A Correctness Result for Reasoning about One-Dimensional Planning Problems

Yuxiao Hu and **Hector J. Levesque**
Department of Computer Science
University of Toronto
Toronto, Ontario M5S 3G4, Canada
{yuxiao, hector} @cs.toronto.edu

Abstract

A plan with rich control structures like branches and loops can usually serve as a general solution that solves multiple planning instances in a domain. However, the correctness of such generalized plans is non-trivial to define and verify, especially when it comes to whether or not a plan works for all of the infinitely many instances of the problem. In this paper, we give a precise definition of a generalized plan representation called an FSA plan, with its semantics defined in the situation calculus. Based on this, we identify a class of infinite planning problems, which we call one-dimensional (1d), and prove a correctness result that 1d problems can be verified by finite means. We show that this theoretical result leads to a practical algorithm that does this verification practically, and a planner based on this verification algorithm efficiently generates provably correct plans for 1d problems.

Introduction

Planning with rich control structures like branches and loops is drawing increasing attention from the AI community (Manna and Waldinger 1987; Levesque 2005; Srivastava, Immerman, and Zilberstein 2008). One advantage of the resulting generalized plan is that the same plan can work in different initial states with very large numbers of objects.

For example, suppose we have the following variant of the logistics problem: There are a number of objects at their source locations (office or home), and the goal is to move them all to their destinations with a truck. The available actions include moving the truck to a location, loading and unloading an object, finding the source and destination locations of an object, and checking whether all objects have been processed. Then, intuitively, the plan in Figure 1 achieves the goal no matter how many objects there are.

Unfortunately, it is not easy to see formally why this plan is *correct* for this problem. The reason is three-fold. First, we need a formal definition of the plan representation; second, we must be precise about what it means for a plan to be correct; finally, we want to be able to conclude that it is correct for all of the infinitely many possible objects and source-destination combinations.

In this paper, we propose a solution to these in terms of an *FSA plan*, a generalized plan representation inspired by

Copyright © 2010, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Figure 1: An example plan for the logistic problem

and shown more expressive than robot programs (Levesque 1996). The semantics of FSA plans is declaratively defined in the situation calculus, so we have a foundation to analyze planning problems, plans, and their correctness. This notion of correctness, although precise and general, has the disadvantage that it uses *second-order logic* and is thus intractable in general. The intractability is inevitable, since FSA plans, like robot programs, are Turing complete in sufficiently expressive action domains (Lin and Levesque 1998).

However, for restricted action domains, we can do much better. A number of real-world problems involve independently processing an unbounded number of objects: delivering packages according to their shipping labels (as in the logistics example above), pushing the buttons on a safe according to the digits of a combination, keeping or discarding eggs according to their smell, *etc*. In this paper, we identify a class of planning problems that includes these, based on the shape of their action theory. We call these action theories *one-dimensional* or *1d*. We introduce a verification procedure, which we prove is correct for all 1d problems. We also present empirical evidence that a planner based on this verification procedure efficiently generates provably correct plans on planning problems that appear to be beyond the algorithmic reach of other planners.

The rest of this paper is organized as follows. First, we review the situation calculus, robot programs and our variant, FSA plans. Based on this, we define 1d planning problems, and prove the main theorems on the correctness of finite reasoning about the correctness of FSA plans. We then present a verification algorithm based on this theoretical result, and show experimentally that a planner using it efficiently gener-

ates provably correct FSA plans. Finally, we sketch possible lines for future work, and conclude.

Problem and Plan Representation

In order to represent and reason about the planning problem sketched above, we need a formal action language. In this paper, we appeal to the situation calculus, although the results introduced here could be adapted to other formalisms like $\mathcal{A}$ (Gelfond and Lifschitz 1993), and the fluent calculus (Thielscher 1998).

The situation calculus is a first-order, multi-sorted logical language with limited second-order features for representing and reasoning about dynamical environments (McCarthy and Hayes 1969; Reiter 2001). Objects in the domain of the logic are of three disjoint sorts: *situation* for situations, *action* for actions and *object* for everything else. The only *situation* constant S_0 denotes the initial situation where no action has yet occurred, and $do(a, s)$ represents the situation after performing action a in situation s. We use $do([a_1, \cdots, a_n], s)$ as an abbreviation for $do(a_n, do(\cdots, do(a_1, s)))$. Functions (relations) whose value may vary from situation to situation are called functional (relational) *fluents*, and denoted by a function (relation) whose last argument is a situation term. Functions and relations whose value do not change across situations are called *rigids*. Without loss of generality, we assume that all fluents are functional. The special relation $Poss(a, s)$ states that action a is executable in situation s, and the function $\text{SR}(a, s)$ indicates the sensing result of a when performed in s. The latter is introduced by Scherl and Levesque (2003) to accommodate knowledge and sensing in the situation calculus. We assume that ordinary actions, not intended for sensing purposes, simply return a fixed value (ok). A formula ϕ is *uniform in s* if it does not mention *Poss*, SR, or any situation term other than s. We call a fluent formula ϕ with all situation arguments eliminated a *situation-suppressed* formula, and use $\phi[s]$ to denote the uniform formula with all situation arguments restored with term s.

The dynamics of a planning problem is formalized by a basic action theory (BAT) of the form

$$\mathcal{D} = \mathcal{FA} \cup \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_{sr} \cup \Sigma_0 \cup \Sigma_{una}$$

where

- $\mathcal{FA}$ is a set of domain-independent axioms defining the legal situations (Reiter 2001).
- Σ_{pre} is a set of action precondition axioms, one for each action symbol of the form $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$. For example, the following says that *load* is possible in s iff the truck is at the source location and not loaded:

  $$Poss(load, s) \equiv loc(s) = src(s) \wedge loaded(s) = \text{FALSE}.$$
- Σ_{post} is a set of successor state axioms (SSAs), one for each fluent symbol f of the form

  $$f(\vec{x}, do(a, s)) = y \equiv \Phi_f(\vec{x}, a, y, s).$$

  For example, the following says that the location of the truck after performing a in s is either the destination of *move* if a is a move action, or its old location if a is not:

  $$loc(do(a, s)) = x \equiv \exists y.\, x = y \wedge a = move(y) \vee x = loc(s) \wedge \neg\exists y.\, (a = move(y)).$$
- Σ_{sr} is a set of sensing result axioms, one for each sensing action of the form $\text{SR}(A(\vec{x}), s) = r \equiv \Theta_A(\vec{x}, r, s)$. For example, the following says that *check* returns *yes* if the number of objects left is 0, and *no* otherwise:

  $$\text{SR}(check, s) = r \equiv r = yes \wedge left(s) = 0 \vee r = no \wedge left(s) \neq 0.$$
- Σ_0 is the initial knowledge base stating facts about S_0.
- Σ_{una} is a set of unique names axioms for actions.

