Planning with Equality

David Plaisted[1] and Stephan Schulz[2]

[1] Department of Computer Science, UNC Chapel Hill, Chapel Hill, NC 27599-3175, U.S.A., plaisted@cs.unc.edu
[2] DHBW Stuttgart, Fakultät Technik, 70174 Stuttgart, Germany, schulz@eprover.org

Abstract. An approach to planning in which states are represented as first-order terms is presented. Fluents can be computed from the term structure, and actions on the states correspond to rewrite rules on the terms. Actions that depend on or influence only a subset of the fluents can often be described as rewrite rules that operate on subterms of the terms. If actions are bidirectional, then efficient completion methods can be used to solve planning problems. Such completion methods first show that a plan exists and then can construct it explicitly. This approach is guaranteed to find a solution to a planning problem if one exists, but not necessarily an optimal one. Some ideas for optimizing plans found in this way are sketched. An implementation shows that this approach can sometimes show quickly that a very long plan exists. The time to find a plan is important as well as the cost of the plan. The time to construct the plan will be proportional to the length of the plan. Some examples are given, and an argument is given that this approach can be more efficient than other common approaches to planning. If this approach quickly show that a plan exists, then the plan has a succinct representation even though the plan might be very long. Also, even before constructing the entire plan, a term representing a state somewhere in the middle of the plan can be computed efficiently.

1 Introduction

The situation calculus permits reasoning about properties of situations that result from a given situation by sequences of actions [MH69]. In the original version of the situation calculus [MH69], the term "situations" refers to states, which represent the entire state of the world at a given time. Later [Rei91] the term situations is used to refer to sequences of states, each state obtained from the previous one by a specified action. We consider the original version of the situation calculus, and use the term "states" instead of "situations" to refer to the entire state of the world at a given time. In order to clarify the distinction, we use the term state calculus instead of situation calculus to refer to what McCarthy and Hayes would have called the situation calculus. In the original situation calculus of McCarthy and Hayes, states (which they called situatons) are represented explicitly by variables, and actions a map states s to states $do(a, s)$. Predicates and functions on a state are called *fluents*. A problem with the situation calculus

or any formalism for reasoning about actions is the necessity to include a large number of *frame axioms* that express the fact that actions do not influence many properties of a state. Since the early days of artificial intelligence research the frame problem has been studied, beginning with McCarthy and Hayes [MH69]. Lin [Lin08] has written a survey of the situation calculus.

Reiter [Rei91] proposed an approach to the frame problem that avoids the need to specify all of the frame axioms. The method of Reiter, foreshadowed by Haas [Haa87], Pednault [Ped89], Schubert [Sch90] and Davis [Dav90], defines situations to be sequences of states or actions, and essentially solves the frame problem by specifying that a change in the truth value of a fluent, caused by an action, is equivalent to a certain condition on the action. In this formalism, it is only necessary to list the actions that change each fluent, and it is not necessary to specify the frame axioms directly. If an action does not satisfy the condition, the fluent is not affected. In the following discussion the term "Reiter's formalism" will be used for simplicity even though others have also contributed to its development. The *fluent calculus* [Thi98] is another interesting approach to the frame problem. In this approach, a state is a conjunction of known facts. The fluent calculus has some similarities to our approach, but our approach makes more use of the subterm relation. Our approach expresses a planning problem as a pure equational theorem proving problem in first-order logic. It could easily be extended to conditional equations, giving more expressiveness.

Rewriting and completion based approaches and implementations of equational first-order logic have become highly efficient. By the known completeness of unfailing completion [BN98],[BDP89,HR87], completion based approaches, given enough time, are guaranteed to find a plan if one exists, but not necessarily an optimal plan. One cannot bound the time that completion based approaches may take, but our examples show that in some cases they can be very fast even for very long plans.

Because completion based approaches might not generate plans that are optimal or near optimal, it is important to consider methods to improve the plans they generate. It is possible to optimize the plan found by completion by locally replacing parts of it by more efficient sequences of actions. One way to improve such a plan is to create plans using several different termination orderings to guide completion. We can then simply accept the most attractive (e.g. the shortest) of the generated plans. These may of course be further optimised. Another optimization is to delete the portion of the plan between duplicated occurrences of the same state, if such exist.

General approaches to planning can guarantee near optimal plans, but even so the cost to use these approaches can be high. Planning can be seen as a search for a connecting path in a graph – ideally the shortest one. The states are nodes of the graph and the actions are the edges. The cost of an edge is the cost of the action. Then a path in the graph, corresponding to a sequence of actions, is a plan. An optimal plan is a shortest (least cost) path. Shortest path algorithms on graphs such as Dijkstra's algorithm [Dij59] generally at least have to read in all the nodes, so they take time at least proportional to the size of

the graph. For planning problems, the size of the graph is at least as large as the number of states, which is often exponential (in the length of the description of the problem). All of the examples given later have exponential search spaces. Thus one would expect that the worst case time for planning would also be exponential. In general, one would expect that finding an optimal path in an exponential graph would be a lot harder than just finding one path, which is what completion does. At least in the switches and Tower of Hanoi examples given below, completion finds a path in polynomial time, though the numbers of states are exponential. For the Tower of Hanoi problem, even the length of the plan is exponential. So we have good reason to believe that our approach would be faster than other plannng methods on these and similar examples, though the final plan might be longer. A* search [HNR68] can be much faster than Dijkstra's algorithm if good heuristics are available, but even A* search takes time proportional to the size of the graph if good heuristics are not available.

Concerning efficiency, the rewriting approach has the potential to reduce the search space by separating the search for sub-plans that are independent of one another. In the equational approach, rewriting on subterms can have the effect of replacing a portion of a state while leaving the rest unchanged. This can permit plans on independent portions of a state to be constructed independently without multiplying the search spaces. It also implicitly gives the effect of many frame axioms.

As far as we are aware the approach of this paper is new. As is common in planning problems, each state specifies the value of all fluents, so that the qualification problem (specifying conditions under which an action can be executed), the ramification problem (deciding the consequences of an action on the values of all fluents), and the frame problem (specifying formally all fluents that an action does not change) do not arise – the latter in particular is addressed by the fact that fluents depend directly on the structure of the encoded terms, and actions only modify these terms locally and as needed.

As for other formalisms, the popular STRIPS formalism [FN71] is basically propositional and each action has an add list of propositions to be added when it is performed and a delete list. The PDDL language [GHK$^+$98] is also popular and much more flexible. GOLOG [LRL$^+$97] permits planning in Reiter's version of the situation calculus and implements a state space search in Prolog. Another logic-based formalism for planning is that of Soutchanski and Young [SY23] which uses A* search in a tree of situations and is also implemented in Prolog. The planning as satisfiability approach [KS92] expresses the planning problem as a satisfiability problem in which a model of a set of propositional clauses can be understood as a plan. This approach takes advantage of the impressive performance of current satisfiability solvers.

2 State Calculus

Instead of the now usual formulation of the situation calculus due to Reiter, in which a situation is a sequence of states or actions, and a state is the entire state

of the world at a given time, we go back to the original formulation of McCarthy, which we call a state calculus and use the term state to refer to the entire state of the world at a given time. We use the term state calculus instead of the term situation calculus to emphasize the distinction. We assume that there is some underlying set $\mathcal{U}$ of axioms in first-order logic concerning states, fluents, and actions. The semantics of this axiomatization will have domains as in many-sorted first-order logic, for states, actions, fluents, and the values of fluents, with fluents mapping from states to various values.

Actions in $\mathcal{U}$ are typically indicated by the letter a, possibly with subscripts, and fluents are typically indicated by the letters p and q, possibly with subscripts. $\mathcal{F}$ is the set of all fluents and $\mathcal{A}$ is the set of actions. The set of fluents and actions are assumed to be finite. States are denoted by s', t', and u', possibly with subscripts. The set of states is $\mathcal{S}$. The values of the fluents are in domain $\mathcal{V}$.

There are also functions do and Φ as follows:

$$do : \mathcal{A} \times \mathcal{S} \to \mathcal{S}$$
$$\Phi : \mathcal{F} \times \mathcal{S} \to \mathcal{V}$$

Thus fluents are essentially functions from states to $\mathcal{V}$. Also $=$ is an equality relation on various domains and the usual equality axioms are assumed. There may be other domains and functions as well. Semantically, an action a in a state s transforms it into another state $do(a, s)$ Also, the fluents represent properties of a state, and $\Phi(p, s)$ is the value of fluent p in state s. Some examples of sets of axioms for state calculi are given later.

Definition 1 *A* state calculus *is a tuple* $(\mathcal{U}, \mathcal{S}, \mathcal{F}, \mathcal{A}, \Phi)$ *with the components as specified above.*

We often use $\mathcal{U}$ to refer to the entire state calculus. Later it will be important to distinguish actions that change something from those that do not.

Definition 2 *For two states* s' *and* t' *in* $\mathcal{S}$, $s' \equiv t'$ *iff for all* $p \in \mathcal{F}$, $\Phi(p, s') = \Phi(p, t')$. *An action* a *in a state calculus* $\mathcal{U}$ *is* stationary *on a state* $s' \in \mathcal{S}$ *if* $s' \equiv do(a, s')$.

We allow for the possibility that $s' \equiv t'$ and $s' \neq t'$ because states might have features other than the fluents, features that are irrelevant for planning.

Frequently actions have preconditions, the intention being that if the preconditions are not satisfied, the action cannot be performed. We assume such actions are reformulated so that they can be performed even if the preconditions are not satisfied, but in that case the action will be stationary.

Definition 3 (Fluent dependence condition) *For all states* s' *and* t', *if* $s' \equiv t'$ *then* $do(a, s') \equiv do(a, t')$. *This can be more formally stated as* $\mathcal{U} \models (\forall s', t' \in \mathcal{S})(s' \equiv t' \to do(a, s') \equiv do(a, t'))$.

We assume that all state calculi satisfy the fluent dependence condition.

3 Term Rewriting Systems

Term rewriting systems[BN98] have a simple syntax and semantics. They are a simple, natural, and sufficient formalism for expressing planning problems. In this paper a state calculus based on first-order term rewriting is presented. States are represented by terms and actions are represented by rewrite rules that operate on terms. This representation permits completion procedures [BN98,DP01] to be used for planning if actions are bidirectional, and also makes use of the subterm structure of terms to separate actions whose effects are independent. This can improve the efficiency of the planning process. This representation is probably most closely related to the fluent calculus of Thielscher [Thi98] among the approaches that have been proposed to date.

The basics of term rewriting systems are as follows.

3.1 Terms

The symbols f, g, h are *function symbols*, x, y, z are *variables*, r, s, t, u, v, w are *terms*, and a, b, c are *individual constants*. Also, i, j, k are variables that are intended to denote integers. F is the set of function symbols and X is the set of variables.

The *arity* of a function symbol is the number of arguments it takes. We assume there is a bound on the maximum arity of any function symbol. Terms are defined as follows: A variable or an individual constant is a term. Also, if f has arity n and $t_1, \ldots, t_n$ are terms then $f(t_1, \ldots, t_n)$ is a term. The set of terms over a set F of function symbols and a set X of variables is denoted $T[F, X]$. A term is a *ground term* if it contains no variables. The notation $s \equiv t$ for terms s and t indicates that the terms are syntactically identical. A term s is a *subterm* of $f(t_1, \ldots, t_n)$ if $s \equiv f(t_1, \ldots, t_n)$ or s is a subterm of t_i for some i. Also, s is a *proper* subterm of $f(t_1, \ldots, t_n)$ if s is a subterm of t_i for some i. A *maximal term* in a set T of terms is a term in t that is not a proper subterm of any other term in T.

A *context* is a term with one occurrence of $\Box$ in it, such as $f(a, \Box, b)$. This is written as $t[\ ]$ and the result of substituting some term u for $\Box$ is written as $t[u]$.

A *substitution* is a replacement of variables x_i by terms t_i; this can be written as $\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ or as $\{t_1/x_1, ...t_n/x_n\}$. Greek symbols such as θ are commonly used for substitutions. The result of applying a substitution θ to a term t is written as $t\theta$. The term $t\theta$ is called an *instance* of t.

3.2 Rewrite Rules

A *rewrite rule* is of the form $r \to s$ where r and s are terms and all variables in s also occur in r. A *term rewriting system* is a finite or infinite set of rewrite rules. The relation $\Rightarrow_R$ is defined by $t_1 \Rightarrow_R t_2$ iff there is a context $t[\ ]$ such that $t_1 \equiv t[r\theta]$ and $t_2 \equiv t[s\theta]$ for some rewrite rule $r \to s$ in R and some substitution θ. For example, if the rule $f(g(x)) \to f(h(x, x))$ is in R, then $f(f(g(h(a, b)))) \Rightarrow_R$

$f(f(h(h(a,b),h(a,b))))$. If q is a rewrite rule then we write $t_1 \Rightarrow_q t_2$ for $t_1 \Rightarrow_{\{q\}} t_2$. Also, $\Rightarrow_R^*$ is the (reflexive) transitive closure of $\Rightarrow_R$. A sequence $t_1 \Rightarrow_R t_2 \Rightarrow_R t_3 \ldots$ is called a *rewrite sequence*. The system R is *terminating* if it has no infinite rewrite sequences. A term s is *reducible* for R if there is a term t such that $s \Rightarrow_R t$; otherwise s is *irreducible*. If $s \Rightarrow_R^* t$ and t is irreducible then one writes $s \Rightarrow!_R t$ and t is called a *normal form* of s. A term rewriting system R is *confluent* if for all terms s, t_1, and t_2, if $s \Rightarrow_R^* t_1$ and $s \Rightarrow_R^* t_2$ then there is a term u such that $t_1 \Rightarrow_R^* u$ and $t_2 \Rightarrow_R^* u$. R is *bidirectional* or *invertible* if $s \Rightarrow_R t$ implies $t \Rightarrow_R s$ for all terms s, t in $T[F, X]$.

