From Macro Plans to Automata Plans

Christer Bäckström [1] and **Anders Jonsson** [2] and **Peter Jonsson** [1]

Abstract. Macros have a long-standing role in planning as a tool for representing repeating subsequences of operators. Macros are useful both for guiding search towards a solution and for representing plans compactly. In this paper we introduce *automata plans* which consist of hierarchies of finite state automata. Automata plans can be viewed as an extension of macros that enables parametrization and branching. We provide several examples of the utility of automata plans, and prove that automata plans are strictly more expressive than macro plans. We also prove that automata plans admit polynomial-time sequential access of the operators in the underlying "flat" plan, and identify a subset of automata plans that admit polynomial-time random access. Finally, we compare automata plans with other representations allowing polynomial-time sequential access.

1 INTRODUCTION

In artificial intelligence planning, it is common to encounter planning problems, or sets of planning problems, whose solutions contain repeating subsequences of operators. Such planning problems present an opportunity to reduce the work of planning algorithms, either by maintaining a library of known repeating subsequences in the hope of reducing the search effort, or simply by obviating the need to store multiple copies of the same subsequence.

Macros have long been a popular tool in planning for representing repeating subsequences of operators. Several researchers have used macros in the context of search [4, 13, 15], where the idea is that longer subsequences of operators can help the search algorithm reach the goal in fewer steps. In some cases [13], the resulting search space can even be exponentially smaller than the original search space.

Macros can also be used as a compact representation of plans with repeating subsequences. Under certain conditions, a macro representation of a plan can be exponentially smaller than the plan itself. Sometimes it is even possible to generate a macro representation of an exponentially long plan in polynomial time [9, 11]. In the latter case, macros can be viewed as a tool for identifying classes of tractable planning problems.

In this paper we introduce the concept of *automata plans*, which are plans represented by hierarchies of finite state automata. Automata plans can be viewed as an extension of macro plans in two dimensions. The first dimension is that automata can be parametrized, making it possible to store families of repeating subsequences compactly, where a family consists of all possible assignments to the variables in the input of the automata. The second dimension is that automata can branch on input, making it possible to represent similar subsequences of operators and distinguish between them by providing different input to the automata.

[1] IDA, Linköping University, SE-581 83 Linköping, Sweden. Email: christer.backstrom@liu.se peter.jonsson@liu.se

[2] DTIC, Universitat Pompeu Fabra, 08018 Barcelona, Spain. Email: anders.jonsson@upf.edu

Finite state automata are commonly used to program behavior in robotics [5] and computer games [6]. In planning, researchers have proposed automata or automata-like representations of the entire planning problem [10, 14] and of individual variables [16]. There also exist algorithms that derive automata-like representations of plans automatically [3, 8] or from examples [17]. However, we are unaware of any application of hierarchical automata in planning.

In this paper we focus on the problem of plan representation, although we note that automata plans may also prove useful during search, or as a tool for defining novel classes of tractable planning problems as in the case of macros [11]. We show that automata plans offer a flexible and powerful way of representing plans, by providing several examples of how automata plans can be used to store plans compactly. We also compare automata plans to HTNs, which are similar in concept but usually viewed as a representation of planning problems as opposed to a plan representation.

We study the theoretical properties of automata plans and compare them to existing compact plan representations. We first show that automata plans are strictly more expressive than macro plans. We then relate automata plans to plan representations that allow polynomial-time random access or sequential access [1]. We show that a subclass of automata plans can be random accessed in polynomial time, and that representations that admit polynomial-time sequential access cannot be converted to automata plans in polynomial time.

The paper is organized as follows. Section 2 introduces notation that is used throughout. Section 3 presents the concept of automata plans, and Section 4 provides examples of their utility. In Sections 5–7 we prove several theoretical results regarding automata plans and related representations. Section 8 concludes with a discussion.

2 NOTATION

Let F be a set of fluents. A literal l is a positive or negative fluent. A set of literals L is *consistent* if $f \notin L$ or $\overline{f} \notin L$ for each $f \in F$. Let $L_+ = \{f \in F : f \in L\}$ and $L_- = \{f \in F : \overline{f} \in L\}$ be the sets of positive and negative fluents in L. A set of literals L *holds* in a state $s \subseteq F$ if $L_+ \subseteq s$ and $L_- \cap s = \varnothing$. Applying L to s results in a new state $(s \setminus L_-) \cup L_+$. Given a set X, let X^* and X^+ denote sequences and non-empty sequences of elements from X.

A STRIPS planning problem with negative pre-conditions is a tuple $\boldsymbol{p} = \langle F, O, I, G\rangle$, where F is a set of fluents, O a set of operators, $I \subseteq F$ an initial state, and $G \subseteq F$ a goal state. Each operator $o = \langle \mathrm{pre}(o), \mathrm{post}(o)\rangle \in O$ has a pre-condition $\mathrm{pre}(o)$ and a post-condition $\mathrm{post}(o)$, both consistent sets of literals. A *plan* for $\boldsymbol{p}$ is a sequence of operators $\omega = \langle o_1, \ldots, o_k\rangle$ such that, for each $1 \leq i \leq k$, $\mathrm{pre}(o_i)$ holds following the application of $o_1, \ldots, o_{i-1}$ to I. We say that ω *solves* $\boldsymbol{p}$ if G holds after applying $o_1, \ldots, o_k$ to I. Given two sequences ω and ω', let $\omega;\omega'$ denote their concatenation.

We also define an untyped STRIPS planning domain as a tuple $\boldsymbol{d} = \langle P, A\rangle$, where P is a set of predicates and A is a set of actions.

Each predicate $p \in P$ and action $a \in A$ has an associated number of parameters $n(p)$ and $n(a)$, respectively. The pre- and post-condition of an action a consist of sets of (positive or negative) predicates, each with an associated function from its parameters to $\{1, \ldots, n(a)\}$.