Given the dynamics, the planning task is to find a plan that is executable in the given environment, and whose execution achieves the desired goal. Here, we only consider planning problems with final-state goals, defined as follows:

Definition 1 (The Planning Problem)**.** A *planning problem* is a pair $\langle \mathcal{D}, G \rangle$, where $\mathcal{D}$ is a basic action theory, and G is a situation-suppressed formula in the situation calculus.

Figure 2 shows the formal specification for the logistic problem above.[1] In this example, there are four fluents, *loc* (the location of the truck), *loaded* (the loading status of the truck), *parcels_left* (the number of parcels remaining to be delivered), and *misplaced* (whether any processed object has been misplaced). The initial value of *parcels_left* is non-negative but unknown, and the goal G is to make $parcels_left = 0$ while keeping $misplaced = \text{FALSE}$. There is a sensing action *check_done* that tells whether or not all parcels have been processed. In addition to the four fluents, we assume there are two rigid functions, *source* and *dest* that provide the shipping label for each object. For example, $dest(7) = home$ would mean that the destination of the 7th object is home. The values of these functions is not specified, but the sensing action *find_src* returns the source for the current object (according to *parcels_left*), and similarly for *find_dest*.

Since the number of objects, their sources, and their destinations are left open, a planning problem like this is not soluble with a sequential plan. We thus need a more general plan representation with branches and loops to handle all the contingencies.

One candidate representation is Levesque's *robot programs* (Levesque 1996), which is inductively defined with the empty program **nil**, sequential execution **seq**, branch **case**, and iteration **loop**. Recently, Hu and Levesque (2009) proposed an alternative representation called the *FSA plan*, and showed that it is more general than robot programs, in that all robot programs have an FSA plan representation but not vice versa. Moreover, they also presented a planning algorithm with this representation, which greatly outperforms the robot-program based KPLANNER (Levesque 2005). As a result, we appeal to FSA plans in this paper.

Definition 2 (FSA Plan (Hu and Levesque 2009))**.**
An *FSA plan* is a tuple $\langle \mathbf{Q}, \gamma, \delta, Q_0, Q_F \rangle$, where

- $\mathbf{Q}$ is a finite set of program states;
- $Q_0 \in \mathbf{Q}$ is an initial program state;

[1] In all example basic action theories presented in this paper, we omit the foundational axioms $\mathcal{FA}$, the unique names axioms Σ_{una}, and any domain closure axiom for objects (as introduced in Definition 6 below).

Precondition Axioms:

$Poss(move(x), s) \equiv \text{TRUE}$

$Poss(load, s) \equiv loc(s) = source(parcels_left(s)) \wedge loaded(s) = \text{FALSE}$

$Poss(unload, s) \equiv loaded(s) = \text{TRUE}$

$Poss(find_src, s) \equiv parcels_left(s) \neq 0$

$Poss(find_dest, s) \equiv parcels_left(s) \neq 0$

$Poss(check_done, s) \equiv \text{TRUE}$

Successor State Axioms:

$loc(do(a, s)) = x \equiv \exists y. x = y \wedge a = move(y) \vee x = loc(s) \wedge a \neq move(y)$

$misplaced(do(a, s)) = x \equiv x = \text{TRUE} \wedge a = unload \wedge loc(s) \neq dest(s) \vee x = misplaced(s) \wedge (a \neq unload \vee loc(s) = dest(s))$

$loaded(do(a, s)) = x \equiv x = \text{TRUE} \wedge a = load \vee x = \text{FALSE} \wedge a = unload \vee x = loaded(s) \wedge a \neq load \wedge a \neq unload$

$parcels_left(do(a, s)) = x \equiv x = parcels_left(s) - 1 \wedge a = unload \vee x = parcels_left(s) \wedge a \neq unload$

Sensing Result Axioms:

$\text{SR}(find_src, s) = r \equiv source(parcels_left(s)) = r$

$\text{SR}(find_dest, s) = r \equiv dest(parcels_left(s)) = r$

$\text{SR}(check_done, s) = r \equiv r = yes \wedge parcels_left(s) = 0 \vee r = no \wedge parcels_left(s) \neq 0$

$\text{SR}(move(x), s) = r \equiv r = ok$

$\text{SR}(load, s) = r \equiv r = ok$

$\text{SR}(unload, s) = r \equiv r = ok$

Initial Situation Axiom:

$\forall n. (source(n) = home \vee source(n) = office) \wedge \forall n. (dest(n) = home \vee dest(n) = office) \wedge loc(S_0) = home \wedge loaded(S_0) = \text{FALSE} \wedge parcels_left(S_0) \geq 0 \wedge misplaced(S_0) = \text{FALSE}$

Goal Condition:

$parcels_left = 0 \wedge misplaced = \text{FALSE}$

Figure 2: Axiomatization of logistic in the situation calculus

- $Q_F \in \mathbf{Q}$ is a final program state;
- $\gamma : \mathbf{Q}^- \rightarrow \mathbf{A}$ is a function, where $\mathbf{Q}^- = \mathbf{Q} \setminus \{Q_F\}$ and $\mathbf{A}$ is the set of primitive actions;
- $\delta : \mathbf{Q}^- \times \mathbf{R} \rightarrow \mathbf{Q}$ is a function, where $\mathbf{R}$ is the set of sensing results, that specifies the program state to transition to for each non-final state and valid sensing result for the associated action.

The execution of an FSA plan starts from $q = Q_0$, and executes the action $\gamma(q)$ associated with program state q. On observing sensing result r, it transitions to the new program state $\delta(q, r)$. This repeats until Q_F is reached.

FSA plans can be visualized graphically, where every node q in the graph is a program state, labeled with its associated action $\gamma(q)$. A directed edge labeled with r exists between q_1 and q_2 iff $\delta(q_1, r) = q_2$. The initial state Q_0 is denoted by an arrow pointing to it, and the final state Q_F by a double border. Figure 1 illustrates an FSA plan for logistic.

In order to represent FSA plans in the situation calculus, we assume that there is a sub-sort of *object* called *program-state*, with Q_0 and Q_F being two constants of this sort, and two rigid function symbols γ and δ. We use a set of sentences *FSA* to axiomatize the plan:

Definition 3. *FSA* is a set of axioms consisting of

1. Domain closure axiom for program states
   $(\forall q). \{q = Q_0 \vee q = Q_1 \vee \cdots \vee q = Q_n \vee q = Q_F\}$;
2. Unique names axioms for program states
   $Q_i \neq Q_j$ for $i \neq j$;
3. Action association axioms, one for each program state other than Q_F, of the form $\gamma(Q) = A$
4. Transition axioms of the form $\delta(Q, R) = Q'$

To capture the desired semantics, we introduce a transition relation $T^\star(q_1, s_1, q_2, s_2)$, which intuitively means that from program state q_1 and situation s_1, the FSA plan will reach q_2 and s_2 at some point during the execution. The formal definition is given in Definition 4.

