# On the Power of Bounded Concurrency III: Reasoning about Programs

Preliminary Report

David Harel* and Roni Rosner†
The Weizmann Institute of Science‡
and
Moshe Vardi§
IBM Almaden Research Center¶

## Abstract

In this paper we continue the study of the inherent power of bounded cooperative concurrency, whereby an automaton can be in some bounded number of states that cooperate in accepting the input. The present paper addresses the difficulty of reasoning about programs. Specifically, we consider the question of whether the additional succinctness that bounded concurrency provides influences the complexity of reasoning about regular computation sequences on the propositional level. Our results concern dynamic, temporal, and process logics, and supply a strongly affirmative answer. In particular, we prove triple-exponential time upper and lower bounds on deciding the validity of propositional dynamic logic with alternating automata enriched with bounded cooperative concurrency, and quadruple-exponential time bounds for deciding validity of branhcing-time and process logics with such automata. In addition to constituting further evidence for the inherent exponential nature of bounded concurrency, the results appear to provide the first examples of natural decision problems that are elementary yet have lower bounds that are higher than double-exponential time.

## 1 Introduction

Classical models of computation, such as Turing machines and 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 is reflected in the separate tapes or stacks that are generated whenever branching (of either kind) takes place. Thus, branching essentially produces separate computations, the results of which 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 multiple states 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 proceeds. In the sequel, we shall use E, A and C, respectively, to denote existential branching (nondeterminism), uni-

*This author's research partially supported by a grant from the Gutwirth Foundation.

†This author's research partially supported by a grant from the Israel Ministry of Science and Development, the National Council for Research and Development.

‡Address: Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot 76100, Israel. E-mail: harel@wisdom.weizmann.ac.IL, roni@wisdom.weizmann.ac.IL

§This author's research carried out in part while visiting the Dept. of Applied Mathematics and Computer Science at the Weizmann Institute of Science, during the Summer of 1989.

¶Address: IBM Almaden Research Center, San Jose, CA 95120-6099. E-mail: vardi@ibm.com

CH2897-7/90/0000/0478$01.00 © 1990 IEEE

versal branching (∀-parallelism), and bounded cooperative concurrency (or simply *bounded concurrency* for short).

Two previous papers [DH89, HH90] have investigated the inherent power of bounded cooperative concurrency, relating it to the two classical kinds of branching. In both papers the criteria used in comparing the power of features is *succinctness*. In [DH89] finite automata were considered, over both finite and infinite words, and in [HH90] pushdown automata were considered, for both deterministic and finite languages. One finding that recurs in all of these cases 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 provide more, less, or the same power. This research is motivated and summarized in a uniform fashion in [Har89].

To help describe the present work we survey some of the results of [DH89]. It is well-known that NFAs are exponentially more succinct than DFAs, in the following upper and lower bound senses (see, e.g., [MF71]):

- 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 [CKS81]). 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.[1]

In [DH89] the effect of adding the C feature was investigated, via the use of statecharts [Har87] as an extension of finite automata. One set of results in [DH89] establishes the vertical lines and the top horizontal lines of Fig. 1, and all the transitivity consequences thereof. 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, the vertical and horizontal lines of Fig. 1, with their transitive extensions, 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 transitive extensions.[2]

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 when 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; in fact, in this paper we define the C feature in terms of a simple extension of finite automata. This extension is an abstraction of the bounded cooperative concurrency feature present in the statechart formalism as well as in bounded versions of conventional formalisms of concurrency such as Petri nets [Rei85] CSP [Hoa78], CCS [Mil80], 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 our work here too.

In the present paper we concern ourselves with the effect these discrepancies in succinctness have on the difficulty of reasoning about regular programs at the propositional level. We consider several formalisms: *linear temporal logic* (LTL) [Pnu77], *branching temporal logic* (BTL) [EH86], *propositional dynamic logic* (PDL) [FL79], and *process logic* (YAPL) [VW83]. For lack of space we focus in this preliminary report on PDL and BTL.

To motivate the work further, it is useful to recall some results on PDL. (Definitions can be found in [FL79, Har84, KT89].) One of the basic questions regarding PDL is the complexity of deciding the validity of formulas. Is the validity problem decidable, and if so is it worse than that of its sublogic, the proposi-

[1] In the figure, transitivity is assumed too; hence, the line labeled 'two-exponentials' that would lead from (E,A) to ∅ is omitted for clarity, despite the fact that it does not follow *a priori*.

[2] Similar results are obtained in [DH89] for the case of infinite words, and in [Hir89] for the case of finite words over a one-letter alphabet.

tional calculus (which is co-NP-complete)? In [FL79] it is shown that the problem is decidable in NEXPTIME, a bound that was later improved by Pratt to DEXPTIME[3] (see, e.g., [Har84, KT89]). A matching lower bound of EXPTIME was also established in [FL79], so that the problem is actually logspace-complete for deterministic exponential time.

An interesting question was raised in [Pra81]. From results in [EZ76] it follows that NFAs are exponentially more succinct than regular expressions, in the upper and lower bounds senses used here[4]. The question in [Pra81] was whether the version of PDL in which the programs are NFAs, instead of regular expressions, is complete for EXPTIME or perhaps requires 2EXPTIME (double-exponential time) — one exponential for transforming the NFAs into regular expressions and the other to apply the exponential time decision procedure for regular PDL[5]. The answer is the former: $\mathrm{PDL}_{\mathrm{E}}$, as we may call it, signifying that the programs are automata enriched with the E feature, is also decidable in EXPTIME (see [Pra81, HS85]). Clearly, this implies EXPTIME decidability for $\mathrm{PDL}_{\emptyset}$ too. (The lower bound of EXPTIME can be easily established for $\mathrm{PDL}_{\emptyset}$ too.) Thus, the differences in succinctness between regular expressions and deterministic or nondeterministic automata do not affect the exponential time decidability of PDL; reasoning about abstract regular programs, given in any of the three media for representation, can be carried out in deterministic exponential time.

