))$. Let FA be the trivial storage type (of the finite automaton): $C = \{c_0\}$, $T = F = \emptyset$. Then $P(FA) = P$, the ordinary pushdown storage type. The k-iterated pushdown storage type P^k is defined for $k \geq 0$ to be $P^k(FA)$, i.e., $P^0 = FA$ and, for $k \geq 1$, $P^k = P^{k-1}(P)$. See [Ma2, DGo] for a direct definition of the k-iterated pushdown. Since P is a well-behaving operation on storage types, all the usual variations on pushdowns are also valid for iterated pushdowns (e.g., stay-operations can be added at each level, $G = \{0,1\}$ may be assumed, etc.). P is well behaving in the sense that if $X_1 \leq X_2$ then $P(X_1) \leq P(X_2)$, which can easily be proved (for P_s instead of P).

For notation on complexity, see, e.g., [HU2]. We will use $\cup\ldots$ to stand for $\cup\{\ldots \mid c > 0\}$.

3. Main results

The first two main results are "taken from" [LLS] and [Ruz]. The first result (Theorem 1(i)(ii)) considers alternating auxiliary pushdown (of X) automata.

Theorem 1(i). For any storage type X and $s(n) \geq \log n$, $ASPACE(s(n))\text{-}P(X) = \cup\, ASPACE(2^{cs(n)})\text{-}X$.

Proof. The proof is a rather straightforward generalization of the proof of Theorem 3.1 of [LLS], where it is shown that $ASPACE(s(n))\text{-}P = \cup\, ASPACE(2^{cs(n)})$, i.e., the result for X = FA.

$\supseteq$: Just as in [LLS]. Given M, M' is constructed such that the symbol part of its pushdown contains $a_0 m_1 a_1 m_2 a_2 \ldots$ where a_i is the worktape configuration of M. Each square of $m_i a_i$ has the same associated X-configuration c_i, viz. the storage configuration of M; this can be realized by duplication, i.e., by push(g).

$\subseteq$: For given M, the notion of surface ID of M can be defined as in [LLS]; it now involves also an X-configuration. For $r \to \{z_1,\ldots,z_t\}$ we only have to consider surface ID's $r, z_1,\ldots,z_t$ with the same X-configuration (because there are no stay-operations). It is now possible to find out whether $r \to \{z_1,\ldots,z_t\}$ using the proof rules (1)-(4) of [LLS] in a backward fashion, nondeterministically. This can be done by an alternating auxiliary X-automaton M' whose computation tree mirrors a proof tree for $r \to \{z_1,\ldots,z_t\}$. M' makes a universal move only when simulating proof rule 2(b). M' keeps the X-configuration part of $r,z_1,\ldots,z_t$ as its own X-configuration and applies f to it when M uses push(g,f), see proof rule 4. M' keeps the worktape part of $r,z_1,\ldots,z_t$ on its own worktape. Since there are at most $2^{ds(n)}$ worktape configurations of M for some $d > 0$, this takes space $(1+2^{ds(n)})s(n) \leq 2^{cs(n)}$ for some $c > 0$. □

Theorem 1(ii). For any storage type X and $r \geq 1$, $2ArH\text{-}P(X) = ASPACE(n^r)\text{-}X$, and $2A1H\text{-}P(X) = 1A\text{-}P(X) = ASPACE(n)\text{-}X$.

Proof. Similar. In the $\subseteq$ - direction, there are cn^r worktape configurations of M; $z_0 \to \{z_1,\ldots,z_t\}$ can be stored by M' in an array A of length cn^r, where A[i] indicates whether the i-th configuration occurs left (and right) of the arrow. Note that in [CKS] a proof of $ASPACE(n) \subseteq 1A\text{-}P$ is given. □

The second result shows the equivalence of alternation and auxiliary pushdown automata.

Theorem 2. For any storage type X and $s(n) \geq \log n$, $NSPACE(s(n))\text{-}P(X) = ASPACE(s(n))\text{-}X$.

Proof. $\supseteq$: As in the proof of Theorem 1 of [Ruz] where it is shown that $NSPACE(s(n))\text{-}P = ASPACE(s(n))$.