In this context, a STRIPS planning problem is induced by a tuple $\langle \Lambda, I, G \rangle$, where Λ is a set of objects that implicitly defines sets of fluents F and operators O by assigning objects to parameters of predicates in P and actions in A, respectively. Each pre- and post-condition of an operator $a(\lambda_1, \ldots, \lambda_{n(a)}) \in O$, where $\lambda_j \in \Lambda$ for each $1 \leq j \leq n(a)$, is given by $p(\lambda_{\varphi(1)}, \ldots \lambda_{\varphi(n(p))}) \in F$, where φ is the function from p's parameters to a's parameters.

Note that, for each predicate $p \in P$ and action $a \in A$ of a planning domain, the planning problem induced by $\langle \Lambda, I, G \rangle$ has $|\Lambda|^{n(p)}$ fluents and $|\Lambda|^{n(a)}$ grounded operators. To avoid an exponential blowup in the size of the planning problem, we assume that $n(p)$ and $n(a)$ are constants that are independent of the size of $\boldsymbol{d} = \langle P, A \rangle$.

3 AUTOMATA PLANS

Let Σ be an alphabet, A a set of actions, and $\mathbf{M}$ a set of automata. Also let $A_\Sigma = \{a[x] : a \in A, x \in \Sigma^*\}$ and $\mathbf{M}_\Sigma = \{M[x] : M \in \mathbf{M}, x \in \Sigma^*\}$. Intuitively, A_Σ corresponds to operators and $\mathbf{M}_\Sigma$ to automata calls. An *automaton* is a tuple $M = \langle G, s_I, s_A \rangle$, where

- $G = (S, E)$ is a graph on a set of states S,
- $s_I \in S$ is the initial state,
- $s_A \in S$ is the accepting state.

Each edge $(s, t) \in E$ has an associated label c/u, where $c \in \Sigma \cup \{\epsilon\}$ is a *condition* and $u \in (A_\Sigma \cup \mathbf{M}_\Sigma)^*$ is a sequence of action symbols (i.e. operators and automata calls). Automata with more than one accepting state can easily be converted to automata with one accepting state, by adding a new accepting state s_A to S and an edge (s, s_A) with label $\epsilon/\langle\rangle$ from each former accepting state s.

The execution model for an automaton M consists of an input string $x \in \Sigma^*$, a current state s_C (initially set to s_I), an index k (initially set to 0), and a sequence of action symbols θ (initially empty). We only consider deterministic automata such that each state $s \in S$ has either no outgoing edge, exactly one outgoing edge with condition ϵ, or $|\Sigma|$ outgoing edges, each with a distinct condition $\sigma \in \Sigma$.

The execution of an automaton proceeds as follows. If s_C has a single outgoing edge (s_C, s) with label ϵ/u, s_C is set to s and u is appended to θ. If s_C has $|\Sigma|$ outgoing edges, the symbol $x[k]$ at index k of the input string x determines which edge to move along. If (s_C, s) is the outgoing edge with label $x[k]/u$, s_C is set to s, k is incremented, and u is appended to θ. If $s_C = s_A$ or s_C has no outgoing edges, execution stops. The result of executing an automaton M on input x is $\mathrm{Apply}(M, x) = \theta$ if $s_C = s_A$ when execution stops, and $\mathrm{Apply}(M, x) = \perp$ otherwise. We only consider automata such that $|\mathrm{Apply}(M, x)| \geq 1$ whenever $\mathrm{Apply}(M, x) \neq \perp$.

Note that our definition forces automata to process the symbols of the input string x in order. We do not, however, require automata to process all symbols of the input string, although it would be trivial to extend our definition to such automata by introducing $|\Sigma|$ edges from s_A to itself, each with label $\sigma/\langle\rangle$ for some $\sigma \in \Sigma$. In contrast, we allow the input strings x' of the action symbols $a'[x']$ and $M'[x']$ in edge labels to freely copy symbols from x in any order.

The *expansion graph* $G_\mathbf{M} = \langle \mathbf{M}, \prec \rangle$ is a directed graph where, for each pair $M, M' \in \mathbf{M}$, $M \prec M'$ if and only if the automata call $M'[x']$ appears in some edge label of M, for any $x' \in \Sigma^*$.

An *automata plan* is a 4-tuple $\mu = \langle \Sigma, A, \mathbf{M}, r \rangle$ where

- Σ, A, $\mathbf{M}$, and each automaton $M \in \mathbf{M}$ are defined as above,
- $G_\mathbf{M}$ is acyclic and its underlying undirected graph is connected,
- $r \in \mathbf{M}_\Sigma$.

We refer to r as the *root* of μ. We define the *expansion function* Exp on $(A_\Sigma \cup \mathbf{M}_\Sigma \cup \{\perp\})^* \cup \{\perp\}$ as follows:

1) $\mathrm{Exp}(\perp) = \perp$,
2) $\mathrm{Exp}(a[x]) = a[x]$ if $a[x] \in A_\Sigma$,
3) $\mathrm{Exp}(M[x]) = \mathrm{Exp}(\mathrm{Apply}(M, x))$ if $M[x] \in \mathbf{M}_\Sigma$,
4) $\mathrm{Exp}(u_1; \ldots; u_k) = \perp$ if $\mathrm{Exp}(u_i) = \perp$ for some $1 \leq i \leq k$,
5) $\mathrm{Exp}(u_1; \ldots; u_k) = \mathrm{Exp}(u_1); \ldots; \mathrm{Exp}(u_k)$ otherwise.

Lemma 1. *For each automata plan* $\mu = \langle \Sigma, A, \mathbf{M}, r \rangle$, $\mathrm{Exp}(M[x]) \in A_\Sigma^+ \cup \{\perp\}$ *for each* $M[x] \in \mathbf{M}_\Sigma$.