Definition 4. We use $T^\star(q_1, s_1, q_2, s_2)$ as abbreviation for $(\forall T).\{\ldots \supset T(q_1, s_1, q_2, s_2)\}$, where the ellipsis is the conjunction of the universal closure of the following:

- $T(q, s, q, s)$
- $T(q, s, q'', s'') \wedge T(q'', s'', q', s') \supset T(q, s, q', s')$
- $\gamma(q) = a \wedge Poss(a, s) \wedge \text{SR}(a, s) = r \wedge \delta(q, r) = q' \supset T(q, s, q', do(a, s))$

Notice that this definition uses second-order quantification to ensure that $T^\star$ is the least predicate satisfying the three properties above. This essentially constrains the set of tuples satisfying $T^\star$ to be the reflexive transitive closure of the one-step transitions in the FSA plan.

With this transition relation, we can now characterize the correctness of FSA plans as follows.

Definition 5 (Plan correctness)**.** Given a planning problem $\langle \mathcal{D}, G \rangle$, a plan axiomatized by *FSA* is correct iff

$$\mathcal{D} \cup FSA \models \exists s. T^\star(Q_0, S_0, Q_F, s) \wedge G[s].$$

The definition essentially says that for an FSA plan to be correct, it must guarantee that for *any* model of $\mathcal{D}$, the execution of the FSA plan will reach the final state Q_F, and the goal is satisfied in the corresponding situation s. (In the case of logistic, a plan needs to work for any initial value of *parcels_left* and any value for the functions *source* and *dest*.)

This criterion of correctness is general and concise, but its second-order quantification and the potential existence of infinitely many models make it less useful algorithmically. Partly for this reason, existing iterative planners based on a similar representation, like KPLANNER (Levesque 2005) and FSAPLANNER (Hu and Levesque 2009), only come with a very weak correctness guarantee: although the generated plan tends to work for all problems in the domain, only

certain instances can be *proven* correct. It is thus interesting to ask whether we can generate provably correct plans for restricted classes of planning problems. The rest of this paper gives a positive answer to this question.

One-Dimensional Planning Problems

The major goal of this paper is to identify a class of planning problems that has a complete procedure to reason about the correctness of solution FSA plans. In this section, we define the class of *1d planning problems*, which is derived from the more restricted *finite problems*.

Definition 6. A planning problem $\langle \mathcal{D}, G\rangle$ is *finite* if $\mathcal{D}$ does not contain any predicate symbol other than *Poss* and equality, and the sort *object* has a domain closure axiom of the form

$$\forall x.\ x = o_1 \vee \cdots \vee x = o_l.$$

Intuitively, a finite problem has finitely many objects in the domain. Therefore, the number of ground fluents as well as their range is finite.

A 1d problem is like a finite problem except that there is a special distinguished fluent (called the *planning parameter*) that takes value from a new sort *natural number*, there is a finite set of distinguished actions (called the *decreasing actions*) which decrement the planning parameter, and some of the functions (called *sequence functions*) have an index argument of sort *natural number*.[2] In the case of the logistic example, the planning parameter is *parcels_left*, the decreasing action is *unload*, and the sequence functions are *source* and *dest*. The idea of a 1d planning problem is that the basic action theory is restricted in how it can use the planning parameter and sequence functions, as follows:

Definition 7. A planning problem $\langle \mathcal{D}', G'\rangle$ is *1d* with respect to an integer-valued fluent p, if there is a finite problem $\langle \mathcal{D}, G\rangle$ whose functions include fluent f_0 and rigids $f_1, \cdots, f_m$, and whose actions include $A_1, \cdots, A_d$, such that $\langle \mathcal{D}', G'\rangle$ is derived from $\langle \mathcal{D}, G\rangle$ as follows:

1. Replace the fluent f_0 with a planning parameter p:
   (a) replace the SSA for f_0 by one for p of the form
   $$p(do(a, s)) = x \equiv x = p(s) - 1 \wedge Dec(a) \vee x = p(s) \wedge \neg Dec(a),$$
   where $Dec(a)$ stands for $(a = A_1 \vee \cdots \vee a = A_d)$;
   (b) replace all atomic formulas involving the term $f_0(s)$ in the Π, Φ, Θ and $G[s]$ formulas by $p(s) = 0$, where s is the free situation variable in those formulas;
   (c) remove all atomic formulas mentioning $f_0(S_0)$ in Σ_0, and add $p(S_0) \geq 0$ instead.
2. Replace the rigids $f_1, \cdots, f_m$ with sequence functions $h_1, \cdots, h_m$:
   (a) replace all terms f_i in the Π, Φ, Θ and $G[s]$ formulas by $h_i(p(s))$, where s is the free variable as above;
   (b) replace all f_i in Σ_0 with $h_i(n)$, where n is a universally quantified variable of sort *natural number*.

[2]For simplicity, we assume in this paper that all sequence functions are rigid, but it is not hard to prove that the definitions and theorems work for sequence fluents as well.

Observe, for example, that in a 1d problem, the occurrence of the integer planning parameter is limited to its own successor state axiom, in Σ_0, and as an argument to a sequence function. Any other use of it is to test whether it is 0. Similarly, we can only apply a sequence function to the current object as determined by the planning parameter (other than in Σ_0 where we must quantify over all natural numbers). This ensures that the objects can be accessed sequentially in descending order, and that they do not interact with one another. It is not hard to see that logistic conforms to these requirements.

Main Theorems

Given a planning problem and a candidate plan, an important reasoning task is to decide whether the plan is guaranteed to achieve the goal according to the action theory. In a 1d setting, we need to ensure that the plan achieves the goal no matter what values the planning parameter p and the sequence functions h_i take. Unfortunately, there are infinitely many values that need to be taken into account.

In this section, we prove a correctness result of the following form: if we can prove that a plan is correct under the assumption that $p(S_0) \leq N$ (for a constant N that we calculate), it will follow that the plan is also correct without this assumption. In other words, correctness of the plan for initial values of p up to N is sufficient.

In the following, we first present a few lemmas characterizing properties of 1d action theories, and then present two theorems that capture the intuitive idea above. The proofs to these lemmas and theorems are sketched in the Appendix.

The first property is that in the execution of an FSA plan, the planning parameter monotonically decreases, and visits all integers between the initial and final values of p.

Lemma 1. *Let M be a model of a 1d action theory $\mathcal{D}$ with planning parameter p. If for some $n, n' \in \mathbb{N}$,*

$$M \models T^\star(q, s, q', s') \wedge p(s) = n \wedge p(s') = n',$$

then $n \geq n'$, and for any n'' satisfying $n' < n'' \leq n$, there exist a constant q'' and a ground situation term s'', such that

$$M \models T^\star(q, s, q'', s'') \wedge T^\star(q'', s'', q', s') \wedge Dec(\gamma(q'')) \wedge p(s'') = n''.$$

During the execution of the plan, there may be multiple program states and situations where the planning parameter has the value n''. Here, $Dec(\gamma(q''))$ identifies a unique configuration where q'' is a decreasing state, *i.e.*, a state whose associated action is among $A_1, \cdots, A_d$.

