On the Power of Bounded Concurrency II: The Pushdown Automata Level †

Tirza Hirst

Dept. of Mathematics & Computer Science
Bar-Ilan University, Ramat Gan, Israel

and

David Harel‡

Dept. of Applied Mathematics & Computer Science
The Weizmann Institute of Science, Rehovot 76100, Israel

Abstract.

This is the second in a series of papers on the inherent power of bounded cooperative concurrency, whereby an automaton can be in some bounded number of states that cooperate in accepting the input. In this paper we deal with the level of pushdown automata. We are interested in differencies in power of expression and in discrepencies in succinctness between variants of pda's that incorporate nondeterminism, pure parallelism and bounded cooperative concurrency. In particular, our results provide further evidence for the general observation that the latter feature provides inherent exponential power, in both upper and lower bound senses, regardless of whether or not the two former features are also present. While we use the language of statecharts to capture these features, our results are extremely robust, and hold also for bounded versions of virtually all other concurrent languages.

1. Introduction

Classical models of computation, such as Turing machines and various kinds of automata, have been enriched with existential and universal branching to capture parallelism. However, unlike the constructs used in the study of real distributed processes and protocols, in these types of branching no cooperation takes place between the spawned processes, except when time comes to decide whether the input should be accepted. In Turing machines and pushdown automata, for example, this fact manifests itself in the totally separate tapes or pushdown stacks that are assumed to be generated whenever branching (of either kind) takes place. Thus, branching essentially produces separate computations, the results of which

†This paper is based on part of the M.Sc. thesis of the first-listed author [Hi], supervised by the second-listed author.

‡This author's research was supported in part by a grant from the Gutwirth Foundation.

are later combined to form the joint result. It would appear that in order to capture real-world concurrency we would want to allow a mechanism to be in more than one state at a time during a single computation, and to enable these states to cooperate in achieving a common goal. This approach, which one might call *cooperative concurrency*, is the dominating one in research on distributed systems, and not the noncooperative concurrency of pure branching. Moreover, in the real world, the number of processors available for simultaneous work is bounded and cannot be assumed to grow as the size of the input grows. One machine of fixed size must solve the algorithmic problem in question for *all* inputs. In contrast, existential and universal branching are unbounded — new processes can be spawned without limit as the computation proceeeds. In the sequel, we shall use E, A and C, respectively, to denote existential branching (nondeterminism), universal branching (∀-parallelism), and bounded cooperative concurrency (or simply *bounded concurrency* for short).

In [DH], we have investigated the inherent power of bounded cooperative concurrency in the realm of finite automata, over both finite and infinite words. The criterion for comparison was *succinctness*. One finding that recurs in all the cases considered therein is that the C feature gives rise to inherently exponential differences in power, in both upper and lower bound senses, regardless of whether E and A are also available or not.

To help describe the present work we survey some of the results of [DH]. (These results, as well as those of the present paper and some of those of [HRV], are motivated and summarized in a uniform fashion in [Ha3].) It is well-known that NFAs are exponentially more succinct than DFAs, in the following upper and lower bound senses (see, e.g., [MF]):

- Any NFA can be simulated by a DFA with at most an exponential growth in size.
- There is a (uniform) family of regular sets, L_n, for $n > 0$, such that each L_n is accepted by an NFA of size $O(n)$ but the smallest DFA accepting it is of size at least 2^n.

The same is true of ∀-automata, namely, the dual machines, in which all branching is universal. It is also true that AFAs, i.e., those that combine *both* types of branching, are exponentially more succinct than both NFAs and ∀-automata, and indeed are *double*-exponentially more succinct than DFAs (see [CKS]). These results also hold in both the upper and lower bound senses described. Thus, in this framework, E and A are exponentially powerful features, independently of each other (that is, whether or not the other is present), and, moreover, their power is additive: the two combined are double-exponentially more succinct than none. Taking a solid arrow to depict the presence of an upper and lower bound of one exponential, the bottom horizontal lines of Fig. 1 summarize these known facts.†

†In the figure, transitivity is assumed too; hence, the line labelled 'two-exponentials' that would

The idea in [DH] was to investigate the effect of introducing the C feature, via the use of statecharts [Ha2] as an extension of finite automata. The first set of results therein establishes the vertical lines and the top horizontal lines of Fig. 1,† as well as the implicit compound lines. Among other things, these include exponential upper and lower bounds for simulating nondeterministic statecharts on NFAs, *double*-exponential bounds for simulating them on DFAs, and *triple*-exponential upper and lower bounds for simulating *alternating* statecharts on DFAs. Thus, these parts of Fig. 1 show that bounded concurrency represents a third, separate, exponentially powerful feature. It is independent of conventional nondeterminism and parallelism, since the savings remain intact in the face of any combination of A and E, and is also additive with respect to them, by virtue of the double- and triple-exponential bounds along the compound lines.

Stronger results are then obtained in [DH], considering the question of how C compares with A and E themselves. These results are summarized by the remaining lines of Fig. 1. Each of the four diagonal arrows denotes exponential upper and lower bounds for the simulation in the downward direction and polynomial (actually, linear) bounds for the upward direction. In particular, nondeterministic statecharts are shown to be exponentially more succinct than AFAs, and the same holds when nonterminism is absent from both. Finally, the line between C and (E,A) represents upper and lower exponential bounds in *both* directions, meaning that alternation and bounded concurrency can be simulated by each other with at most an exponential growth in size, and that, in general, neither of these gaps can be reduced. These results show that while bounded concurrency is actually more powerful than each of $\forall$-parallelism or nondeterminism taken alone, it is incompatible, in terms of expressive power, with the combination of both.‡

We should add that all these results are extremely robust, in that they are insensitive to the particular mechanism of cooperation adopted. In many of the lower bound proofs the main use of cooperation is merely to pass along carries in the process of counting in binary — an extremely simple form of cooperation. Consequently, the results do not depend on the choice of statecharts as the language for describing computations; they could have been phrased for bounded versions of standard models such as Petri nets [Re], CSP [Ho], CCS [Mi], or the concurrent versions of standard programming languages such as Pascal or Prolog. As the reader will be able to see quite easily, this robustness carries over to the present work too.