Proof. We prove the lemma for all automata plans $\mu = \langle \Sigma, A, \mathbf{M}, r \rangle$ and all choices of $M[x] \in \mathbf{M}_\Sigma$ by induction over $|\mathbf{M}|$. If $|\mathbf{M}| = 1$, since $G_\mathbf{M}$ is acyclic, $\mathrm{Apply}(M, x)$ is either $\perp$ or a sequence of operators in A_Σ^+. In either case, $\mathrm{Exp}(M[x]) = \mathrm{Apply}(M, x)$.

If $|\mathbf{M}| = n > 1$, $\mathrm{Apply}(M, x)$ is either $\perp$, in which case $\mathrm{Exp}(M[x]) = \perp$, or a sequence of action symbols $u_1; \ldots; u_k \in (A_\Sigma \cup (\mathbf{M} \setminus \{M\})_\Sigma)^+$. Since $|\mathbf{M} \setminus \{M\}| = n - 1$, by hypothesis of induction $\mathrm{Exp}(u_i) \in A_\Sigma^+ \cup \{\perp\}$ for each $1 \leq i \leq k$ such that $u_i = M'[x']$ is an automaton. On the other hand, if $u_i = a'[x']$ is an operator, $\mathrm{Exp}(u_i) = u_i \in A_\Sigma$. If $\mathrm{Exp}(u_i) = \perp$ for some $1 \leq i \leq k$, then $\mathrm{Exp}(M[x]) = \perp$, else $\mathrm{Exp}(M[x]) \in A_\Sigma^+$. □

An automata plan μ represents an operator sequence ω if and only if $\mathrm{Exp}(r) = \omega$. We remark that a macro plan is a special case of an automata plan with empty input strings and such that each automaton has a single edge from s_I to s_A with condition ϵ. We show that just as for macros, the operators represented by an automata plan can be sequentially accessed with polynomial delay.

Lemma 2. *Given a polynomial-size automata plan* μ *for a plan* ω*, we can output the operators of* ω *sequentially with polynomial delay.*

Proof sketch. We can output the operators represented by μ in sequential order by maintaining a stack of execution models (current input string x, current state s_C, current index k) for each automaton recursively called by the root automaton. Since we require the expansion of each automaton to contain at least one operator, an automaton never has to make more than one additional recursive call (that might propagate down to a leaf) to reach the next operator. Combined with the fact that the expansion graph $G_\mathbf{M}$ is acyclic and has polynomial size, and that the size of each automaton is polynomially bounded, we can always output the next operator in polynomial time. □

4 EXAMPLES

In this section we show several examples of the expressive power of automata plans. Just like macros, automata plans can compactly represent plans that are exponentially long. Figure 1 shows an automaton M_n for moving n discs from peg x to peg y via peg z in Towers of Hanoi (ToH). $\mathsf{A}_n[xy]$ is the action for moving disc n from x to y. For $n = 1$ the edge label should be $\epsilon/\langle \mathsf{A}_1[xy] \rangle$. It is not hard to show that the automata plan $\mu = \langle \{1, 2, 3\}, \{\mathsf{A}_1, \ldots, \mathsf{A}_N\}, \{\mathsf{M}_1, \ldots, \mathsf{M}_N\}, \mathsf{M}_N[132] \rangle$ is a plan for the ToH instance with N discs. Unlike macro solutions for ToH [11], the automata plan has a single automaton for each number n of discs.

The ability to parametrize automata also makes it possible to represent other types of plans compactly. Figure 2 shows an automaton

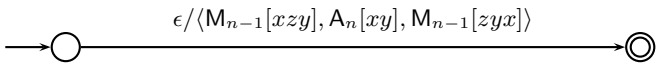

Figure 1. Automaton $\mathsf{M}_n[xyz]$ for Towers of Hanoi.

D for delivering a package in Logistics. The set of symbols Σ contains the objects of the problem: packages, airplanes, trucks, cities, and locations. The input to D is the package p to be delivered, an airplane a and two trucks t_f, t_t, two cities c_f and c_t, the current and target location l_f and l_t of the package, the current location v of the airplane and w, x of the two trucks, and intermediate airports y, z.

Figure 2 also shows the automaton T for moving a package using a truck. DT, LT, and UT stand for DriveTruck, LoadTruck, and UnloadTruck, respectively. The automaton A for moving a package using an airplane is almost identical. These three automata can be used to move any package between any two locations given the initial location of the different objects. Note that the validity of the solution depends on DT and FA (FlyAirplane) working properly even if the current and target location are the same.

The ability to branch on input also makes it possible for automata plans to represent more complex plans. For example, in contingent planning, a plan is a tree that branches on the observations made during execution. We can represent a contingent plan as an automata plan with symbols $\{0, 1\}$, such that the input string of the root automaton determines the chain of observations made during plan execution (the automata plan thus represents a single branch of the contingent plan tree). The meaning of each observation need not be known. In the worst case, the automata plan is as big as the original contingent plan. However, if the contingent plan contains subtrees that repeat themselves, each such subtree can be represented as a single automaton, causing the plan to be more compact. This is true even if the actions in two different subtrees have different parameters.

Finally, it is possible to define recursive automata that call themselves, branching on the input in a way that causes recursion to stop when the base case is reached. For example, we could modify the ToH example so that the solution is represented by a single recursive automaton M such that the number of discs n is part of the input string. Note, however, that some of the properties we later prove regarding automata plans do not apply if we allow recursive automata.

Automata plans are similar in concept to Hierarchical Task Networks (HTNs), in that both are hierarchical formalisms for planning. However, HTNs are usually viewed as a representation of planning problems, while automata plans are a representation of plans (i.e. solutions to planning problems). In this respect, an important difference is that HTNs need to keep track of the current state to search for a plan, something automata plans (as defined in this paper) cannot do. To keep track of the effect of operators on the current state, each automaton would need a mechanism for returning an output string that can be interpreted by its "parent" automata.