Given the succinctness results of Fig. 1, new questions arise. Does the A feature make a difference? How about the C feature? What happens when two or three of the features are present? Our main result is that, in contrast with E, the A and C features *do* make a difference. We show that the succinctness that these features provide carries over to the problem of reasoning about computation sequences enriched by the corresponding automata, causing the decision problems to be much harder. Specifically, we show that A and C add an exponential to the complexity of the decision problems for the formalisms we investigate, independently of each other, and in an additive manner. Our results, depicted in Fig. 2, are summarized in Fig. 3.

The outline of the paper is as follows. The technical issues that cause the decision problems to become hard are isolated in Section 2. It turns out that it suffices to be able to carry out a uniform kind of counting and marking states in the models. Whenever this is doable, the appropriate lower bounds follow.

We consider PDL in Section 3. Our results are summarized in the penultimate column of Fig. 3. It is not too difficult to see that the upper bounds follow from those of Fig. 1. For example, that $\mathrm{PDL}_{\mathrm{E,A,C}}$ can be decided in triple-exponential time follows from the ability to remove the A and C features at a cost of two exponentials, and to then apply the exponential decision procedure of [HS85, Pra81]. We thus concentrate on lower bounds, which require a nontrivial combination of techniques from [Abr80, DH89, FL79, VS85]. (The main results of Section 3 appeared in preliminary form in [Har89].)

We next turn to CTL*, a branching version of temporal logic in which formulas can quantify over paths in the model [EH86] and YAPL, a restricted form of process logic [VW83]. For both of these, an upper bound of 2EXPTIME was established in [EJ88, EJ89] and a lower bound of 2EXPTIME was established in [VS85]. In Section 4 we extend these logics too with the E, A and C features. Our results concerning the added complexity of these features are analogous to those for PDL, as can be seen in the last column in Fig. 3. Since the basic decision problem here is 2EXPTIME, all the complexity bounds are one exponential higher. In particular, CTL* and YAPL with alternating statecharts is shown to be complete for 4EXPTIME.

Our bounds for CTL* apply also to the problem of *program synthesis*, studied in [PR89a, PR89b]. It is shown there how, given a linear temporal specification $\varphi$, a branching temporal formula $\psi$ can be derived, whose validity expresses the *implementability* of the specification $\varphi$. On the propositional level, this form of the synthesis problem is 2EXPTIME-complete. We observe that our proofs of the CTL* case entail analogous results concerning the synthesis problem. Specifically, we can show that for specifications given in linear temporal logic augmented with automata temporal connectives, the complexity of synthesis is identical to that of CTL* augmented with the corresponding automata. This applies to any combination of E, A and C.

We consider the results of the paper to be interesting in several respects. First, they provide additional strong evidence for the inherent exponential power of bounded cooperative concurrency. Second, they show that the relationship between the E, A, and C features, when considered in the present framework, differs from that obtained when investigating succinctness alone as in [DH89, Hir89]. This is demonstrated by the difference between Fig. 1 and Fig. 2. Finally, the 2EXPSPACE, 3EXPTIME and 4EXPTIME re-

[3] We shall simply write EXPTIME for short.

[4] For DFAs, there are exponential lower bounds in *both* directions.

[5] We should add that representing regular programs by automata, rather than by regular expressions, is tantamount to moving from *while*-programs to flowcharts.

sults appear to be the first examples of natural decision problems in logic that are solvable in elementary time, but whose lower bounds are higher than 2EXPTIME.

## 2 Counting and Comparing Counters

As our basis, we use finite-state automata on finite words. We briefly describe an extension thereof, similar to that of [CKS81], but enriched with the C operator of [DH89], with the intention of providing a uniform approach to the different notions of concurrency. An automaton in our extended model consists of the following components: A finite alphabet $\Sigma$; a set $S$ of states partitioned into three types: E-states, A-states and C-states; two designated sets of initial and of terminal subsets of $S$; and a transition function, assigning to each pair of states (i.e., a *transition*) a Boolean expression over $B = \Sigma \cup S$. Here, E stands for existential (unbounded) nondeterminism, A stands for universal (unbounded) nondeterminism, and C stands for *bounded cooperative concurrency*. A transition $\langle s_1, s_2 \rangle$ is said to be *enabled* on a set of states $S'$ and a symbol $a \in \Sigma$, if the transition function assigns to it a true expression under the characteristic function of $S' \cup \{a\}$ in $B$. A *run* of an automation $A$ on a word $a_1 a_2 \cdots a_k$ is a finite frontiered tree (called a *trace*) of depth $k$, each node of which contains a non-empty set of states. The root of the trace contains some initial subset of $S$. A node on level $i$ of the trace containing the set $S'$ is expanded to its level $i+1$ successors by *simultaneously* applying *some* transition that is enabled on $S'$ and $a_i$ to each E-state in $S'$, and *all* enabled transitions to every C-state and A-state in $S'$. A C-state is expanded by replacing it by *all* of its successors, but without splitting the node. An A-state, on the other hand, is expanded by splitting the node in question into several nodes, each of which contains the expansion of the original set of states but with the A-state replaced by one of its successors. This captures the subtle difference between C and A: the former denotes parallelism within a single computation path, whereas the latter denotes branching into separate paths. $A$ accepts a word if there is some accepting run of $A$ on it, i.e., a trace in which all the nodes along its frontier contain terminal sets of states. (We note that there is an obvious correspondence between these automata and the (A,E)-statecharts of [DH89].) The language of $A$, $\mathcal{L}(A)$, is the set of all words accepted by $A$.