In this paper we report on the results of a similar investigation, in which the basic mechanism is that of a pushdown automaton. Here, however, there are differencies in power of expression: for example, E-machines (i.e., nondeterminis-

---

lead from (E,A) to $\emptyset$ is omitted for clarity, despite the fact that it does not follow *a priori*.

†The single dot along these lines denotes one exponential gaps.

‡Similar results are obtained in [DH] for the case of infinite words, and in [Hi] for the case of finite languages and for the case of finite words over a one-letter alphabet.

tic pda's) accept the context-free languages, whereas, of course, $\emptyset$-machines (i.e., dpda's) accept a strictly smaller set. Accordingly, in Section 3 we consider power of expression, and show that whatever combination of the E and A features is considered, the C feature adds nothing to the class of languages accepted. We then turn to succinctness, with the goal of finding out exactly how powerful the three features are when used to accept sets of languages that are common to all variants of the machines. We consider deterministic context-free languages and regular languages in Section 4 and finite languages in Section 5.

When viewed together, our succintness results further confirm the general conclusion reached in [DH, Ha3], namely, that bounded concurrency provides inherent exponential power, regardless of whether or not E and/or A are also present. The pushdown automata case is particularly interesting since, in contrast to finite automata, where the E and A features are also of exponential power (as illustrated in Fig. 1), here E and A give rise to *unlimited* differences in succinctness, whereas the C feature stands fast as being inherently exponential in all cases. In [HRV] we provide yet more evidence for this general conclusion, by showing that the C feature causes exponential differences in the time-complexity of deciding validity in a number of logics of programs.

2. Preliminaries

The *size* of context-free grammars and pda's are defined in the usual way (see, for example, [Hr]), as are acceptance by empty stack or final state. $T(A)$ is the set accepted by the pda A by final state, $N(A)$ is the set accepted by empty stack, and $L(A)$ is the set accepted by both final state and empty store simultaneously. There are polynomial translations between these acceptence criteria, so that in the sequel we shall blur the distinctions between them. Often, the translations are in fact linear. Moreover, to simplify the exposition of our results we shall emphasis in all places only the difference between polynomial gaps and exponential or higher gaps, although in many cases the polynomials in question are really linear. We let $L(G)$ be the language defined by the grammar G. The following results are adapted from [Hr].

Definition. A pda $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is *moderate* if $(q', \alpha) \in \delta(q, a, Z)$ implies $|\alpha| \leq 2$.

Proposition 1. There is a polynomial p, such that for each context-free grammar G of size n there is a pda A of size $p(n)$, such that $L(G) = N(A)$.

Proposition 2. There is a polynomial p, such that for each (deterministic) pda A of size n there is a moderate (deterministic) pda B of size $p(n)$ that is equivalent to A in whichever form of acceptance is being used.

Proposition 3. There is a polynomial p, such that for each pda A of size n there is a context-free grammar G of size $p(n)$ with $L(G) = N(A)$.

Proof. Let A be a pda of size n. Simulate A by a moderate-pda A' of size polynomial in n, and then simulate A' by a polynomial-sized CF-grammar G, as shown, e.g., in [Hr, pp. 151–152]. $\triangle$

Proposition 4. There is a polynomial p, such that for each context-free grammar G of size n, there is a grammar G' of size $p(n)$ in Chomsky normal form (CNF), such that $L(G') = L(G)$.

We now consider extensions of the basic model of pushdown automata, by adding the C feature. This can be done in many ways without influencing the results of the paper. For example, we could have added pushdowns to bounded-token Petri nets [Re], to a bounded version of finite-state CSP or CCS programs [Ho, Mi], or to bounded versions of any of the standard kinds of parallel programming languages. As in [DH], we too shall use the statecharts of [Ha1] as the basis of our extension. The details of the language can be found. e.g., in [Ha1, Ha2], and the way the E and A features are combined with the bounded concurrency of statecharts is described in [DH]. Here we consider a version of statecharts that allows the standard operations on a pushdown stack to be present along the transitions between states. By convention, if more than one transition is to be taken in one step (by virtue of the machine being in more than one state at once) and if the stack operations called for by these transitions are contradictory (i.e., they ask for different sequences of symbols to be pushed), we agree that the machine immediately stops and rejects the input. Thus, in the sequel, a C-pda is a deterministic pushdown statechart, an E-pda is a usual nondeterministic pda, an (E,A)-pda is an alternating pushdown automaton (see [LLS]), a $\emptyset$-pda is a dpda, an (E,C)-pda is a nondeterministic pushdown statechart, an (E,A,C)-pda is an alternating pushdown statechart, etc. The size of these statecharts is defined in the obvious way (number of states + number of transitions + sum of lengths of the labels on transitions (including the stack operations)).

Let ξ be any subset of {E,A,C}. We denote by ξ-PDA the class of pda's enriched with the ξ features, and by ξ-LAN the set of languages accepted by the machines in the class ξ-PDA .

Proposition 5. There is a polynomial p, such that for any (E,A)-pda A of size n there is an (E,A)-pda B of size $p(n)$ with $T(A) = \Sigma^* - T(B)$. Moreover, A is an E-pda iff B is an A-pda.

Sketch of proof. Let A be an (E,A)-pda of size n. B is constructed by exchanging the accepting states with the non-accepting states, and the $\forall$-states with the $\exists$-states. $\triangle$

Let C_1, C_2 be two classes of machines.