The next few properties deal with replacing a situation in an interpretation with a similar situation in a different interpretation. This similarity measure is defined as follows.

Definition 8 ($\langle i, j\rangle$-similarity)**.** Let M_1 and M_2 be models of $\mathcal{D}$, s_1 and s_2 be ground situation terms, and i, j be natural numbers. *Situation s_1 in interpretation M_1 is $\langle i, j\rangle$-similar to s_2 in M_2*, denoted by $M_1\langle s_1, i\rangle \sim M_2\langle s_2, j\rangle$, if

- M_1 and M_2 have identical domain for *action*s and *object*s;
- for all rigid finite functions r, $r^{M_1} = r^{M_2}$;
- for all rigid sequence functions h, $h^{M_1}(i) = h^{M_2}(j)$;

- for all finite fluents f, $f^{M_1}(s_1^{M_1}) = f^{M_2}(s_2^{M_2})$;
- for p, $p^{M_1}(s_1^{M_1}) = i$ and $p^{M_2}(s_2^{M_2}) = j$.

Intuitively, the situations s_1 and s_2 are indistinguishable in their respective interpretations, except for the differences in the planning parameter, since all finite fluents, as well as sequence functions at the current indexes, take identical values. As a result, the execution of an FSA plan at a particular program state will be indistinguishable in the two cases.

The $\langle i, j\rangle$-similarity relation is commutative and transitive, as formalized in Lemma 2. The proof of this lemma simply follows from multiple applications of Definition 8.

Lemma 2.

1. *If $M_1\langle s_1, i\rangle \sim M_2\langle s_2, j\rangle$, then $M_2\langle s_2, j\rangle \sim M_1\langle s_1, i\rangle$.*
2. *If $M_1\langle s_1, i\rangle \sim M_2\langle s_2, j\rangle$ and $M_2\langle s_2, j\rangle \sim M_3\langle s_3, k\rangle$, then $M_1\langle s_1, i\rangle \sim M_3\langle s_3, k\rangle$.*

The $\langle i, j\rangle$-similarity relation only require the sequence functions to agree at indexes i and j, respectively. Sometimes, we want to compare, between models, the values of sequence functions at multiple indexes. For this purpose, we introduce the following ξ-relationship.

Definition 9. Let $\xi \subset \mathbb{N}\times\mathbb{N}$ be a set of pairs of natural numbers, and M_1 and M_2 be two models of $\mathcal{D}$. We say that M_1 *ξ-relates to* M_2, denoted by $M_1\{\xi\}M_2$, if for all sequence functions h and all $\langle i, j\rangle \in \xi$, $h^{M_1}(i) = h^{M_2}(j)$.

Notice that $\langle i, j\rangle$-similarity relates two situations (in their respective interpretations), whereas ξ-relationship compares two interpretations. Two interpretations can be ξ-related, even if the finite fluents and the planning parameters are very different.

Given the concepts of $\langle i, j\rangle$-similarity and ξ-relationship, the next lemma captures the following intuition: given a model M of $\mathcal{D}$, we can construct another model M' of $\mathcal{D}$, which is almost the same as M, but with different values for the planning parameter and sequence functions.

Definition 10. A binary relation R is *functional*, if for any x, y, z such that $\langle x, z\rangle \in R$ and $\langle y, z\rangle \in R$, $x = y$ holds.

Lemma 3. *For any 1d theory $\mathcal{D}$, if M is a model of $\mathcal{D}$ with $M \models p(S_0) = n$, then for any $n' \in \mathbb{N}$ and functional relation $\xi \subset \mathbb{N}\times\mathbb{N}$ such that $\langle n, n'\rangle \in \xi$, there is another model M' of $\mathcal{D}$ such that $M\{\xi\}M'$ and $M\langle S_0, n\rangle \sim M'\langle S_0, n'\rangle$.*

Intuitively, M' in Lemma 3 is similar to M, in the sense that the initial values of all finite fluents are identical in both interpretations, the planning parameters is n' instead of n, and the values of the sequence functions in M' at some indexes are mapped from their values in M at possibly other indexes according to ξ. Later in the proof of the theorems, we shall use this property and show that the (in)correctness of an FSA plan for a "larger" interpretation can be reduced to the (in)correctness for a "smaller" interpretation.

Finally, we are ready to present a property about the execution of an FSA plan in a 1d theory. Suppose we have two situations in two models where the planning parameter is positive in both, all sequence functions agree at these and smaller indexes, and all the other fluents have identical values. Then starting from the same plan state, the execution of the FSA plan will be the same in each model. Lemma 4 formalizes this intuition.

Lemma 4. *Let M_1 and M_2 be two models of a 1d action theory $\mathcal{D}$, and s_1 and s_2 be two situation terms. Suppose $M_1\langle s_1, n_1\rangle \sim M_2\langle s_2, n_2\rangle$ for $n_1, n_2 > k$, and $M_1\{\xi\}M_2$ where $\langle n_1 - i, n_2 - i\rangle \in \xi$ for all $0 < i \leq k$. For any states q, q' and action sequence σ such that $p(s_i') = n_i - k$, where $s_i' = do(\sigma, s_i)$, we have*

$$M_1 \models T^\star(q, s_1, q', s_1') \text{ iff } M_2 \models T^\star(q, s_2, q', s_2').$$

Furthermore, $M_1\langle s_1', n_1 - k\rangle \sim M_2\langle s_2', n_2 - k\rangle$.

A special case of Lemma 4 is when $n_1 = n_2$. Then s in M and s' in M' become isomorphic, that is, they are not only similar, but also have the same value for p. In this case, the execution trace will be identical, even after the planning parameter reaches 0. This leads to the following:

Corollary 1. *Let $M_1\langle s_1, n\rangle \sim M_2\langle s_2, n\rangle$ and $M_1\{\xi\}M_2$, where $\langle i, i\rangle \in \xi$ for all $0 \leq i \leq n$. For any constants q, q' and action sequence σ, let $s_i' = do(\sigma, s_i)$, then*

$$M_1 \models T^\star(q, s_1, q', s_1') \;\text{ iff }\; M_2 \models T^\star(q, s_2, q', s_2').$$

Moreover, $M_1\langle s_1', n'\rangle \sim M_2\langle s_2', n'\rangle$, for some n'.

With the properties above, we are ready to present the two main theorems of this paper.