Call the automaton $A$ a $\Gamma$-*automaton*, for $\Gamma \subseteq \{\text{A,E,C}\}$, if $A$ employs the *features* listed in $\Gamma$ (i.e., $A$ has states of the corresponding types). An automaton $A$ operating on an alphabet $\Sigma = \{a_0, \ldots a_{k-1}\}$, denoted $A(a_0, \ldots, a_{k-1})$, is viewed as a $k$-ary operator. We briefly describe the formalism; the reader is referred to [HS85, VW83, WVS83] for more details.

We consider propositional logics of programs with two types of elements: *programs* and *formulas*. We obtain programs by applying $k$-ary automata $A(a_0, \ldots, a_{k-1})$ to $k$ pairs $\langle X_i, b_i \rangle$, where $X_i$ is a *propositional formula* and $b_i$ is an *atomic program*. The intended semantics of this program is a set of paths, where every such path corresponds to a word in the language accepted by $A$. The letter $a_i$ corresponds to a $b_i$-transition from a state satisfying $X_i$. This definition is in the spirit of defining programs over sequences of alternating *tests* and letters, which is more suitable for our approach of unifying the dynamic and temporal formalisms.

Formulas are formed from atomic propositions $P, Q, \ldots$, and programs, by Boolean connectives $\neg, \vee$, and the $<>$ modality over pairs of programs and formulas. All formulas are *state formulas*, i.e., their intended semantics is a set of states, except for programs, which are *path formulas*, i.e., their intended semantics is a set of paths.

Process logic (YAPL) is the largest formalism admitting all the above defined formulas. Here, $< \alpha > X$ requires $X$ to be evaluated over an infinite path a prefix of which is admitted by $\alpha$. Propositional dynamic logic (PDL) is obtained as the special case, where modalities are applied only to state formulas, with $< \alpha > X$ requiring $X$ to hold at some end-state of an $\alpha$ path. In branching temporal logic (BTL) on the other hand, there is only one implicit atomic program, say $a$, and thus a single modality $< a^* >$ (denoted by the path quantifier $\exists$), while linear temporal logic (LTL) is further restricted to linear models (and hence also drops the path quantifier $\exists$). A subscript $\Gamma$ to the above logics refers to the automata features admitted.

Let $\Sigma = \{0, 1\}$ be our basic alphabet. We consider automata with alphabet $\times_{i=0}^{l-1} \Sigma = \Sigma^{(l)}$, $l > 0$. A letter in $\Sigma^{(l)}$ is to be viewed as encoding the truth values of $l$ propositions. Let $D$ be an automaton over the alphabet $\Sigma$. For a letter $b \in \Sigma^{(l)}$ define the *b-discriminating* expansion of $D$, to be the automaton $D^b$ over the alphabet $\Sigma^{(l)}$, derived from $D$ by replacing any 1-edge in $D$ by a $b$-edge, and any 0-edge in it by the set of all possible $\Sigma^{(l)} - \{b\}$ edges.

For an integer $k \geq 0$, define the function $2^{(k)} : \mathcal{N} \to \mathcal{N}$ as follows: $2^{(0)}(m) = m$, and $2^{(k+1)}(m) = 2^{2^{(k)}(m)}$.

Let L be any version of $\text{LTL}_\Gamma$, $\text{PDL}_\Gamma$, $\text{BTL}_\Gamma$, or $\text{YAPL}_\Gamma$, with $\Gamma \subseteq \{\text{E,A,C}\}$. Let $R$ be an atomic proposition and $f : \mathcal{N} \to \mathcal{N}$ an integer function. Define an *R-f-counter* to be a family $\varphi = \{\varphi_n \mid n > 0\}$ of L-formulas that can be generated in polynomial (in

$n$) time, such that:

- every model for $\varphi_n$ has at least $f(n)$ distinct states in which $R$ holds (call such states *R-states*); and
- for every $m > 0$, there exists a model for $\varphi_n$ and a cycle in this model, along which there are precisely $mf(n)$ distinct $R$-states.

Intuitively, the formula $\varphi_n$ counts reference points marked by the proposition $R$ along paths, so that every cycle has a count of 0 modulo $f(n)$.

Given such a counter, define an *R-f-positioner* to be a family $\psi(X) = \{\psi_n(X) \mid n > 0\}$ of L-formulas that can be generated in polynomial (in $n$) time, such that: for any L-formula $\chi$, a model $\mathcal{M}$ for $\varphi_n$, and $R$-state $s_0$, if $\mathcal{M}, s_0 \models \psi_n(\chi)$, then along every path in the model, formed by atomic programs from $\psi$, and rooted at $s_0$, the $f(n)$-th $R$-state satisfies $\chi$. Intuitively, the formula $\psi_n(\chi)$ imposes the truth of $\chi$ at the $f(n)$-th reference point along any appropriate path.

The idea underlying our lower bounds is that a pair consisting of an $R$-$f$-counter and an $R$-$f$-positioner, enables us to describe Turing machines behaviors structured as either sequences or trees, with fixed size configurations at each node. The size of all configurations is a multiple $g(n) = mf(n)$, for some fixed $m > 0$. To be able to describe paths along such trees we require that every configuration follows from the previous one by 'local' transformations only, i.e., the $j$'th point in a configuration can be determined by propositional reasoning from the $j$'th point in the previous configuration. Indeed, Turing machines transitions are local in that sense.