5 AUTOMATA PLANS AND MACROS

In this section we show that automata plans are strictly more expressive than macros. We first show that any macro plan can be converted to an automata plan. We then prove that there are small automata plans that cannot be converted to any macro plan of polynomial size.

Lemma 3. *Every* STRIPS *plan ω that can be represented using polynomially many macros has an automata plan μ of polynomial size.*

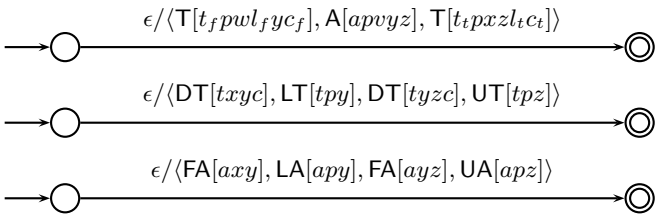

Figure 2. Automata $\mathsf{D}[pat_f t_t c_f c_t l_f l_t vwxyz]$ for delivering a package and $\mathsf{T}[tpxyzc]$, $\mathsf{A}[apxyz]$ for moving a package using a truck/airplane.

Proof. Replace every macro with an automaton having a single edge from s_I to s_A with condition ϵ. □

Definition 4. *Let R be a type of plan representation. Then the following problem is defined for R:*

Operator in Interval
INSTANCE: *A* STRIPS *problem $\boldsymbol{p}$, an R representation ρ of an operator sequence $\omega \in O^*$, an operator $o \in O$ and two integers i and j such that $1 \leq i < j \leq |\omega|$.*
QUESTION: *Does o occur in position k of ω for some $i \leq k \leq j$?*

Lemma 5. Operator in Interval *is in $\boldsymbol{P}$ for macro plans.*

Proof sketch. We can compute the length of the full expansion of all macros in polynomial time [2], by viewing macro plans as context free grammars with the non-terminal symbols being macros, the terminal symbols operators, the production rules the sequences associated with each macro, and the start symbol the root macro. Given i and j we can find all macros that are used in the expansion of the subsequence from i to j. It is sufficient to check if o occurs in the direct expansion of any of these macros. If the index i is in the middle of a macro expansion, we should only recursively check the macros that contain operators from i forward (the opposite is true for j). □

We next construct a planning instance $\boldsymbol{p}_n$ corresponding to the set of all 3SAT instances on n variables. We show that the solution to $\boldsymbol{p}_n$ can be represented by a small automata plan.

Construction 6. *For an arbitrary positive integer n, define the set $X_n = \{x_1, \ldots, x_n\}$ of atoms and the corresponding set $L_n = \{\ell_1, \ldots, \ell_{2n}\}$ of literals, where $\ell_{2i-1} = \overline{x_i}$ and $\ell_{2i} = x_i$ for each i. Also define a total order $<$ on L_n such that $\ell_i < \ell_j$ if and only if $i < j$. Let $C_n = \{c_1, \ldots, c_{m(n)}\}$ be the set of all 3-literal clauses over L_n, where each clause is represented as a tuple $c_k = \langle \ell_k^1, \ell_k^2, \ell_k^3\rangle$ such that $\ell_k^1, \ell_k^2, \ell_k^3 \in L_n$ and $\ell_k^1 \leq \ell_k^2 \leq \ell_k^3$.*

Construct a STRIPS *instance $\boldsymbol{p}_n = \langle F_n, O_n, I_n, G_n\rangle$, where $F_n = \{fe, fx, fs, sat, e_1, \ldots, e_{m(n)}, x_1, \ldots, x_n, v_0, \ldots, v_{m(n)}\}$, $I_n = \varnothing$, $G_n = \{fe, e_1, \ldots, e_{m(n)}, x_1, \ldots, x_n\}$, and O_n given by*

$$
\begin{aligned}
os &= \langle\{\overline{fe}, \overline{v_0}\}, \{v_0, fs\}\rangle \\
ol_k^1 &= \langle\{v_{k-1}, \overline{v_k}, e_k, \ell_k^1\}, \{v_k\}\rangle \\
ol_k^2 &= \langle\{v_{k-1}, \overline{v_k}, e_k, \overline{\ell_k^1}, \ell_k^2\}, \{v_k\}\rangle \\
ol_k^3 &= \langle\{v_{k-1}, \overline{v_k}, e_k, \overline{\ell_k^1}, \overline{\ell_k^2}, \ell_k^3\}, \{v_k\}\rangle \\
on_k &= \langle\{v_{k-1}, \overline{v_k}, e_k, \overline{\ell_k^1}, \overline{\ell_k^2}, \overline{\ell_k^3}\}, \{v_k, \overline{fs}\}\rangle \\
ov_k &= \langle\{v_{k-1}, \overline{e_k}\}, \{v_k\}\rangle \\
ot &= \langle\{v_{m(n)}, fs\}, \{fx, \overline{v_0}, \ldots, \overline{v_{m(n)}}, sat\}\rangle \\
of &= \langle\{v_{m(n)}, \overline{fs}\}, \{fx, \overline{v_0}, \ldots, \overline{v_{m(n)}}\}\rangle \\
ox_j &= \langle\{fx, \overline{x_j}, x_{j+1}, \ldots, x_n\}, \{fe, \overline{fx}, x_j, \overline{x_{j+1}}, \ldots, \overline{x_n}\}\rangle \\
oe_i &= \langle\{fe, x_1, \ldots, x_n, \overline{e_i}, e_{i+1}, \ldots, e_{m(n)}\}, \\
&\qquad \{\overline{fe}, \overline{sat}, \overline{x_1}, \ldots, \overline{x_n}, e_i, \overline{e_{i+1}}, \ldots, \overline{e_{m(n)}}\}\rangle
\end{aligned}
$$

