Decidability and Undecidability Results for Planning with Numerical State Variables

Malte Helmert
Institut für Informatik, Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee, Gebäude 052, 79110 Freiburg, Germany
helmert@informatik.uni-freiburg.de

Abstract

These days, propositional planning can be considered a quite well-understood problem. Good algorithms are known that can solve a wealth of very different and sometimes challenging planning tasks, and theoretical computational properties of both general STRIPS-style planning and the best-known benchmark problems have been established.

However, propositional planning has a major drawback: The formalism is too weak to allow for the easy encoding of many genuinely interesting planning problems, specifically those involving numbers. A recent effort to enhance the PDDL planning language to cope with (among other additions) numerical state variables, to be used at the third international planning competition, has increased interest in these issues.

In this contribution, we analyze "STRIPS with numbers" from a theoretical point of view. Specifically, we show that the introduction of numerical state variables makes the planning problem undecidable in the general case and many restrictions thereof and identify special cases for which we can provide decidability results.

Introduction

In recent years, we have seen significant progress on the solution of propositional planning tasks. Fully automated planning systems such as FF (Hoffmann & Nebel 2001) and systems using hand-tailored control knowledge such as **TALplanner** (Doherty & Kvarnström 2001) show impressive performance on the problem suite of the second international planning competition. Many researchers have seen this as an indication that it is time to move on to richer – and perhaps more interesting – domain definition languages than the STRIPS or ADL subsets of PDDL, and consequently, the PDDL language has been extended to allow for numerical state variables, explicit models of concurrency, and specification of other metrics of plan quality than sequential length.

The result of this effort, the PDDL2.1 language defined by Fox and Long (2001), has the potential to become *the* representation language for planning problems involving numerical conditions and effects, especially since it is going to be used for the third international planning competition at AIPS 2002.

From a practician's point of view, this means that it is time some new planning algorithms capable of handling numerical features were developed, and indeed a number of recent publications in this area can be found (Haslum & Geffner 2001; Do & Kambhampati 2001).

From a theoretician's point of view, this is a good opportunity to look into decidability and complexity issues that arise in the context of numerical state variables. There is no work that we are aware of in the planning literature that addresses these issues, so in this paper we make a first attempt at closing this gap.

In the following analysis, we restrict ourselves to what Fox and Long have called "level 2" of PDDL2.1, not considering the PDDL2.1 formulation of concurrency or plan metrics. There are five main reasons behind that decision.

First, only investigating one new language feature at a time makes it easier to judge the impact of the addition of that feature. If we presented results for "full" PDDL2.1 instead, it might not be immediately obvious which new features actually contribute to the increased hardness of the task.

Second, related to this, restricting ourselves to one feature makes the results of our analysis more accessible, easier to understand, and allows for providing a more in-depth picture than we would have been able to give if we had chosen a panorama view of PDDL2.1.

Third, we think that the addition of numerical state variables is the feature of PDDL2.1 that will have the highest number of supporters, because it is immediately plausible and there are few competing models. In contrast, Fox and Long themselves have presented an interesting alternative model for concurrent planning domains with their PDDL+ language, which favors modeling actions of the planning agent as instantaneous and modeling change over time through environmental processes. It is not obvious if and how decidability results for PDDL2.1 with concurrency carry over to PDDL+, and vice versa.

Fourth, some of the other additions, especially concerning plan metrics, are only of importance when we are concerned with *optimal* or *near-optimal* planning. Our analysis is concerned with the more fundamental problem of finding *any* plan, because we feel that at this time, we cannot reasonably expect to optimally solve bigger problems with numerical features. While there are first results for optimally solving

planning problems of this kind (Haslum & Geffner 2001), just finding some plan for these domains seems a goal that is ambitious enough at the moment.

Last, we will see in the following sections that the addition of numerical features to the representation language is already sufficient to witness a sharp increase in the hardness of the planning task. Thus, we feel no immediate need to investigate even harder planning tasks.

The rest of this paper is structured as follows: In the following two sections, we introduce a formal notion of planning with numerical state variables and a convenient way of specifying restrictions to the domain definition language. After that, we prove some undecidability and decidability results for the general case and restrictions thereof. Finally, we discuss the implications of our analysis, have a look at related work, and conclude.

Planning with Numbers

To get started, we must introduce a notion of planning with numerical state variables. Our planning formalisms are based on propositional STRIPS, enhanced with numerical preconditions, numerical goal conditions, and numerical effects. We will look into a variety of different planning formalisms, each sharing the same basic propositional planning properties, but differing in what kind of numerical features they exhibit. For that reason, we will identify planning formalisms with their numerical features.

Definition 1 ***Planning formalism***
*A **planning formalism** is a triple $\mathcal{F} = (\mathcal{G}, \mathcal{P}, \mathcal{E})$, where $\mathcal{G}, \mathcal{P}, \mathcal{E} \subseteq \bigcup_{n \geq 1}(\mathbb{Q}^n \to \mathbb{Q})$. The three components of $\mathcal{F}$ are called its **numerical goal condition functions**, **numerical (operator) precondition functions**, and **numerical effect functions**, respectively.*

The sets of functions that define a planning formalism characterize the calculations that are allowed to be part of a condition or effect evaluation. To see how this works, we first need some auxiliary definitions.