Theorem 1. *Suppose $\langle \mathcal{D}, G\rangle$ is a 1d planning problem with planning parameter p, and that $\mathcal{D}$ contains FSA axioms for some plan. Let $N_0 = 2 + k_0 \cdot l^m$, where k_0 is the number of decreasing program states in the FSA plan, m is the total number of finite and sequence functions, and l is the total number of values that they can take. Then we have:*

If $\mathcal{D} \cup \{p(S_0) \leq N_0\} \models \exists s. T^\star(Q_0, S_0, Q_F, s) \wedge G[s]$,
then $\mathcal{D} \models \exists s. T^\star(Q_0, S_0, Q_F, s) \wedge G[s]$.

The intuition behind this theorem is that both the number of program states and the number of indistinguishable situations are finite, so if the initial planning parameter is large enough, we will discover two identical configurations during the execution of the FSA plan. This is illustrated by the two black dots in the original execution line in Figure 3, representing the configurations after executing γ_1 and γ_2 from the initial situation, respectively. This enables us to construct a new initial state with smaller p, where the execution of γ_1 directly leads to $\langle q, s_2\rangle$, so that γ_3 can follow and achieve the goal. In this way, the correctness of the FSA plan for a "larger" model can be reduced to that of a "smaller" model, which has been verified according to the assumptions of the theorem. The proof of Theorem 1 is elaborated in the Appendix.

The bound N_0 in this theorem is exponential in the number of ground functions, and thus can be extremely large even for relatively simple action theories.

In order to obtain a tighter bound, we can narrow down the number of values finite and sequence functions may take by observing their successor state axioms, instead of assuming that they range over all finite objects. Suppose function f_j

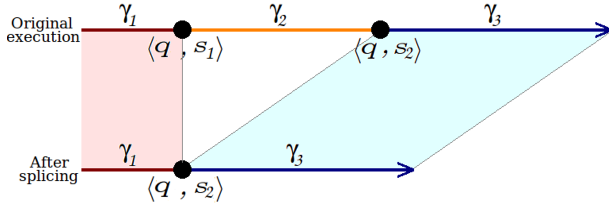

Figure 3: The intuition behind Theorem 1.

(or $h_j(n)$) only takes l_j different values, then we can prove a variant of Theorem 1 with a bound $N_0' = 2 + k_0 \prod_{j=1}^{m} l_j$, which is usually much smaller than N_0. We omit the details here, as N_0' is still exponential in the number of functions.

To obtain a more practical bound in a similar flavor, we introduce another theorem, where we do not declaratively specify the bound, but instead only spell out the necessary condition for an integer N_t to be a valid bound.

Theorem 2. *Suppose $\langle \mathcal{D}, G \rangle$ is a 1d planning problem with planning parameter p, and that $\mathcal{D}$ contains FSA axioms for some plan. Let $Seen(q, s)$ be the abbreviation for*

$$\exists s'.\, T^\star(Q_0, S_0, q, s') \wedge p(s') > 1 \wedge \bigwedge f(s) = f(s') \wedge \bigwedge h(p(s)) = h(p(s'))$$

where the first conjunction is over the finite fluents f, and the second over sequence functions h. Suppose $N_t > 0$ satisfies

$$\mathcal{D} \cup \{p(S_0) = N_t\} \models \forall q, s.\, T^\star(Q_0, S_0, q, s) \wedge p(s) = 1 \supset Seen(q, s)$$

Then we have the following:

If $\mathcal{D} \cup \{p(S_0) \leq N_t\} \models \exists s. T^\star(Q_0, S_0, Q_F, s) \wedge G[s]$,
then $\mathcal{D} \models \exists s. T^\star(Q_0, S_0, Q_F, s) \wedge G[s]$.

Intuitively, N_t has to be large enough so that a similar situation to the one that decrements the planning parameter from 1 to 0 occurs earlier in the execution trace. The proof sketch in the Appendix shows that this condition alone suffices to guarantee the existence of the splicing points illustrated in Figure 3 above.

Experimental Results

Given an FSA plan for a 1d planning problem, Theorems 1 and 2 suggest two algorithms to verify its correctness, which can then be used for plan generation.

Plan verification

To utilize the idea in Theorem 1, we only need to execute the FSA plan for $p(S_0) = 0, 1, \cdots, N_0$ (or up to N_0'). If the goal is achieved in all cases, then the FSA plan is correct in general, according to the theorem, and otherwise, it is incorrect. However, when the bound is large, this algorithm becomes impractical, since the number of possible initial worlds is exponential in the planning parameter. In the logistic example, for instance, each parcel has four possible source-destination combinations, so if we consider a problem containing 514 parcels (see the bounds for logistic below), the total number of possible combinations would be 4^{514}.

Fortunately, the bound N_0 is a very loose, worst-case estimate, and Theorem 2 offers a much tighter bound. It suggests an algorithm, shown in Figure 4, to incrementally verify the FSA plan, until a number satisfying the N_t criterion is reached.

```
function verify(P)
  Conf := {}; q := Q_0;
  for N_t = 1, 2, 3, ··· , N_0
    modified := FALSE;
    while q ≠ Q_F
      a := γ(q); c := ⟨q, v⃗⟩;*
      if a = A_i and p = 1 and c ∉ Conf then
        modified := TRUE;
        Conf := Conf ∪ {c};
      endIf
      execute action a
      if error then return FALSE
      forEach possible sensing result r
        q := δ(q, r);
    endWhile
    if modified = FALSE then return TRUE
  endFor
  return TRUE
```

*: $\vec{v}$ is the list values of all finite and sequence functions in the state.

Figure 4: The verification algorithm using Theorem 2

In practice, we start from $p(S_0) = 0, 1, 2, \cdots$, and run the FSA plan for each value. In each execution, whenever the planning parameter p decreases from 1 to 0, we record the program state, as well as the value of all finite and sequence functions in a table. If for some N_t, the execution for $p(S_0) = N_t$ does not add any new row into the table, then this N_t satisfies the criterion of Theorem 2, and thus, the plan is guaranteed to be correct in general. If the FSA plan fails before reaching such an N_t, then it is proved incorrect. Notice that when the plan is correct, the algorithm will terminate, since in the worst case, if we verify till $N_t = N_0$, then it is guaranteed correct by Theorem 1.

Plan generation

With the complete verification algorithms in hand, we can now generate plans that are correct for 1d planning problems. This is done by slightly modifying FSAPLANNER introduced by Hu and Levesque (2009).

The FSAPLANNER works by alternating between a *generation* and a *testing* phase: it generates plans for values of the planning parameter up to a lower bound, and then tests the resulting candidate plans for a higher value of the planning parameter. Although this appears to work for many applications, it has at least two serious problems: (1) the lower and higher bounds must be set by hand and (2) the only formal guarantee is that the plan works for the given values.