4 State Rewriting System

Based on a state calculus $\mathcal{U}$ we construct a term-rewriting system $R_{\mathcal{U}}$ to simulate the actions of $\mathcal{U}$ using rewrite rules. In Definition 4, $\mathcal{T}$ is a set of ground terms that represent states, σ is a function from such ground terms to states that they represent, and for all states s' there is a term in $\mathcal{T}$ that essentially represents it. Definition 5 states that given any collection of values for fluents, it is possible to compute a ground term that represents a corresponding state if such a state exists.

Definition 4 *Given a state calculus* $(\mathcal{U}, \mathcal{S}, \mathcal{F}, \mathcal{A}, \Phi)$*, a* state representing system for $\mathcal{U}$ *is a tuple* $(\mathcal{T}, \sigma, \hat{\Phi})$ *such that* $\mathcal{T} \subseteq T[F, X]$*,* $\mathcal{T}$ *is a set of ground terms,* $\sigma : \mathcal{T} \rightarrow \mathcal{S}$ *is a function, for all* $s' \in \mathcal{S}$ *there is a* $t \in \mathcal{T}$ *such that* $\sigma(t) \equiv s'$*,* $\hat{\Phi}(p,t) : \mathcal{F} \times \mathcal{T} \rightarrow \mathcal{V}$ *is a function, for all fluents* $p \in \mathcal{F}$ *and terms* t*,* $\hat{\Phi}(p,t) = \Phi(p, \sigma(t))$*, and* $\hat{\Phi}(p,t)$ *is computable, that is, the values of the fluents can be determined from* p *and the term structure of* t*.*

Definition 5 *Let* Find *be a function which, given a set* $\{(p_1, v_1), \cdots, (p_n, v_n)\}$ *of fluents and their values, finds a term* t *such that* $\Phi(p_i, \sigma(t)) = v_i$ *for all* i *if such a term exists, else returns "none." We assume that the function* Find *is computable.*

4.1 Rewrite Rules for State Calculus

Definition 6 *A term-rewriting system* R represents *the state calculus* $(\mathcal{U}, \mathcal{S}, \mathcal{F}, \mathcal{A}, \Phi)$ *if* $(\mathcal{T}, \sigma, \hat{\Phi})$ *is a state representing system for* $\mathcal{U}$ *and if* $R = E_\sigma \cup A_\sigma$ *where* E_σ *and* A_σ *are sets of rewrite rules satisfying Equations 1, 2, 3, and 4 below. We also call the tuple* $((\mathcal{U}, \mathcal{S}, \mathcal{F}, \mathcal{A}, \Phi), \mathcal{T}, \sigma, \hat{\Phi}, R)$ *a* state rewriting system*. We frequently abbreviate* $(\mathcal{U}, \mathcal{S}, \mathcal{F}, \mathcal{A}, \Phi)$ *by* $\mathcal{U}$ *and* $((\mathcal{U}, \mathcal{S}, \mathcal{F}, \mathcal{A}, \Phi), \mathcal{T}, \sigma, \hat{\Phi}, R)$ *by* $R_{\mathcal{U}}$*.*

The rules E_σ are *rearrangement rules* that reformat the term without affecting the state; for example, they might permit a list of terms to be rearranged in an arbitrary order. The rules A_σ are *action rules* that simulate actions on states. It is assumed that rules in E_σ are invertible. These rules E_σ and A_σ are assumed to satisfy the following axioms. The first axiom says that if a term s rewrites to t and s represents a state then t does also. The second axiom says

that a term s rewrites to a term t using the set E_σ of rewrite rules if s and t have the same values of all fluents.

$$\text{If } q \in E_\sigma \cup A_\sigma \text{ and } s \in \mathcal{T} \text{ and } t \text{ is a term such that } s \Rightarrow_q t \text{ then } t \in \mathcal{T} \text{ also.} \quad (1)$$

$$\text{For terms } s, t \in \mathcal{T},\ s \Rightarrow^*_{E_\sigma} t \text{ iff } \sigma(s) \equiv \sigma(t). \quad (2)$$

For a single rule q in A_σ there can be many ground terms s and t such that $s \Rightarrow_q t$ and these various terms can correspond to different actions a. Thus we have the following axiom:

$$\text{If } q \in A_\sigma \text{ then for all } s, t \in \mathcal{T}, \text{ if } s \Rightarrow_q t \text{ then there exists } a \in \mathcal{A} \text{ such that } \sigma(t) \equiv do(a, \sigma(s)). \quad (3)$$

In fact we assume that some such action a is computable, given s, t, and $q \in A_\sigma$.

We also allow the possibility that actions in $\mathcal{A}$ correspond to more than one rule in A_σ.

Definition 7 *For $s, t \in \mathcal{T}$, $s \stackrel{\sim}{\Rightarrow}_R t$ if there are terms $u, v \in \mathcal{T}$ such that $s \Rightarrow^*_{E_\sigma} u \Rightarrow_{A_\sigma} v \Rightarrow^*_{E_\sigma} t$.*

$$\text{For all } a \in \mathcal{A} \text{ and all } s, t \in \mathcal{T}, \text{ if } \sigma(t) \equiv do(a, \sigma(s)) \text{ and } a \text{ is non-stationary on } \sigma(s) \text{ then } s \stackrel{\sim}{\Rightarrow}_R t. \quad (4)$$

Although $\mathcal{T}$ is a set of ground terms, the rewrite rules in $R_\mathcal{U}$ need not be ground rules, which will be clear from the examples.

4.2 Planning Using Term Rewriting

Definition 8 *Suppose $(\mathcal{U}, \mathcal{S}, \mathcal{F}, \mathcal{A}, \Phi)$ is a state calculus and s' and t' are states in $\mathcal{S}$. Then a* plan from s' to t' *is a sequence of states $s'_1, s'_2, \cdots, s'_n$ and actions $a_1, a_2, \cdots, a_{n-1}$ with $s' = s'_1$ and $t' = s'_n$ such that for all i, $1 \leq i \leq n-1$, $s'_{i+1} = do(a_i, s'_i)$.*

Theorem 1 (Planning Theorem). *Suppose s_1 and s_n are terms in $\mathcal{T}$ and there is a plan from $\sigma(s_1)$ to a state s'_n with $s'_n \equiv \sigma(s_n)$. Then $s_1 \stackrel{\sim}{\Rightarrow}^*_R s_n$. The converse is also true.*

Proof. If there is a plan then there is a plan without stationary actions. Let $s'_1, s'_2, \cdots, s'_n$ be the states in the plan as in Definition 8 with $s'_1 = \sigma(s_1)$ and $s'_n \equiv \sigma(s_n)$. By Definition 4 there are terms s_i, $1 \leq i \leq n$ such that $\sigma(s_i) \equiv s'_i$. The conclusion follows by Definition 7 and Equation 4. For the converse, let $s_1, \cdots, s_n$ be such that $s_i \stackrel{\sim}{\Rightarrow}_R s_{i+1}, 1 \leq i \leq n-1$. Let a_i be actions such that $\sigma(s_{i+1}) \equiv do(a_i, \sigma(s_i))$, $1 \leq i \leq n-1$ (by Equations 2 and 3) and note that these actions are computable by Equation 3. Let $s'_1, s'_2, \cdots, s'_n$ be states such that $s'_1 = \sigma(s_1)$ and for $1 \leq i \leq n-1$ $s'_{i+1} = do(a_i, s'_i)$. Then by repeated use of the fluent dependence condition, $s'_i \equiv \sigma(s_i)$. Thus $s'_n \equiv \sigma(s_n)$ and the s'_i form a plan from $\sigma(s_1)$ to a state equivalent to $\sigma(s_n)$. □

Thus, given the values of the fluents on s'_1 and s'_n, to see if it is possible to reach a state equivalent to s'_n from s'_1 by a sequence of actions in $\mathcal{U}$, one can construct terms s_1 and s_n as in the theorem using *Find* and test if $s_1 \stackrel{\sim}{\Rightarrow}^*_R s_n$.

Corollary 1. *With conditions as in the planning theorem, if A_σ is invertible, then $s_1 \stackrel{\sim}{\Leftrightarrow}^*_R s_n$, and if $s_1 \stackrel{\sim}{\Leftrightarrow}^*_R s_n$ then a plan exists as in the theorem.*

Corollary 2. *Unfailing completion can find a proof that $s_1 \stackrel{\sim}{\Leftrightarrow}^*_R s_n$ iff there is a plan from $\sigma(s_1)$ to a state s'_n with $s'_n \equiv \sigma(s_n)$.*

Proof. By the planning theorem and the soundness and completeness of unfailing completion.

This means that rewrite strategies such as completion and unfailing completion [BN98] [BDP89,HR87] can be used to test if such a plan exists. These rewriting based approaches can show the existence of a plan even without explicitly contstructing it. Sometimes the time to show the existence of a plan can be much smaller than the time to construct or even print out the plan.

Theorem 2. *Suppose $R_\mathcal{U}$ is a rewriting system that represents the theory $\mathcal{U}$ and suppose A_σ is invertible. If s and t are ground terms then from a proof using unfailing completion and rewriting that $s \stackrel{\sim}{\Leftrightarrow}^* t$ it is possible to construct a sequence s_i, $1 \leq i \leq n$ of ground terms such that $s_1 = s$ and $s_n = t$ and $s_i \stackrel{\sim}{\Leftrightarrow} s_{i+1}, 1 \leq i < n$ and a sequence of actions $a_1, a_2, \cdots, a_{n-1}$ such that $\sigma(s_{i+1}) \equiv do(a_i, \sigma(s_i))$ for i, $1 \leq i \leq n-1$.*

Proof. From such an unfailing completion proof it is possible to construct a sequence $s_1, s_2, \cdots, s_n$ of ground terms such that $s_i \stackrel{\sim}{\Leftrightarrow} s_{i+1}, 1 \leq i < n$, using e.g. the system demonstrated in [DS96]. Then by repeated use of the decidability assumption following Equation 3, one can find a sequence $a_1, a_2, \cdots, a_{n-1}$ of actions such that $\sigma(s_{i+1}) \equiv do(a_i, \sigma(s_i))$ for all i, $1 \leq i \leq n-1$. □

This plan might not be optimal, but it might be possible to improve it after it is found. The particular plan that is found can be influenced by the choice of a *termination ordering* [BN98]. The use cases presented below show that it is plausible to assume that the invertibility assumption holds in many interesting domain. One large class of problems we can handle are in logistics with constraints ("Bring a wolf, a goat and a cabbage over the river...").

Definition 9 *Suppose $a, b \in \mathcal{A}$ and for all states $r', s' \in \mathcal{S}$, $s' \equiv do(a, r')$ iff $r' \equiv do(b, s')$. Then a^R is defined to be some such b.*

In some cases a^R is not defined because such an action b might not exist.

To directly extract a plan from a proof that $s_1 \stackrel{\sim}{\Leftrightarrow}^* s_n$, the notation $s \stackrel{\alpha_1 \cdots \alpha_k}{\underset{R^*_\mathcal{U}}{\Rightarrow}} t$ can be used indicating a sequence of actions leading from $\sigma(s)$ to $\sigma(t)$. These actions can be carried along in the completion procedure. We have the following rules:

$$\text{If } r \overset{\alpha_1 \cdots \alpha_m}{\underset{R^*_{\mathcal{U}}}{\Rightarrow}} s \text{ and } s \overset{\alpha_{m+1} \cdots \alpha_n}{\underset{R^*_{\mathcal{U}}}{\Rightarrow}} t \text{ then } r \overset{\alpha_1 \cdots \alpha_n}{\underset{R^*_{\mathcal{U}}}{\Rightarrow}} t. \tag{5}$$

$$\text{If } s \overset{\alpha_1 \cdots \alpha_k}{\underset{R^*_{\mathcal{U}}}{\Rightarrow}} t \text{ then } t \overset{\alpha_k^R \cdots \alpha_2^R \alpha_1^R}{\underset{R^*_{\mathcal{U}}}{\Rightarrow}} s \text{ assuming } A_\sigma \text{ is invertible and all } \alpha_i^R \text{ exist.} \tag{6}$$

$$\text{For all } s, t \text{ in } \mathcal{T} \text{ if } s \Rightarrow^*_{E_\sigma} t \text{ then } s \overset{\epsilon}{\underset{R^*_\sigma}{\Rightarrow}} t \text{ where } \epsilon \text{ is the empty sequence.} \tag{7}$$

The above equation says that rules in E_σ correspond to empty actions which might change the term but do not change the state.

$$\text{If } \sigma(t) \equiv do(a, \sigma(s)) \text{ and } s \Rightarrow_{A_\sigma} t \text{ then } s \overset{a}{\underset{R^*_{\mathcal{U}}}{\Rightarrow}} t. \tag{8}$$

However, there might be more than one action corresponding to a given rewrite rule in A_σ. This can cause a problem if the rule is non-ground. The relevant ground instance of the rewrite rule might not be known until the entire proof has been constructed.

5 Examples

Some examples will illustrate the properties of this approach to the state calculus. The default rewriting relation $s \Rightarrow t$ in these examples will refer to rewriting in the system $R_{\mathcal{U}}$. The approach used for rewriting is unfailing completion[BDP89,HR87], which, in the limit, produces a term rewriting system that is confluent by ordered rewriting. We assume that $R_{\mathcal{U}}$ is bidirectional.