Definition 2 ***Some terminology***
*For the rest of this definition, let V_P and V_N be disjoint finite sets called **propositional state variables** and **numerical state variables**, respectively.*

*The set of **states** (of (V_P, V_N)) is defined as $S = \{\ (\alpha, \beta) \mid \alpha : V_P \to \{\bot, \top\}, \beta : V_N \to \mathbb{Q}\ \}$.*

*A **propositional condition** is given by a propositional variable $v \in V_P$. It is written as $v = \top$. It is **satisfied** by state (α, β) iff $\alpha(v) = \top$.*

*For a set of rational functions (in one or more variables) $\mathcal{C}$, a **numerical condition** over $\mathcal{C}$ is given by an n-ary function $f \in \mathcal{C}$, n numerical state variables $v_1, \ldots, v_n \in V_N$ and a relational operator $\mathbf{relop} \in \{=, \neq, <, \leq, \geq, >\}$. It is written as $f(v_1, \ldots, v_n)\ \mathbf{relop}\ 0$, and it is satisfied by state (α, β) iff $f(\beta(v_1), \ldots, \beta(v_n))\ \mathbf{relop}\ 0$.*

*A **propositional effect** is given by a propositional variable $v \in V_P$ and a truth value $t \in \{\top, \bot\}$. It is written as $v \leftarrow t$ and v is called its **affected variable**.*

*For a set of rational functions (in one or more variables) $\mathcal{E}$, a **numerical effect** over $\mathcal{E}$ is given by an n-ary function $f \in \mathcal{E}$ and n numerical state variables $v_1, \ldots, v_n \in V_N$. It is written as $v_1 \leftarrow f(v_1, \ldots, v_n)$ and v_1 is its affected variable.*

*An **operator** over sets of rational functions $\mathcal{C}$ and $\mathcal{E}$ is a pair (Pre, Eff), where Pre is a finite set of propositional conditions and numerical conditions over $\mathcal{C}$ and Eff is a finite set of propositional effects and numerical effects over $\mathcal{E}$ such that different effects have different affected variables.*

This is all we need to define planning tasks:

Definition 3 ***Planning task***
*A **planning task** over a planning formalism $\mathcal{F} = (\mathcal{G}, \mathcal{P}, \mathcal{E})$ is a 5-tuple $T = (V_P, V_N, Init, Goal, Ops)$, such that $Init$ is a state, $Goal$ is a finite set of conditions over $\mathcal{G}$, and Ops is a finite set of operators over $\mathcal{P}$ and $\mathcal{E}$, with V_P and V_N being the underlying state variables (propositional and numerical, respectively).*

*$Init$ is called the **initial state**, $Goal$ the set of **goal conditions**, and Ops the **operator set** of T.*

The next definition captures the semantics of applying an operator to a state.

Definition 4 ***State transition graph***
*Let $T = (V_P, V_N, Init, Goal, Ops)$ be a planning task over a planning formalism $(\mathcal{G}, \mathcal{P}, \mathcal{E})$, and let S denote the set of states of T. The **state transition graph** of T is defined as the digraph with vertex set S containing the arc $((\alpha, \beta), (\alpha', \beta'))$ iff there is an operator $(Pre, Eff) \in Ops$ such that:*

(α, β) satisfies all conditions in Pre.

For all propositional effects $v \leftarrow t \in Eff$: $\alpha'(v) = t$.

For all $v \in V_P$ that are not affected by any effect in Eff: $\alpha'(v) = \alpha(v)$.

For all numerical effects $v_1 \leftarrow f(v_1, \ldots, v_n) \in Eff$: $\beta'(v_1) = f(\beta(v_1), \ldots, \beta(v_n))$.

For all $v \in V_N$ that are not affected by any effect in Eff: $\beta'(v) = \beta(v)$.

We complete our formal definitions by specifying the family of decision problems we want to analyze.

Definition 5 PLANEX-$\mathcal{F}$
For a planning formalism $\mathcal{F} = (\mathcal{G}, \mathcal{P}, \mathcal{E})$, the decision problem PLANEX-$\mathcal{F}$ *is defined as follows:*
***Given:** A planning task T over $\mathcal{F}$.*
***Question:** In the state transition graph of T, is there a directed path from the initial state to some state (α, β) satisfying all goal conditions?*

To provide some examples, PLANEX-$(\emptyset, \emptyset, \emptyset)$ is plan existence for propositional STRIPS (while there *are* numerical state variables according to our definition, they are not referred to at all and hence can be ignored), and PLANEX-$(\emptyset, \{id\}, \{inc, dec\})$, where $id : \mathbb{Q} \to \mathbb{Q}$ is defined by $x \mapsto x$, $inc : \mathbb{Q} \to \mathbb{Q}$ is defined by $x \mapsto x + 1$, and $dec : \mathbb{Q} \to \mathbb{Q}$ is defined by $x \mapsto x - 1$, is the plan existence problem for a planning formalism where no numerical conditions can be evaluated as part of the goal test, preconditions can test numerical state variables against zero, and effects can increase or decrease numerical state variables by one.

A Hierarchy of Planning Formalisms

Having defined the framework, we now have to answer the question which planning formalisms we are interested in analyzing. It is evident that we have to pose some restrictions on the sets of functions that define a planning formalism. If, for example, we included functions that are not computable, such as the *busy beaver* function (Boolos & Jeffrey 1989), we could easily prove undecidability results without getting any real insight into the problem of planning with numerical state variables.

It seems natural to restrict ourselves to functions that can be expressed by only using the usual arithmetic operators $\{+, -, \cdot, /\}$, which is in fact the approach of PDDL2.1. For our purposes, the problem of that approach is that many functions that make use of the division operator are not total, because division by zero is not possible.

Partial functions are forbidden in our definition of a planning formalism for a reason: Consider an operator which updates the value of state variable v by setting $\beta'(v) = \beta(v)^2/\beta(v)$. While this looks like a no-op, it is in fact a *hidden precondition* for the operator, because this calculation can only be performed if $\beta(v) \neq 0$. We do not want operator effects to play the part of operator preconditions because this is not what they are meant for. Thus, we further restrict the class of functions we want to analyze by only allowing division *by constants* other than zero, rather than division by values of state variables. This means that the most general class of functions we will investigate for operator effects is $\mathbb{Q}[x_1, x_2, x_3, \ldots]$, the class of multi-variable polynomials over the rational numbers.