Definition. We write $C_1 \dashrightarrow C_2$ (or $C_1 \stackrel{\cdots}{\longrightarrow} C_2$, $C_1 \stackrel{\cdots}{\dashrightarrow} C_2$, respectively), if there is a polynomial p, such that for any $M_1 \in C_1$ of size n that accepts a deterministic language, there is an equivalent $M_2 \in C_2$ of size no more than $2^{p(n)}$ (or $2^{2^{p(n)}}$, $2^{2^{2^{p(n)}}}$, respectively).

Definition. We write $C_1 \xrightarrow{\cdot} C_2$ (or $C_1 \xrightarrow{\cdot\cdot} C_2$, $C_1 \xrightarrow{\cdot\cdot\cdot} C_2$, respectively) if there is a family of deterministic languages L_n, for $n > 0$, and a polynomial p, such that L_n is accepted by some $M_1 \in C_1$ of size $p(f(n))$ for some function f, but the smallest $M_2 \in C_2$ accepting it is at least of size $2^{f(n)}$ (or $2^{2^{f(n)}}$, $2^{2^{2^{f(n)}}}$, respectively).

Definition. We write $C_1 \xrightarrow[\star]{} C_2$ if for any recursive function g, there is a family of deterministic languages L_n, for $n > 0$, such that, for each n, L_n is accepted by some $M_1 \in C_1$ of size $O(n)$, but the smallest $M_2 \in C_2$ accepting it is at least of size $g(n)$.

When a small f is added as a subscript to the arrows in these definitions, they are to be considered as applying to finite languages, rather than to deterministic ones.

3. Results on Expressive Power

Proposition 6.

$$\begin{aligned} \emptyset\text{-LAN} &= \text{C-LAN} = \text{determinstic CFLs;} \\ \text{E-LAN} &= \text{(E,C)-LAN} = \text{CFLs;} \\ \text{A-LAN} &= \text{(A,C)-LAN} = 2^{\Sigma^*} - \text{CFLs;} \\ \text{(E,A)-LAN} &= \text{(E,A,C)-LAN} = \text{DEXPTIME.} \end{aligned}$$

Proof. The C feature adds nothing to the expressive power of any of the E and/or A variants of pda's, as shown in Prop. 10 of Section 4. The characterization of DEXPTIME appears in [CKS, LLS]. $\triangle$

Proposition 7. $\emptyset$-LAN $\subset$ (E-LAN $\cap$ A-LAN).

Proof. Clearly, $\emptyset$-LAN $\subseteq$ (E-LAN $\cap$ A-LAN). To prove the inequality we exhibit an inherently nondeterministic CFL L such that $\overline{L} = \Sigma^* - L$ is also a CFL. Thus, by Prop. 5, L is accepted by an A-pda too.

Let L be the inherently nondeterministic language $\{ww^R \mid w \in \{a,b\}^*\}$ (see [Hr, p. 390]). To show that $\overline{L} = \Sigma^* - L$ is also a CFL, consider the grammar:

$$\begin{aligned} S &\to aSa \mid bSb \mid aAb \mid bAa \mid a \mid b \\ A &\to aA \mid bA \mid \Lambda \end{aligned} \qquad \triangle$$

Proposition 8. (E-LAN $\cup$ A-LAN) $\subset$ (E,A)-LAN .

Proof. Clearly, (E-LAN $\cup$ A-LAN) $\subseteq$ (E,A)-LAN . We exhibit a language L that is accepted by an (E,A)-pda, but is not a CFL or the complement of one. Define $L = \{a^{2^n} \mid n > 0\}$. The (E,A)-pda for L is illustrated in Fig. 2. It pushes

the first half of the input word onto the stack by guessing the middle. It then checks, using a ∀-state, if the length of the rest of the word is equal to the length of the stack, and repeats the whole process for the second half, recursively.

It is not too difficult to see that neither L nor $\overline{L}$ is a CFL, by using simple pumping arguments. △

Turning to one-letter alphabets, every context-free language $L \subseteq \{a\}^*$ is regular, by Parikh's theorem [Pa]. From Prop. 5 and the closure of regular sets under completion, it follows that every A-LAN language $L \subseteq \{a\}^*$ is also regular. (E,A)-LAN , on the other hand, contains one-letter languages that are non-regular.

Proposition 9. There is a non-regular language (which is therefore non-context-free) $L \subseteq \{a\}^*$ that is accepted by an (E,A)-pda.

Proof. As in the previous proof, let $L = \{a^{2^n} \mid n > 0\}$. △

4. Bounds for Deterministic and Regular Languages

The results of this section are illustrated in Fig. 3. The text deals with deterministic languages, but all the results hold for regular languages too, since the upper bounds obviously apply to regular languages too, and the example languages used for the lower bounds are all, in fact, regular. We first show that C can be eliminated at the expense of at most an exponential blowup.

Proposition 10. Let ξ be any subset of {E,A}. Then

$$(\xi,\text{C})\text{-PDA} \longrightarrow \xi\text{-PDA}\,.$$

Proof. Let M be a (ξ,C)-pda of size n. The proof uses a simple product construction, as in [DH]. A configuration of a conventional pda contains the input word, the state, the position of the head, and the contents of the stack. A configuration of a (ξ,C)-pda differs in that instead of a single state it may be thought of as containing a subset of atomic-states.