Lemma 7. *For each positive integer n, the* STRIPS *instance $\boldsymbol{p}_n$ according to Construction 6 always has a unique plan $\omega_n = \langle o_1, \ldots, o_h \rangle$ with the following property: For every 3SAT instance $\boldsymbol{s}$ with n variables there are two polynomial-time computable indices i and j such that $\boldsymbol{s}$ is satisfiable if and only if the subplan $o_i, \ldots, o_j$ contains one or more occurrences of operator ot.*

Proof sketch. The instance $\boldsymbol{p}_n$ has a unique solution ω_n of the form

$$\begin{aligned} \omega_n &= E_0, oe, E_1, oe, \ldots, oe, E_{2^{m(n)}-1}, \\ E_i &= V_i^0, ox, V_i^1, ox, \ldots, ox, V_i^{2^n-1}, \\ V_i &= os, oy_1, oy_2, \ldots, oy_{m(n)}, oz. \end{aligned}$$

The variables $e_1, \ldots, e_{m(n)}$ and $x_1, \ldots, x_n$ are used as two binary counters e and x and the plan can be viewed as a nested loop. Each oe operator is a deterministic choice among $oe_1, \ldots, oe_{m(n)}$, and the same holds for ox. The outer loop enumerates all values from 0 to $2^{m(n)} - 1$ for e. There is one variable e_i for each clause in C_n, so this loop enumerates all 3SAT instances over X_n. That is, each E_i block corresponds to a unique 3SAT instance $\mathbf{s}_i$.

For each such instance, the inner loop enumerates all possible assignments to the variables in X_n. There is a V_i^j block for each assignment whose purpose is to check if $\mathbf{s}_i$ is satisfied for the current assignment. A V_i^j block contains exactly one operator oy_k for each of the $m(n)$ clauses, checking each of the clauses in order. If clause c_k is not "enabled" (that is, e_k is false) then $oy_k = ov_k$ which "skips over" the clause. Otherwise, oy_k = on_k if the clause is not satisfied in the current assignment x, and either of ol_k^1, ol_k^2, ol_k^3 if it is satisfied. Note that the latter three operators are mutually exclusive so the choice is deterministic. Each V_i^j block ends with ot if all enabled clauses were satisfied for the current assignment and of otherwise.

The variable fs keeps track of whether all clauses were satisfied. The variable sat is false at the start of every E_i block and is true at the end if and only if all clauses were satisfied for some assignment to x. The only action that makes sat true is ot, so $\mathbf{s}_i$ is satisfiable if and only if ot occurs in block E_i. Since the plan has a regular structure and all blocks of the same type have the same length, it is trivial to compute the indices for the start and end of an E_i block. □

Note that the variable sat is not part of any precondition or the goal; it is the operator ot itself that we use in the proof of the next lemma.

Lemma 8. *Unless the polynomial hierarchy collapses there is no polynomial p such that for every positive integer n, the plan ω_n for $\boldsymbol{p}_n$ according to Lemma 7 has a macro plan of size at most $p(||\boldsymbol{p}_n||)$.*

Proof. Suppose there is a polynomial p such that ω_n has a macro plan μ_n of size at most $p(||\boldsymbol{p}_n||)$ for each $n > 0$. Construct an advice-taking deterministic Turing machine M with input i on the form $I_n^i = \langle \boldsymbol{p}_n, i \rangle$, where n and i are integers such that $n > 0$ and $0 \leq i < 2^{m(n)}$. Let i be represented in binary using $m(n)$ bits. Then the input size $s_n = ||I_n^i||$ is strictly increasing in n and does not depend on i. Define the advice function a such that $a(s_n) = \mu_n$. Since M chooses advice based only on the size of its input the choice of advice depends entirely on n and is independent of i.

Given an arbitrary 3SAT instance $\mathbf{s}$ we can compute n and i such that $\mathbf{s}$ corresponds to block E_i in plan ω_n and thus compute I_n^i, all in polynomial time. Lemma 7 says that $\mathbf{s}$ is satisfiable if and only if block E_i of ω_n contains operator ot. Since the advice $a(s_n) = \mu_n$ is macro plan for ω_n and the advice is given to M for free, it follows from Lemma 5 that we can use M to decide satisfiability for an arbitrary 3SAT instance in polynomial time. However, that means **NP** $\subseteq$ **P/poly**, which is impossible unless the polynomial hierarchy collapses [12, Theorem 6.1], thus contradicting that p exists. □

Note that this proof does not make any assumption about the time complexity of computing μ_n, just that such a macro plan exists.

We say that an automata plan is *append restricted* if whenever an automaton with input string x calls another automaton it can only pass as input a constant string or x with a constant string appended. Note that this imposes a strong condition on automata.

Lemma 9. *There is a polynomial p such that for each $n > 0$, the plan ω_n for* STRIPS *instance $\boldsymbol{p}_n$ according to Lemma 7 has an automata plan ρ of size at most $p(||\boldsymbol{p}_n||)$, even if ρ is append restricted.*

Proof. For each $n > 0$, there exists an automata plan $\rho_n = \langle \{0,1\}, \{\mathsf{E}_i\} \cup \{\mathsf{X}_j\} \cup \{\mathsf{S}_k\} \cup \{\mathsf{U}_k\}, O_n, \mathsf{E}_1[\,] \rangle$, shown in Figure 3, that represents ω_n. Since $m(n) < 8n^3$ there is some polynomial p such that $||\rho_n|| \leq p(||\boldsymbol{p}_n||)$ for each n.

The automata plan works as follows. The automata $\mathsf{E}_1, \ldots, \mathsf{E}_{m(n)}$ enumerate all combinations of values for the e variables, and the automata $\mathsf{X}_1, \ldots, \mathsf{X}_n$ enumerate all combinations of values for the x variables. Whenever $\mathsf{S}_1[x]$ is called, the input string consists of $m(n)$ literals for the e variables in order, followed by n literals for the x variables in order.