Although there is no immediate need to be equally restrictive with regard to the sets of functions allowed for goal conditions and operator preconditions, we will do so for a simple reason: A condition that uses division arbitrarily can easily be transformed into a set of polynomial preconditions by multiplying with the squares of all denominators[1] and adding preconditions stating that the denominators must be different from zero. For example, the precondition $2/(x-1)+1 > 0$ can be replaced by the two polynomial preconditions $2(x-1)+(x-1)^2 > 0$ and $x-1 \neq 0$.

We will see that even with these restrictions, the PLANEX problem is undecidable, and thus we will investigate more specialized planning formalisms to identify the boundary of decidability. Specifically, we will use the following classes of functions for numerical goal conditions and numerical preconditions[2]:

- $\mathcal{C}_\emptyset = \emptyset$: No numerical conditions.
- $\mathcal{C}_0 = \{x \mapsto x\}$: Compare with zero. Numerical conditions are of the type v **relop** 0.
- $\mathcal{C}_c = \{\ x \mapsto x - c \mid c \in \mathbb{Q}\ \}$: Compare with a constant. Numerical conditions are of the type $v - c$ **relop** 0, for which we will use the alternative notation v **relop** c.
- $\mathcal{C}_= = \{(x_1, x_2) \mapsto x_1 - x_2\}$: Compare with another numerical state variable. Numerical conditions are of the type $v_1 - v_2$ **relop** 0, for which we will use the alternative notation v_1 **relop** v_2.
- $\mathcal{C}_p = \mathbb{Q}[x]$: Compare a polynomial of a state variable with zero. Numerical conditions are of the type $p(v)$ **relop** 0, where p is a polynomial.
- $\mathcal{C}_{p+} = \mathbb{Q}[x_1, x_2, x_3, \ldots]$: Compare a polynomial of many state variables to zero. Numerical conditions are of the type $p(v_1, \ldots, v_n)$ **relop** 0, where p is a polynomial.

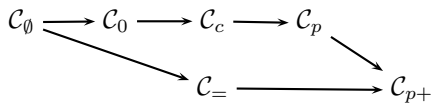

Figure 1: Containment for the function classes used in conditions. Arrows indicate subset relations.

For numerical effects, we will investigate the following classes of functions:

- $\mathcal{E}_\emptyset = \emptyset$: No numerical effects.
- $\mathcal{E}^{=c} = \{\ x \mapsto c \mid c \in \mathbb{Q}\ \}$: Assign a constant. Numerical effects are of the type $v \leftarrow c$.
- $\mathcal{E}_{+1} = \{x \mapsto x + 1\}$: Increase by one. Numerical effects are of the type $v \leftarrow v + 1$.
- $\mathcal{E}^{=c}_{+1} = \mathcal{E}^{=c} \cup \mathcal{E}_{+1}$. Numerical effects are either of the type $v \leftarrow c$ or of the type $v \leftarrow v + 1$.
- $\mathcal{E}_{\pm 1} = \{\ x \mapsto x + c \mid c \in \{-1, +1\}\ \}$: Increase or decrease by one. Numerical effects are either of the type $v \leftarrow v + 1$ or of the type $v \leftarrow v - 1$.
- $\mathcal{E}^{=c}_{\pm 1} = \mathcal{E}^{=c} \cup \mathcal{E}_{\pm 1}$. Numerical effects are either of the type $v \leftarrow c$, or of the type $v \leftarrow v+1$, or of the type $v \leftarrow v-1$.
- $\mathcal{E}_{+c} = \{\ x \mapsto x + c \mid c \in \mathbb{Q}_+\ \}$: Add a positive constant. Numerical effects are of the type $v \leftarrow v + c$ for $c \in \mathbb{Q}_+$.
- $\mathcal{E}^{=c}_{+c} = \mathcal{E}^{=c} \cup \mathcal{E}_{+c}$. Numerical effects are either of the type $v \leftarrow c$ for $c \in \mathbb{Q}$ or of the type $v \leftarrow v + c$ for $c \in \mathbb{Q}_+$.
- $\mathcal{E}_{\pm c} = \{\ x \mapsto x + c \mid c \in \mathbb{Q}\ \}$: Add an arbitrary constant. Numerical effects are of the type $v \leftarrow v + c$.
- $\mathcal{E}^{=c}_{\pm c} = \mathcal{E}^{=c} \cup \mathcal{E}_{\pm c}$. Numerical effects are either of the type $v \leftarrow c$ or of the type $v \leftarrow v + c$.
- $\mathcal{E}_p = \mathbb{Q}[x]$: Assign a polynomial of the old value. Numerical effects are of the type $v \leftarrow p(v)$, where p is a polynomial.
- $\mathcal{E}_{p+} = \mathbb{Q}[x_1, x_2, x_3, \ldots]$: Assign a polynomial of the old values of multiple state variables. Numerical effects are of the type $v_1 \leftarrow p(v_1, \ldots, v_k)$, where p is a polynomial.

With six options each for goal conditions and operator preconditions and twelve options for operator effects, there is a total of 432 different planning formalisms that we can define using combinations of the above classes of functions (not all of them interesting, of course). The following result shows that they need not be considered in isolation:

[1] We use squares in order to be sure not to multiply with a negative number, which is important if the relational operator is from the set $\{<, \leq, \geq, >\}$.

[2] We write $x \mapsto f(x)$ for the function in $\mathbb{Q} \to \mathbb{Q}$ which maps x to $f(x)$, and $(x_1, \ldots, x_n) \mapsto f(x_1, \ldots, x_n)$ for the function in $\mathbb{Q}^n \to \mathbb{Q}$ which maps $(x_1, \ldots, x_n)$ to $f(x_1, \ldots, x_n)$.