Precondition Axioms:

$Poss(look, s) \equiv \text{TRUE}$

$Poss(chop, s) \equiv chops_needed(s) \neq 0 \wedge axe(s) = out$

$Poss(store, s) \equiv axe(s) = out$

Successor State Axioms:

$axe(do(a, s)) = x \equiv x = stored \wedge a = store \vee$
$\quad x = axe(s) \wedge a \neq store$

$chops_needed(do(a, s)) = x \equiv$
$\quad x = chops_needed(s) - 1 \wedge a = chop \vee$
$\quad x = chops_needed(s) \wedge a \neq chop$

Sensing Result Axioms:

$\text{SR}(look, s) = r \equiv r = up \wedge chops_needed \neq 0 \vee$
$\quad r = down \wedge chops_needed = 0$

$\text{SR}(chop, s) = r \equiv r = ok$

$\text{SR}(store, s) = r \equiv r = ok$

Initial Situation Axiom:

$axe(S_0) = out \wedge chops_needed(S_0) \geq 0$

Goal Condition:

$axe = stored \wedge chops_needed = 0$

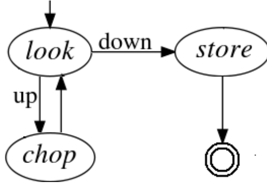

Figure 5: Axioms and a solution plan for treechop

The verification algorithms proposed above resolve both of these problems. The idea is to replace the test phase of FSAPLANNER by this verification. Then whenever a plan passes the testing phase, it is guaranteed to be correct. Notice that in both cases, the bounds N_0 and N_t can be obtained mechanically from the planning problem itself without manual intervention. The former only depends on the number of fluents and constants that appear in Σ_0 and Σ_{post}, whereas the latter is identified by table saturation.

We ran several experiments with variants of FSAPLANNER on four example domains: treechop, variegg, safe and logistic. (The first two are adapted from (Levesque 2005).)

treechop: The goal is to chop down a tree, and put away the axe. The number of chops needed to fell the tree is unknown, but a *look* action checks whether the tree is up or down. Intuitively, a solution involves first *look* and then *chop* whenever *up* is sensed. This repeats until *down* is sensed, in which case we *store* the axe, and are done. Figure 5 shows the problem definition and a solution FSA plan for treechop.

variegg: The goal is to get enough good eggs in the bowl from a sequence of eggs, each of which may be either good or bad, in order to make an omelette. A sensing action *check_bowl* tests if there are enough eggs in the bowl, and another *smell_dish* tests whether the egg in the dish is good or bad. Other actions include breaking an egg in the sequence to the dish, moving the egg from dish to bowl and dumping the dish. Figure 6 shows the problem definition and a solution FSA plan for variegg.

safe: The goal is to open a safe whose secret combination is written on a piece of paper as a binary string. The action *pick_paper* picks up the paper, and the sensing action *read* reads the first unmarked bit of the combination and return either 0 or 1, or "done" if the end of string is reached. The action $process(x)$ crosses the current bit on the paper, and pushes button x on the safe, where x can be 0 or 1. Finally, the actions *open* unlocks the safe if the correct combination is pushed, and jams the safe otherwise. Figure 7 shows the problem definition and a solution FSA plan for safe.

We summarize the parameters/bounds and computation times on the four sample problems in Figure 8. Here, N_{man} is the manually specified test parameter in the original FSAPLANNER, N_0 and N_0' are the exponential bounds obtained from Theorem 1, and N_t is the tighter bound based on table-saturation derived from Theorem 2. The corresponding CPU time to generate a correct plan is listed below each parameter/bound. (All runs are in SWI-Prolog under Ubuntu Linux 8.04 on a Intel Core2 3.0GHz CPU machine with 3.2GB memory.)

Comparing the three bounds that have guarantees, N_t is much tighter than N_0', which is in turn much tighter than N_0. The loose bounds N_0 and N_0' become impractical for larger planning problems like safe and logistic, whereas N_t is consistently small for all problems. Note that this planner can do even better than the original FSAPLANNER in cases where the manually specified test bound is overestimated. In sum, the table saturation based verification algorithm enables us to efficiently generate correctness-guaranteeing FSA plans for these 1d problems.

Related Work

The work most similar to ours in this paper is the theorem that "simple problems" can be finitely verified (Levesque 2005). However, the definition of simple problems is based on properties of the plan, and thus somewhat *ad hoc*. Our definition of 1d problems, in contrast, is rooted in the situation calculus, and therefore inherits its rigorous proofs.

Another closely related work is Lin's proof technique for goal achievability for rank 1 action theories by model subsumption (Lin 2008). His rank 1 action theory is more general than our 1d theory, but the type of plan that can be reasoned about is more restricted: plans with all actions located in a non-nested loop. Efficiently generating iterative plans is also outside of the scope of his work.

The planner Aranda (Srivastava, Immerman, and Zilberstein 2008) learns "generalized plans" that involve loops by using abstraction on an example plan. They prove that their

Precondition Axioms:

$Poss(next_to_dish, s) \equiv dish(s) = empty \land eggs_left \neq 0$

$Poss(dump_dish, s) \equiv dish(s) \neq empty$

$Poss(dish_to_bowl, s) \equiv dish(s) \neq empty$

$Poss(sniff_dish, s) \equiv dish(s) \neq empty$

$Poss(check_bowl, s) \equiv \text{TRUE}$

Successor State Axioms:

$$dish(do(a,s)) = x \equiv \\ x = empty \land (a = dish_to_bowl \lor a = dump_dish) \lor \\ x = egg_seq(eggs_left(s)) \land a = next_to_dish \lor \\ x = dish(s) \land a \neq dish_to_bowl \land a \neq dump_dish \land \\ a \neq next_to_dish$$

$$misplaced(do(a,s)) = x \equiv \\ x = yes \land \\ (a = dump_dish \land dish(s) = good_egg \lor \\ a = dish_to_bowl \land dish(s) = bad_egg) \lor \\ x = misplaced(s) \land \\ \neg(a = dump_dish \land dish(s) = good_egg \lor \\ a = dish_to_bowl \land dish(s) = bad_egg)$$

$$eggs_left(do(a,s)) = x \equiv \\ x = eggs_left(s) - 1 \land a = next_to_dish \lor \\ x = eggs_left(s) \land a \neq next_to_dish$$

Sensing Result Axioms:

$$\text{SR}(check_bowl, s) = r \equiv \\ r = enough_eggs \land eggs_left(s) = 0 \lor \\ r = need_eggs \land eggs_left(s) \neq 0$$

$$\text{SR}(sniff_dish, s) = r \equiv r = dish(s) \land eggs_left(s) \neq 0 \\ r = good_egg \land eggs_left(s) = 0 \lor$$

$\text{SR}(next_to_dish, s) = r \equiv r = ok$

$\text{SR}(dump_dish, s) = r \equiv r = ok$

$\text{SR}(dish_to_bowl, s) = r \equiv r = ok$

Initial Situation Axiom:

$$\forall n. (egg_seq(n) = good_egg \lor egg_seq(n) = bad_egg) \land \\ dish(S_0) = empty \land eggs_left(S_0) \geq 0 \land \\ misplaced(S_0) = no$$