5.1 Switches Example

In this example, there are m switches which can be on or off. For this example, the state calculus $\mathcal{U}$ is as follows:

The fluents are $sw_1, sw_2, \cdots, sw_m$ and the values of the fluents are "on" and "off." The actions are "turnon(i)" and "turnoff(i)" for $1 \leq i \leq m$. The axioms are as follows:

$(\exists s' \in \mathcal{S})[(\Phi(sw_1, s') = \text{"}off\text{"}) \wedge \cdots \wedge (\Phi(sw_m, s') = \text{"}off\text{"})]$ (Existence of a state in which all switches are off)

$i \neq j \rightarrow \Phi(sw_i, do(turnon(j), s')) = \Phi(sw_i, s')$ (Frame axiom for turning a switch on)

$i \neq j \rightarrow \Phi(sw_i, do(turnoff(j), s')) = \Phi(sw_i, s')$ (Frame axiom for turning a switch off)

$\Phi(sw_i, do(turnon(i), s')) = \text{"}on\text{"}$ (After a switch is turned on it is on)

$\Phi(sw_i, do(turnoff(i), s')) = \text{"}off\text{"}$ (After a switch is turned off it is off)

There are also the equality axioms on the domains.

This set of axioms essentially assumes that the initial state has all switches off. An axiom can be added stating that no state exists with all switches on, or whatever the goal state is. As for the state rewriting system $R_{\mathcal{U}}$, the state of the switches is represented by a term in $\mathcal{T}$ of the form $f(x_1, x_2, \cdots, x_m)$ where all x_i can be "on" or "off". Also $\hat{\Phi}(sw_i, f(b_1, \cdots, b_m)) = b_i$.

As for the semantics, an action turns any particular switch on or off. One way to represent this example is by rewrite rules in $R_{\mathcal{U}}$ of the form

$$f(x_1, \cdots, x_{i-1}, \text{off}, x_{i+1}, \cdots, x_m) \rightarrow f(x_1, \cdots, x_{i-1}, \text{on}, x_{i+1}, \cdots, x_m)$$

to turn the i^{th} switch on and a rule

$$f(x_1, \cdots, x_{i-1}, \text{on}, x_{i+1}, \cdots, x_m) \rightarrow f(x_1, \cdots, x_{i-1}, \text{off}, x_{i+1}, \cdots, x_m)$$

to turn it off . The search space is exponential because each of the m switches can either be on or off. Stationary actions (such as turning a switch on that is already on) correspond to rewrite rules that cannot be applied to a term. This set of rules is already bidirectional.

Now, suppose the terms s and t as in Theorem 2 are $f(u_1, \cdots, u_m)$ and $f(v_1, \cdots, v_m)$ where the u_i and v_i are each either "on" or "off". The two rewrite rules per switch will be represented by a single equation

$$f(x_1, \cdots, x_{i-1}, \text{off}, x_{i+1}, \cdots, x_m) = f(x_1, \cdots, x_{i-1}, \text{on}, x_{i+1}, \cdots, x_m)$$

which during unfailing completion will be oriented in either direction, depending on the termination ordering. There are a quadratic number of critical pairs between these equations, but they are all joinable, so completion terminates quickly. Then the reductions to normal form will take worst case time proportional to m, which is linear time.

Suppose that "off" is larger than "on" in the termination ordering. Then both s which is $f(u_1, \cdots, u_m)$ and t which is $f(v_1, \cdots, v_m)$ will rewrite to $f(\text{on}, \cdots, \text{on})$, leading to a rewriting sequence $f(u_1, \cdots, u_m) \Rightarrow^* f(\text{on}, \cdots, \text{on}) \Leftarrow^* f(v_1, \cdots, v_m)$, so that the plan consists of turning all the switches u_i on that are off, and then turn off all the switches v_i that are on. This could result in a switch that is off in both starting and ending states to be turned on and then off again. This plan can be optimized by removing such pairs of actions, resulting in a reasonable plan. The length of the plan will be at most $2m$, regardless of the positions of the switches in the original and final states. However, a breadth-first search for a plan will result in an exponential search space.

5.2 Tower of Hanoi

For this example, there are m disks of different sizes on three pegs. On each peg the disks have to be in order of size, with the largest disk on the bottom. Only the top disk on a peg can be moved from one peg to another, and it has to be the smallest disk on the peg it is moved to. The problem is to rearrange the disks on the pegs, typically moving all of them from one peg to another.

For this example, the state calculus $\mathcal{U}$ is as follows:

The fluents are $on(k,j), 1 \leq k \leq m, 1 \leq j \leq 3$ indicating that disk k is on peg j. The values of the fluents are "true" and "false." The actions are $move(i,j,k), 1 \leq i \leq m, 1 \leq j \leq 3$ to move disk k from peg i to peg j. This action has no effect unless disk k is not already on peg j and disk k is smaller than any other disk on peg i and smaller than any disk on peg j. The smallest disk is 1 and the largest is n.

The axioms of $\mathcal{U}$ are as follows:

$(\forall i,j,1 \leq i,j \leq 3)(\forall k,1 \leq k \leq m)(\forall s' \in \mathcal{S})[i \neq j \wedge \Phi(on(k,i),s') = "true" \rightarrow \neg\Phi(on(k,j),s') = "true"]$ (Disk on at most one peg)

$(\forall k,1 \leq k \leq m)(\forall s' \in \mathcal{S})[(\Phi(on(k,1),s') = "true") \vee (\Phi(on(k,2),s') = "true") \vee (\Phi(on(k,3),s') = "true")]$ (Disk on at least one peg)

$(\exists s' \in \mathcal{S})[(\Phi(on(1,1),s') = "true") \wedge \cdots \wedge (\Phi(on(m,1),s') = "true")]$ (Existence of a state)

$(\forall i,j,k)(\forall s' \in \mathcal{S})[(1 \leq k \leq m) \wedge (1 \leq i,j \leq 3) \wedge min(k,i,s') \wedge min(k,j,s') \wedge \Phi(on(k,i),s') = "true") \rightarrow \Phi(on(k,j),do(move(i,j,k),s')) = "true"]$ (Effect of a move)

$(\forall i,j,k)(\forall s' \in \mathcal{S})[(1 \leq k \leq m) \wedge (1 \leq i,j \leq 3) \wedge (\neg min(k,i,s') \vee \neg min(k,j,s')) \wedge \Phi(on(k,i),s') = "true") \rightarrow \Phi(on(k,i),do(move(i,j,k),s')) = "true"]$ (Preconditions of a move not satisfied)

$(\forall h,i,j,k,d)(\forall s' \in \mathcal{S})[(1 \leq d,k \leq m) \wedge (1 \leq i,j,h \leq 3) \wedge ((d \neq k) \vee (i \neq h)) \wedge \Phi(on(d,h),s') = "true" \rightarrow \Phi(on(d,h),do(move(i,j,k),s')) = "true")$ (Frame axiom)

$(\forall i,j,1 \leq i,j \leq 3)(\forall k,1 \leq k \leq m)(min(k,i,s') \equiv [\Phi(on(k-1,i),s') = "false" \wedge \cdots \wedge \Phi(on(2,i),s') = "false" \wedge \Phi(on(1,i),s') = "false"])$ (Definition of smallest disk)

There are also the equality axioms on the domains.

The search space is exponential because the largest disk can be on any of the three pegs, the next largest disk can be on any of the three pegs, possibly on top of the largest disk, and so on. But completion for this example takes polynomial time, as we shall see.

The optimal sequence to move m disks from peg i to peg j, for $i \neq j$, consists of moving the $m-1$ smallest disks from peg i to peg k, $k \neq i$ and $k \neq j$, then moving the largest disk from peg i to peg j, then moving the $m-1$ smallest disks from peg k to peg j. If $m = 1$ this consists of one move, so the general sequence has $2^m - 1$ moves.

For the syntax of this problem, states can be encoded as terms $f(u_1, u_2, \cdots, u_m)$, where each u_i is either 1, 2, or 3 depending on which peg the disks are on, and u_1 refers to the largest disk, u_2 to the next largest, and so on. The actions are of the form move(i, j, k) to move disk k from peg i to peg j.

The rules in $R_{\mathcal{U}}$ are of the form

$$f(i,j,j,\cdots,j) \rightarrow f(k,j,j,\cdots,j)$$

to move the largest disk and, more generally, rules are of the form

$$f(x_1,x_2,\cdots,i,j,j,\cdots,j) \rightarrow f(x_1,x_2,\cdots,k,j,j,\cdots,j)$$

where i, j, and k are distinct elements of $\{1, 2, 3\}$. The i^{th} largest disk can move only from peg i to peg k if all the smaller disks are on peg j, because otherwise, a smaller disk will be on peg i or k, preventing the move. For example, consider the largest disk. To move it from peg 1 to peg 2, there cannot be any smaller disks on peg 1 because they would be on top of it, and only the top disk can be moved. There also cannot be any smaller disks on peg 2, because then a larger disk would be put on top of a smaller one, which is not permitted. Now consider the smallest disk. It is always on top of one of the piles, and at any time it can move to any other pile. In general, a disk is constrained in moving only by disks that are smaller than it is. Stationary actions (such as trying to move a disk that is not the smallest one on its peg) correspond to rewrite rules that cannot be applied to a term in this formalization.

Completing this set of rules generates $2m$ rules in all.

In general, completing the system requires a quadratic number of rewrite operations even though the optimal action sequence requires an exponential number of actions. Finally, rewriting the starting term and the goal term to a common term requires a number of rewrites that is linear in m.

5.3 Blocks world

In this example, there are a fixed number of positions, each having a stack of blocks. The blocks can be piled in any order, and at any time the top block in any stack can be moved on top of any other stack. Then one wants a plan to transform some specified starting state to a goal state.

We do not give a formal representation of the theory $\mathcal{U}$ for this example. As for $R_{\mathcal{U}}$, states can be represented by lists $f(h(s_1, u_1), f(h(s_2, u_2), \cdots, f(h(s_m, u_m), \perp) \cdots))$ where the s_i are lists of blocks and the u_i are their locations. A stack of blocks is represented as a list $g(b_1, g(b_2, \cdots \perp \cdots))$ where b_1 is the top block, b_2 is next under it, and so on. The actions include permuting the lists of blocks: $f(t_1, f(t_2, u)) \rightarrow f(t_2, f(t_1, u))$ to exchange adjacent stacks of blocks (an action in E_σ) and an action $f(h(g(b_1, s_1), u_1), f(h(s_2, u_2), z)) \rightarrow f(h(s_1, u_1), f(h(g(b_1, s_2), u_2), z))$ to move the top block b_1 from the stack $g(b_1, s_1)$ at u_1 to the stack s_2 at u_2. This action could be represented as *movetop*(u_1, u_2) to move the top block from the stack at u_1 to the stack at u_2. In this example there cannot be two different blocks on the same block.

5.4 Generating term representations

The question arises how one can find such representations of states as terms. One can show formally that in many cases one can construct such systems $R_{\mathcal{U}}$ but a lack of space prohibits a full discussion of this topic. The basic idea of representing states as terms is to represent the physical structure of the state as a corresponding term structure. A stack of objects can be represented as a list of terms, one for each object. A sequence of objects can also be reprsented as a list in this way. Thus a sequence of switches can be represented as a list of terms, each being "on" or "off." A stack of blocks can be represented as a list of

terms, where each term is a label for a particular block. A collection of stacks of blocks can be reprsented as a list of lists, one list for each stack of blocks. Because the collection of stacks is unordered, it is possible to reorder the lists of blocks arbitrarily, though the blocks in each list may not be reordered except by actions. It is also possible to use another representation if the physical structure of the problem can be determined from it. This explains the representation of the Tower of Hanoi problem, which is more convenient than representing the problem as a sequence of three lists, one list for each peg. Once the term representation is chosen, it is fairly straightforward to construct the term rewriting system.

In general, it is always possible to represent states of $\mathcal{U}$ by terms if actions are deterministic. The idea is to give the values of all the fluents p_i of a state s in a list $f(v_1, \cdots, v_n)$ where $v_i = \Phi(p_i, s)$. If there is some redundancy in this list then it can be compressed.

6 Experimental results

To put the above ideas to the test, we have performed a series of experiments in the *switches* and the *Tower of Hanoi* domain. In particular, we have created several variants of the problems in different sizes, and used the theorem prover E [Sch02,SCV19] to find equational plans for these problems. We have recorded various statistics of the problems and the run time, and we used post-processing of the proof object generated by the prover to determine the number of rewrite steps in the proof object and the corresponding number of actions (i.e., single applications of the original axioms encoding actions) in these proofs.

Test examples were created with Python scripts available in the form of a Jupyter notebook at http://www.eprover.eu/E-eu/eqplanning.html. A full description of the results, also showing examples of the problem encoding, is available in the appendix at the same address. We used the standard release of E 3.0 available at https://www.eprover.org. All experiments were run on a MacBook Pro with M1Pro CPU and 32GB of RAM.

Unless otherwise specified, E ran a simple symbol-counting clause selection function (`eprover -sR --proof-object -H'(1*weight11_ugg)' <problem>`). The options `-sR` select low-output mode during search and output of resource usage at the end. For some experiments we explicitly modify the term ordering via the option `--precedence`. These cases are noted below.