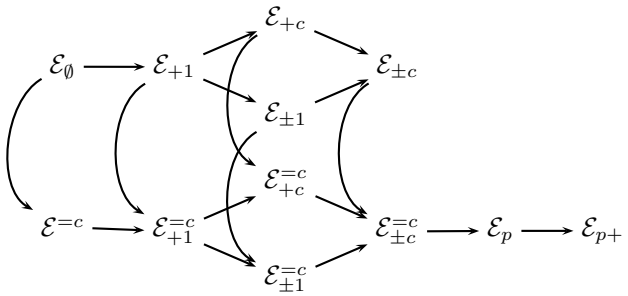

Figure 2: Containment for the function classes used in effects. Arrows indicate subset relations.

Proposition 6 ***Generalization/specialization***
For planning formalisms $\mathcal{F} = (\mathcal{G}, \mathcal{P}, \mathcal{E})$, $\mathcal{F}' = (\mathcal{G}', \mathcal{P}', \mathcal{E}')$ satisfying $\mathcal{G} \subseteq \mathcal{G}'$, $\mathcal{P} \subseteq \mathcal{P}'$, and $\mathcal{E} \subseteq \mathcal{E}'$, the following holds: PLANEX-$\mathcal{F} \leq$ PLANEX-$\mathcal{F}'$.

This is immediately obvious because if these containment conditions hold, the one decision problem is just a special case of the other one, and the identity function can be used as a membership-preserving mapping. This proposition makes it worthwhile to look at the containment graphs for the function classes we have defined. They are illustrated in Figure 1 for the conditions and Figure 2 for the effects.

Undecidability Results

We have now finished our preparations and can start proving results. In this section, we prove undecidability for five of the planning formalisms introduced. Together with Proposition 6, this allows us to conclude that for 258 of the 432 different variants of planning with numbers, complete algorithms for plan generation cannot be found. The remaining 174 variants will then be analyzed in the following section.

Our first result shows that general goals, with individual goal conditions referring to polynomials of multiple state variables, lead to undecidability even if no numerical operator preconditions and only a very weak type of numerical effects are present. The proof is based on the decision problem DIOPHANT$_{\mathbb{N}_0}$, which asks if a given Diophantine equation (polynomial in multiple variables) has a solution in the natural numbers. This problem is known to be undecidable (Cutland 1980).[3]

Theorem 7 PLANEX-$(\mathcal{C}_{p+}, \mathcal{C}_\emptyset, \mathcal{E}_{+1})$ ***is undecidable.***
Proof: Let $p \in \mathbb{Q}[x_1, \ldots, x_n]$ be a polynomial. The equation $p(x_1, \ldots, x_n) = 0$ has a solution in the natural numbers iff the planning task T described below is solvable:
$T = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$, where
$V_P = \emptyset$,
$V_N = \{v_1, \ldots, v_n\}$,
$\mathit{Init} = (\emptyset, \{(v_1, 0), \ldots, (v_n, 0)\})$,
$\mathit{Goal} = \{p(v_1, \ldots, v_n) = 0\}$, and
$\mathit{Ops} = \{\ (\emptyset, \{v_i \leftarrow v_i + 1\}) \mid i \in \{1, \ldots, n\}\ \}$.

This is not hard to see: If $(m_1, \ldots, m_n)$ is a solution to p, then applying the operator affecting v_1 m_1 times, applying the operator affecting v_2 m_2 times, and so on, results in a plan satisfying the goal. On the other hand, the values of the state variables at the end of the execution of any plan that solves the task form a solution to the Diophantine equation.

It is evident that this transformation is computable and that the resulting planning task obeys the constraints of the planning formalism. Thus, DIOPHANT$_{\mathbb{N}_0} \leq$ PLANEX-$(\mathcal{C}_{p+}, \mathcal{C}_\emptyset, \mathcal{E}_{+1})$, proving the theorem. ∎

Note that the planning tasks constructed in the proof even obey some further constraints: There are no operator preconditions whatsoever, there is only one goal condition, and there is only one effect per operator. This emphasizes the fact that goal conditions referring to multiple numerical state variables are indeed a great challenge.

Corresponding results could be shown for multi-variable polynomials in the preconditions rather than the goal, or for multi-variable polynomials in the numerical effects and tests against 0 as the only valid goal conditions, with only a few adjustments to the proof. However, we do not provide these proofs since the following theorems already entail those undecidability results.

While the previous result is not too limiting – polynomials in multiple variables are quite powerful mathematical objects and thus it does not seem unreasonable to disallow them –, the following result uncovers more severe restrictions. But first we have to introduce the decision problem that the reduction is based on.

Definition 8 01-MPCP
Given: *A modified PCP (MPCP) over the alphabet {0, 1}, i. e. non-empty words $a_1, \ldots, a_n, b_1, \ldots, b_n$ over {0, 1}.*
Question: *Is there a solution for the MPCP, i. e. a sequence of indices $i_1, \ldots, i_m$ such that the concatenation of the words $a_{i_1}, \ldots, a_{i_m}$ equals the concatenation of the words $b_{i_1}, \ldots, b_{i_m}$, and such that $i_1 = 1$?*

This problem is undecidable, even if n is fixed to be 7 (Matiyasevich & Senizerguez 1996), a special case we denote as 01-MPCP[7].