Goal Condition:

$eggs_left = 0 \land misplaced = no$

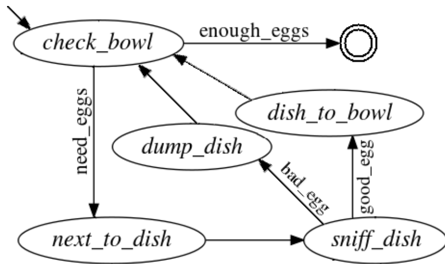

Figure 6: Axioms and a solution plan for variegg

Precondition Axioms:

$Poss(pick_paper, s) \equiv have_paper(s) = \text{FALSE}$

$Poss(open, s) \equiv safe(s) = locked$

$Poss(read, s) \equiv have_paper(s) = \text{TRUE}$

$Poss(process(x), s) \equiv buttons_left \neq 0$

Successor State Axioms:

$$have_paper(do(a,s)) = y \equiv \\ a = pick_paper \land y = \text{TRUE} \lor \\ a \neq pick_paper \land y = have_paper(s)$$

$$safe(do(a,s)) = y \equiv a = open \land buttons_left(s) = 0 \land \\ (mistkn(s) = \text{TRUE} \supset y = jammed) \land \\ (mistkn(s) = \text{FALSE} \supset y = unlocked) \lor \\ (a \neq open \lor buttons_left(s) \neq 0) \land y = safe(s)$$

$$mistkn(do(a,s)) = y \equiv \\ y = \text{TRUE} \land \\ \exists x.\, a = process(x) \land bit_seq(buttons_left(s)) \neq x \lor \\ y = mistkn(s) \land \\ \neg\exists x.\, a = process(x) \land bit_seq(buttons_left(s)) \neq x$$

$$buttons_left(do(a,s)) = y \equiv \\ \exists x.\, a = process(x) \land y = buttons_left(s) - 1 \lor \\ \neg\exists x.\, a = process(x) \land y = buttons_left(s)$$

Sensing Result Axiom:

$$\text{SR}(read, s) = r \equiv buttons_left(s) = 0 \land r = done \lor \\ buttons_left(s) \neq 0 \land r = bit_seq(buttons_left(s))$$

$\text{SR}(pick_paper, s) = r \equiv r = ok$

$\text{SR}(open, s) = r \equiv r = ok$

$\text{SR}(process(x), s) = r \equiv r = ok$

Initial Situation Axiom:

$$have_paper(S_0) = \text{FALSE} \land safe(S_0) = locked \land \\ mistkn(S_0) = \text{FALSE} \land buttons_left(S_0) \geq 0 \land \\ \forall n. \big(bit_seq(n) = 0 \lor bit_seq(n) = 1\big)$$

Goal Condition:

$safe = unlocked$

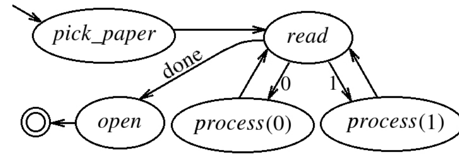

Figure 7: Axioms and a solution plan for safe

Problem	treechop	variegg	safe	logistic
N_{man}	100	6	4	5
Time (secs)	0.1	0.12	0.09	3.93
N_0	18	345	4098	514
Time (secs)	0.03	>1 day	>1 day	>1 day
N_0'	4	12	36	34
Time (secs)	0.01	8.68	>1 day	>1 day
N_t	2	3	2	2
Time (secs)	0.01	0.08	0.08	3.56

Figure 8: Comparison of FSAPLANNER using different verification modules

planner generates correct plans for problems in "extended-LL" domains. However, it not clear what sort of action theories can or cannot be characterized as extended-LL. It is thus interesting future work to compare the relative expressiveness between extended-LL and 1d problems, and identify a more general class that accommodates both formalisms.

There is also important work on planning in domains where loops are required but correctness in general is not considered at all. The planner loopDistill (Winner and Veloso 2007) learns from an example partial-order plan. Similarly, the planner introduced by Bonet, Palacios and Geffner (2009) synthesizes finite-state controllers via conformant planning. In both cases, the resulting plans can usually solve problems similar to the examples used to generate them, but under what conditions they will be applicable is not addressed.

Earlier work on deductive synthesis of iterative or recursive plans represents another approach to the problem based on theorem proving. For example, Manna and Waldinger (1987) finds recursive procedures to clear blocks in the classical blocks world, and the resulting plan comes with a strong correctness guarantee. Unfortunately, the price to pay is typically manual intervention (for example, to identify induction hypotheses) and poor performance. Magnusson and Doherty recently proposed to use heuristics to automatically generate induction hypotheses for temporally-extended maintenance goals (2008). However, their planner is incomplete, and for which subclass their approach is complete remains to be investigated.

Finally, there is a separate branch of research in model checking for automatically verifying correctness of computer programs (Clarke, Grumberg, and Peled 1999). It is concerned with correctness of programs in predefined computer languages instead of general action domains, and does not aim for program synthesis. However, results and techniques from this community may shed light on our goal of iterative plan verification and generation in the long run.

Conclusion and Future Work

In this paper, we identified a class of planning problems which we called 1d, and proved that plan correctness for unbounded 1d problems could be checked in a finite and practical way. Based on this theoretical result, we developed a variant of FSAPLANNER, and showed that it efficiently generates provably correct plans for 1d problems.

In the future, we intend to investigate planning problems beyond the 1d class. Consider, for example, the following:

> *We start with a stack A of blocks, with the same number of blue and red ones. We can pick up a block from stack A or B, and put a block on stack B or C. We can also sense when a stack is empty and the color of a block being held. The goal is to get all the blocks onto stack C, alternating in color, with red on the bottom.*

What makes this problem challenging is that we may need to put a block aside (onto stack B) and deal with any number of other blocks before we can finish with it. In a still more general example, consider the Towers of Hanoi. In this case, we spend almost all our time finding a place for disks that are not ready to be moved to their final location. In the future, we hope to develop finite techniques for such problems too.

References

Bonet, B.; Palacios, H.; and Geffner., H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proceedings of International Conference on Automated Planning and Scheduling*.

Clarke, E. M.; Grumberg, O.; and Peled, D. A. 1999. *Model Checking*. MIT Press.

Gelfond, M., and Lifschitz, V. 1993. Representing actions and change by logic programs. *Journal of Logic Programming* 301–323.

Hu, Y., and Levesque, H. 2009. Planning with loops: Some new results. In *ICAPS Workshop on Generalized Planning: Macros, Loops, Domain Control*.

Levesque, H. 1996. What is planning in the presence of sensing. In *Proceedings of National Conference on Artificial Intelligence*.