$\subseteq$: Special case of the $\subseteq$-part of the proof of Theorem 1(i). We only have to find out whether $r \to \{z\}$, i.e., to store two surface ID's in space 2s(n). □

Together with Cook's result [Coo] that $NSPACE(s(n))\text{-}P = \cup\, DTIME(2^{cs(n)})$, Theorem 1(i,ii) and 2 suffice to prove the results in the Table of Fig. 1 (by induction), except for the nondeterministic r-head P^k-automata. This case will now be treated, together with iterated stack automata.

The storage type SA(X), stack of X-configurations, can be defined as P(X) with the additional ability of reading in the stack. Similarly one can define NSA(X), nested stack of X-configurations: the automaton can in addition create and destruct new stacks. These storage types are all just as for the classical (nested) stack automata [GGH, Aho], except that each stack square contains both a symbol and an X-configuration (which may be tested when reading in the stack). Iterated stack automata and iterated nested stack automata can be defined as for pushdowns: SA^k and NSA^k denote $SA^k(FA)$ and $NSA^k(FA)$, respectively. Elementary equivalence results, as for P(X), can also be shown for SA(X) and NSA(X); note that, trivially, $P(X) \leq SA(X) \leq NSA(X)$.

The 1-way nested stack automaton and the 1-way 2-iterated pushdown automaton are very close (both recognize the indexed languages); in fact it is quite straightforward to show the following theorem.

Theorem 3. For every storage type X, NSA(X) and P(P(X)) are equivalent storage types. □

This shows that the Table for NSA^k (iterated nested stack automata) can be obtained from the Table of Fig.1 by replacing every k by 2k. Since the simulation of P(P(X)) by NSA(X), although straightforward (a nested stack is just a space-efficient representation of a pushdown of pushdowns: if two consecutive pushdowns have a common prefix, then only the remaining part of the second pushdown is repeated, nested in the first pushdown, after the prefix), involves a number of technical details, we only show the simulation of NSA(X) by P(P(X)) which suffices for the purposes of this paper (in fact, if NSA were not considered, it would suffice to simulate SA(X) by P(P(X))).

Lemma 1. For every storage type X, $NSA(X) \leq P(P(X))$.

Proof. It is rather easy to see that $SA(X) \leq P(P(X))$. The stack in Fig.2(a) is simulated by the pushdown of pushdowns in Fig.2(b). Each pushdown is a prefix of its left neighbors. A move down in the

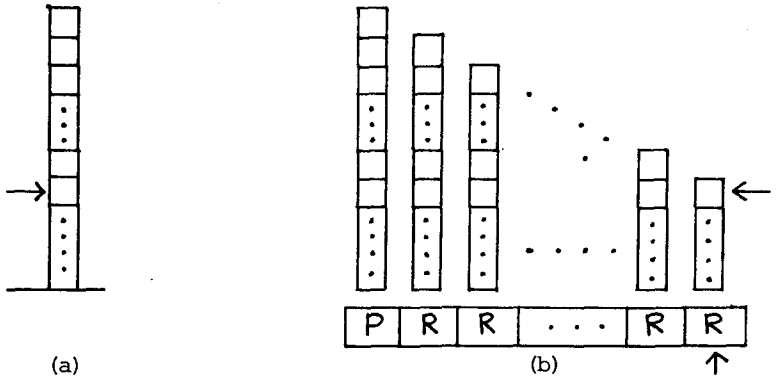

Fig.2. Simulation of a stack by a 2-iterated pushdown. Each blank square is a pair <g,c>, where g is a stack symbol and c is an X-configuration. Each square with P or R, together with the pushdown above it, is an element of the "main" pushdown in (b).

stack is simulated by push(R,pop), provided the pop does not empty the pushdown (this detail should be taken care of); a move up is simulated by pop, provided top = R (reading mode); a push(g,f) by stay(P,push(g,f)), and a pop by stay(P,pop), provided top = P (push/pop mode); a test stacksymbol = g? is simulated by test(top = g?), empty? by test(empty?), provided top = P, and test(t) by test(test(t)).