Theorem 9 PLANEX-$(\mathcal{C}_=, \mathcal{C}_\emptyset, \mathcal{E}_p)$ ***is undecidable.***
Proof: Let $((a_1, b_1), \ldots, (a_7, b_7))$ be a binary MPCP. We assume that a_1 and b_1 start with the same digit (otherwise the problem is trivially unsolvable and can be mapped to a trivially unsolvable planning task), and that this digit is 1 (otherwise 0 and 1 are flipped in all words). We write $\#w$ to denote the number that has the word $w \in \{0, 1\}^*$ as its binary notation, e. g. $\#110$ equals 6. The MPCP can be solved iff the planning task T described below has a solution:
$T = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$, where
$V_P = \emptyset$,
$V_N = \{a, b\}$,
$\mathit{Init} = (\emptyset, \{(a, \#a_1), (b, \#b_1)\})$,
$\mathit{Goal} = \{a = b\}$, and

[3] The reference only gives a proof of undecidability for solutions in the *integers* rather than in the natural numbers, but it is easy to see that $p(x_1, \ldots, x_n) = 0$ has an integer solution if and only if one of the polynomials that can be obtained from p by substituting some of the variables x_i by $-x_i$ has a solution in the natural numbers, providing a Turing reduction.

$Ops = \{\ op_i \mid i \in \{1, \ldots, 7\}\ \}$, where
$op_i = (\emptyset, \{a \leftarrow 2^{|a_i|}a + \#a_i, b \leftarrow 2^{|b_i|}b + \#b_i\})$.

Again, it is easy to see that this transformation is a computable function and that it maps to a planning instance satisfying the requirements for the chosen planning formalism (note that $2^{|a_i|}$ and $2^{|b_i|}$ are *constants*, as a_i and b_i are *not* state variables).

To see that the mapping is solution-preserving, observe that after applying an operator sequence $op_{i_2}, \ldots, op_{i_m}$ for $i_j \in \{1, \ldots, 7\}$ to the initial state, the state variables a and b, written down as binary numbers, equal the concatenation of $a_{i_1}, \ldots, a_{i_m}$ and $b_{i_1}, \ldots, b_{i_m}$, respectively, for $i_1 = 1$.

Thus, if $i_1, \ldots, i_m$ is a solution to the MPCP, then applying $op_{i_2}, \ldots, op_{i_m}$, in sequence, to the initial state solves the planning task, and vice versa.

This shows 01-MPCP[7] $\leq$ PLANEX-$(\mathcal{C}_=, \mathcal{C}_\emptyset, \mathcal{E}_p)$, proving the theorem. ∎

This result is complemented by the following:

Theorem 10 PLANEX-$(\mathcal{C}_0, \mathcal{C}_\emptyset, \mathcal{E}_p)$ ***is undecidable.***
Proof: In this case, we only need to slightly adjust the mapping from the previous proof to accommodate for the different type of goal. We only describe the changes to the previous mapping. First, we introduce two propositional variables, $Work$, with an initial value of $\top$, and $Finish$, with an initial value of $\bot$. The goal is adjusted to require a and b to both have values of 0, rather than just be equal.

We make $Work = \top$ a precondition of each of the operators op_i, $i \in \{1, \ldots, 7\}$, and introduce two new operators, $op_S = (\{Work = \top\}, \{Work \leftarrow \bot, Finish \leftarrow \top\})$ and $op_D = (\{Finish = \top\}, \{a \leftarrow a - 1, b \leftarrow b - 1\})$.

It is not hard to see that this is again a solution-preserving reduction. If the MPCP is solvable, then applying a sequence of operators over $\{op_1, \ldots, op_7\}$ leads to equal values for a and b. Then, op_S can be applied once and op_D a number of times until both values are 0, meeting the goal. On the other hand, it is easy to see that valid plans can be transformed into MPCP solutions in the same way as in the previous proof if the trailing instances of op_S and op_D are removed.

Thus, 01-MPCP[7] $\leq$ PLANEX-$(\mathcal{C}_0, \mathcal{C}_\emptyset, \mathcal{E}_p)$, proving the theorem. ∎

Similar results hold for preconditions (rather than goal conditions) from $\mathcal{C}_=$ and $\mathcal{C}_0$, but again, those are entailed by results to come. Note from the two proofs that the results still hold if only polynomials of degree 1 are allowed in numerical effects. This basically shows that from the effect function sets in our hierarchy, everything that is beyond assignment, addition and subtraction of constants is too hard, even for the most simple types of pre- and goal conditions. In fact, the proof of Theorem 9 even shows undecidability for the very restricted case of no preconditions, only seven operators, and only two state variables, if polynomials of degree 1 are allowed for effects.

For the last two results in this section, we first have to define abacus programs.

Definition 11 ***Abacus program***
*An **abacus program** is a 5-tuple (V, L, l_0, l_H, P), where V and L are disjoint finite sets called **variables** and **labels**, respectively, $l_0 \in L$ is called the **initial label**, $l_H \in L$ is called the **halt label**, and $P : L \setminus \{l_H\} \rightarrow C_{V,L}$ is called the **program**. $C_{V,L}$ is defined as follows:*
$C_{V,L} = \{\ \textbf{INC}\ v; \rightarrow l \mid v \in V, l \in L\ \}$
$\quad \cup\ \{\ \textbf{DEC}\ v; \rightarrow l_>, l_= \mid v \in V, l_>, l_= \in L\ \}$

We cannot afford the space to formally introduce the semantics of abacus programs, so we hope that the following informal account will suffice.

The variables of the program correspond to registers of a virtual machine which can hold arbitrary natural numbers. They are initialized to zero. Initially, the current label is l_0. If at any time the current label is l_H, execution terminates. Otherwise, if the current label is l, and $P(l) = \textbf{INC}\ v; \rightarrow l'$, the value of variable v is increased by one and l' becomes the new current label. If $P(l) = \textbf{DEC}\ v; \rightarrow l_>, l_=$, the value of variable v is inspected. If it is greater than zero, it is decreased by one and $l_>$ becomes the current label. Otherwise, it is not changed and $l_=$ becomes the current label. The process is repeated with the new current label.