Accordingly, the states of the equivalent ξ-pda M' are subsets of the atomic-states of M (numbering at most 2^n), whose transitions are defined according to those of M in the following fashion. The configuration Q_2' follows Q_1' in M' iff Q_2 follows Q_1 in M, where Q_1' and Q_2' contain the same input word, the same head position and the same stack as Q_1, and Q_2, respectively, and the states of Q_1' and Q_2' represent the same subsets of atomic-states as Q_1 and Q_2, respectively. The start state of M' and its accepting states are defined in a straightforward way as those representing the corresponding configurations in M. △

Corollary 11. Let ξ be E or A. Then

$$\begin{aligned} \text{C-PDA} &\dashrightarrow \xi\text{-PDA}\,, \\ \text{C-PDA} &\dashrightarrow \text{(E,A)-PDA}\,, \\ (\xi,\text{C})\text{-PDA} &\dashrightarrow \text{(E,A)-PDA}\,. \end{aligned}$$

In contrast to the removal of the C feature, E and A cannot be removed with any limited blowup. We prove this even for the replacement of one by the other:

Proposition 12.

$$\text{A-PDA} \xrightarrow[\star]{} \text{E-PDA} \qquad \text{and}$$

$$\text{E-PDA} \xrightarrow[\star]{} \text{A-PDA}\,.$$

Proof. Let g be a recursive function, and let M be a Turing machine that starts on a tape containing 1^n and ends with a tape containing $1^{g(n)}$. Let the relevant computation sequence of M be $q_0 1^n \vdash \alpha_2 \vdash \cdots \vdash \alpha_k \vdash q_1 1^{g(n)}$, where q_1 is the machine's final state. Let L_n be the singleton $\{q_0 1^n \not\in \alpha_2^R \not\in \alpha_3 \cdots \alpha_k^R \not\in q_1 1^{g(n)} \not\in \not\in\}$, where α^R is the configuration α in reverse. (For simplicity we assume that k is even.) Then, an A-pda of size $O(\log n)$ accepts L_n as follows. It checks that each block follows the previous one according to the rules of M. This check is carried out using two $\forall$-branches to check the even numbered blocks in parallel with the odd numbered ones, and each block is pushed on the stack, with the adjacent block being checked against it for complience. This part of the A-pda is of constant size. In parallel, the machine verifies that there are n 1's in the first block with a $\emptyset$-pda component of size $O(\log n)$, as described in the proof of Prop. 14. (There we verify 2^n occurences of 1 with a $\emptyset$-pda of size $O(n)$.)

In contrast, the smallest CNF-grammar that generates L_n must contain at least $\log(g(n))$ variables. Otherwise, we could generate more words, by pumping the derivation-tree of the single $w \in L_n$. Thus, by Prop. 3, the size of the smallest E-pda accepting L_n must be polynomially related to $\log(g(n))$. Carrying out the construction for a function g' that is related to g in a similarly polynomial manner concludes the proof.

For the second case, let $\overline{L}_n = \Sigma^* - L_n$. The result then follows from Prop. 5. $\triangle$

Corollary 13. Let ξ be A or E, and let each of μ and ν be C or $\emptyset$. Then

$$(\text{E,A},\mu)\text{-PDA} \xrightarrow[\star]{} (\xi,\nu)\text{-PDA}\,,$$

$$(\xi,\mu)\text{-PDA} \xrightarrow[\star]{} (\{\text{E,A}\} - \xi\,,\ \nu)\text{-PDA}\,,$$

$$(\text{E,A})\text{-PDA} \xrightarrow[\star]{} \emptyset\text{-PDA}\,, \qquad \text{and}$$

$$\xi\text{-PDA} \xrightarrow[\star]{} \emptyset\text{-PDA}\,.$$

In order to establish the exponential lower bounds represented in the three front downward vertical lines of Fig. 3, it suffices to establish the exponential lower bounds for the two diagonals representing the simulation of a C-pda by an E-pda and an A-pda:

Proposition 14. Let ξ be E or A. Then

$$\text{C-PDA} \not\longrightarrow \xi\text{-PDA}\,.$$

Proof. Let $L_n = \{1^{2^n}\}$. There is a $\emptyset$-pda M that uses $n+1$ stack-symbols, $s_0, \ldots, s_n$, and accepts L_n. First, it pushes s_n onto the stack. Subsequently, if it sees s_i, $i \geq 1$, at the top of the stack it replaces it by $s_{i-1}s_{i-1}$. This continues until M sees s_0 at the top of the stack, at which time it pops s_0 and reads one character in the input-word. When the stack is empty, all 2^n characters of the input-word have been read.

We can now exhibit a C-pda of size $O(\log(n))$ that does the same. It uses only two stack-symbols a and b, encoding s_i by a^ib. Identifying a^ib at the top of the stack, popping it off and pushing two sequences of the form $a^{i-1}b$ instead, can be carried out using the ability of the statecharts to count up to n with $O(\log(n))$ states [DH].

In order to show that the smallest E-pda accepting L_n must be of size polynomially related to n (i.e., one of the two is polynomial in the other), it suffices to show that the smallest CNF-grammar generating it is at least of size n; see Prop. 3. Indeed, the smallest CNF grammar that generates L_n must contain at least n non-terminals. Otherwise, in the derivation-tree of the word 1^{2^n} there would be a path of length at least n, which would then have to contain two occurrences of some nonterminal. We could then apply a pumping argument to generate other words that are not in the language.

For the case of $\xi = A$, let $\overline{L}_n = \Sigma^* - L_n$. A C-pda accepting $\overline{L}_n$ is similar to the previous one for L_n. However, the size of the smallest A-pda accepting it[†] must be polynomially related to n, by Prop. 5. $\triangle$

Corollary 15. Let ξ be E, A or $\emptyset$. Then

$$(\xi\text{,C})\text{-PDA} \not\longrightarrow \xi\text{-PDA}\,.$$

We conjecture that the exponential lower bounds apply not only to the replacement of C by E or A, but also to the replacement of C by both E *and* A, as was proved in [Hi] for finite automata (see [DH]):

Conjecture 16. C-PDA $\not\longrightarrow$ (E,A)-PDA .

[†]This is an example of a case in which we blur the distinction between acceptance by empty stack and by final state. The differences have no effect on the results.