Switch banks We have created versions of the problem with the action rules encoded in any of three different representations, further described in the appendix: Each individual action encoded by a dedicated rule mapping one ground state to the successor state (resulting in an exponential number of rules), using variables to represent the unchanged switches (resulting in a linear number of rules), and using subterm rewriting, i.e. representing all actions by the single equation `off=on`. The concrete problem instance is described as a disequation between encodings of the start state and the final state. We consider two different problem types: A minimal problem, where initial state and goal state differ

Instance	Ground			Variable			Subterm		
size	RW	Actions	Time	RW	Actions	Time	RW	Actions	Time
1	1	1	0.003	1	1	0.006	1	1	0.006
2	2	2	0.002	2	2	0.005	2	2	0.005
3	3	3	0.002	3	3	0.004	3	3	0.004
4	4	4	0.002	4	4	0.003	4	4	0.003
5	5	5	0.003	5	5	0.003	5	5	0.003
6	6	6	0.004	6	6	0.003	6	6	0.003
7	7	7	0.009	7	7	0.003	7	7	0.003
8	8	8	0.020	8	8	0.003	8	8	0.003
9	9	9	0.046	9	9	0.003	9	9	0.003
10	10	10	0.114	10	10	0.003	10	10	0.002
11	11	11	0.290	11	11	0.003	11	11	0.002
12	12	12	0.794	12	12	0.003	12	12	0.002
13	13	13	2.272	13	13	0.003	13	13	0.002
14	14	14	6.850	14	14	0.003	14	14	0.002
15	15	15	22.405	15	15	0.003	15	15	0.002
16	16	16	93.106	16	16	0.003	16	16	0.002
17	17	17	407.151	17	17	0.003	17	17	0.002
18	18	18	1845.499	18	18	0.003	18	18	0.002
19	19	19	8076.025	19	19	0.003	19	19	0.002
20	20	20	42027.167	20	20	0.003	20	20	0.002

Times are are given as total CPU time in seconds.

Table 1. Switch data (all switches different)

by only a single switch setting, and a maximal problem, where all switches differ between initial and final state (in this case, both states alternate *on* and *off*, but in complementary positions). For the former, the optimal plan has length 1, for the latter length n.

We present results for sizes 1 to 20 in Table 1. Several observations can be made. First, the prover always finds an optimal plan. Also, we cannot observe a *plan compression* via interreduction and completion – each action corresponds to exactly one rewrite step in the the proof object. Finally, we can see that for the variable and subterm encodings, run times are within the measurement noise of the operating system. These plans are found basically instantly. For the full ground encoding, we see that the sheer processing of an exponentially growing set of axioms also results in an exponential increase in runtime.

In a second setting, we consider the case where in the initial state all switches are *on*, while in the final state one single switch is off. An optimal plan would just switch the single different switch, independent of the instance size. Again, E finds a plan for all instances and in all encodings. However, the quality of the plan depends on the term ordering. If *off*$>$*on*, then the prover will find an optimal plan. If, on the other hand, *on*$>$*off*, then the prover will reduce both initial and final state to an intermediate state where all switches are off, and thus use *2n-1* actions. Full results are available in the appendix.

Instance size	Flat encoding			Recursive encoding		
	RW	Actions	Time	RW	Actions	Time
1	1	1	0.008	1	1	0.007
2	6	6	0.005	4	4	0.004
5	33	147	0.003	25	157	0.003
10	118	35000	0.003	100	39356	0.003
20	418	2324522914	0.004	400	2324522914	0.004
30	928	137260754729736	0.008	900	137260754729736	0.007

Table 2. Tower of Hanoi

Tower of Hanoi The Tower of Hanoi problem shows the power of completion-based approaches to very efficiently find plans even when the size of plans grows exponentially. Table 2 shows some selected results (full results are in the appendix) for two encodings, flat and recursive, which are described in the appendix. The ideal plan for a Hanoi problem of size n has $2^n - 1$ actions – or, for an instance of size 20, about 1 million moves, for an instance of size 30 about 1 billion moves. E finds solutions to these planing problems in less than 10 milliseconds. We can observe an extreme *compression* of the plans – the interreduction of rewrite rules leads to effective macro rules that represent very large number of individual actions. While the proof objects even in the largest case has less than 1000 rewrite applications, the plans represented by these proof objects are humungous. We believe that the major reason for that is the aggressive simplification used in all modern provers – every term is rewritten to its normal form, even if that is not required by the proof itself. However, this very effect also opens the opportunity for local optimisations of the plan. When reconstructing the actual equational chain, every segment connecting two identical terms can simply be dropped.

7 Plan Representation

Earlier we discussed how to find representations of states as terms. There is also the question how one can represent the plans. Some of the plans produced for the Tower of Hanoi problem are very large, possibly too large to print out. However, in general, if each rewrite rule in $R_{\mathcal{U}}$ corresponds to a single action, and a^R exists for all actions $a \in \mathcal{A}$, then a plan from term r to term s can be expressed more succinctly by a series of equations (1) of the form $x = a$ for an action a, (2) of the form $x = y^R$ indicating that the plan x is the reverse of plan y, with all actions reversed, or (3) of the form $x = y \circ z$ indicating the plan x is the concatenation of plans y and z. Such a listing will be about the same length as the number of rewrite steps in the proof so it will be much shorter than a full plan. Also, we show how to get a term t that represents a state somewhere in the middle of the plan, so that one can then recursively optimize the plans to get from r to t and from t to s.

Theorem 3. *Suppose that for all rules $u \rightarrow v$ in $R_{\mathcal{U}}$ there is an action $a \in \mathcal{A}$ such that for all ground instances $u\theta \rightarrow v\theta$ of $u \rightarrow v$, $\sigma(v\theta) = do(a, \sigma(u\theta))$. Then from a rewrite and unfailing completion proof that there is a plan between terms r and s it is possible to construct a sequence of equations as mentioned above in which the number of equations is at most double the number of rewrite steps in the proof.*

Proof. For simplicity we represent equations $u = v$ as the rules $u \rightarrow v$ and $v \rightarrow u$, and vice versa. These imply $u \Rightarrow v$ and $v \Rightarrow u$, respectively. From such expressions $u \Rightarrow v$ and $v \Rightarrow u$, a rewrite proof can be expressed in terms of applying substitutions and the equality axioms. That is, the steps on the proof are of the form

$$\frac{}{u \Rightarrow u} \quad \frac{u \Rightarrow v}{v \Rightarrow u} \quad \frac{u \Rightarrow v \wedge v \Rightarrow w}{u \Rightarrow w} \quad \frac{u \Rightarrow v}{f(\cdots u \cdots) \Rightarrow f(\cdots v \cdots)} \quad \frac{u \Rightarrow v}{u\theta \Rightarrow v\theta}$$

Applying a substitution does not change the plan because of the assumption about rules in $R_{\mathcal{U}}$. The identity axiom $u = u$ corresponds to an empty plan. The functionally reflexive axioms do not change the plan. The symmetry axiom reverses the plan. The transitivity axiom results in the concatenation of two plans. The number of applications of the transitivity axiom is at most the number of rewrite or critical pair steps. It is not necessary to apply the symmetry axiom more than once in succession. □

Unfailing completion makes use of semi-critical pair operations.

Theorem 4. *Suppose that unfailing completion after some work produces a proof consisting of one rewrite operation from r to s or from s to r. Then in the proof of the existence of a plan from r to s there is at least one rewrite operation on the large side of a rule or a (semi-) critical pair operation between rules or equations. Suppose the (temporally) last such operation is between rules or oriented equations $r_1 \rightarrow s_1$ and $r_2 \rightarrow s_2$. Suppose wlog. that a subterm of r_1 unifies with r_2 and that α is the mgu. Let the normal form of the resulting equation be $u_1 = u_2$ as used for the proof. Let $u_1\beta = u_2\beta$ be the instance of this equation that is used in rewriting r and s to a common term. Suppose $r \equiv v[u_1\beta]$ and $s \equiv v[u_2\beta]$. Then $v[r_1\alpha\beta]$ represents a term in the plan, and this term is not the same as r or s.*

Proof. Note that if the final unfailing completion proof that $r \downarrow s$ is longer than one rewrite then some term in the middle represents a state in the middle of the plan. Without loss of generality suppose $u_1\beta = u_2\beta$ is used in rewriting r to a normal form and $r \not\Rightarrow^* s$ using $R_{\mathcal{U}}$. Because the final proof consists of a single rewrite, $r \equiv v[u_1\beta]$ for some v and $v[r_1\alpha\beta] \Rightarrow^* v[u_1\beta]$ and $v[u_2\beta] \equiv s$. Also $v[r_1\alpha\beta] \not\equiv v[u_1\beta]$ for then the rule $r_2 \rightarrow s_2$ would already rewrite r to s. $v[r_1\alpha\beta] \not\equiv s$ because $v[r_1\alpha\beta]$ rewrites to s in at least one rewrite step. □

8 Conclusion

After a brief survey of the situation calculus, term-rewriting systems are introduced and situation calculus concepts are presented. Instead of situation calculus the terminology state calculus is used to avoid misunderstanding. The approach presented here finds a plan that solves a planning problem, but not necessarily an optimal plan. Not only the cost of a plan but the time to find it are important parameters of a planner. A general argument is given that this approach can be much faster than breadth-first style approaches such as Dijkstra's method and the A* method, and an implementation shows that the existence of a very long plan can sometimes be shown rapidly by this approach. An approach to encoding the state calculus by term rewriting is presented. A general method for planning using this approach is given. Three examples illustrate the properties of this approach. We present a number of experiments on the first two classes of planning problems, demonstrating that even very long plans can be found in very short times, and that the quality of the resulting plans can be improved, by choosing a good rewrite ordering. We show that if a plan can be found quickly by this approach then it has a succinct representation, even if the plan is very long. Also, even without actually constructing the whole plan, it is possible to find a term representing a state somewhere in the middle of the plan rapidly.

There is a need for more work on optimizing the plans found by this method, as well as comparing this method to other existing approaches on a variety of problems. In addition, the formalism can be extended to conditional term rewriting systems. Another possible extension would be to actions that are not bidirectional.

References

[BDP89] Leo Bachmair, N. Dershowitz, and D. Plaisted. Completion without failure. In Hassan Aït-Kaci and Maurice Nivat, editors, *Resolution of Equations in Algebraic Structures 2: Rewriting Techniques*, pages 1–30, New York, 1989. Academic Press.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[Dav90] E. Davis. *Representations of Commonsense Knowledge.* Morgan Kaufmann, 1990.

[Dij59] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[DP01] N. Dershowitz and D. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 9, pages 535–610. Elsevier Science, 2001.

[DS96] Jörg Denzinger and Stephan Schulz. Recording and Analysing Knowledge-Based Distributed Deduction Processes. *Journal of Symbolic Computation*, 21(4/5):523–541, 1996.

[FN71] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208, 1971.

[GHK+98] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—The Planning Domain Definition Language, 1998.

[Haa87] A. R. Haas. The case for domain-specic frame axioms. In F. M. Brown, editor, *The Frame Problem in artificial intelligence. Proceedings of the 1987 workshop*, pages 343–348. Morgan Kaufmann, 1987.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[HR87] J. Hsiang and M. Rusinowitch. On Word Problems in Equational Theories. In *Proc. of the 14th ICALP, Karlsruhe*, volume 267 of *LNCS*, pages 54–71. Springer, 1987.

[KS92] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *European Conference on Artificial Intelligence*, 1992.

[Lin08] Fangzhen Lin. Situation calculus. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, pages 649–669. Elsevier, 2008.

[LRL+97] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. Golog: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 31(1):59–83, 1997. Reasoning about Action and Change.

[MH69] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[Ped89] E. P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the International Conference on Principles of Knowledge Representation (KR-98)*, pages 324–332. Morgan Kaufmann, Inc., 1989.

[Rei91] Raymond Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.

[Sch90] Lehnart Schubert. Monotonic solution of the frame problem in the situation calculus: An efficient method for worlds with fully specified actions. In Henry E. Kyburg, Ronald P. Loui, and Greg N. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, volume Volume 5, pages 23–67. Kluwer Academic Publishers, Dordrecht / Boston / London, 1990.

[Sch02] Stephan Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.

[SCV19] Stephan Schulz, Simon Cruanes, and Petar Vukmirović. Faster, higher, stronger: E 2.3. In Pascal Fontaine, editor, *Proc. of the 27th CADE, Natal, Brasil*, number 11716 in LNAI, pages 495–507. Springer, 2019.

[SY23] Mikhail Soutchanski and Ryan Young. Planning as theorem proving with heuristics, 2023.

[Thi98] Michael Thielscher. Introduction to the fluent calculus. *Electron. Trans. Artif. Intell.*, 2:179–192, 1998.

A Full experimental data

This appendix will be made available online at

http://www.eprover.eu/E-eu/eqplanning.html

Unless otherwise specified, all results are obtained with E 3.0 using the command line `eprover -sR --proof-object -H'(1*weight11_ugg)' <problem>`. All times are in seconds. Since timing by the operating system is not exact, there is some noise, and differences in the few millisecond range are not significant.