Abacus machines are as powerful as Turing Machines (Boolos & Jeffrey 1989), which implies that it is algorithmically impossible to decide whether execution of a given abacus program stops or not, i. e. the halting problem for abacus programs, HALT-ABACUS, is undecidable, following from a well-known theorem about Turing Machines (Boolos & Jeffrey 1989). This leads to the following result:

Theorem 12 PLANEX-$(\mathcal{C}_\emptyset, \mathcal{C}_0, \mathcal{E}_{\pm 1})$ ***is undecidable.***
Proof: Let (V, L, l_0, l_H, P) be an abacus program. Then the planning task T is defined as follows:
$T = (V_P, V_N, Init, Goal, Ops)$, $Init = (\alpha_I, \beta_I)$, where
$V_P = L$,
$V_N = V$,
$\alpha_I(l_0) = \top$; $\alpha_I(l) = \bot$ for all other $l \in L$,
$\beta_I(v) = 0$ for all $v \in V$,
$Goal = \{l_H = \top\}$, and
Ops contains the following operators:
For l, l', v such that $P(l) = \textbf{INC}\ v; \rightarrow l'$:
$(\{l = \top\}, \{l \leftarrow \bot, l' \leftarrow \top, v \leftarrow v + 1\})$.[4]
For l, $l_>$, $l_=$, v such that $P(l) = \textbf{DEC}\ v; \rightarrow l_>, l_=$:
$(\{l = \top, v > 0\}, \{l \leftarrow \bot, l_> \leftarrow \top, v \leftarrow v - 1\})$ and
$(\{l = \top, v = 0\}, \{l \leftarrow \bot, l_= \leftarrow \top\})$.

To understand the mapping, note that states of the planning task correspond to execution states of the abacus program, where propositional variable l is true iff l is the current label. It is easy to verify that plans for this task correspond to execution traces of the abacus program, and that the goal will be met iff the halt label ever becomes the current label, i. e. iff the abacus program halts.

Thus, HALT-ABACUS$\leq$ PLANEX-$(\mathcal{C}_\emptyset, \mathcal{C}_0, \mathcal{E}_{\pm 1})$, proving the theorem. ∎

We consider it interesting to point out that the values of the numerical state variables in a goal state correspond to

[4] If $l = l'$, the propositional effects are omitted. Similar comments apply to the other operators if $l = l_>$ or $l = l_=$, respectively.

the values of the abacus program variables upon termination. This close correspondence means that we could employ a planning formalism as a programming language, computing functions of the natural numbers. The previous proof shows that, viewed as a programming language, the formalism is Turing-complete (like abacus programs).

The last result in this section is a minor variation of the previous theorem.

Theorem 13 PLANEX-$(\mathcal{C}_\emptyset, \mathcal{C}_=, \mathcal{E}_{+1})$ ***is undecidable.***
Proof: We use the same reduction as in the proof of the preceding theorem, with two changes. First, for each numerical state variable $v \in V$, we introduce another numerical state variable v^-, also initialized to 0. Second, in the operators corresponding to decrement statements in the abacus program, the preconditions that compare v against zero are changed to compare v against v^-, and the effect that decreases v is changed to an effect that increases v^-.

Again, we can verify that plans correspond to abacus program execution traces. To understand this, observe that the value of the abacus program variable v in state (α, β) can be calculated as $\beta(v) - \beta(v^-)$. Thus, following the same reasoning as above, HALT-ABACUS$\leq$ PLANEX-$(\mathcal{C}_\emptyset, \mathcal{C}_=, \mathcal{E}_{+1})$, proving the theorem. ∎

This completes our list of undecidability results. The following section will provide some good news.

Decidability Results

We begin the discussion of decidable planning formalisms in our numerical planning framework by mentioning two trivial results.

Theorem 14 PLANEX-$(\mathcal{C}_\emptyset, \mathcal{C}_\emptyset, \mathcal{E}_{p+})$ ***is decidable.***
Proof: Since the planning formalism does not feature numerical goal conditions or operator preconditions, all numerical state variables and effects can simply be ignored. The problem therefore reduces to plan existence for propositional STRIPS, which is known to be decidable, in fact **PSPACE**-complete (Bylander 1994). ∎

The preceding result is not of much relevance and was only included for sake of completeness. The next result is a bit more interesting, but still obvious.

Theorem 15 PLANEX-$(\mathcal{C}_{p+}, \mathcal{C}_{p+}, \mathcal{E}^{=c})$ ***is decidable.***
Proof: In this planning formalism, only constants are assigned to numerical state variables. Since there is only a finite set C of such constants in each planning task, no numerical state variable can take on more than $|C|+1$ different values, namely its initial value and the values from C. The set of states reachable from the initial state is thus finite and the reachable part of the state transition graph can be explicitly constructed and the problem then solved using standard graph search techniques. ∎

Having considered the trivial cases, we will now look at more interesting planning formalisms and provide some simplification results. To do so, we must first introduce a new term.

Definition 16 ***Scalable function sets***
A set of rational functions $\mathcal{F}$ is called ***scalable*** *if for each $q \in \mathbb{Q}_+$ and for each n-ary function $f \in \mathcal{F}$ there exists some function $f_{[q]} \in \mathcal{F}$ such that for all $x_1, \ldots, x_n \in \mathbb{Q}$, $\mathrm{sgn}(f(x_1, \ldots, x_n)) = \mathrm{sgn}(f_{[q]}(qx_1, \ldots, qx_n))$.*[5]

Some examples of scalable function sets:
$\mathcal{C}_p$: $(x \mapsto p(x))_{[q]} = (x \mapsto p(\frac{x}{q}))$
$\mathcal{C}_c$: $(x \mapsto x - c)_{[q]} = (x \mapsto x - qc)$
$\mathcal{C}_=$: $((x_1, x_2) \mapsto x_1 - x_2)_{[q]} = ((x_1, x_2) \mapsto x_1 - x_2)$

Our first simplification result shows that in certain cases we can restrict the domain of numerical state variables to the set of integers.