The example that prompted us to make this conjecture is the language

$$L_n = \{w\#w_1 \mid w \in \{0,1\}^n,\ w_1 \in \{0,1,\#\}^*,\ w \text{ appears in } w_1 \text{ exactly } 2^n \text{ times}\}$$

On the one hand there is a linear-sized C-pda accepting L_n. It keeps the first w in n orthogonal yes/no components for comparison with subsequent words, and uses a counter of size $O(n)$ to count the number of subwords equal to w. (Note that the stack is not used at all.) On the other hand, we conjecture that any (E,A)-pda accepting L_n must be of exponential size. Here is why. M has to compare and count the occurrences of w along a single path, and, since E and A have the effect of splitting the computation into separate paths, it would appear that these features cannot help in reducing the size. As to exploiting the stack, if M uses it for either comparing or counting, it appears that the other of these tasks cannot be supported at the same time by the stack unless 2^n stack-symbols or 2^n states are used.

If the conjecture is true, then obviously if ξ is E, A or (E,A), we also have:

$$(\xi, \text{C})\text{-PDA} \quad \dot{\longrightarrow} \quad \text{(E,A)-PDA}\,.$$

5. Bounds for Finite Languages

The results of this section are illustrated in Fig. 4.

The exponential upper bounds for simulating (C,ξ)-pda's by ξ-pda's clearly apply to finite languages too. However, nondeterminism can be eliminated with a double-exponential blowup, so that the gap between E and $\emptyset$ is not unlimited, as it was in the case of deterministic languages. To establish this we first show how to replace E by C with an exponential blowup[†]:

Proposition 17.

$$\text{E-PDA} \quad \dot{\longrightarrow}_f \quad \text{C-PDA}\,.$$

Proof. Let M be an E-pda of size n. Without loss of generality, assume that M is moderate. The stack can expand up to the number of transition rules of M (which is less than n), otherwise some rule would be used at least twice, which would cause M to accept an infinite number of words.

From M we can obtain an (E,C)-finite-automaton M' by adding orthogonal state components for simulating the contents of the stack, and another component describing the height of the stack at any given moment, and changing the

[†]Recall the use of an f subscript on the arrows, indicating finite languages.

transitions accordingly. M' is of size polynomial in n. We now simulate M' by a C-automaton of exponential size, by removing the nondeterminism as in [DH]. $\triangle$

Corollary 18. Let ξ be $\emptyset$ or C. Then

$$(\mathrm{E},\xi)\text{-PDA} \stackrel{\cdots}{\longrightarrow}_f \xi\text{-PDA}\,.$$

$$(\mathrm{E},\xi)\text{-PDA} \stackrel{\cdots}{\longrightarrow}_f (\mathrm{A},\xi)\text{-PDA}\,.$$

Corollary 19. Let ξ be $\emptyset$ or A. Then

$$(\mathrm{E},\mathrm{C})\text{-PDA} \stackrel{\cdots}{\longrightarrow}_f \xi\text{-PDA}\,.$$

In contrast to the double-exponential transition from E to A, the reverse transition is of unlimited complexity:

Proposition 20.

$$\mathrm{A}\text{-PDA} \stackrel{}{\longrightarrow}_{\star f} \mathrm{E}\text{-PDA}\,.$$

Proof. The proof is the same as the analogous proof in Prop. 12, since the language

$$L_n = \{q_0 1^n \not\alpha_2^R \not\cdots \alpha_k^R \not q_1 1^{g(n)} \not\not\}$$

used therein is obviously finite. (In contrast, when simulating an E-pda by an A-pda in Prop. 12 we used the complement, $\overline{L}_n$, which is not finite.) $\triangle$

Corollary 21. Let ξ_1 be $\emptyset$ or C, and let ξ_2 be any subset of {E,C}. Then

$$(\mathrm{A},\xi_1)\text{-PDA} \stackrel{}{\longrightarrow}_{\star f} \xi_2\text{-PDA}\,.$$

It is of particular interest that, although nondeterminism can be eliminated with a limited blowup (Corollary 18), this is not true in the presence of the A feature:

Proposition 22.

$$(\mathrm{E},\mathrm{A})\text{-PDA} \stackrel{}{\longrightarrow}_{\star f} \mathrm{A}\text{-PDA}\,.$$

Proof. Let g be any recursive function. Let T be a Turing machine that, for each $n > 0$, computes $g(n)$ according to the computation sequence represented by the following word:

$$\alpha_n = q_0 1^n \not\beta_2^R \not\beta_3 \not\cdots \not\beta_k^R \not q_1 1^{g(n)} \not\not,$$

where q_1 is the machine's final state. Let

$$L_n = \{w_1\$w_2\$w_3 \mid \text{ for } 1 \le i \le 3,\ |w_i| \le |\alpha_n|,\ w_1 = \alpha_n \text{ or } w_3 = \alpha_n, \text{and } w_2 \ne \alpha_n\}$$
$$\cup\ \{w_1\$w_2 \mid\ \ |w_i| \le 2|\alpha_n| \text{ for } i = 1,2,\ \ w_1 = \alpha_n \text{ or } w_2 = \alpha_n.\}$$

We now construct a small (E,A)-pda M that accepts L_n. M guesses nondeterministically whether the input-word w contains one '\$' symbol or two. Say it has decided the latter, so that the input word should be $w = w_1\$w_2\$w_3$. M now guesses which of w_1 or w_3 must equal α_n. Say it decides w_1. It then verifies the equality $w_1 = \alpha_n$, using parallelism to check that each pair of blocks separated by $\not\in$'s in fact represents two successive configurations of the Turing machine T, laid back to back, and it verifies the inequality $w_2 \neq \alpha_n$, using nondeterminism to find a pair of such blocks that do not thus represent successive configurations. Both these checks can be carried out with an (E,A)-pda of constant size, using the stack to arrive at the corresponding point in the second of the two blocks being checked. We also have to check, in parallel to the verification of $w_1 = \alpha_n$, that the initial sequence of 1's is of length n. This is carried out by a deterministic component of logarithmic size, as in the proofs of Props. 12 and 14. In parallel to these activities, we use the stack to verify that $|w_2|, |w_3| \leq |\alpha_n|$. This is done by pushing w_1, which is really just α_n, onto the stack, and then comparing its length to the lengths of w_2 and w_3, in parallel. The size of this (E,A)-pda component is also constant. Hence, the total size of M is only $O(\log n)$.