We now extend this to show that $NSA(X) \leq P(PX))$. In addition to the tests and operations of SA(X), which are simulated as above, NSA(X) has operations create(g,f), create(g), and destruct (with $g \in G$, $f \in F$); create(g,f), and create(g), creates below the current square a new, nested, stack of length 1 with stack symbol g and configuration f(c), c respectively, where c is the configuration of the current square; destruct does the inverse operation. The operation create(g,f) is simulated by push(P); stay(P,stay(#g,f)),

where # indicates that this is the bottom of the stack, and similarly for create(g). The operation destruct is simulated by a pop, provided top = P and test(top $\in$ #G). Note that the function h of this simulation is one-to-one and h^{-1} is obtained as follows. Each reachable configuration c of NSA(X) can be built up in a unique way by push, move down, and create operations only; by executing the corresponding simulating programs of P(P(X)), as indicated above, in the same order, one arrives at the corresponding configuration $h^{-1}(c)$ of P(P(X)); note that the number of symbols P in the main pushdown of $h^{-1}(c)$ equals the number of stacks of c. □

We now show that in all cases mentioned in Fig.1 (except of course the 1-way nondeterministic case) the P(P(X))-automata can even be simulated by the corresponding SA(X)-automata. Together with Lemma 1 this shows that in these cases the P(P(X)), NSA(X), and SA(X) automata all have the same power (this gives other proofs of some results of [Bee] and extends them to the alternating case).

Theorem 4. For any storage type X, $s(n) \geq \log n$, and $r \geq 1$,
(i) NSPACE(s(n))-SA(X) = NSPACE(s(n))-P(P(X)), and
(ii) 2NrH-SA(X) = 2NrH-P(P(X)) = ASPACE(n^{2r})-X.

Proof. (i) By Theorems 1 and 2, NSPACE(s(n))-P(P(X)) = $\cup$ NSPACE($2^{cs(n)}$)-P(X). Hence, by Lemma 1, it suffices to show that NSPACE($2^{cs(n)}$)-P(X) $\subseteq$ NSPACE(s(n))-SA(X).
This is easy: the worktape configurations of the pushdown automaton can be put on the stack, and compared in space s(n).

(ii) Similar to the second part of the proof of Theorem 2, it can be shown that 2NrH-P(X) $\subseteq$ 2A(2r)H-X. Hence 2NrH-P(P(X)) $\subseteq$ 2A(2r)H-P(X), which, by Theorems 1(ii) and 2, equals ASPACE(n^{2r})-X = NSPACE(n^{2r})-P(X). Again, as in (i) above, this last class is included in 2NrH-SA(X). See [HU1] and Lemma 3 of [Coo] to understand that one head suffices to copy worktape configurations of length n^2; similarly r heads can do length $(n^r)^2$. □

Theorem 4(ii) shows that 2NrH-P^k is as given in the Table of Fig.1; the proof of the Table is now completed. Next we consider alternating SA and NESA (non-erasing stack automata, i.e., without pop operation). We note that when NESA is dropped from the next theorem, its proof is entirely the same as that of Theorem 4(i). The result, as it is now, is "taken from" Theorem 5.1 of [LLS].

Theorem 5. For any storage type X, $s(n) \geq \log n$, and $r \geq 1$,
(i) ASPACE(s(n))-NESA(X) = ASPACE(s(n))-SA(X) = ASPACE(s(n))-P(P(X)).
(ii) 2ArH-NESA(X) = 2ArH-SA(X) = 2ArH-P(P(X)).
(iii) 1A-NESA(X) = 1A-SA(X) = 1A-P(P(X)).

Proof. (i) From Theorem 1(i) we know that ASPACE(s(n))-P(P(X)) = $\cup$ ASPACE($2^{2^{cs(n)}}$)-X. A proof that ASPACE($2^{2^{cs(n)}}$)-X $\subseteq$ ASPACE(s(n))-NESA(X) is entirely analogous to that of Lemma 5.2 of [LLS], where X = FA. By Lemma 1, this proves (i). Note that we do not need a proof of the analogue of Lemma 5.3 of [LLS] : the inclusion ASPACE(s(n))-SA(X) $\subseteq$ $\cup$ ASPACE($2^{cs(n)}$)-P(X) follows from Lemma 1 and Theorem 1(i).