The logic L is *linearly-i-succinct*, $i \geq 0$, if there exist in L,

- an $R$-$2^{(i)}$-counter $\varphi$; and
- an $R$-$2^{(i)}$-positioner $\psi(X)$.

L is *i-succinct* if it is linearly-$i$-succinct, and is interpreted over branching structures, i.e., L is *not* LTL. L is *exactly (linearly-) i-succinct* if $i$ is the largest $j$ such that L is (linearly-) $j$-succinct.

The next results isolate the computational aspects of our proofs, by generalizing the simulation of various Turing machines using appropriate powerful logics.

**Proposition 2.1** *If* L *is k-succinct, then*

- *the satisfiability problem for* L *is logspace hard for* $2^{(k+1)}$-DTIME; *and*
- *the size of models for formulas of length n is* $\Omega(2^{(k+1)})$.

**Sketch of Proof:**
We generalize techniques used in [FL79] for proving the one-exponential lower bound for regular PDL, and ideas that are reminiscent of the double-exponential bounds given in [Abr80] for PDL with Boolean variables, and in [VS85] for CTL*.

Given an arbitrary $2^{(k)}(cn)$-space-bounded alternating Turing machine $M$, with $c$ a constant, and an input $x$ of size $cn$, we construct a polynomial-size formula $\varphi_{M,x}$ in the logic L, and show that $M$ accepts $x$ iff $\varphi_{M,x}$ is satisfiable. The idea is to encode each configuration of $M$ by a sequence of $2^{(k)}(cn)$ $R$-states in an L model, identified by an appropriate $R$-$2^{(k)}$-counter. The key thing that $\varphi_{M,x}$ must be able to do (all the rest follows more or less standard PDL or CTL* techniques), is to jump from an arbitrary point in one configuration to the corresponding point in a successor configuration, in order to verify compliance with the transition table of $M$. This is done by using the corresponding $R$-$2^{(k)}$-positioner.

The total size of the resulting formula can then be shown to be polynomial in $n$. ∎

By restricting these techniques to linear models, we may similarly encode computations of nondeterministic space-bounded Turing machines. Hence the following corollary.

**Corollary 2.2** *If* L *is linearly-k-succinct, then*

- *the satisfiability problem for* L *is logspace hard for* $2^{(k)}$-SPACE; *and*
- *the size of models for formulas of length n is* $\Omega(2^{(k+1)})$.

Having thus set up the general results, we are left with having to show succinctness of the different versions of the particular logics we are interested in. This is done in the following two sections.

## 3 PDL and LTL with Concurrent Automata

The following proposition follows immediately from the one-exponential decision procedure for $\mathrm{PDL_E}$ presented in [Pra81, HS85], and the exponential elimination of each of A or C, given in [DH89].

**Proposition 3.1** *For* $\Gamma \subseteq \{\mathrm{A},\mathrm{C}\}$ *with* $|\Gamma| = k$, *and each of the logics* $\mathrm{PDL}_\Gamma$ *and* $\mathrm{PDL}_{\Gamma,\mathrm{E}}$,

- *satisfiability is decidable in* $2^{(k+1)}$-DTIME; *and*
- *every satisfiable formula has a model of size* $2^{(k+1)}$.

We now consider lower bounds.

**Proposition 3.2** *For* $\Gamma \subseteq \{A,C\}$ *with* $|\Gamma| = k$, $\mathrm{PDL}_\Gamma$ *and* $\mathrm{PDL}_{\Gamma,E}$ *are exactly* $k$*-succinct.*

**Proof:**
The upper bounds of Proposition 3.1 imply that the addition of E to $\Gamma$ makes no difference in succinctness, since otherwise we would have higher lower bounds according to Proposition 2.1. Moreover, since these lower bounds match the upper bounds, the above $k$'s are maximal. Thus, it suffices to prove succinctness to establish the exact succinctness of the logics.

We first construct the appropriate counter and positioner for $\mathrm{PDL}_\emptyset$, in order to establish its succinctness. For simplicity, we assume the argument $n$ is a power of 2, i.e., $n = 2^l$ for some $l > 0$, and employ the propositions $P_0, \ldots, P_{l-1}$, as well as the unary alphabet $\Sigma = \{a\}$.

The following $\mathrm{PDL}_\emptyset$ families $\varphi$ and $\psi(X)$ represent an $R$-$2^{(0)}$-counter of size $O(\log n)$ and an $R$-$2^{(0)}$-positioner of size $O(n \log n)$, respectively. The regular programs appearing in them can be easily implemented by deterministic automata.

$$\varphi_n : \quad \bigwedge_{i=0}^{l-1} \neg P_i \ \wedge$$

$$[a^*]\,( \quad R \ \wedge$$

$$\bigwedge_{i=0}^{l-1} (\langle a \rangle P_i \rightarrow [a] P_i) \ \wedge$$

$$([a]\vec{P} = \vec{P} + 1))$$

$$\psi_n(X) : \left[ \bigcup_{\vec{t} \in \{\mathbf{T},\mathbf{F}\}^l} (\vec{P} = \vec{t})(\vec{P} \neq \vec{t})^+ (\vec{P} = \vec{t}) \right] X$$

Where $([a]\vec{P} = \vec{P} + 1)$ stands for

$$(P_0 \equiv [a]\neg P_0) \wedge$$

$$\bigwedge_{i=1}^{l-1} ((P_i \equiv [a]P_i) \equiv (\neg P_{i-1} \vee (P_{i-1} \equiv [a]P_{i-1})));$$