We now show that the smallest A-pda accepting L_n must be polynomially related to $\log(g(n))$. By Props. 3–5, it suffices to show that there cannot be a CNF grammar G generating $\overline{L}_n$ with less than $\log(g(n))/2$ variables. Indeed, let $k(G)$ be the constant from the iteration theorem (see, e.g., [Hr, p. 186]). We have $k(G) \leq g(n)$. Let $w = \alpha_n\$\alpha_n\$\alpha_n = w_1\$w_2\$w_3 \in \overline{L}_n$, and let us mark the set of positions of w_2 in w. Obviously, $|w_2| = |\alpha_n| > g(n) \geq k(G)$. Hence, by the theorem, there is a partition $v_1, \ldots, v_5$ of w such that: $\forall i \geq 0$, $v_1 v_2^i v_3 v_4^i v_5 \in \overline{L}_n$, where $v_2 \neq \Lambda$ or $v_4 \neq \Lambda$, and is taken entirely from w_2. Assume it is v_2. If v_4 is empty, or if it is not empty and is taken entirely from w_2, then in $w' = v_1 v_2^0 v_3 v_4^0 v_5$, we have $w_1 = w_3 = \alpha_n$, $w_2 \neq \alpha_n$, and $|w_2| < |\alpha_n|$. Thus $w' \in L_n$. Therefore, $w' \notin \overline{L}_n$, which is a contradiction. If $v_4 \neq \Lambda$ and is taken entirely from w_3, then in $w' = v_1 v_2^0 v_3 v_4^0 v_5$ we have $w_1 = \alpha_n$, $w_2 \neq \alpha_n$, $w_3 \neq \alpha_n$, and $|w_2|, |w_3| < |\alpha_n|$. Thus $w' \in L_n$ again. Finally, if v_4 contains a '\$', then $w' = v_1 v_2^0 v_3 v_4^0 v_5$ contains only one '\$'. Thus $w' = w_1\$w_2$, where $w_1 = \alpha_n$ and $|w_2| \leq 2|\alpha_n|$. Therefore, $w' \in L_n$, which, once again, is a contradiction. $\triangle$

Corollary 23. Let ξ_1 and ξ_2 be either C or $\emptyset$. Then

$$(\text{E,A},\xi_1)\text{-PDA} \xrightarrow[\star]{}_f (\text{A},\xi_2)\text{-PDA}\,.$$

Here, too, we have exponential lower bounds on replacing C by E or A, and the proof is a finitary version of the proof of the analogous Prop. 14:

Proposition 24. Let ξ be E or A. Then

$$\text{C-PDA} \xrightarrow{\cdot}_f \xi\text{-PDA}\,.$$

Proof. Let $L_n = \{1^k \mid k \leq 2^n\}$. The C-pda accepting L_n is similar to that of the proof of Prop. 14, except for the final states. All the states will be final,

except one "sink" state, that will be reached if the stack empties before the entire input-word has been read.

In contrast, we now show that any CNF grammar that generates L_n or $\overline{L}_n$ contains at least n variables. By Props. 3–5, this will complete the proof.

Assume that there is such a CNF grammar with less than n variables. Then in the derivation tree of 1^{2^n} (or $1^{2^{n+1}}$, respectively) in L_n (or $\overline{L}_n$, respectively), there would be a path of length at least n. Thus, this path would contain some variable appearing at least twice, and we could pump the derivation-tree of 1^{2^n}, accepting other words that are not in L_n. Similarly, we could pump-out the derivation tree of $1^{2^{n+1}}$ in order to get a smaller word which is not in $\overline{L}_n$. $\triangle$

Corollary 25. Let ξ be E, A or $\emptyset$. Then

$$(\xi,\text{C})\text{-PDA} \quad \dot{\longrightarrow}_f \quad \xi\text{-PDA}\,.$$

As in the previous section, we have the following conjecture:

Conjecture 26.

$$\text{C-PDA} \quad \dot{\longrightarrow}_f \quad (\text{E,A})\text{-PDA}\,.$$

Here now is a lower bound that establishes the tightness of the upper bounds of Prop. 17 and Corollaries 18 and 19:

Proposition 27.

$$(\text{E,C})\text{-PDA} \quad \dot{\dot{\dot{\longrightarrow}}}_f \quad \text{A-PDA}\,.$$

Proof. Let

$$L_n = \{w \mid\ w \in \{0,1,\$\}^*, \text{ and either } |w| < 4n+3 \text{ or}$$
$$(w = w_1\$w_2\$w_3\$w_4,\ \ |w| = 4n+3,\ [w_1 \neq w_2 \text{ or } w_2 \neq w_3 \text{ or } w_3 \neq w_4])\}$$

We shall first exhibit an (E,C)-pda M of size $O(\log\log n)$ that accepts L_n. By the proof of Prop. 14 we can count up to n with a C-pda of size $O(\log\log n)$. Thus, a C-pda of size $O(\log\log n)$ can check if $|w| = 4n+3$ or $|w| < 4n+3$. To check the inequalities in the former case, M uses nondeterminsm to guess the offending bit, and then uses the $O(\log\log n)$ counting ability to reach the corresponding bit in the next word and check complience (see [DH]). This is done in parallel to the equality check, $|w| = 4n+3$.

In contrast, we now show that any CNF-grammar that generates $\overline{L}_n$ must contain at least 2^n variables. As before, the desired result will follow from Props. 3–5.