(ii, iii). Similar, using Theorem 1(ii). □

Theorems 4 and 5 imply that SA(X) is equivalent to P(P(X)), and so to NSA(X), for all types of automata in the Table of Fig.1. Hence the Table for SA^k (iterated stack automata) can be obtained by replacing k by 2k.

4. Applications to 1-way automata

Characterizations of 2-way or multi-head X-automata can be used to obtain hard problems concerning 1-way X-automata (cf., e.g., [Hun,En2]), and to prove proper inclusions between classes of 1-way automaton

languages (cf.[En2]). We quote some results from [En2]. First a slightly improved version of Theorem 22 of [En2]; let 2GSM be the class of all nondeterministic 2-way gsm-mappings.

Proposition 1. For any storage type X, $2N1H\text{-}X = 2GSM^{-1}(1N\text{-}X) = 2GSM^{-1}(1D\text{-}X)$.

Proof. See [En2]; note that $L_{F'}$, defined there as the set of all successful instruction sequences of X, can be accepted by a deterministic 1-way X-automaton. □

A similar result holds for 2NMH-X and nondeterministic log-space transducers.

This proposition can be used to translate proper inclusions between classes of 2-way automata down to proper inclusions between classes of 1-way automata.

Theorem 6. The diagram in Fig.3 is correct (i.e., ascending lines denote proper inclusions; classes that are not connected by ascending lines are incomparable). Informally: $\{1D\text{-}P^k\}$ is a "small" hierarchy inside the hierarchy $\{1N\text{-}P^k\}$.

Proof. By the time-complexity hierarchy (see, e.g., [HU2]) and the Table of Fig.1, the classes $2N1H\text{-}P^k$ form a proper hierarchy (i.e., $2N1H\text{-}P^k \subsetneq 2N1H\text{-}P^{k+1}$ for all k). Hence, by Proposition 1, $1D\text{-}P^{k+1}$ is not included in $1N\text{-}P^k$. It remains to show that 1N-P is not included in $\cup\{1D\text{-}P^k \mid k \geq 1\}$, i.e., that there is a context-free language that cannot be recognized by any 1-way deterministic P^k automaton. In [En3] the more general fact is shown that there is a fixed linear context-free language L_0 (with a grammar of 12 productions) such that for every storage type X, if $1A\text{-}X \subsetneq RE$ (or $1N\text{-}P(X) \subsetneq RE$), then $L_0 \notin 1D\text{-}X$. By the Table, $1A\text{-}P^k \subsetneq RE$. □

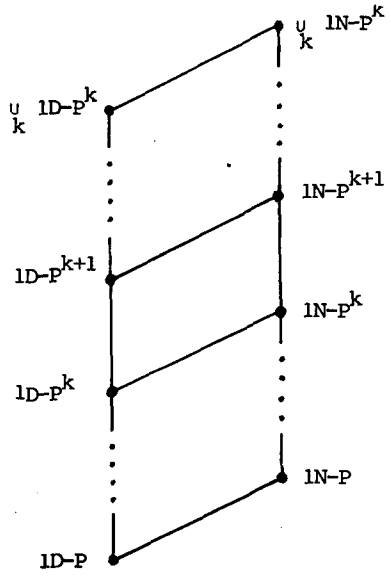

Fig.3. The hierarchies of 1-way iterated pushdown languages.

In [DGo] it is shown that $1N\text{-}P^k = k\text{-}OI$, where k-OI is the class of k-level OI-languages (the k-th class in the OI-hierarchy). Damm has shown in [Dam] that $k\text{-}OI \subsetneq (2k+1)\text{-}OI$ by the method of rational index. Theorem 6 shows that in fact $k\text{-}OI \subsetneq (k+1)\text{-}OI$. A concrete language in $1N\text{-}P^{k+1}$ but not in $1N\text{-}P^k$ is of course the usual ([Gin]) generator of the full principal AFL $1N\text{-}P^{k+1}$. More easily describable languages in (2k+1)-OI but not in k-OI are given in [Dam]; essentially these languages are of the form $\{a^m \mid m = \exp_{2k}(n)$ for some $n > 0\}$, and one may wonder whether these counter-examples can also be obtained from our complexity considerations.