Each clause c_k is verified by either S_k or U_k, which are almost identical. They first check the e literals to see if clause c_k is enabled (1) or not (0). If it is enabled, then they continue to the end of the e literals and then check the x literals. This can be done in a similar fashion since we have assumed that the literals of a clause are ordered. The purpose of the automata fragments η_1, η_2, ξ_1, ξ_2, and ξ_3 is simply to consume the correct number of input symbols. For reference, η_1 appears at the bottom of Figure 3.

The symbol ℓ_k^1 on labels in S_k or U_k represents $\underline{1}$ if $\ell_k^1 = x_i$ for some i, and 0 otherwise. The opposite is true for $\overline{\ell_k^1}$. If c_k is satisfied by one of ℓ_k^1, ℓ_k^2, ℓ_k^3, then $\mathsf{S}_k[x]$ calls $\mathsf{S}_{k+1}[x]$ to verify the next clause, otherwise it calls $\mathsf{U}_{k+1}[x]$, while $\mathsf{U}_k[x]$ always calls $\mathsf{U}_{k+1}[x]$.

Automaton $\mathsf{S}_{k+1}[x]$ is called if and only if $c_1, \ldots, c_k$ are either satisfied or disabled. As soon as we find a clause that is enabled but not satisfied we shift from S automata to U automata and can never shift back. This constitutes a simple memory to keep track of whether all clauses were satisfied or not. Note that the U automata must still check each clause and output the correct operator in order to represent the exact plan ω_n. The only difference is that the S "branch" and the U "branch" output different operators at the end. □

In Section 6 we show that we can randomly access the operators of the automata plan ρ_n from the proof of Lemma 9 in polynomial time. This does not contradict Lemma 8 (in fact, Operator in Interval is NP-hard for automata plans, which we leave without proof). Intuitively, even though we can access individual operators in polynomial time, the interval $[i, j]$ can be exponentially large, and for the given interval, each automaton is called with exponentially many different input strings, which makes it hard to determine whether a given operator is part of an interval.

Theorem 10. *Automata plans are strictly more compact than macro plans. This holds even for append restricted automata plans.*

Proof. Follows directly from Lemmas 3, 8, and 9. □

6 AUTOMATA WITH UNIFORM EXPANSION

A CRAR [1] is any polynomial representation of a plan that allows polynomial-time random access of its operators. Any plan that can

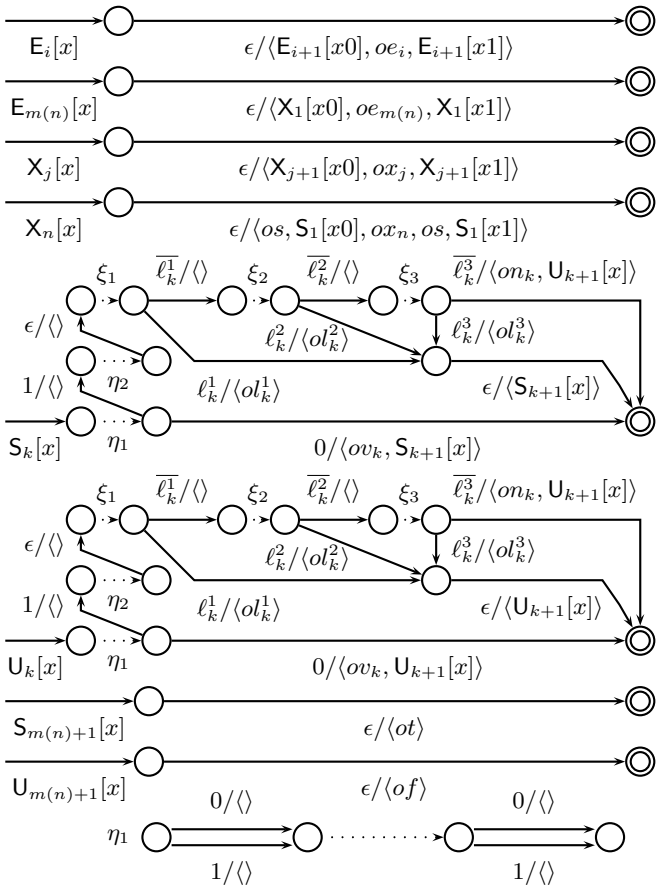

Figure 3. Automata used in the proof of Lemma 9.

be represented by a polynomial-size macro plan has this property [2] (again, by viewing macros as context free grammars). In this section we show that automata plans can also be random accessed in polynomial time, provided that they have a certain attribute that we define.

We say that an automata plan $\langle \Sigma, A, \mathbf{M}, r\rangle$ has *uniform expansion* if and only if for each $M \in \mathbf{M}$ there exists a number ℓ_M such that $|\mathrm{Exp}(M[x])| = \ell_M$ for each $x \in \Sigma^*$ such that $\mathrm{Exp}(M[x]) \neq \bot$. In this section we show that the operators of an automata plan with uniform expansion can be randomly accessed in polynomial time.

Note that all automata used in the proof of Lemma 9 have uniform expansion. For each S_k, $1 \leq k \leq m(n)$, and each x, $\mathrm{Apply}(\mathsf{S}_k, x)$ contains exactly one operator among $ov_k, ol_k^1, ol_k^2, ol_k^3, on_k$, followed by either $\mathsf{S}_{k+1}[x]$ or $\mathsf{U}_{k+1}[x]$. The same is true for U_k.

Theorem 11. *Let p and q be arbitrary polynomials. Assume X is a family of* STRIPS *instances satisfying the following conditions:*

1. *every solvable instance $\boldsymbol{p} \in X$ has a plan ω of length $\leq 2^{||\boldsymbol{p}||}$ with a corresponding automata plan $\mu = \langle \Sigma, A, \mathbf{M}, r\rangle$,*
2. *μ is of size $O(p(||\boldsymbol{p}||))$,*
3. *each $M \in \mathbf{M}$ has size $O(q(||\boldsymbol{p}||))$, and*
4. *μ has uniform expansion.*

Then each solvable instance $\boldsymbol{p} \in X$ has a plan ω with a CRAR.