$(\vec{P} = \vec{t})$ stands for

$$\left( \bigwedge_{i=0}^{l-1} (P_i \equiv t_i) \right)?\ a$$

and $(\vec{P} \neq \vec{t})$ stands for

$$\left( \bigvee_{i=0}^{l-1} \neg (P_i \equiv t_i) \right)?\ a.$$

We now have to show that by employing either the A or C features, the above formulas can be exponentially compactified. This is done as follows. The automata we used in the above formulas can be uniformly decomposed into the initial state $s$ and the automata $D^b$ for all $b$ in the alphabet $\Sigma^{(l)}$, where the $D^b$ are the $b$-discriminating expansions of some $D$ over the alphabet $\Sigma$. We now employ $l$ components operating concurrently (via either an A or C branching) which, for any $b$-transition, $b = \langle a_0, \ldots, a_{l-1} \rangle$, the $i$th component is 'sensitive' to $a_i$ only. Initially, for any such $b$-transition emanating from $s$, the $i$th component records $a_i$, and then proceeds to simulate $D$, with $a_i$ substituted for 1. The joint action of the $l$ components correctly simulates its behavior, and this applies to all $b$. This process enables us to use a $2^{(k)}$-positioner to construct a $2^{(k+1)}$-counter, while keeping the polynomial bound. Moreover, we can then construct a $2^{(k+1)}$-positioner from the $2^{(k+1)}$-counter.

Analogously, the addition of the C feature to the above $2^{(k+1)}$-positioner of $\mathrm{L_A}$, enables us to compactify A-automata into equivalent {A,C}-automata, from which we can construct the appropriate $2^{(k+2)}$-counter and $2^{(k+2)}$-positioner as formulas in $\mathrm{L_{A,C}}$. ∎

Our main result concerning PDL with concurrent automata is obtained by combining Propositions 2.1, 3.1 and 3.2.

**Theorem 3.3** *For* $\Gamma \subseteq \{A,C\}$ *with* $|\Gamma| = k$, *and each of the logics* $\mathrm{PDL}_\Gamma$ *and* $\mathrm{PDL}_{\Gamma,E}$,

- *the satisfiability problem is logspace complete for* $2^{(k+1)}$-DTIME; *and*
- *the size of models for formulas of length* $n$ *is* $\Theta(2^{(k+1)}(n))$.

Linear temporal logic can be handled similarly.

**Theorem 3.4** *For* $\Gamma \subseteq \{A,C\}$ *with* $|\Gamma| = k$, *and each of the logics* $\mathrm{LTL}_\Gamma$ *and* $\mathrm{LTL}_{\Gamma,E}$,

- *the satisfiability problem is logspace complete for* $2^{(k)}$-SPACE; *and*
- *the size of models for formulas of length* $n$ *is* $\Theta(2^{(k+1)}(n))$.

**Sketch of Proof:**
The upper bounds can be established by combining the PSPACE completeness of $\mathrm{LTL}_\emptyset$ proved in [WVS83] and the translations between automata of [DH89]. For the lower bounds, the counters and positioners of Proposition 3.2 can be easily transformed into appropriate LTL ones, thus establishing the linear-succinctness of LTL. The lower bounds now follow from Corollary 2.2.

Actually, the linear-$2^{(0)}$-succinctness (and hence also the PSPACE-hardness) holds for the less expressive version of LTL restricted to the standard *until* and *next* operators, thus reestablishing the PSPACE completeness proved in [HR83, SC85]. ⌟

# 4 BTL and YAPL with Concurrent Automata

**Proposition 4.1** *For* $\Gamma \subseteq \{A,C\}$ *with* $|\Gamma| = k$, *and each of the logics* $BTL_\Gamma$ *and* $BTL_{\Gamma,E}$,

- *satisfiability is decidable in* $2^{(k+2)}$-DTIME; *and*
- *every satisfiable formula has a model of size* $2^{(k+2)}$.

**Sketch of Proof:**
A double-exponential decision procedure for $BTL_E$ can be obtained by first using an exponential translation into $\Delta$-$PDL_E$ as described in [VW83], and then an exponential decision procedure, similar to the procedure presented in [EJ89] for $\Delta$-PDL (PDL with programs represented by regular expressions plus the *repeat* operation denoted $\Delta$). Now we can proceed by the standard exponential elimination of each of A or C, given in [DH89]. ⌟

**Proposition 4.2** *For* $\Gamma \subseteq \{A,C\}$ *with* $|\Gamma| = k$, $BTL_\Gamma$ *and* $BTL_{\Gamma,E}$ *are exactly* $(k+1)$*-succinct.*

**Proof:**
Similarly to the proof of Proposition 3.2, we focus first on the $(k+1)$-succinctness of $BTL_\emptyset$. We construct appropriate counter and positioner with the reference proposition $R$.

Assume the argument $n$ is a power of 2, i.e., $n = 2^l$ for some $l > 0$, and employ the propositions $P_0, \ldots, P_{l-1}$ accordingly, as well as several additional propositions: $I, J, Q$ and $R_1$. We introduce several abbreviations: By $\vec{P} = 0$ we abbreviate

$$\bigwedge_{i=0}^{l-1} \neg P_i \ ;$$

and by $\bigcirc\vec{P} = \vec{P} + 1$ we abbreviate

$$(P_0 \equiv \bigcirc\neg P_0) \wedge$$

$$\bigwedge_{i=1}^{l-1}((P_i \equiv \bigcirc P_i) \equiv (\neg P_{i-1} \vee (P_{i-1} \equiv \bigcirc P_{i-1}))).$$