Levesque, H. 2005. Planning with loops. In *Proceedings of International Joint Conference on Artificial Intelligence*.

Lin, F., and Levesque, H. 1998. What robots can do: robot programs and effective achievability. *Artificial Intelligence*.

Lin, F. 2008. Proving goal achievability. In *Proceedings of International Conference on the Principles of Knowledge Representation and Reasoning*.

Magnusson, M., and Doherty, P. 2008. Deductive planning with inductive loops. In *Proceedings of National Conference on Artificial Intelligence*.

Manna, Z., and Waldinger, R. 1987. How to clear a block: a theory of plans. *Journal of Automated Reasoning* 3(4):343–377.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence* 463–502.

Pirri, F., and Reiter, R. 1999. Some contributions to the metatheory of the situation calculus. *Journal of the ACM* 46(3):261–325.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Scherl, R., and Levesque, H. 2003. Knowledge, action, and the frame problem. *Artificial Intelligence* 144.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2008. Learning generalized plans using abstract counting. In *Proceedings of National Conference on Artificial Intelligence*.

Thielscher, M. 1998. Introduction to the fluent calculus. *Electronic Transactions on Artificial Intelligence* 2(3-4):179–192.

Winner, E., and Veloso, M. 2007. LoopDISTILL: Learning domain-specific planners from example plans. In *Proceedings of ICAPS-07 Workshop on AI Planning and Learning*.

Appendix

Proof of Theorems

Proof of Lemma 1. This follows from the shape of the SSA for p, which constrains the actions to either leave the value of p unchanged or decrease it by 1. □

Proof of Lemma 3. Let M_0 be the sub-model of M with all objects involving non-initial situations removed. We need only construct a model M_0', which is the same as M_0 except that $M_0' \models p(S_0) = n'$ and the sequence functions h all satisfy this: for any k and any $\langle i, j\rangle \in \xi$, if $M_0 \models h(i, S_0) = k$ then $M_0' \models h(j, S_0) = k$. The existence of M' from M then follows by applying the Relative Satisfiability Theorem (Pirri and Reiter 1999). □

Proof of Lemma 4. By induction over the length of σ. □

Proof of Theorem 1. Suppose, for the sake of contradiction, that $\mathcal{D} \not\models \exists s.T^\star(Q_0, S_0, Q_F, s) \wedge G[s]$. Then there is a smallest $n > N_0$ such that for some model M of $\mathcal{D}$ we have $M \models p(S_0) = n \wedge \neg\big(\exists s.T^\star(Q_0, S_0, Q_F, s) \wedge G[s]\big)$.

By Lemma 3, there exists another model M_1 of $\mathcal{D}$ satisfying $M\langle S_0, n\rangle \sim M_1\langle S_0, n-1\rangle$ and $M\{\xi_1\}M_1$, where $\langle i, i-1\rangle \in \xi_1$ for all $1 \le i \le n$.

Since $M_1 \models p(S_0) = n-1 < n$, according to our assumption, $M_1 \models T^\star(Q_0, S_0, Q_F, s) \wedge G[s]$ for some situation term s. Based on the value of $p(s)$, there are two cases:

Case 1: $M_1 \models p(s) = n' > 0$:
By Lemma 4, $M \models T^\star(Q_0, S_0, Q_F, s)$, and furthermore $M_1\langle s, n'\rangle \sim M\langle s, n'+1\rangle$, so $M \models G[s]$. This contradicts the assumption that $M \not\models \exists s.T^\star(Q_0, S_0, Q_F, s) \wedge G[s]$.

Case 2: $M_1 \models p(s) = 0$:
By Lemma 1, there is an action sequence σ and decreasing state q such that

$$M_1 \models T^\star(Q_0, S_0, q, do(\sigma, S_0)) \wedge p(do(\sigma, S_0)) = 1.$$

By Lemma 4,

$$M \models T^\star(Q_0, S_0, q, do(\sigma, S_0)) \wedge p(do(\sigma, S_0)) = 2.$$

Since σ reduces p from n to 2 in M, by Lemma 1, it must contain more than $k_0 \cdot l^m$ decreasing actions. So there must be two points in the execution of σ with the same state: there is some program state q' and $\sigma = \alpha\beta\gamma$ such that

$$\begin{aligned} M \models\ & T^\star(Q_0, S_0, q', do(\alpha, S_0)) \ \wedge \\ & T^\star(q', do(\alpha, S_0), q', do(\alpha\beta, S_0)), \end{aligned}$$

where $M\langle do(\alpha, S_0), u\rangle \sim M\langle do(\alpha\beta, S_0), v\rangle\rangle$, for some $u > v$. Furthermore,

$$M \models \neg\exists s.T^\star(q', do(\alpha\beta, S_0), Q_F, s) \wedge G[s].$$

By Lemma 3 again, there is a model M_u of $\mathcal{D}$ such that $M\langle S_0, n\rangle \sim M_u\langle S_0, n-(u-v)\rangle$ and $M\{\xi_u\}M_u$, where $\langle i, i-(u-v)\rangle \in \xi_u$ for all $u \le i \le n$ and $\langle i, i\rangle \in \xi_u$ for all $0 \le i < v$. By Lemma 4 again, we have that

$$M_u \models T^\star(Q_0, S_0, q', do(\alpha, S_0)).$$

By Lemma 2, $M\langle do(\alpha\beta, S_0), v\rangle \sim M_u\langle do(\alpha, S_0), v\rangle$. Therefore, by Corollary 1,

$$M_u \models \neg\exists s.T^\star(q', do(\alpha, S_0), Q_F, s) \wedge G[s].$$

So we obtain

$$M_u \models p(S_0) = n-(u-v) \wedge \neg\exists s.T^\star(Q_0, S_0, Q_F, s) \wedge G[s],$$

which contradicts the assumption that $M \models p(S_0) = n$ has the smallest n that fails the FSA plan. □

Proof of Theorem 2. The proof is the same as the one for Theorem 1, except for the way we decompose σ.

Let M be the smallest interpretation failing the FSA plan, and $M\langle S_0, n\rangle \sim M_t\langle S_0, N_t\rangle$. Further let

$$M_t \models T^\star(Q_0, S_0, q, do(\sigma, S_0)) \wedge p(do(\sigma, S_0)) = 1$$

where q is a decreasing state, then by the constraint on N_t, there must be a decomposition $\sigma = \alpha\beta$ such that

$$\begin{aligned} M_t \models\ & T^\star(Q_0, S_0, q, do(\alpha, S_0)) \ \wedge \\ & T^\star(q, do(\alpha, S_0), q, do(\alpha\beta, S_0)), \end{aligned}$$

and $M_t\langle do(\alpha, S_0), u\rangle \sim M_t\langle do(\alpha\beta, S_0), v\rangle$, where $u > 1$ and $v = 1$. The rest of the proof follows as in Theorem 1. □