A.1 Switches – all different

Full ground encoding In this setting, the initial and final states differ in all switches, and the actions are encoded as equations between full ground states. An example of size 3 is presented below. Results are in Table 3.

```
% Simple switchbank example. There are 3 switches
% that can be on or off. Actions flip a single switch.
%
% This is the ground version.
%
% Each possible action in each possible state is encoded
% in one rule, i.e. we have 2^n*n=24 action rules.

cnf(to_plan, negated_conjecture, f(on,off,on)!=f(off,on,off)).
cnf(g_0_on_on_on, axiom, f(on,on,on)=f(off,on,on)).
cnf(g_1_on_on_on, axiom, f(on,on,on)=f(on,off,on)).
cnf(g_2_on_on_on, axiom, f(on,on,on)=f(on,on,off)).
cnf(g_0_on_on_off, axiom, f(on,on,off)=f(off,on,off)).
cnf(g_1_on_on_off, axiom, f(on,on,off)=f(on,off,off)).
cnf(g_2_on_on_off, axiom, f(on,on,off)=f(on,on,on)).
cnf(g_0_on_off_on, axiom, f(on,off,on)=f(off,off,on)).
cnf(g_1_on_off_on, axiom, f(on,off,on)=f(on,on,on)).
cnf(g_2_on_off_on, axiom, f(on,off,on)=f(on,off,off)).
cnf(g_0_on_off_off, axiom, f(on,off,off)=f(off,off,off)).
cnf(g_1_on_off_off, axiom, f(on,off,off)=f(on,on,off)).
cnf(g_2_on_off_off, axiom, f(on,off,off)=f(on,off,on)).
cnf(g_0_off_on_on, axiom, f(off,on,on)=f(on,on,on)).
cnf(g_1_off_on_on, axiom, f(off,on,on)=f(off,off,on)).
cnf(g_2_off_on_on, axiom, f(off,on,on)=f(off,on,off)).
cnf(g_0_off_on_off, axiom, f(off,on,off)=f(on,on,off)).
cnf(g_1_off_on_off, axiom, f(off,on,off)=f(off,off,off)).
cnf(g_2_off_on_off, axiom, f(off,on,off)=f(off,on,on)).
cnf(g_0_off_off_on, axiom, f(off,off,on)=f(on,off,on)).
cnf(g_1_off_off_on, axiom, f(off,off,on)=f(off,on,on)).
cnf(g_2_off_off_on, axiom, f(off,off,on)=f(off,off,off)).
cnf(g_0_off_off_off, axiom, f(off,off,off)=f(on,off,off)).
cnf(g_1_off_off_off, axiom, f(off,off,off)=f(off,on,off)).
cnf(g_2_off_off_off, axiom, f(off,off,off)=f(off,off,on)).
```

	Instance size	Rewrite steps	Action equivalents	Prover time
alldiff_switch_gnd001.p.prf	1	1	1	0.003
alldiff_switch_gnd002.p.prf	2	2	2	0.002
alldiff_switch_gnd003.p.prf	3	3	3	0.002
alldiff_switch_gnd004.p.prf	4	4	4	0.002
alldiff_switch_gnd005.p.prf	5	5	5	0.003
alldiff_switch_gnd006.p.prf	6	6	6	0.004
alldiff_switch_gnd007.p.prf	7	7	7	0.009
alldiff_switch_gnd008.p.prf	8	8	8	0.020
alldiff_switch_gnd009.p.prf	9	9	9	0.046
alldiff_switch_gnd010.p.prf	10	10	10	0.114
alldiff_switch_gnd011.p.prf	11	11	11	0.290
alldiff_switch_gnd012.p.prf	12	12	12	0.794
alldiff_switch_gnd013.p.prf	13	13	13	2.272
alldiff_switch_gnd014.p.prf	14	14	14	6.850
alldiff_switch_gnd015.p.prf	15	15	15	22.405
alldiff_switch_gnd016.p.prf	16	16	16	93.106
alldiff_switch_gnd017.p.prf	17	17	17	407.151
alldiff_switch_gnd018.p.prf	18	18	18	1845.499
alldiff_switch_gnd019.p.prf	19	19	19	8076.025
alldiff_switch_gnd020.p.prf	20	20	20	42027.167

Table 3. Results on switch banks with full ground encoding

Action encoding with variables In this setting, the initial and final states differ in all switches, and the actions are encoded as equations using variables to represent the unchanging parts of the state. An example of size 3 is presented below. Results are in Table 4.

```
% Simple switchbank example. There are 3 switches
% that can be on or off. Actions flip a single switch.
%
% This is the version using first-order variables.
%
% The states of non-relevant switches are captured (and
% preserved) by variables, so we only need one rule to
% turn each switch on, and one to turn it off. Thus
% we have 2n=6 action rules.

cnf(to_plan, negated_conjecture, f(on,off,on)!=f(off,on,off)).
cnf(n_3_0_off, axiom, f(on,X1,X2) = f(off,X1,X2)).
cnf(n_3_0_on,  axiom, f(off,X1,X2) = f(on,X1,X2)).
cnf(n_3_1_off, axiom, f(X0,on,X2) = f(X0,off,X2)).
cnf(n_3_1_on,  axiom, f(X0,off,X2) = f(X0,on,X2)).
cnf(n_3_2_off, axiom, f(X0,X1,on) = f(X0,X1,off)).
cnf(n_3_2_on,  axiom, f(X0,X1,off) = f(X0,X1,on)).
```

	Instance size	Rewrite steps	Action equivalents	Prover time
alldiff_switch_var001.p.prf	1	1	1	0.006
alldiff_switch_var002.p.prf	2	2	2	0.005
alldiff_switch_var003.p.prf	3	3	3	0.004
alldiff_switch_var004.p.prf	4	4	4	0.003
alldiff_switch_var005.p.prf	5	5	5	0.003
alldiff_switch_var006.p.prf	6	6	6	0.003
alldiff_switch_var007.p.prf	7	7	7	0.003
alldiff_switch_var008.p.prf	8	8	8	0.003
alldiff_switch_var009.p.prf	9	9	9	0.003
alldiff_switch_var010.p.prf	10	10	10	0.003
alldiff_switch_var011.p.prf	11	11	11	0.003
alldiff_switch_var012.p.prf	12	12	12	0.003
alldiff_switch_var013.p.prf	13	13	13	0.003
alldiff_switch_var014.p.prf	14	14	14	0.003
alldiff_switch_var015.p.prf	15	15	15	0.003
alldiff_switch_var016.p.prf	16	16	16	0.003
alldiff_switch_var017.p.prf	17	17	17	0.003
alldiff_switch_var018.p.prf	18	18	18	0.003
alldiff_switch_var019.p.prf	19	19	19	0.003
alldiff_switch_var020.p.prf	20	20	20	0.003

Table 4. Results on switch banks with frame variable encoding

Actions rewrite subterms In this setting, the initial and final states differ in all switches, and the actions are encoded as equations directly changing the value of a switch at the subterm level. An example of size 3 is presented below. Results are in Table 5.

```
% Simple switchbank example. There are 3 switches
% that can be on or off. Actions flip a single switch.
%
% This is the version using rewriting at subterm positions,
% using a single action rule that can flip any switch.
%
% There should be two axioms, on->off and off->on. However,
% since we require bidirectionality, both are covered by a
% single equation.

cnf(to_plan, negated_conjecture, f(on,off,on)!=f(off,on,off)).
cnf(onoff,axiom, on=off).
```

A.2 Switches – minimal difference, default ordering

The next three examples are encoded as above, but with a conjecture representing different initial and goal states. We present only one example (in the subterm action encoding) to demonstrate this difference. Results are in Tables 6, 7 and

	Instance size	Rewrite steps	Action equivalents	Prover time
alldiff_switch_sub001.p.prf	1	1	1	0.006
alldiff_switch_sub002.p.prf	2	2	2	0.005
alldiff_switch_sub003.p.prf	3	3	3	0.004
alldiff_switch_sub004.p.prf	4	4	4	0.003
alldiff_switch_sub005.p.prf	5	5	5	0.003
alldiff_switch_sub006.p.prf	6	6	6	0.003
alldiff_switch_sub007.p.prf	7	7	7	0.003
alldiff_switch_sub008.p.prf	8	8	8	0.003
alldiff_switch_sub009.p.prf	9	9	9	0.003
alldiff_switch_sub010.p.prf	10	10	10	0.002
alldiff_switch_sub011.p.prf	11	11	11	0.002
alldiff_switch_sub012.p.prf	12	12	12	0.002
alldiff_switch_sub013.p.prf	13	13	13	0.002
alldiff_switch_sub014.p.prf	14	14	14	0.002
alldiff_switch_sub015.p.prf	15	15	15	0.002
alldiff_switch_sub016.p.prf	16	16	16	0.002
alldiff_switch_sub017.p.prf	17	17	17	0.002
alldiff_switch_sub018.p.prf	18	18	18	0.002
alldiff_switch_sub019.p.prf	19	19	19	0.002
alldiff_switch_sub020.p.prf	20	20	20	0.002

Table 5. Results on switch banks with subterm action encoding

8. The prover nearly always generates the optimal ordering by default, but in the onediff_switch_gnd002 example (Table 6) it generates the less than optimal *on>off*.

A.3 Switches – minimal difference, different orderings

These problems use the same encoding as above, but we explicitly specify the term ordering, using `--precedence='f>off>on'` for the first (Table 9) and `--precedence='f>on>off'` for the second (Table 10) set of experiments.

A.4 Tower of Hanoi

Flat encoding In this encoding, a state is a flat term of the form $f(X_1, \ldots, X_n)$, where X_i encodes the ith disk and can take one of the values p_1, p_2, p_3, denoting peg one, two or three, respectively. The task is to move all disks from peg one to peg two. Results are in Table 11.

An example of size 3 is presented below.

	Instance size	Rewrite steps	Action equivalents	Prover time
onediff_switch_gnd001.p.prf	1	1	1	0.003
onediff_switch_gnd002.p.prf	2	3	3	0.002
onediff_switch_gnd003.p.prf	3	1	1	0.002
onediff_switch_gnd004.p.prf	4	1	1	0.002
onediff_switch_gnd005.p.prf	5	1	1	0.003
onediff_switch_gnd006.p.prf	6	1	1	0.004
onediff_switch_gnd007.p.prf	7	1	1	0.008
onediff_switch_gnd008.p.prf	8	1	1	0.019
onediff_switch_gnd009.p.prf	9	1	1	0.046
onediff_switch_gnd010.p.prf	10	1	1	0.111
onediff_switch_gnd011.p.prf	11	1	1	0.283
onediff_switch_gnd012.p.prf	12	1	1	0.767
onediff_switch_gnd013.p.prf	13	1	1	2.136
onediff_switch_gnd014.p.prf	14	1	1	6.257
onediff_switch_gnd015.p.prf	15	1	1	20.225
onediff_switch_gnd016.p.prf	16	1	1	83.521
onediff_switch_gnd017.p.prf	17	1	1	346.459
onediff_switch_gnd018.p.prf	18	1	1	1607.884
onediff_switch_gnd019.p.prf	19	1	1	6946.928
onediff_switch_gnd020.p.prf	20	1	1	31387.790

Table 6. Results on switch banks with one switch difference, ground encoding

	Instance size	Rewrite steps	Action equivalents	Prover time
onediff_switch_var001.p.prf	1	1	1	0.005
onediff_switch_var002.p.prf	2	1	1	0.004
onediff_switch_var003.p.prf	3	1	1	0.003
onediff_switch_var004.p.prf	4	1	1	0.003
onediff_switch_var005.p.prf	5	1	1	0.002
onediff_switch_var006.p.prf	6	1	1	0.003
onediff_switch_var007.p.prf	7	1	1	0.003
onediff_switch_var008.p.prf	8	1	1	0.003
onediff_switch_var009.p.prf	9	1	1	0.003
onediff_switch_var010.p.prf	10	1	1	0.003
onediff_switch_var011.p.prf	11	1	1	0.003
onediff_switch_var012.p.prf	12	1	1	0.003
onediff_switch_var013.p.prf	13	1	1	0.003
onediff_switch_var014.p.prf	14	1	1	0.003
onediff_switch_var015.p.prf	15	1	1	0.003
onediff_switch_var016.p.prf	16	1	1	0.003
onediff_switch_var017.p.prf	17	1	1	0.003
onediff_switch_var018.p.prf	18	1	1	0.003
onediff_switch_var019.p.prf	19	1	1	0.003
onediff_switch_var020.p.prf	20	1	1	0.003

Table 7. Results on switch banks with one switch difference, frame variable encoding

	Instance size	Rewrite steps	Action equivalents	Prover time
onediff_switch_sub001.p.prf	1	1	1	0.007
onediff_switch_sub002.p.prf	2	1	1	0.004
onediff_switch_sub003.p.prf	3	1	1	0.004
onediff_switch_sub004.p.prf	4	1	1	0.003
onediff_switch_sub005.p.prf	5	1	1	0.003
onediff_switch_sub006.p.prf	6	1	1	0.003
onediff_switch_sub007.p.prf	7	1	1	0.003
onediff_switch_sub008.p.prf	8	1	1	0.003
onediff_switch_sub009.p.prf	9	1	1	0.003
onediff_switch_sub010.p.prf	10	1	1	0.003
onediff_switch_sub011.p.prf	11	1	1	0.003
onediff_switch_sub012.p.prf	12	1	1	0.002
onediff_switch_sub013.p.prf	13	1	1	0.002
onediff_switch_sub014.p.prf	14	1	1	0.002
onediff_switch_sub015.p.prf	15	1	1	0.002
onediff_switch_sub016.p.prf	16	1	1	0.002
onediff_switch_sub017.p.prf	17	1	1	0.002
onediff_switch_sub018.p.prf	18	1	1	0.002
onediff_switch_sub019.p.prf	19	1	1	0.002
onediff_switch_sub020.p.prf	20	1	1	0.002

Table 8. Results on switch banks with one switch difference, subterm action encoding

	Instance size	Rewrite steps	Action equivalents	Prover time
off_ononediff_switch_sub001.p.prf	1	1	1	0.003
off_ononediff_switch_sub002.p.prf	2	1	1	0.003
off_ononediff_switch_sub003.p.prf	3	1	1	0.003
off_ononediff_switch_sub004.p.prf	4	1	1	0.002
off_ononediff_switch_sub005.p.prf	5	1	1	0.002
off_ononediff_switch_sub006.p.prf	6	1	1	0.002
off_ononediff_switch_sub007.p.prf	7	1	1	0.002
off_ononediff_switch_sub008.p.prf	8	1	1	0.002
off_ononediff_switch_sub009.p.prf	9	1	1	0.002
off_ononediff_switch_sub010.p.prf	10	1	1	0.002
off_ononediff_switch_sub011.p.prf	11	1	1	0.002
off_ononediff_switch_sub012.p.prf	12	1	1	0.002
off_ononediff_switch_sub013.p.prf	13	1	1	0.002
off_ononediff_switch_sub014.p.prf	14	1	1	0.002
off_ononediff_switch_sub015.p.prf	15	1	1	0.002
off_ononediff_switch_sub016.p.prf	16	1	1	0.002
off_ononediff_switch_sub017.p.prf	17	1	1	0.002
off_ononediff_switch_sub018.p.prf	18	1	1	0.002
off_ononediff_switch_sub019.p.prf	19	1	1	0.002
off_ononediff_switch_sub020.p.prf	20	1	1	0.002