Proof. Let $\boldsymbol{p}$ be a STRIPS instance with solution ω, represented by an automata plan $\mu = \langle \Sigma, A, \mathbf{M}, r\rangle$ that satisfies the requirements above. Since μ has uniform expansion there exist numbers

```
1  function Find(i,u)
2    if u is an operator
3      then return u
4    else (* u = M[x] *)
5      ⟨u_1, ..., u_k⟩ := Apply(M, x)
6      s := 0, j := 1
7      while s + ℓ(u_j) ≤ i do
8        s := s + ℓ(u_j), j := j + 1
9      return Find(i − s,u_j)
```

Figure 4. Algorithm for using an automata plan as a CRAR.

ℓ_M, $M \in \mathbf{M}$, such that $|\mathrm{Exp}(M[x])| = \ell_M$ for each $x \in \Sigma^*$ such that $\mathrm{Exp}(M[x]) \neq \bot$. Note that for each $M \in \mathbf{M}$, $\ell_M \leq 2^{||\boldsymbol{p}||}$, implying that ℓ_M can be represented by at most $||\boldsymbol{p}||$ bits. Without loss of generality, we assume that we have access to these numbers.

We prove that the recursive algorithm Find in Figure 4 has the following properties, by induction over the number of recursive calls:

1) for each $u \in \mathbf{M}_\Sigma$ such that $\mathrm{Exp}(u) = \langle a_1[x_1], \ldots, a_k[x_k]\rangle \neq \bot$, Find($i$, u) returns operator $a_i[x_i]$ for $1 \leq i \leq k$, and
2) for each $a[x] \in A_\Sigma$, Find(i, $a[x]$) returns $a[x]$.

Basis: If Find(i, u) does not call itself recursively, then u must be an operator. By definition, $\mathrm{Exp}(u) = u$ since $u \in A_\Sigma$.

Induction step: Suppose the claim holds when Find makes at most n recursive calls for some $n \geq 0$. Assume Find(i, $M[x]$) makes $n+1$ recursive calls. Let $\langle u_1, \ldots, u_k\rangle = \mathrm{Apply}(M, x)$ and, for each $1 \leq i \leq k$, $\ell(u_i) = 1$ if $u_i \in A_\Sigma$ and $\ell(u_i) = \ell_{M'}$ if $u_i = M'[x'] \in \mathbf{M}_\Sigma$. Lines 6–8 computes s and j such that either

1) $j = 1$, $s = 0$ and $i < \ell(u_1)$ or
2) $j > 1$, $s = \ell(u_1) + \ldots + \ell(u_{j-1}) \leq i < \ell(u_1) + \ldots + \ell(u_j)$.

By definition, $\mathrm{Exp}(u) = \mathrm{Exp}(u_1); \ldots; \mathrm{Exp}(u_k)$ so operator i in $\mathrm{Exp}(u)$ is operator $i - s$ in $\mathrm{Exp}(u_j)$. It follows from the induction hypthesis that the recursive call Find($i - s$, u_j) returns this operator.

To prove that Find runs in polynomial time, note that Find calls itself recursively at most once for each $M \in \mathbf{M}$ since $G_\mathbf{M}$ is acyclic. Moreover, the complexity of generating $\mathrm{Apply}(M, x)$, as well as its length k, are polynomial in $O(q(||\boldsymbol{p}||))$, the size of automaton M. The loop on line 7 runs at most k times. Since μ has size $O(p(||\boldsymbol{p}||))$ by assumption, Find is guaranteed to run in polynomial time.

We have showed that μ together with the procedure Find and the values ℓ_M, $M \in \mathbf{M}$, constitute a CRAR for ω. Since only $M \cdot ||\mu||$ bits are needed to represent the values and the procedure Find obviously runs in polynomial space (in the size of μ and consequently in $||\boldsymbol{p}||$), this CRAR is polynomial both in time and space. □

7 AUTOMATA AND SEQUENTIAL ACCESS

In this section we prove that CSARs cannot be converted to automata plans in polynomial time, unless an unlikely complexity-theoretic collapse occurs. A CSAR [1] is any polynomial representation of a plan that allows sequential access of the operators in polynomial time. Together with Lemma 2, this implies that automata plans and CSARs have different computational properties and are, thus, not equivalent notions of compact representations.

Definition 12. *Let R be a type of plan representation. Then the following problem is defined for R:*

Last Operator
INSTANCE: *A* STRIPS *instance* $\boldsymbol{p}$*, an* R *representation* ρ *of an operator sequence* $\omega \in O^*$*, and an operator* $o \in O$*.*
QUESTION: *Is* o *the last operator in* ω*?*

Theorem 13. *If there is a polynomial-time algorithm for transforming any* CSAR *into an equivalent automata plan, then* **P = PSPACE***.*

Proof. We prove the theorem by showing that Last Operator is in **P** for automata plans, but **PSPACE**-hard for CSARs. The given algorithm could solve Last Operator for CSARs in polynomial time, by transforming a CSAR to an automata plan and solving Last Operator for the automata plan. This is only possible if **P** = **PSPACE**.

We first show that Last Operator is in **P** for any automata plan $\mu = \langle \Sigma, \mathbf{M}, A, r \rangle$. For each $M[x] \in \mathbf{M}_\Sigma$, let $\mathrm{Apply}(M, x) = \langle u_1, \ldots, u_k \rangle$. The last operator in $\mathrm{Exp}(M[x])$ has to equal the last operator in $\mathrm{Exp}(u_k)$. We can thus define a recursive procedure for finding the last operator, and apply this procedure to the root automaton r. Since the expansion graph $G_\mathbf{M}$ is acyclic and has polynomial size, the number of calls to this recursive procedure is polynomially bounded. Since the automata have polynomial size, generating $\mathrm{Apply}(M, x)$ also takes polynomial time.