We mention that another "1-way result" that can be obtained in a way similar to Theorem 6, is : if K is a full semi-AFL included in $\cup\{1N\text{-}P^k \mid k \geq 1\}$, then $2GSM^n(K) \subsetneq 2GSM^{n+1}(K)$ for all $n \geq 0$; for a proof, see Theorems 14 and 49 of [En2].

Next we quote Theorem 50 of [En2]. A storage type is finitely encoded if it has a finite number of tests and operations ([Gin]).

Proposition 2. Let X be a finitely encoded storage type. The non-emptiness problem for 1N-X automata is log-space complete in the class of languages accepted by 2NMH-X automata. □

The proposition also holds for the general membership problem of 1N-X automata, and for the non-

emptiness problem of 1D-X automata. Since we may assume that iterated pushdown automata are finitely encoded (take G = {0,1}), we obtain immediately from the Table of Fig.1 that for $k \geq 1$ the emptiness problem for 1N-P^k automata (and for 1D-P^k automata) is log-space complete in DTIME($\exp_{k-1}$(poly)), and similarly for the general membership problem of 1N-P^k automata. This can be sharpened to

Theorem 7. For $k \geq 2$,

(i) the emptiness problem for 1N-P^k automata is in $\cup$ DTIME($\exp_{k-1}(cn^2)$);

(ii) for every $e > 0$, the emptiness problem for 1D-P^k automata is not in $\cup$ DTIME($\exp_{k-1}(cn^{2-e})$).

Proof. (i) This non-emptiness problem can be recognized by a 2N1H-P^k automaton (the main pushdown can be used to store the encoded state of the 1N-P^k automaton, cf. Proposition 4.5 of [Hun]). (ii) The proof of Proposition 2 shows that the log-space reduction of 2N1H-P^k languages can be done in time O(nlogn). □

The decidability of the emptiness-problem for 1N-P^k languages is shown in [Dam] (using algebraic methods, for the k-level OI grammars).

Theorem 7 gives some natural automaton-theoretic decision problems of high intractability. By combining them, a nonelementary problem is even obtained (cf. the result of Meyer and Stockmeyer in Section 11.4 of [AHU]). Note that $\cup\{2NMH\text{-}P^k \mid k \geq 1\}$ is the class of elementary problems. Let an iterated pushdown automaton be a P^k-automaton for any $k \geq 1$.

Corollary 1. The emptiness problem for 1-way (non)deterministic iterated pushdown automata is decidable but not elementary. □

We finally note that the results of this section can be extended to 1-way SA^k automata; e.g., there is a 1D-SA^{k+1} language not in 1N-P^{2k+1}.

5. Conclusion

We have left some questions unanswered. First we note that it can be shown from the hierarchy result in Section 4 that the k-level monadic program schemes (k = 1 : recursive procedures without parameters, k = 2 : recursive procedures with parameters and result of type "procedure without parameters", see [Enl]) form a proper hierarchy. (A similar result, for another type of program schemes, is given in [Dam]). In fact, programming the multi-head finite automaton by the k-level monadic program schemes gives the 2NMH-P^k languages. Since these schemes are close to the typed λ-calculus with fixed-point operator (cf.[Dam]), one may wonder what the relationship is to the result of [FLO] that every function representable in the (simply) typed λ-calculus is bounded by an elementary function.

Other questions are:

(1) Are the deterministic and nondeterministic multi-head P^k automata equivalent? For k = 1 this is shown in [Coo]. What is the power (in terms of complexity classes) of deterministic r-head P^k-automata? These questions could not be answered in a uniform manner (as in Theorems 1 and 2) because auxiliary X-automata may not be always-halting.

(2) Are the 1N-P^k languages context-sensitive? This is mentioned in [Gre] as having been shown by Aho and Ullman. Is 1N-SA^k properly included in 1N-P^{2k}? Note that, of course, $\cup\{1N\text{-}SA^k \mid k \geq 1\} = \cup\{1N\text{-}P^k \mid k \geq 1\} = \cup\{1N\text{-}NSA^k \mid k \geq 1\}$.