Table 9. Results on switch banks with one switch difference, subterm action encoding, $off > on$

	Instance size	Rewrite steps	Action equivalents	Prover time
on_offonediff_switch_sub001.p.prf	1	1	1	0.005
on_offonediff_switch_sub002.p.prf	2	3	3	0.003
on_offonediff_switch_sub003.p.prf	3	5	5	0.003
on_offonediff_switch_sub004.p.prf	4	7	7	0.003
on_offonediff_switch_sub005.p.prf	5	9	9	0.002
on_offonediff_switch_sub006.p.prf	6	11	11	0.002
on_offonediff_switch_sub007.p.prf	7	13	13	0.002
on_offonediff_switch_sub008.p.prf	8	15	15	0.002
on_offonediff_switch_sub009.p.prf	9	17	17	0.002
on_offonediff_switch_sub010.p.prf	10	19	19	0.002
on_offonediff_switch_sub011.p.prf	11	21	21	0.002
on_offonediff_switch_sub012.p.prf	12	23	23	0.002
on_offonediff_switch_sub013.p.prf	13	25	25	0.002
on_offonediff_switch_sub014.p.prf	14	27	27	0.002
on_offonediff_switch_sub015.p.prf	15	29	29	0.002
on_offonediff_switch_sub016.p.prf	16	31	31	0.002
on_offonediff_switch_sub017.p.prf	17	33	33	0.002
on_offonediff_switch_sub018.p.prf	18	35	35	0.002
on_offonediff_switch_sub019.p.prf	19	37	37	0.002
on_offonediff_switch_sub020.p.prf	20	39	39	0.002

Table 10. Results on switch banks with one switch difference, subterm action encoding, *on>off*

```
% This is a flat encoding of the Tower of Hanoi puzzle.
% There are 3 differently sized disks sitting on one of the
% 3 pegs p1, p2, p3. A bigger disk may never sit on a smaller
% disk. One can move the top disk from one peg to another peg
% if this does nor violate the size constraint.
% The goal is to move all disks from p1 to p2.
%
% The flat encoding represents the disks as argument position
% in a term of the form f(arg1, arg2, ..., argn), where each
% arg can take the value p1, p2 or p3. The largest disk is on
% the left, the smallest on the right.
%
% There should be 6n=18 axioms (plus inital and goal state), one each
% to move disk k from any peg to any other peg (for all 6
% combinations of two distinct pegs). However, because the axioms
% are bidirectional, half of them are redundant, so we only need
% 3n = 9 axioms.

cnf(to_plan, negated_conjecture, f(p1,p1,p1)!=f(p2,p2,p2)).
cnf(h3_0p1p2p3, axiom, f(p1,p3,p3)=f(p2,p3,p3)).
cnf(h3_1p1p2p3, axiom, f(X0,p1,p3)=f(X0,p2,p3)).
cnf(h3_2p1p2p3, axiom, f(X0,X1,p1)=f(X0,X1,p2)).
cnf(h3_0p1p3p2, axiom, f(p1,p2,p2)=f(p3,p2,p2)).
cnf(h3_1p1p3p2, axiom, f(X0,p1,p2)=f(X0,p3,p2)).
cnf(h3_2p1p3p2, axiom, f(X0,X1,p1)=f(X0,X1,p3)).
cnf(h3_0p2p3p1, axiom, f(p2,p1,p1)=f(p3,p1,p1)).
cnf(h3_1p2p3p1, axiom, f(X0,p2,p1)=f(X0,p3,p1)).
cnf(h3_2p2p3p1, axiom, f(X0,X1,p2)=f(X0,X1,p3)).
```

	Instance size	Rewrite steps	Action equivalents	Prover time
hanoi_var001.p.prf	1	1	1	0.008
hanoi_var002.p.prf	2	6	6	0.005
hanoi_var003.p.prf	3	13	17	0.004
hanoi_var004.p.prf	4	27	47	0.004
hanoi_var005.p.prf	5	33	147	0.003
hanoi_var006.p.prf	6	46	436	0.004
hanoi_var007.p.prf	7	61	1301	0.004
hanoi_var008.p.prf	8	78	3894	0.003
hanoi_var009.p.prf	9	97	11671	0.003
hanoi_var010.p.prf	10	118	35000	0.003
hanoi_var011.p.prf	11	141	104985	0.003
hanoi_var012.p.prf	12	163	321495	0.003
hanoi_var013.p.prf	13	193	944795	0.004
hanoi_var014.p.prf	14	208	3188632	0.003
hanoi_var015.p.prf	15	238	9565923	0.004
hanoi_var016.p.prf	16	270	28697798	0.003
hanoi_var017.p.prf	17	304	86093425	0.004
hanoi_var018.p.prf	18	340	258280308	0.004
hanoi_var019.p.prf	19	378	774840959	0.004
hanoi_var020.p.prf	20	418	2324522914	0.004
hanoi_var021.p.prf	21	460	6973568781	0.004
hanoi_var022.p.prf	22	504	20920706384	0.005
hanoi_var023.p.prf	23	550	62762119195	0.005
hanoi_var024.p.prf	24	598	188286357630	0.005
hanoi_var025.p.prf	25	648	564859072937	0.006
hanoi_var026.p.prf	26	700	1694577218860	0.006
hanoi_var027.p.prf	27	754	5083731656631	0.007
hanoi_var028.p.prf	28	810	15251194969946	0.007
hanoi_var029.p.prf	29	868	45753584909893	0.007
hanoi_var030.p.prf	30	928	137260754729736	0.008

Table 11. Results on Tower of Hanoi, flat encoding, default ordering

Flat encoding, different orderings Tables 12 to 17 show the results for the Tower of Hanoi problem with flat encoding and different term orderings.

List (recursive) encoding with subterm actions In this encoding of the Tower of Hanoi problem, a state is a recursively encoded list of positions, $f(X_1, f(X_2, f(\ldots, \bot)))$, where again X_i encodes the i^{th} disk and can take one of the values p_1, p_2, p_3. The task is to move all disks from peg one to peg two. The largest disk is on the left, the smallest on the right.

A bigger disk may never sit on a smaller disk. One can move the top disk from one peg to another peg if this does nor violate the size constraint.The goal is to move all disks from p_1 to p_2. Results are in Table 18.

There should be $6n = 18$ axioms (plus inital and goal state), one each to move disk k from any peg to any other peg (for all 6 combinations of two distinct

	Instance size	Rewrite steps	Action equivalents	Prover time
p1p2p3_hanoi_var001.p.prf	1	1	1	0.009
p1p2p3_hanoi_var002.p.prf	2	10	13	0.004
p1p2p3_hanoi_var003.p.prf	3	11	14	0.002
p1p2p3_hanoi_var004.p.prf	4	20	33	0.002
p1p2p3_hanoi_var005.p.prf	5	30	110	0.002
p1p2p3_hanoi_var006.p.prf	6	42	339	0.002
p1p2p3_hanoi_var007.p.prf	7	56	1024	0.002
p1p2p3_hanoi_var008.p.prf	8	72	3077	0.002
p1p2p3_hanoi_var009.p.prf	9	90	9234	0.002
p1p2p3_hanoi_var010.p.prf	10	110	27703	0.003
p1p2p3_hanoi_var011.p.prf	11	132	83108	0.003
p1p2p3_hanoi_var012.p.prf	12	155	354304	0.003
p1p2p3_hanoi_var013.p.prf	13	182	747958	0.003
p1p2p3_hanoi_var014.p.prf	14	209	3188658	0.003
p1p2p3_hanoi_var015.p.prf	15	239	9565951	0.003
p1p2p3_hanoi_var016.p.prf	16	271	28697828	0.003
p1p2p3_hanoi_var017.p.prf	17	305	86093457	0.004
p1p2p3_hanoi_var018.p.prf	18	341	258280342	0.004
p1p2p3_hanoi_var019.p.prf	19	379	774840995	0.004
p1p2p3_hanoi_var020.p.prf	20	419	2324522952	0.004
p1p2p3_hanoi_var021.p.prf	21	461	6973568821	0.005
p1p2p3_hanoi_var022.p.prf	22	505	20920706426	0.005
p1p2p3_hanoi_var023.p.prf	23	551	62762119239	0.005
p1p2p3_hanoi_var024.p.prf	24	599	188286357676	0.005
p1p2p3_hanoi_var025.p.prf	25	649	564859072985	0.006
p1p2p3_hanoi_var026.p.prf	26	701	1694577218910	0.006
p1p2p3_hanoi_var027.p.prf	27	755	5083731656683	0.007
p1p2p3_hanoi_var028.p.prf	28	811	15251194970000	0.007
p1p2p3_hanoi_var029.p.prf	29	869	45753584909949	0.007
p1p2p3_hanoi_var030.p.prf	30	929	137260754729794	0.008

Table 12. Results on Tower of Hanoi, flat encoding, $p_1 > p_2 > p_3$

pegs). However, because the axioms are bidirectional, half of them are redundant, so we need only $3n = 9$ axioms. The axioms also are shortened with identical variables to the left removed so for example $f(X_1, f(X_2, f(p_1, f(p_2, \bot)))) = f(X_1, f(X_2, f(p_3, f(p_2, \bot)))$ can be shortened to $f(p_1, f(p_2, \bot)) = f(p_3, f(p_2, \bot))$.

An example of the recursive ordering of size 3 is presented below. In the example of size 3 given below, there are 3 differently sized disks sitting on one of the 3 pegs p_1, p_2, p_3.

	Instance size	Rewrite steps	Action equivalents	Prover time
p1p3p2_hanoi_var001.p.prf	1	1	1	0.008
p1p3p2_hanoi_var002.p.prf	2	7	8	0.004
p1p3p2_hanoi_var003.p.prf	3	13	17	0.002
p1p3p2_hanoi_var004.p.prf	4	28	51	0.002
p1p3p2_hanoi_var005.p.prf	5	34	155	0.002
p1p3p2_hanoi_var006.p.prf	6	47	462	0.002
p1p3p2_hanoi_var007.p.prf	7	62	1381	0.002
p1p3p2_hanoi_var008.p.prf	8	79	4136	0.002
p1p3p2_hanoi_var009.p.prf	9	98	12399	0.002
p1p3p2_hanoi_var010.p.prf	10	119	37186	0.003
p1p3p2_hanoi_var011.p.prf	11	142	111545	0.003
p1p3p2_hanoi_var012.p.prf	12	168	354301	0.003
p1p3p2_hanoi_var013.p.prf	13	194	1003843	0.003
p1p3p2_hanoi_var014.p.prf	14	353	3188646	0.003
p1p3p2_hanoi_var015.p.prf	15	408	9565937	0.003
p1p3p2_hanoi_var016.p.prf	16	467	28697814	0.003
p1p3p2_hanoi_var017.p.prf	17	530	86093441	0.004
p1p3p2_hanoi_var018.p.prf	18	597	258280326	0.004
p1p3p2_hanoi_var019.p.prf	19	668	774840977	0.004
p1p3p2_hanoi_var020.p.prf	20	743	2324522934	0.004
p1p3p2_hanoi_var021.p.prf	21	822	6973568801	0.005
p1p3p2_hanoi_var022.p.prf	22	905	20920706406	0.005
p1p3p2_hanoi_var023.p.prf	23	992	62762119217	0.005
p1p3p2_hanoi_var024.p.prf	24	1083	188286357654	0.005
p1p3p2_hanoi_var025.p.prf	25	1178	564859072961	0.006
p1p3p2_hanoi_var026.p.prf	26	1277	1694577218886	0.006
p1p3p2_hanoi_var027.p.prf	27	1380	5083731656657	0.007
p1p3p2_hanoi_var028.p.prf	28	1487	15251194969974	0.007
p1p3p2_hanoi_var029.p.prf	29	1598	45753584909921	0.007
p1p3p2_hanoi_var030.p.prf	30	1713	137260754729766	0.008

Table 13. Results on Tower of Hanoi, flat encoding, $p_1 > p_3 > p_2$

```
List (recursive) encoding of Tower of Hanoi for size 3
cnf(to_plan, negated_conjecture, f(p1,f(p1,f(p1,bot)))!=f(p2,f(p2,f(p2,bot)))).
cnf(h3_0p1p2p3, axiom, f(p1,f(p3,f(p3,bot)))=f(p2,f(p3,f(p3,bot)))).
cnf(h3_1p1p2p3, axiom, f(p1,f(p3,bot))=f(p2,f(p3,bot))).
cnf(h3_2p1p2p3, axiom, f(p1,bot)=f(p2,bot)).
cnf(h3_0p1p3p2, axiom, f(p1,f(p2,f(p2,bot)))=f(p3,f(p2,f(p2,bot)))).
cnf(h3_1p1p3p2, axiom, f(p1,f(p2,bot))=f(p3,f(p2,bot))).
cnf(h3_2p1p3p2, axiom, f(p1,bot)=f(p3,bot)).
cnf(h3_0p2p3p1, axiom, f(p2,f(p1,f(p1,bot)))=f(p3,f(p1,f(p1,bot)))).
cnf(h3_1p2p3p1, axiom, f(p2,f(p1,bot))=f(p3,f(p1,bot))).
cnf(h3_2p2p3p1, axiom, f(p2,bot)=f(p3,bot)).
```

Recursive encoding, different orderings Tables 19 to 24 show the results for the Tower of Hanoi problem with recursive encoding and different term orderings.