Assume that G is a CNF-grammar for $\overline{L}_n$, with less than 2^n variables. Let $T_1, \ldots T_{2^n}$ be the derivation-trees of the 2^n words in the set:

$$A_n = \{w_1\$w_2\$w_3\$w_4 \mid\ w_i \in \{0,1\}^n, \text{ for } 1 \leq i \leq 4, \text{ and } w_1 = w_2 = w_3 = w_4\}.$$

Notice that $A_n \subseteq \overline{L}_n$. Since G is in Chomsky normal form, each node of T_i, $1 \leq i \leq 2^n$, splits into two nodes at most. Assume the root S splits into A and B. If A (or B) generates a part of w_1 (or w_4), but not all of it, we refer to B (or A) as a root, and continue recursively until, for the first time, we reach a variable V_1 that splits into V_2 and V_3 so that V_2 (or V_3) generates the whole of w_2 (or w_3) and do not generate any character from w_4 (or w_1). Thus, each tree T_i, $1 \leq i \leq 2^n$, has a least-common-ancestor of w_2 or w_3, which does not generate any character of w_4 or w_1, respectively. We denote these nodes by B_i, $1 \leq i \leq 2^n$, and the words generated from them by $w(B_i)$, $1 \leq i \leq 2^n$. Since there are less than 2^n variables in G, there must exist $1 \leq i, j \leq 2^n$, $i \neq j$, such that, $B_i = B_j$. If $|w(B_i)| < |w(B_j)|$, then by replacing $w(B_j)$ by $w(B_i)$ in the word $w_j\$w_j\$w_j\$w_j$, the grammar would generate this new word, which is of length less than $4n+3$, and is thus not in $\overline{L}_n$ (since it is in L_n). If the lengths of $w(B_i)$ and $w(B_j)$ are equal but the positions of the '\$' in them are not, then by exchanging them we also obtain words that are not in $\overline{L}_n$. Otherwise, if $|w(B_i)| = |w(B_j)|$ and the positions of the '\$' in them are equal, then, by replacing $w(B_j)$ by $w(B_i)$ in $w_j\$w_j\$w_j\$w_j$, we obtain either the word $w_j\$w'\$w_i\$w''$ or the word $w'\$w_i\$w''\$w_j$, with $w', w'' \in \{0,1\}^n$ (depending on which of w_2 or w_3 is generated by B_j in $w_j\$w_j\$w_j\$w_j$), both of which are not in $\overline{L}_n$. $\triangle$

Corollary 28. Let ξ_1 be C or $\emptyset$ and ξ_2 be A or $\emptyset$. Then

$$\begin{aligned} \text{(E,C)-PDA} &\dashrightarrow_f \emptyset\text{-PDA}\,, \\ \text{(E, }\xi_1\text{)-PDA} &\dashrightarrow_f (\xi_1, \xi_2)\text{-PDA}\,, \\ \text{E-PDA} &\dashrightarrow_f \text{(C, }\xi_2\text{)-PDA}\,. \end{aligned}$$

6. Conclusion

As mentioned in the introduction, our results provide further evidence of the inherent exponential power of bounded cooperative concurrency (see [Ha3]). As shown, the C feature retains its exponential power in all cases we have considered on the pushdown automata level, although E and A are much more powerful. In fact, one might argue that E and A are *too* powerful as features for modelling parallelism in pushdown automata. They capture more languages than the basic version of deterministic machines, and provide unlimited (and one might add, unreasonable) power of succinctness when restricted to the class of deterministic languages. The situation for finite languages is less symmetric, and somewhat more difficult, as shown in Section 5.

We plan to carry out a similar investigation of pushdown automata on infinite words and on trees.

References

[CKS] Chandra, A.K., D. Kozen, and L. J. Stockmeyer, "Alternation", *J. Assoc. Comput. Mach.* **28** (1981), 114–133.

[DH] Drusinsky, D. and D. Harel, "On the Power of Bounded Concurrency I: The Finite Automata Level", submitted, 1989. (Preliminary version appeared as: "On the Power of Cooperative Concurrency", *Proc. Concurrency '88*, Lecture Notes in Computer Science **335**, Springer-Verlag, New York, 1988, pp. 74–103.)

[Ha1] Harel, D., "Statecharts: A Visual Formalism for Complex Systems", *Sci. of Comput. Prog.* **8** (1987), 231–274.

[Ha2] Harel, D., "On Visual Formalisms", *Comm. Assoc. Comput. Mach.* **31** (1988), 514–530.

[Ha3] Harel, D., "A Thesis for Bounded Concurrency", *Proc. 14th Symp. on Math. Found. of Comput. Sci.*, Lecture Notes in Computer Science, Vol. 379, Springer-Verlag, New York, 1989, pp. 35–48.

[HRV] Harel, D., R. Rosner and M. Vardi, "On the Power of Bounded Concurrency III: Reasoning about Programs", *Proc. 5th IEEE Symp. on Logic in Computer Science*, to appear, 1990.

[Hr] Harrison, M. A., *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA, 1978.

[Hi] Hirst, T., "Succinctness Results for Statecharts", M.Sc. Thesis, Bar-Ilan University, Ramat Gan, Israel, 1989 (in Hebrew).

[Ho] Hoare C.A.R, "Communicating Sequential Processes", *Comm. Assoc. Comput. Mach.* **21**, (1978), 666–677.

[HU] Hopcroft, J. E., and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

[LLS] Ladner, R. E., R. J. Lipton and L. J. Stockmeyer, "Alternating Pushdown Automata", *Proc. 19th IEEE Symp. on Found. of Comput. Sci.*, 1978, pp. 92–106.

[MF] Meyer, A. R. and M. J. Fischer, "Economy of Description by Automata, Grammars, and Formal Systems", *Proc. 12th IEEE Symp. on Switching and Automata Theory*, 1971, pp. 188–191.

[Mi] Milner, R., *A Calculus of Communicating Systems*, Lecture Notes in Computer Science, Vol. 94, Springer-Verlag, New York, 1980.