Algorithm 17 ***Domain simplification***
Let T be a planning task over $(\mathcal{G}, \mathcal{P}, \mathcal{E})$ such that $\mathcal{G}$ and $\mathcal{P}$ are scalable and $\mathcal{E} \in \{\mathcal{E}^{=c}, \mathcal{E}_{+c}, \mathcal{E}^{=c}_{+c}, \mathcal{E}_{\pm c}, \mathcal{E}^{=c}_{\pm c}\}$. Then the following "domain simplification" algorithm translates T into an equivalent task over $(\mathcal{G}, \mathcal{P}, \mathcal{E})$ such that all numerical effects are of the type $v \leftarrow v + c$ or $v \leftarrow c$ for $c \in \mathbb{Z}$ and all initial values of numerical state variables are integers. Thus, for each reachable state (α, β), the domain of β is a subset of $\mathbb{Z}$.

Let $T = (V_P, V_N, Init, Goal, Ops)$, $Init = (\alpha_I, \beta_I)$.

Let $M = \{\ \beta_I(v) \mid v \in V_N\ \} \cup$
$\{\ c \mid \exists (Pre, Eff) \in Ops\ \exists v \in V_N : v \leftarrow c \in Eff\ \} \cup$
$\{\ c \mid \exists (Pre, Eff) \in Ops\ \exists v \in V_N : v \leftarrow v + c \in Eff\ \}$.

In words, M contains all the constants that appear in the initial state and operator effects.

Let $d \in \mathbb{N}_+$ be a common denominator of all $q \in M$.

Replace, for all $v \in V_N$, initial values $\beta_I(v)$ by $d\beta_I(v)$, effects of type $v \leftarrow c$ by $v \leftarrow dc$, effects of type $v \leftarrow v + c$ by $v \leftarrow v + dc$, and conditions $f(v_1, \ldots, v_n)$ $\mathbf{reqop}$ 0 by $f_{[d]}(v_1, \ldots, v_n)$ $\mathbf{reqop}$ 0.

This satisfies the requirements, which can be shown by verifying that the function which maps (α, β) to $(\alpha, d\beta)$ is a state isomorphism from the original to the modified planning task, i.e. initial and goal states are mapped to initial and goal states, and $((\alpha, \beta), (\alpha', \beta'))$ is an arc in the original state transition graph iff $((\alpha, d\beta), (\alpha', d\beta'))$ is an arc in the modified state transition graph.

Furthermore, the requirements of integrality and staying within $(\mathcal{G}, \mathcal{P}, \mathcal{E})$ are met.

Our second simplification result, which is applicable under the same conditions as domain simplification, shows that in many cases numerical conditions using polynomials can be replaced by simple comparisons.

Theorem 18 ***Condition simplification***
Let $(\mathcal{G}, \mathcal{P}, \mathcal{E})$ be a planning formalism such that $\mathcal{G}$ and $\mathcal{P}$ are scalable and $\mathcal{E} \in \{\mathcal{E}^{=c}, \mathcal{E}_{+c}, \mathcal{E}^{=c}_{+c}, \mathcal{E}_{\pm c}, \mathcal{E}^{=c}_{\pm c}\}$. Then
PLANEX-$(\mathcal{G}, \mathcal{P}, \mathcal{E}) \leq_T$ PLANEX-$(\mathcal{G}', \mathcal{P}', \mathcal{E})$
for $\mathcal{G}' = (\mathcal{G} \setminus \mathcal{C}_p) \cup \mathcal{C}_c$ and $\mathcal{P}' = (\mathcal{P} \setminus \mathcal{C}_p) \cup \mathcal{C}_c$.
Proof: The following algorithm provides the required Turing reduction. First, apply domain simplification to the input

[5] sgn denotes the signum, or sign function. To be strict, we must also require that there is an algorithm that calculates a representation of $f_{[q]}$ from representations of f and q. However, as the most general class of functions we are concerned with in this paper are polynomials, we can safely assume that this is the case.

planning task. We can now assume all numerical state variables to only take on integral values, which allows to translate conditions of the type $p(v)$ **relop** 0 for polynomials p into disjunctions of simpler conditions of type v **relop**$_i$ c_i in the following way.[6]

Using standard numerical algorithms, calculate integers l and u such that for all x_0 solving $p(x_0) = 0$, $l < x_0 < u$ (we ignore the trivial case $p(x) = 0$ for all x). Because of the intermediate value theorem and the assumption $\beta(v) \in \mathbb{Z}$ for all states (α, β), $p(v)$ **relop** 0 is equivalent to

$(v \leq l \wedge p(l)$ **relop** $0)$
$\vee\ (v = l + 1 \wedge p(l + 1)$ **relop** $0)$
$\vee \ldots$
$\vee\ (v = u - 1 \wedge p(u - 1)$ **relop** $0)$
$\vee\ (v \geq u \wedge p(u)$ **relop** $0)$,

where the subexpressions $p(k)$ **relop** 0 can be evaluated statically to simplify the expression to a simple disjunction.

Once this has been done for all polynomial conditions, operators with disjunctive preconditions can be translated to sets of standard operators using well-known techniques (Nebel 1999), and the goal can be translated into a set of conjunctive goals analogously. We can then use the subroutine for PLANEX-$(\mathcal{G}', \mathcal{P}', \mathcal{E})$ to check if at least one of those conjunctive goals can be satisfied. If so, we succeed, otherwise we fail. ∎

Our last simplification result shows that in many cases numerical operator preconditions can be compiled away completely if numerical state variables can only be assigned constants or increased.