	Instance size	Rewrite steps	Action equivalents	Prover time
p2p1p3_hanoi_var001.p.prf	1	1	1	0.005
p2p1p3_hanoi_var002.p.prf	2	9	9	0.004
p2p1p3_hanoi_var003.p.prf	3	10	10	0.002
p2p1p3_hanoi_var004.p.prf	4	19	25	0.002
p2p1p3_hanoi_var005.p.prf	5	29	82	0.002
p2p1p3_hanoi_var006.p.prf	6	41	251	0.002
p2p1p3_hanoi_var007.p.prf	7	55	756	0.002
p2p1p3_hanoi_var008.p.prf	8	71	2269	0.002
p2p1p3_hanoi_var009.p.prf	9	89	6806	0.002
p2p1p3_hanoi_var010.p.prf	10	109	20415	0.003
p2p1p3_hanoi_var011.p.prf	11	131	61240	0.003
p2p1p3_hanoi_var012.p.prf	12	154	236206	0.003
p2p1p3_hanoi_var013.p.prf	13	181	551130	0.003
p2p1p3_hanoi_var014.p.prf	14	208	2125776	0.003
p2p1p3_hanoi_var015.p.prf	15	238	6377305	0.003
p2p1p3_hanoi_var016.p.prf	16	270	19131890	0.003
p2p1p3_hanoi_var017.p.prf	17	304	57395643	0.004
p2p1p3_hanoi_var018.p.prf	18	340	172186900	0.004
p2p1p3_hanoi_var019.p.prf	19	378	516560669	0.004
p2p1p3_hanoi_var020.p.prf	20	418	1549681974	0.004
p2p1p3_hanoi_var021.p.prf	21	460	4649045887	0.005
p2p1p3_hanoi_var022.p.prf	22	504	13947137624	0.005
p2p1p3_hanoi_var023.p.prf	23	550	41841412833	0.005
p2p1p3_hanoi_var024.p.prf	24	598	125524238458	0.006
p2p1p3_hanoi_var025.p.prf	25	648	376572715331	0.006
p2p1p3_hanoi_var026.p.prf	26	700	1129718145948	0.006
p2p1p3_hanoi_var027.p.prf	27	754	3389154437797	0.007
p2p1p3_hanoi_var028.p.prf	28	810	10167463313342	0.007
p2p1p3_hanoi_var029.p.prf	29	868	30502389939975	0.007
p2p1p3_hanoi_var030.p.prf	30	928	91507169819872	0.008

Table 14. Results on Tower of Hanoi, flat encoding, $p_2 > p_1 > p_3$

	Instance size	Rewrite steps	Action equivalents	Prover time
p2p3p1_hanoi_var001.p.prf	1	1	1	0.005
p2p3p1_hanoi_var002.p.prf	2	6	6	0.003
p2p3p1_hanoi_var003.p.prf	3	13	17	0.002
p2p3p1_hanoi_var004.p.prf	4	27	47	0.002
p2p3p1_hanoi_var005.p.prf	5	33	147	0.002
p2p3p1_hanoi_var006.p.prf	6	46	436	0.002
p2p3p1_hanoi_var007.p.prf	7	61	1301	0.002
p2p3p1_hanoi_var008.p.prf	8	78	3894	0.002
p2p3p1_hanoi_var009.p.prf	9	97	11671	0.002
p2p3p1_hanoi_var010.p.prf	10	118	35000	0.003
p2p3p1_hanoi_var011.p.prf	11	141	104985	0.003
p2p3p1_hanoi_var012.p.prf	12	163	321495	0.003
p2p3p1_hanoi_var013.p.prf	13	193	944795	0.003
p2p3p1_hanoi_var014.p.prf	14	208	3188632	0.003
p2p3p1_hanoi_var015.p.prf	15	238	9565923	0.003
p2p3p1_hanoi_var016.p.prf	16	270	28697798	0.003
p2p3p1_hanoi_var017.p.prf	17	304	86093425	0.004
p2p3p1_hanoi_var018.p.prf	18	340	258280308	0.004
p2p3p1_hanoi_var019.p.prf	19	378	774840959	0.004
p2p3p1_hanoi_var020.p.prf	20	418	2324522914	0.004
p2p3p1_hanoi_var021.p.prf	21	460	6973568781	0.004
p2p3p1_hanoi_var022.p.prf	22	504	20920706384	0.005
p2p3p1_hanoi_var023.p.prf	23	550	62762119195	0.005
p2p3p1_hanoi_var024.p.prf	24	598	188286357630	0.005
p2p3p1_hanoi_var025.p.prf	25	648	564859072937	0.006
p2p3p1_hanoi_var026.p.prf	26	700	1694577218860	0.006
p2p3p1_hanoi_var027.p.prf	27	754	5083731656631	0.006
p2p3p1_hanoi_var028.p.prf	28	810	15251194969946	0.007
p2p3p1_hanoi_var029.p.prf	29	868	45753584909893	0.007
p2p3p1_hanoi_var030.p.prf	30	928	137260754729736	0.008

Table 15. Results on Tower of Hanoi, flat encoding, $p_2 > p_3 > p_1$

	Instance size	Rewrite steps	Action equivalents	Prover time
p3p1p2_hanoi_var001.p.prf	1	1	1	0.004
p3p1p2_hanoi_var002.p.prf	2	5	6	0.003
p3p1p2_hanoi_var003.p.prf	3	11	17	0.002
p3p1p2_hanoi_var004.p.prf	4	19	53	0.002
p3p1p2_hanoi_var005.p.prf	5	29	161	0.002
p3p1p2_hanoi_var006.p.prf	6	41	485	0.002
p3p1p2_hanoi_var007.p.prf	7	55	1457	0.002
p3p1p2_hanoi_var008.p.prf	8	71	4373	0.002
p3p1p2_hanoi_var009.p.prf	9	89	13121	0.002
p3p1p2_hanoi_var010.p.prf	10	109	39365	0.003
p3p1p2_hanoi_var011.p.prf	11	131	118097	0.003
p3p1p2_hanoi_var012.p.prf	12	156	354293	0.003
p3p1p2_hanoi_var013.p.prf	13	181	1062881	0.003
p3p1p2_hanoi_var014.p.prf	14	353	3188646	0.003
p3p1p2_hanoi_var015.p.prf	15	408	9565937	0.003
p3p1p2_hanoi_var016.p.prf	16	467	28697814	0.003
p3p1p2_hanoi_var017.p.prf	17	530	86093441	0.004
p3p1p2_hanoi_var018.p.prf	18	597	258280326	0.004
p3p1p2_hanoi_var019.p.prf	19	668	774840977	0.004
p3p1p2_hanoi_var020.p.prf	20	743	2324522934	0.004
p3p1p2_hanoi_var021.p.prf	21	822	6973568801	0.005
p3p1p2_hanoi_var022.p.prf	22	905	20920706406	0.005
p3p1p2_hanoi_var023.p.prf	23	992	62762119217	0.005
p3p1p2_hanoi_var024.p.prf	24	1083	188286357654	0.005
p3p1p2_hanoi_var025.p.prf	25	1178	564859072961	0.006
p3p1p2_hanoi_var026.p.prf	26	1277	1694577218886	0.006
p3p1p2_hanoi_var027.p.prf	27	1380	5083731656657	0.007
p3p1p2_hanoi_var028.p.prf	28	1487	15251194969974	0.007
p3p1p2_hanoi_var029.p.prf	29	1598	45753584909921	0.007
p3p1p2_hanoi_var030.p.prf	30	1713	137260754729766	0.008

Table 16. Results on Tower of Hanoi, flat encoding, $p_3 > p_1 > p_2$

	Instance size	Rewrite steps	Action equivalents	Prover time
p3p2p1_hanoi_var001.p.prf	1	1	1	0.005
p3p2p1_hanoi_var002.p.prf	2	4	4	0.003
p3p2p1_hanoi_var003.p.prf	3	11	17	0.002
p3p2p1_hanoi_var004.p.prf	4	19	53	0.002
p3p2p1_hanoi_var005.p.prf	5	29	161	0.002
p3p2p1_hanoi_var006.p.prf	6	41	485	0.002
p3p2p1_hanoi_var007.p.prf	7	55	1457	0.002
p3p2p1_hanoi_var008.p.prf	8	71	4373	0.002
p3p2p1_hanoi_var009.p.prf	9	89	13121	0.002
p3p2p1_hanoi_var010.p.prf	10	109	39365	0.003
p3p2p1_hanoi_var011.p.prf	11	131	118097	0.003
p3p2p1_hanoi_var012.p.prf	12	154	314927	0.003
p3p2p1_hanoi_var013.p.prf	13	181	1062881	0.003
p3p2p1_hanoi_var014.p.prf	14	208	3188632	0.003
p3p2p1_hanoi_var015.p.prf	15	238	9565923	0.003
p3p2p1_hanoi_var016.p.prf	16	270	28697798	0.003
p3p2p1_hanoi_var017.p.prf	17	304	86093425	0.003
p3p2p1_hanoi_var018.p.prf	18	340	258280308	0.004
p3p2p1_hanoi_var019.p.prf	19	378	774840959	0.004
p3p2p1_hanoi_var020.p.prf	20	418	2324522914	0.004
p3p2p1_hanoi_var021.p.prf	21	460	6973568781	0.004
p3p2p1_hanoi_var022.p.prf	22	504	20920706384	0.005
p3p2p1_hanoi_var023.p.prf	23	550	62762119195	0.005
p3p2p1_hanoi_var024.p.prf	24	598	188286357630	0.005
p3p2p1_hanoi_var025.p.prf	25	648	564859072937	0.006
p3p2p1_hanoi_var026.p.prf	26	700	1694577218860	0.006
p3p2p1_hanoi_var027.p.prf	27	754	5083731656631	0.006
p3p2p1_hanoi_var028.p.prf	28	810	15251194969946	0.007
p3p2p1_hanoi_var029.p.prf	29	868	45753584909893	0.007
p3p2p1_hanoi_var030.p.prf	30	928	137260754729736	0.008

Table 17. Results on Tower of Hanoi, flat encoding, $p_3 > p_2 > p_1$

	Instance size	Rewrite steps	Action equivalents	Prover time
hanoi_sub001.p.prf	1	1	1	0.007
hanoi_sub002.p.prf	2	4	4	0.004
hanoi_sub003.p.prf	3	9	15	0.004
hanoi_sub004.p.prf	4	16	50	0.004
hanoi_sub005.p.prf	5	25	157	0.003
hanoi_sub006.p.prf	6	36	480	0.003
hanoi_sub007.p.prf	7	49	1451	0.003
hanoi_sub008.p.prf	8	64	4366	0.003
hanoi_sub009.p.prf	9	81	13113	0.003
hanoi_sub010.p.prf	10	100	39356	0.003
hanoi_sub011.p.prf	11	121	118087	0.003
hanoi_sub012.p.prf	12	144	354282	0.003
hanoi_sub013.p.prf	13	169	1062869	0.003
hanoi_sub014.p.prf	14	196	3188632	0.003
hanoi_sub015.p.prf	15	225	9565923	0.003
hanoi_sub016.p.prf	16	256	28697798	0.003
hanoi_sub017.p.prf	17	289	86093425	0.003
hanoi_sub018.p.prf	18	324	258280308	0.004
hanoi_sub019.p.prf	19	361	774840959	0.004
hanoi_sub020.p.prf	20	400	2324522914	0.004
hanoi_sub021.p.prf	21	441	6973568781	0.004
hanoi_sub022.p.prf	22	484	20920706384	0.004
hanoi_sub023.p.prf	23	529	62762119195	0.004
hanoi_sub024.p.prf	24	576	188286357630	0.005
hanoi_sub025.p.prf	25	625	564859072937	0.005
hanoi_sub026.p.prf	26	676	1694577218860	0.005
hanoi_sub027.p.prf	27	729	5083731656631	0.005
hanoi_sub028.p.prf	28	784	15251194969946	0.006
hanoi_sub029.p.prf	29	841	45753584909893	0.006
hanoi_sub030.p.prf	30	900	137260754729736	0.007

Table 18. Results on Tower of Hanoi, recursive encoding, default ordering

	Instance size	Rewrite steps	Action equivalents	Prover time
p1p2p3_hanoi_sub001.p.prf	1	1	1	0.009
p1p2p3_hanoi_sub002.p.prf	2	5	6	0.003
p1p2p3_hanoi_sub003.p.prf	3	10	21	0.003
p1p2p3_hanoi_sub004.p.prf	4	18	68	0.002
p1p2p3_hanoi_sub005.p.prf	5	28	211	0.002
p1p2p3_hanoi_sub006.p.prf	6	40	642	0.002
p1p2p3_hanoi_sub007.p.prf	7	54	1937	0.002
p1p2p3_hanoi_sub008.p.prf	8	70	5824	0.002
p1p2p3_hanoi_sub009.p.prf	9	88	17487	0.002
p1p2p3_hanoi_sub010.p.prf	10	108	52478	0.003
p1p2p3_hanoi_sub011.p.prf	11	130	157453	0.003
p1p2p3_hanoi_sub012.p.prf	12	154	472380	0.003
p1p2p3_hanoi_sub013.p.prf	13	180	1417163	0.003
p1p2p3_hanoi_sub014.p.prf	14	208	4251514	0.003
p1p2p3_hanoi_sub015.p.prf	15	238	12754569	0.004
p1p2p3_hanoi_sub016.p.prf	16	270	38263736	0.004
p1p2p3_hanoi_sub017.p.prf	17	304	114791239	0.004
p1p2p3_hanoi_sub018.p.prf	18	340	344373750	0.004
p1p2p3_hanoi_sub019.p.prf	19	378	1033121285	0.005
p1p2p3_hanoi_sub020.p.prf	20	418	3099363892	0.005
p1p2p3_hanoi_sub021.p.prf	21	460	9298091715	0.005
p1p2p3_hanoi_sub022.p.prf	22	504	27894275186	0.006
p1p2p3_hanoi_sub023.p.prf	23	550	83682825601	0.006
p1p2p3_hanoi_sub024.p.prf	24	598	251048476848	0.007
p1p2p3_hanoi_sub025.p.prf	25	648	753145430591	0.007
p1p2p3_hanoi_sub026.p.prf	26	700	2259436291822	0.007
p1p2p3_hanoi_sub027.p.prf	27	754	6778308875517	0.008
p1p2p3_hanoi_sub028.p.prf	28	810	20334926626604	0.008
p1p2p3_hanoi_sub029.p.prf	29	868	61004779879867	0.009
p1p2p3_hanoi_sub030.p.prf	30	928	183014339639658	0.009