[Pa] Parikh, R. J., "On Context-Free Languages", *J. Assoc. Comput. Mach.* **13** (1966), 570–581.

[Re] Reisig, W., *Petri Nets: An Introduction*, Springer-Verlag, Berlin, 1985.

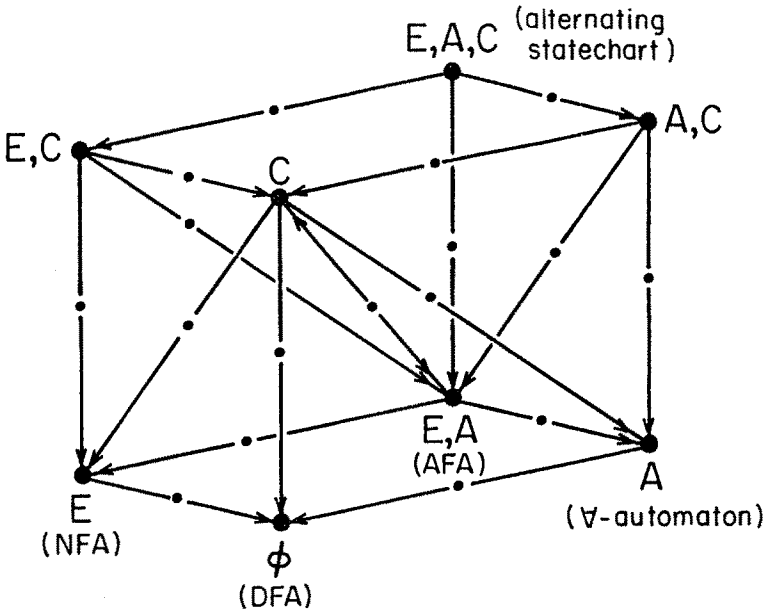

Figure 1. Results for finite automata (see [DH])

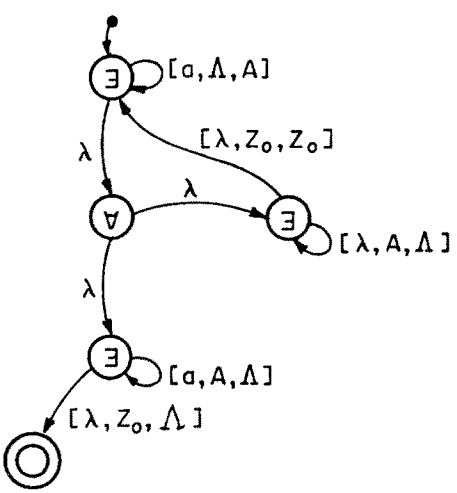

Figure 2. An (E,A)-pda for $L = \{a^{2^n} \mid n > 0\}$

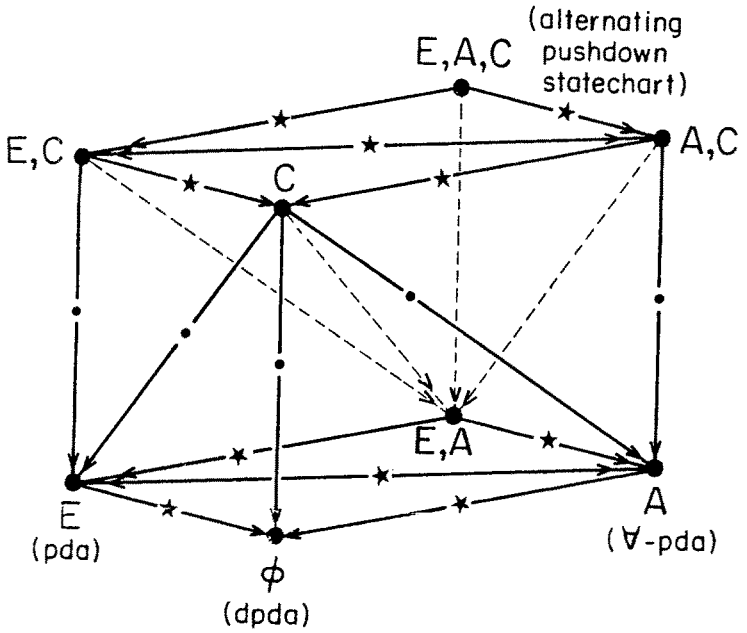

Figure 3. Summary of results for deterministic and regular languages

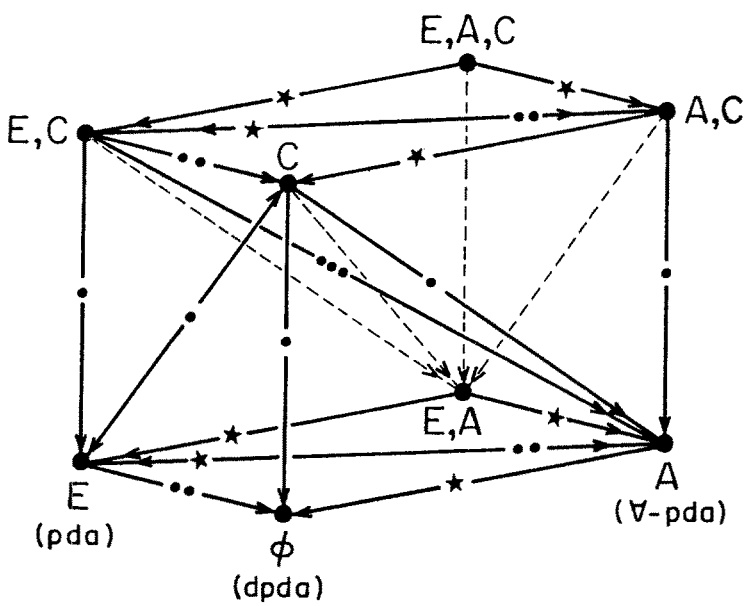

Figure 4. Summary of results for finite languages