For formulas $Y_1, Y_2$, we abbreviate by $I \downarrow Y_1, Y_2$ the scheme

$$Y_1 \equiv \Diamond(I \wedge \bigcirc\neg I \wedge Y_2),$$

stating that presently $Y_1$ holds iff eventually $Y_2$ holds in a state at which the proposition $I$ 'falls' (i.e., $I$ changes its truth value from true in the indicated state to false in the next one).
By $I \updownarrow \vec{P}$ we abbreviate

$$(\neg R)\mathcal{U}(R \wedge \bigcirc((\neg R)\mathcal{U}(I \wedge \bigcirc\neg I))) \wedge$$

$$\bigwedge_{i=0}^{l-1}(I \downarrow P_i, P_i)$$

stating that eventually $I$ falls with precisely one $R$-state before the fall, while the counter $\vec{P}$ has the same value in the present state and in the falling state.
Also, for any formulas $Y_1, Y_2$, we abbreviate by $I \Downarrow Y_1, Y_2$ the scheme

$$((R \wedge Y_1) \vee \bigcirc((\neg R)\mathcal{U}(\neg R \wedge Y_1))) \equiv$$

$$\Diamond(Y_2 \wedge \bigcirc((\neg R)\mathcal{U}(R \wedge I \wedge \bigcirc\neg I))).$$

In this scheme and in the following formula we consider intervals which begin with an $R$-state and end just before an $R$-state. The scheme states that $Y_1$ holds in some state of the first interval, iff eventually $Y_2$ holds in some state of an interval immediately after which $I$ falls. Finally, by $I \Updownarrow \vec{Q}$, we abbreviate

$$(\neg R_1)\mathcal{U}(R_1 \wedge (\bigcirc\neg R_1)\mathcal{U}(R \wedge I \wedge \bigcirc\neg I)) \wedge$$

$$\bigwedge_{\vec{t} \in \{T,F\}^l} \Big(I \Downarrow ((\vec{P} = \vec{t}) \wedge J), ((\vec{P} = \vec{t}) \wedge Q)\Big)$$

where $(\vec{P} = \vec{t})$ stands for

$$\bigwedge_{i=0}^{l-1}(P_i \equiv t_i).$$

This formula states that eventually $I$ falls in some $R$-state, with precisely one $R_1$-state before the fall, and that the first and the last intervals before $I$ falls are related in such a way that for every value $\vec{t}$ of the counter $\vec{P}$, a $\vec{t}$-state in the first interval satisfies $J$ iff a $\vec{t}$-state in the last interval satisfies $Q$. The proposition $J$ will be used to mark those positions in an interval (representing the binary number $\vec{Q}$) at which there is a 1 bit in the binary number $\vec{Q} - 1$. Thus, $I \Updownarrow \vec{Q}$ will imply that the value of $\vec{Q}$ in the last interval before $I$ falls is 1 less than its value at the first interval.

The following $BTL_\emptyset$ families $\varphi$ and $\psi(X)$ represent an $R$-$2^{(1)}$-counter of size $O(n \log n)$ and an $R$-$2^{(1)}$-positioner of size $O(n \log n)$, respectively. The temporal (path) operators appearing in them can be easily implemented by the appropriate combinatorial deterministic automata. Moreover, the presentation of the formulas in the standard temporal logic CTL*,

can serve as an alternative proof for the succinctness of standard CTL*, whose expressive power is known to be strictly smaller than that of its automata version $\mathrm{BTL}_{\emptyset}$.

$$\varphi_n: \quad R_1 \quad \wedge$$

$$\forall\Box( \quad (R_1 \equiv R \wedge (\neg Q)\mathcal{U}(\neg Q \wedge \bigcirc R)) \ \wedge$$

$$(R \equiv I \wedge (\vec{P} = 0)) \ \wedge$$

$$(I \equiv \exists \bigcirc I) \wedge (I \rightarrow \exists \bigcirc \neg I) \ \wedge$$

$$(J \equiv (Q \equiv \bigcirc((\neg R)\mathcal{U}(\neg R \wedge Q)))) \ \wedge$$

$$(\bigcirc \vec{P} = \vec{P} + 1) \ \wedge$$

$$((I \updownarrow \vec{P}) \rightarrow ((I \downarrow Q, \neg Q) \equiv \bigcirc(Q \mathcal{U} R)))$$

$$\psi_n(X): \ R \ \wedge \ (I \Updownarrow \vec{Q}) \rightarrow (I \downarrow \mathrm{T}, X)$$

We can now show that the above formulas can be compactified as in the proof of Proposition 3.2. ∎

Our main result concerning BTL with concurrent automata is directly obtained from Propositions 2.1, 4.1 and 4.2.

**Theorem 4.3** *For $\Gamma \subseteq \{\mathrm{A},\mathrm{C}\}$ with $|\Gamma| = k$, and each of the logics $\mathrm{BTL}_{\Gamma}$ and $\mathrm{BTL}_{\Gamma,\mathrm{E}}$,*

- *the satisfiability problem is logspace complete for $2^{(k+2)}$-DTIME; and*
- *the size of models for formulas of length $n$ is $\Theta(2^{(k+2)})$.*

Inspecting the details of the proof of Theorem 4.3, we observe that they can be combined with the results of [PR89a, PR89b] to yield the next corollary.

**Corollary 4.4** *For $\Gamma \subseteq \{\mathrm{A},\mathrm{C}\}$ with $|\Gamma| = k$, and each of the specification languages $\mathrm{LTL}_{\Gamma}$ and $\mathrm{LTL}_{\Gamma,\mathrm{E}}$,*

- *the implementability problem is logspace complete for $2^{(k+2)}$-DTIME; and*
- *the size of implementations for formulas of length $n$ is $\Theta(2^{(k+2)})$.*

Moreover, the BTL succinctness results translate easily to similar YAPL ones, as indicated in [VW83]. The double exponential upper bound for $\mathrm{YAPL}_{\mathrm{E}}$ is proved in [EJ89].