We next show that Last Operator is **PSPACE**-hard for CSARs. Bylander [7] proved that STRIPS planning is **PSPACE**-hard by presenting a polynomial-time reduction from polynomial-space DTM acceptance to STRIPS plan existence. The details of this reduction is not important in this proof but we note that it has the following property: if there is a path from the initial state I to some state s in the state-transition graph, at most one operator is applicable in s. This implies that if I has a solution, then there is a unique path in the state-transition graph from I to G.

We provide a polynomial-time reduction from polynomial-space DTM acceptance to Last Operator for CSARs. Given such a DTM, construct (in polynomial time) the corresponding STRIPS instance $\boldsymbol{p} = \langle F, O, I, G \rangle$ according to Bylander. Construct a new instance $\boldsymbol{p}' = \langle F', O', I, G' \rangle$ where $F' = F \cup \{NA_o \mid o \in O\} \cup \{OK\}$, $O' = O'' \cup \bigcup_{o \in O} X_o \cup \{yes, no\}$, and $G' = \{OK\}$. The variables NA_o will be used for indicating that operator o is Not Applicable. Define $O'' = \{\langle pre \cup \{\overline{NA_o} \mid o \in O\}, post \rangle \mid \langle pre, post \rangle \in O\}$. Given an operator $o \in O$, let X_o contain operator $\langle \{\overline{x}\}, \{NA_o\} \rangle$ for each literal $x \in \mathrm{pre}(o)$. Finally, let $yes = \langle G, \{OK\} \rangle$ and $no = \langle \{NA_o \mid o \in O\}, \{OK\} \rangle$.

If the DTM does not accept its input, then there is a path (using operators in O'' only) in the state transition graph from I to some state s where no operator in O'' is applicable. In state s, at least one operator in each set X_o is applicable so we can make all NA_o variables true and reach the goal state G' by applying operator no. If the DTM accepts its input, then there is a path from I to G using operators in O'' only. Furthermore, there is no state on this path where at least one operator in each X_o is applicable. Consequently, there is only one path from I to G' and this path ends with the operator yes.

Finally, we note that there is a simple polynomial CSAR for $\boldsymbol{p}'$. This CSAR selects the only applicable operator in O'', as long as such an operator exists, and an applicable operator in X_o otherwise, for some $o \in O$. The CSAR finishes with operator yes or no. We have thus shown that Last Operator is **PSPACE**-hard for CSARs, which concludes the proof of the theorem. □

8 CONCLUSION

We have introduced the novel concept of automata plans, i.e. plans represented by hierarchies of finite state automata. Automata plans extend macro plans by allowing parametrization and branching, and can be used to represent solutions to a variety of planning problems. We have showed that automata plans are strictly more expressive than macro plans, and related automata plans to the recent concepts of polynomial-time random access and sequential access.

Out of several possible extensions, possibly the most interesting one is to endow automata with the ability to produce output other than the sequence of action symbols. This could be used to keep track of the current state in automata plans, by updating the state each time an operator is encountered and returning the updated state to the previous automaton on the call stack. The ability to branch on the current state would cause automata plans to resemble HTNs, and further research is needed to clarify similarities and differences.

ACKNOWLEDGMENTS

A. Jonsson is partially supported by grants TIN2009-10232, MICINN, Spain, and EC-7PM-SpaceBook.

REFERENCES

[1] C. Bäckström and P. Jonsson, 'Limits for Compact Representation of Plans', in *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, (2011).
[2] P. Bille, G. Landau, R. Raman, K. Sadakane, S. Satti, and O. Weimann, 'Random access to grammar-compressed strings', in *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 373–389, (2011).
[3] B. Bonet, H. Palacios, and H. Geffner, 'Automatic Derivation of Finite-State Machines for Behavior Control', in *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI)*, (2010).
[4] A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer, 'Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators', *Journal of Artificial Intelligence Research*, **24**, 581–621, (2005).
[5] R. Brooks, 'A robot that walks; emergent behaviours from a carefully evolved network', *Neural Computation*, **1**, 253–262, (1989).
[6] M. Buckland, *Programming Game AI by Example*, Wordware Publishing, Inc, 2004.
[7] T. Bylander, 'The computational complexity of propositional STRIPS planning', *Artificial Intelligence*, **69**, 165–204, (1994).
[8] A. Cimatti, M. Roveri, and P. Traverso, 'Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains', in *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, pp. 875–881, (1998).
[9] O. Giménez and A. Jonsson, 'The Complexity of Planning Problems with Simple Causal Graphs', *Journal of Artificial Intelligence Research*, **31**, 319–351, (2008).
[10] S. Hickmott, J. Rintanen, S. Thiébaux, and L. White, 'Planning via Petri Net Unfolding', in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1904–1911, (2007).
[11] A. Jonsson, 'The Role of Macros in Tractable Planning', *Journal of Artificial Intelligence Research*, **36**, 471–511, (2009).
[12] R. Karp and R. Lipton, 'Some connections between nonuniform and uniform complexity classes', in *Proceedings of the 12th ACM Symposium on Theory of Computing (STOC)*, pp. 302–309, (1980).
[13] R. Korf, 'Planning as search: A quantitative approach', *Artificial Intelligence*, **33(1)**, 65–88, (1987).
[14] S. LaValle, *Planning Algorithms*, Cambridge Press, 2006.
[15] S. Minton, 'Selectively generalizing plans for problem-solving', in *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 596–599, (1985).
[16] D. Toropila and R. Barták, 'Using Finite-State Automata to Model and Solve Planning Problems', in *Proceedings of the 11th Italian AI Symposium on Artificial Intelligence (AI*IA)*, pp. 183–189, (2010).
[17] E. Winner and M. Veloso, 'DISTILL: Towards learning domain-specific planners by example', in *Proceedings of the 20th International Conference on Machine Learning (ICML)*, pp. 800–807, (2003).