Table 19. Results on Tower of Hanoi, recursive encoding, $p_1 > p_2 > p_3$

	Instance size	Rewrite steps	Action equivalents	Prover time
p1p3p2_hanoi_sub001.p.prf	1	1	1	0.006
p1p3p2_hanoi_sub002.p.prf	2	5	6	0.003
p1p3p2_hanoi_sub003.p.prf	3	11	17	0.002
p1p3p2_hanoi_sub004.p.prf	4	21	54	0.002
p1p3p2_hanoi_sub005.p.prf	5	35	161	0.002
p1p3p2_hanoi_sub006.p.prf	6	53	486	0.002
p1p3p2_hanoi_sub007.p.prf	7	75	1457	0.002
p1p3p2_hanoi_sub008.p.prf	8	101	4374	0.002
p1p3p2_hanoi_sub009.p.prf	9	131	13121	0.002
p1p3p2_hanoi_sub010.p.prf	10	165	39366	0.002
p1p3p2_hanoi_sub011.p.prf	11	203	118097	0.002
p1p3p2_hanoi_sub012.p.prf	12	245	354294	0.003
p1p3p2_hanoi_sub013.p.prf	13	291	1062881	0.003
p1p3p2_hanoi_sub014.p.prf	14	341	3188646	0.003
p1p3p2_hanoi_sub015.p.prf	15	395	9565937	0.003
p1p3p2_hanoi_sub016.p.prf	16	453	28697814	0.003
p1p3p2_hanoi_sub017.p.prf	17	515	86093441	0.003
p1p3p2_hanoi_sub018.p.prf	18	581	258280326	0.003
p1p3p2_hanoi_sub019.p.prf	19	651	774840977	0.004
p1p3p2_hanoi_sub020.p.prf	20	725	2324522934	0.004
p1p3p2_hanoi_sub021.p.prf	21	803	6973568801	0.004
p1p3p2_hanoi_sub022.p.prf	22	885	20920706406	0.004
p1p3p2_hanoi_sub023.p.prf	23	971	62762119217	0.005
p1p3p2_hanoi_sub024.p.prf	24	1061	188286357654	0.005
p1p3p2_hanoi_sub025.p.prf	25	1155	564859072961	0.005
p1p3p2_hanoi_sub026.p.prf	26	1253	1694577218886	0.005
p1p3p2_hanoi_sub027.p.prf	27	1355	5083731656657	0.006
p1p3p2_hanoi_sub028.p.prf	28	1461	15251194969974	0.006
p1p3p2_hanoi_sub029.p.prf	29	1571	45753584909921	0.006
p1p3p2_hanoi_sub030.p.prf	30	1685	137260754729766	0.007

Table 20. Results on Tower of Hanoi, recursive encoding, $p_1 > p_3 > p_2$

	Instance size	Rewrite steps	Action equivalents	Prover time
p2p1p3_hanoi_sub001.p.prf	1	1	1	0.005
p2p1p3_hanoi_sub002.p.prf	2	4	4	0.003
p2p1p3_hanoi_sub003.p.prf	3	8	13	0.002
p2p1p3_hanoi_sub004.p.prf	4	14	38	0.002
p2p1p3_hanoi_sub005.p.prf	5	22	111	0.002
p2p1p3_hanoi_sub006.p.prf	6	32	328	0.002
p2p1p3_hanoi_sub007.p.prf	7	44	977	0.002
p2p1p3_hanoi_sub008.p.prf	8	58	2922	0.002
p2p1p3_hanoi_sub009.p.prf	9	74	8755	0.002
p2p1p3_hanoi_sub010.p.prf	10	92	26252	0.002
p2p1p3_hanoi_sub011.p.prf	11	112	78741	0.003
p2p1p3_hanoi_sub012.p.prf	12	134	236206	0.003
p2p1p3_hanoi_sub013.p.prf	13	158	708599	0.003
p2p1p3_hanoi_sub014.p.prf	14	184	2125776	0.003
p2p1p3_hanoi_sub015.p.prf	15	212	6377305	0.003
p2p1p3_hanoi_sub016.p.prf	16	242	19131890	0.004
p2p1p3_hanoi_sub017.p.prf	17	274	57395643	0.004
p2p1p3_hanoi_sub018.p.prf	18	308	172186900	0.004
p2p1p3_hanoi_sub019.p.prf	19	344	516560669	0.004
p2p1p3_hanoi_sub020.p.prf	20	382	1549681974	0.005
p2p1p3_hanoi_sub021.p.prf	21	422	4649045887	0.005
p2p1p3_hanoi_sub022.p.prf	22	464	13947137624	0.005
p2p1p3_hanoi_sub023.p.prf	23	508	41841412833	0.006
p2p1p3_hanoi_sub024.p.prf	24	554	125524238458	0.006
p2p1p3_hanoi_sub025.p.prf	25	602	376572715331	0.006
p2p1p3_hanoi_sub026.p.prf	26	652	1129718145948	0.007
p2p1p3_hanoi_sub027.p.prf	27	704	3389154437797	0.007
p2p1p3_hanoi_sub028.p.prf	28	758	10167463313342	0.008
p2p1p3_hanoi_sub029.p.prf	29	814	30502389939975	0.008
p2p1p3_hanoi_sub030.p.prf	30	872	91507169819872	0.008

Table 21. Results on Tower of Hanoi, recursive encoding, $p_2 > p_1 > p_3$

	Instance size	Rewrite steps	Action equivalents	Prover time
p2p3p1_hanoi_sub001.p.prf	1	1	1	0.004
p2p3p1_hanoi_sub002.p.prf	2	4	4	0.003
p2p3p1_hanoi_sub003.p.prf	3	9	15	0.002
p2p3p1_hanoi_sub004.p.prf	4	16	50	0.002
p2p3p1_hanoi_sub005.p.prf	5	25	157	0.002
p2p3p1_hanoi_sub006.p.prf	6	36	480	0.002
p2p3p1_hanoi_sub007.p.prf	7	49	1451	0.002
p2p3p1_hanoi_sub008.p.prf	8	64	4366	0.002
p2p3p1_hanoi_sub009.p.prf	9	81	13113	0.002
p2p3p1_hanoi_sub010.p.prf	10	100	39356	0.002
p2p3p1_hanoi_sub011.p.prf	11	121	118087	0.002
p2p3p1_hanoi_sub012.p.prf	12	144	354282	0.003
p2p3p1_hanoi_sub013.p.prf	13	169	1062869	0.003
p2p3p1_hanoi_sub014.p.prf	14	196	3188632	0.003
p2p3p1_hanoi_sub015.p.prf	15	225	9565923	0.003
p2p3p1_hanoi_sub016.p.prf	16	256	28697798	0.003
p2p3p1_hanoi_sub017.p.prf	17	289	86093425	0.003
p2p3p1_hanoi_sub018.p.prf	18	324	258280308	0.003
p2p3p1_hanoi_sub019.p.prf	19	361	774840959	0.004
p2p3p1_hanoi_sub020.p.prf	20	400	2324522914	0.004
p2p3p1_hanoi_sub021.p.prf	21	441	6973568781	0.004
p2p3p1_hanoi_sub022.p.prf	22	484	20920706384	0.004
p2p3p1_hanoi_sub023.p.prf	23	529	62762119195	0.005
p2p3p1_hanoi_sub024.p.prf	24	576	188286357630	0.005
p2p3p1_hanoi_sub025.p.prf	25	625	564859072937	0.005
p2p3p1_hanoi_sub026.p.prf	26	676	1694577218860	0.005
p2p3p1_hanoi_sub027.p.prf	27	729	5083731656631	0.005
p2p3p1_hanoi_sub028.p.prf	28	784	15251194969946	0.006
p2p3p1_hanoi_sub029.p.prf	29	841	45753584909893	0.006
p2p3p1_hanoi_sub030.p.prf	30	900	137260754729736	0.006

Table 22. Results on Tower of Hanoi, recursive encoding, $p_2 > p_3 > p_1$

	Instance size	Rewrite steps	Action equivalents	Prover time
p3p1p2_hanoi_sub001.p.prf	1	1	1	0.004
p3p1p2_hanoi_sub002.p.prf	2	5	6	0.003
p3p1p2_hanoi_sub003.p.prf	3	11	17	0.002
p3p1p2_hanoi_sub004.p.prf	4	21	54	0.002
p3p1p2_hanoi_sub005.p.prf	5	35	161	0.002
p3p1p2_hanoi_sub006.p.prf	6	53	486	0.002
p3p1p2_hanoi_sub007.p.prf	7	75	1457	0.002
p3p1p2_hanoi_sub008.p.prf	8	101	4374	0.002
p3p1p2_hanoi_sub009.p.prf	9	131	13121	0.002
p3p1p2_hanoi_sub010.p.prf	10	165	39366	0.002
p3p1p2_hanoi_sub011.p.prf	11	203	118097	0.002
p3p1p2_hanoi_sub012.p.prf	12	245	354294	0.003
p3p1p2_hanoi_sub013.p.prf	13	291	1062881	0.003
p3p1p2_hanoi_sub014.p.prf	14	341	3188646	0.003
p3p1p2_hanoi_sub015.p.prf	15	395	9565937	0.003
p3p1p2_hanoi_sub016.p.prf	16	453	28697814	0.003
p3p1p2_hanoi_sub017.p.prf	17	515	86093441	0.003
p3p1p2_hanoi_sub018.p.prf	18	581	258280326	0.003
p3p1p2_hanoi_sub019.p.prf	19	651	774840977	0.004
p3p1p2_hanoi_sub020.p.prf	20	725	2324522934	0.004
p3p1p2_hanoi_sub021.p.prf	21	803	6973568801	0.004
p3p1p2_hanoi_sub022.p.prf	22	885	20920706406	0.004
p3p1p2_hanoi_sub023.p.prf	23	971	62762119217	0.005
p3p1p2_hanoi_sub024.p.prf	24	1061	188286357654	0.005
p3p1p2_hanoi_sub025.p.prf	25	1155	564859072961	0.005
p3p1p2_hanoi_sub026.p.prf	26	1253	1694577218886	0.005
p3p1p2_hanoi_sub027.p.prf	27	1355	5083731656657	0.005
p3p1p2_hanoi_sub028.p.prf	28	1461	15251194969974	0.006
p3p1p2_hanoi_sub029.p.prf	29	1571	45753584909921	0.006
p3p1p2_hanoi_sub030.p.prf	30	1685	137260754729766	0.006

Table 23. Results on Tower of Hanoi, recursive encoding, $p_3 > p_1 > p_2$

	Instance size	Rewrite steps	Action equivalents	Prover time
p3p2p1_hanoi_sub001.p.prf	1	1	1	0.003
p3p2p1_hanoi_sub002.p.prf	2	4	4	0.004
p3p2p1_hanoi_sub003.p.prf	3	9	15	0.002
p3p2p1_hanoi_sub004.p.prf	4	16	50	0.002
p3p2p1_hanoi_sub005.p.prf	5	25	157	0.002
p3p2p1_hanoi_sub006.p.prf	6	36	480	0.002
p3p2p1_hanoi_sub007.p.prf	7	49	1451	0.002
p3p2p1_hanoi_sub008.p.prf	8	64	4366	0.002
p3p2p1_hanoi_sub009.p.prf	9	81	13113	0.002
p3p2p1_hanoi_sub010.p.prf	10	100	39356	0.002
p3p2p1_hanoi_sub011.p.prf	11	121	118087	0.002
p3p2p1_hanoi_sub012.p.prf	12	144	354282	0.003
p3p2p1_hanoi_sub013.p.prf	13	169	1062869	0.003
p3p2p1_hanoi_sub014.p.prf	14	196	3188632	0.003
p3p2p1_hanoi_sub015.p.prf	15	225	9565923	0.003
p3p2p1_hanoi_sub016.p.prf	16	256	28697798	0.003
p3p2p1_hanoi_sub017.p.prf	17	289	86093425	0.003
p3p2p1_hanoi_sub018.p.prf	18	324	258280308	0.003
p3p2p1_hanoi_sub019.p.prf	19	361	774840959	0.004
p3p2p1_hanoi_sub020.p.prf	20	400	2324522914	0.004
p3p2p1_hanoi_sub021.p.prf	21	441	6973568781	0.004
p3p2p1_hanoi_sub022.p.prf	22	484	20920706384	0.004
p3p2p1_hanoi_sub023.p.prf	23	529	62762119195	0.005
p3p2p1_hanoi_sub024.p.prf	24	576	188286357630	0.005
p3p2p1_hanoi_sub025.p.prf	25	625	564859072937	0.005
p3p2p1_hanoi_sub026.p.prf	26	676	1694577218860	0.005
p3p2p1_hanoi_sub027.p.prf	27	729	5083731656631	0.005
p3p2p1_hanoi_sub028.p.prf	28	784	15251194969946	0.006
p3p2p1_hanoi_sub029.p.prf	29	841	45753584909893	0.006
p3p2p1_hanoi_sub030.p.prf	30	900	137260754729736	0.006

Table 24. Results on Tower of Hanoi, recursive encoding, $p_3 > p_2 > p_1$