**Corollary 4.5** *For $\Gamma \subseteq \{\mathrm{A},\mathrm{C}\}$ with $|\Gamma| = k$, and each of the logics $\mathrm{YAPL}_{\Gamma}$ and $\mathrm{YAPL}_{\Gamma,\mathrm{E}}$,*

- *the satisfiability problem is logspace complete for $2^{(k+2)}$-DTIME; and*
- *there is a lower bound of $2^{(k+2)}$ on the size of models.*

## Acknowledgements

We would like to thank Moria Levi for helping with some of the details of an early proof of the double-exponential lower bound on $\mathrm{PDL}_{\mathrm{C}}$.

## References

[Abr80] K. Abrahamson, *Decidability and Expressiveness of Logics of Programs*, Ph.D. thesis, Univ. of Washington, Seattle, 1980. Available as Technical Report 80-08-01.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer, Alternation, *J. ACM* **28**, 1981, pp. 114–133.

[DH89] D. Drusinsky 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", in *Proc. Intl. Conf. on Concurrency, Concurrency 88*, Lec. Notes in Comp. Sci. **335**, Springer, 1988, pp. 74–103).

[EH86] E.A. Emerson and J.Y. Halpern, 'Sometimes' and 'not never' revisited: On branching time versus linear time, *J. ACM* **33**, 1986, pp. 151–178.

[EJ88] E.A. Emerson and C.S. Jutla, The complexity of tree automata and logic of programs, *Proc. 29th IEEE Symp. Found. of Comp. Sci.*, 1988, pp. 328–337.

[EJ89] E.A. Emerson and C.S. Jutla, On simultaneously determinizing and complementing $\omega$-automata, *Proc. 4th IEEE Symp. Logic in Comp. Sci.*, 1989.

[EZ76] A. Ehrenfeucht and P. Zeiger, Complexity measures for regular expressions, *J. Comp. Sys. Sci.* **12**, 1976, pp. 134–146.

[FL79] M.J. Fischer and R.E. Ladner, Propositional dynamic logic of regular programs, *J. Comp. Sys. Sci.* **18**, 1979, pp. 194–211.

[Har84] D. Harel, Dynamic logic, *Handbook of Philosophical Logic Vol. II* (D. Gabbay and F. Guenthner, eds.), pp. 497–604, Reidel, 1984.

[Har87] D. Harel, Statecharts: A visual formalism for complex systems, *Sci. Comp. Prog.* **8**, 1987, pp. 231–274.

[Har89] D. Harel, A thesis for bounded concurrency, *Proc. 14th Symp. Math. Found. Comput. Sci.*, Lec. Notes in Comp. Sci. 379, Springer, 1989, pp. 35–48.

[HH90] T. Hirst and D. Harel, On the power of bounded concurrency II: The pushdown automata level, *Proc. 15th Coll. Trees in Algebra and Programming*, Lec. Notes in Comp. Sci., Springer, 1990. To appear.

[Hir89] T. Hirst, *Succinctness Results for Statecharts*, Master's thesis, Bar-Ilan Univ., Ramat-Gan, Israel, 1989. In Hebrew.

[Hoa78] C.A.R Hoare, Communicating sequential processes, *Comm. ACM* **21**, 1978, pp. 666–677.

[HR83] J. Y. Halpern and J. H. Reif, The propositional dynamic logic of deterministic well-structured programs, *Theoretical Computer Science* **27**, 1983, pp. 127-165.

[HS85] D. Harel and R. Sherman, Propositional dynamic logic of flowcharts, *Inf. and Cont.* **64**, 1985, pp. 119–135.

[KT89] D.C. Kozen and J. Tiuryn, Logics of programs, *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), North-Holland, Amsterdam, 1989. To appear.

[MF71] A.R. Meyer and M.J. Fischer, Economy of description by automata, grammars, and formal systems, *Proc. 12th IEEE Symp. Switching and Automata Theory*, 1971, pp. 188–191.

[Mil80] R. Milner, *A Calculus of Communicating Systems*, Lec. Notes in Comp. Sci. 94, Springer, 1980.

[Pnu77] A. Pnueli, The temporal logic of programs, *Proc. 18th IEEE Symp. Found. of Comp. Sci.*, 1977, pp. 46–57.

[PR89a] A. Pnueli and R. Rosner, On the synthesis of a reactive module, *Proc. 16th ACM Symp. Princ. of Prog. Lang.*, 1989, pp. 179–190.

[PR89b] A. Pnueli and R. Rosner, On the synthesis of an asynchronous reactive module, *Proc. 16th Int. Colloq. Aut. Lang. Prog.*, Lec. Notes in Comp. Sci. 372, Springer, 1989, pp. 652–671.

[Pra81] V.R. Pratt, Using graphs to understand PDL, *Proc. Workshop on Logics of Programs* (D.C. Kozen, ed.), Lec. Notes Comp. Sci. 131, Springer, 1981, pp. 387–396.

[Rei85] W. Reisig, *Petri Nets: An Introduction*, Springer, Berlin, 1985.

[SC85] A.P. Sistla and E.M. Clarke, The complexity of propositional linear temporal logic, *J. ACM* **32**, 1985, pp. 733–749.

[VS85] M.Y. Vardi and L.J. Stockmeyer, Improved upper and lower bounds for modal logics of programs, *Proc. 17th ACM Symp. Theory of Comp.*, 1985, pp. 240–251.

[VW83] M.Y. Vardi and P. Wolper, Yet another process logic, *Proc. Workshop on Logics of Programs* (E.M. Clarke and D.C. Kozen, eds.), Lec. Notes Comp. Sci. 164, Springer, 1983, pp. 501–512.