(3) Is it possible to find automaton-theoretic characterizations of complexity classes larger than the class $\cup\{DTIME(\exp_k(n)) \mid k \geq 1\}$ of elementary languages? One idea is to define an iterated pushdown automaton with an arbitrary number of levels of pushdowns: it should have operations to go up and down in level. Unfortunately, this automaton would be able to accept all recursively enumerable languages (each

level would simulate one square of a Turing machine tape). One could think of bounding the number of levels in terms of the length of the input.

(4) In general, which complexity classes can be characterized by 2-way or multi-head automata, and, vice versa, for which storage types X are 2N1H-X and 2NMH-X complexity classes?

References

[Aho] A.V. Aho; Nested stack automata; JACM 16 (1969), 383-406.

[AHU] A.V. Aho, J.E. Hopcroft, J.D. Ullman; "The design and analysis of computer algorithms"; Addison-Wesley, Reading, Mass., 1974.

[Bee] C. Beeri; Two-way nested stack automata are equivalent to two-way stack automata; JCSS 10 (1975), 317-339.

[CKS] A.K. Chandra, D.C. Kozen, L.J. Stockmeyer; Alternation; JACM 28(1981), 114-133.

[Coo] S.A. Cook; Characterizations of pushdown machines in terms of time-bounded computers; JACM 18 (1971), 4-18.

[Dam] W. Damm; The IO- and OI-hierarchies; TCS 20 (1982), 95-207.

[DGo] W. Damm, A. Goerdt; An automata-theoretic characterization of the OI-hierarchy; Proc. of the 9-th ICALP, Aarhus, 1982, pp. 141-153.

[En1] J. Engelfriet; Simple program schemes and formal languages; Lecture Notes in Computer Science 20, Springer-Verlag, Berlin, 1974.

[En2] J. Engelfriet; Two-way automata and checking automata; in: Foundations of Computer Science III (eds. J.W. de Bakker, J. van Leeuwen), Mathematical Centre Tracts 108, pp. 1-69, Amsterdam, 1979.

[En3] J. Engelfriet; A note on 1-way deterministic recognition of context-free languages; manuscript, 1983.

[FLO] S. Fortune, D. Leivant, M. O'Donnell; The expressiveness of simple and second order type structures; IBM Report RC 8542, 1980.

[Gin] S. Ginsburg; "Algebraic and automata-theoretic properties of formal languages"; North-Holland, Amsterdam, 1975.

[GGH] S. Ginsburg, S.A. Greibach, M.A. Harrison; Stack automata and compiling; JACM 14 (1967), 389-418.

[Gol] J. Goldstine; Automata with data storage; Proc. of A Conference on Theoretical Computer Science, Waterloo, 1977, pp. 239-246.

[Gre] S.A. Greibach; Full AFLs and nested iterated substitution; Inf. and Control 16 (1970), 7-35.

[Hu1] J.E. Hopcroft, J.D. Ullman; Nonerasing stack automata; JCSS 1 (1967), 166-186.

[Hu2] J.E. Hopcroft, J.D. Ullman; "Introduction to Automata theory, Languages, and Computation"; Addison-Wesley, Reading, Mass., 1979.

[Hun] H.B. Hunt III; On the complexity of finite, pushdown, and stack automata; Math. Syst. Theory 10 (1976), 33-52.

[Iba] O.H. Ibarra; Characterizations of some tape and time complexity classes of Turing machines in terms of multi-head and auxiliary stack automata; JCSS 5 (1971), 88-117.

[LLS] R.E. Ladner, R.J. Lipton, L.J. Stockmeyer; Alternating pushdown and stack automata; IBM Report RC 9415, 1982; see also Proc. 19th FOCS, pp.92-106, 1978.

[Ma1] A.N. Maslov; The hierarchy of indexed languages of an arbitrary level; Soviet Math. Dokl. 15 (1974), 1170-1174.

[Ma2] A.N. Maslov; Multi-level stack automata; Probl. of Inf. Transm. 12 (1976), 38-43.

[PDS] R. Parchmann, J. Duske, J. Specht; On deterministic indexed languages; Inf. and Control 45 (1980), 48-67.

[Ruz] W.L. Ruzzo; Tree-size bounded alternation; JCSS 21 (1980), 218-235.

[Sco] D. Scott; Some definitional suggestions for automata theory; JCSS 1 (1967), 187-212.