[WVS83] P. Wolper, M.Y. Vardi, and A.P. Sistla, Reasoning about infinite computation paths, *Proc. 24th IEEE Symp. Found. of Comp. Sci.*, 1983, pp. 185–194.

## Figures

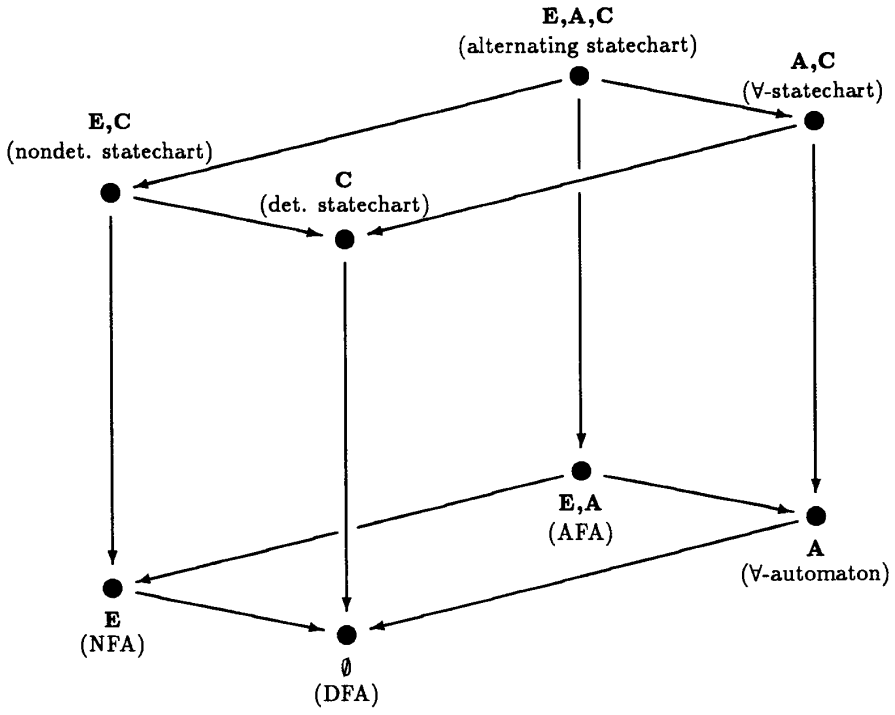


Figure 1: *Succinctness results for finite automata over* $\Sigma^*$ (see [DH89])

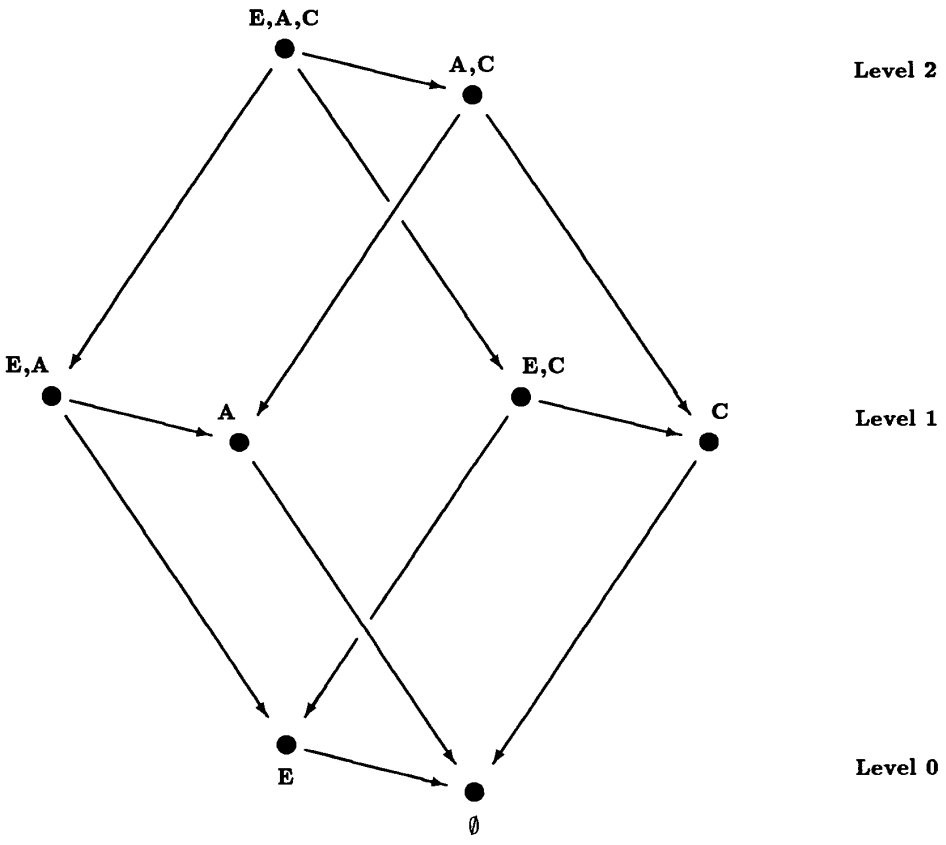


Figure 2: *The three levels of automata features*

| automata features | linear formalisms | | branching formalisms | |
|---|---|---|---|---|
| | FA | LTL | PDL | BTL, YAPL, Synthesis |
| level 2 | EXPSPACE | 2EXPSPACE | 3EXPTIME | 4EXPTIME |
| level 1 | PSPACE | EXPSPACE | 2EXPTIME | 3EXPTIME |
| level 0 | LOGSPACE | PSPACE | EXPTIME | 2EXPTIME |

Figure 3: *Summary of results* (all entries denote logspace completeness)