Theorem 19 ***Precondition elimination***
Let $\mathcal{G}$ be a scalable function set and $\mathcal{E} \in \{\mathcal{E}^{=c}, \mathcal{E}_{+c}, \mathcal{E}^{=c}_{+c}\}$. Then PLANEX-$(\mathcal{G}, \mathcal{C}_p, \mathcal{E}) \leq_T$ PLANEX-$(\mathcal{G}, \mathcal{C}_\emptyset, \mathcal{E})$.

Proof: Because the previous theorem implies the relationship PLANEX-$(\mathcal{G}, \mathcal{C}_p, \mathcal{E}) \leq_T$ PLANEX-$(\mathcal{G}, \mathcal{C}_c, \mathcal{E})$, we only need to eliminate operator preconditions that compare to constants. First, we apply domain simplification, obtaining a task $T = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ over $(\mathcal{G}, \mathcal{C}_c, \mathcal{E})$, with $\mathit{Init} = (\alpha_I, \beta_I)$. Assume $V_N \neq \emptyset$ (otherwise, this is trivial).

Let $l = \min(\{\ \beta_I(v) \mid v \in V_N\ \} \cup$
$\{\ c \mid \exists(\mathit{Pre}, \mathit{Eff}) \in \mathit{Ops}\ \exists v \in V_N : v \leftarrow c \in \mathit{Eff}\ \})$.

l is the least initial value of any numerical state variable, or the least value that can be assigned to a state variable, whichever is lower. Values of numerical state variables will never be below l.

Let $u = 1 + \max\{\ c \mid \exists(\mathit{Pre}, \mathit{Eff}) \in \mathit{Ops}\ \exists v \in V_N$
$\exists \mathbf{relop} \in \{=, \neq, <, \leq, \geq, >\} : v$ **relop** $c \in \mathit{Pre}\ \}$
(assuming this set is non-empty, otherwise there is nothing to do). u is one greater than the highest number that is compared to in any numerical precondition. For evaluating numerical preconditions, values beyond u need not be distinguished. Our key idea is to keep track of numerical state variables having values between l and u by using new *propositional* variables $eq_{v,k}$ for each $v \in V_N$, $k \in \{l, \ldots, u\}$[7], where $eq_{v,k}$ is to be read as "the value of v is k" with the exception of $k = u$, where it means "the value of v is at least u".

The initial value of these new propositional variables is defined by $\alpha_I(eq_{v,k}) = \top \leftrightarrow \beta_I(v) = k$. Their meaning is maintained throughout operator applications by adjusting operator definitions as follows:

For operators having an effect $v \leftarrow c$, add the effects $eq_{v,c} \leftarrow \top$ and $eq_{v,k} \leftarrow \bot$ for each $k \in \{l, \ldots, u\} \setminus \{c\}$.

For operators having an effect $v \leftarrow v + c$, for each $\mathit{old}, \mathit{new} \in \{l, \ldots, u\}$ add a *conditional* propositional effect **IF** $eq_{v,\mathit{old}} = \top$ **THEN** $eq_{v,\mathit{new}} \leftarrow t$, where $t = \top$ if the equation $\min(\mathit{old} + c, u) = \mathit{new}$ is true, and $t = \bot$ otherwise.

Once this has been done, each numerical precondition v **relop** c can be replaced by the following disjunctive propositional precondition:
$\bigvee_{\{k \in \{l,\ldots,u\}\ \mid\ k\ \mathbf{relop}\ c\ \}} eq_{v,k} = \top$.

Finally, disjunctive preconditions and conditional effects can be compiled away (Nebel 1999). ∎

Now that we have provided a number of simplification results, it is time we presented the main result of this section. In the following, let $T = (V_P, V_N, \mathit{Init}, \mathit{Goal}, \mathit{Ops})$ be a planning task over the formalism $\mathcal{F} = (\mathcal{C}_c \cup \mathcal{C}_=, \mathcal{C}_\emptyset, \mathcal{E}^{=c}_{\pm c})$. Note that there are no numerical preconditions in this formalism.

Definition 20 ***Propositional states***
For states (α, β), α is called a ***propositional state****.*

For propositional states α and α', an operator sequence is called a ***propositionally acyclic path from α to α'*** *if there is a corresponding path in the state transition graph from (α, β) to (α', β') (for some β, β') such that all nodes in the path have different propositional states.*

For propositional states α, a non-empty operator sequence is called a ***propositional cycle in*** *α if it corresponds to a path in the state transition graph from (α, β) to (α, β') such that all nodes in the path except the start and end node have different propositional states.*

The planning task T has $2^{|V_P|}$ propositional states, a finite number. Without numerical preconditions, knowing the propositional state is sufficient for deciding if an operator sequence is applicable in a given state, and the values of the numerical state variables are only important for checking goal conditions. Thus, we will also talk about an operator or sequence of operators being *applicable in a propositional state*.

Note that the sets of propositionally acyclic paths and propositional cycles can be computed by analyzing the (finite) transition graph of the planning task obtained by deleting all numerical state variables, effects and goal conditions from T. Only operator sequences of length at most $2^{|V_P|}$ must be taken into account.

[6] Note that we need domain simplification: If x can take on any rational value, than conditions like $x^2 - 2 \geq 0$ cannot be translated into a finite number of simple comparisons of x to rational numbers.

[7] For simplicity of notation, we assume $l < u$, which can be enforced by increasing u if needed.

Definition 21 ***Additive and assigning effects***
For operators $o = (Pre, Eff)$ and numerical state variables v, the ***increase*** *of v by o is defined as follows:*
$incr(o, v) = c$ *if* $v \leftarrow v + c \in Eff$.
$incr(o, v) = 0$ *if no effect in Eff affects v.*
$incr(o, v)$ *is undefined if* $v \leftarrow c \in Eff$ *for some* $c \in \mathbb{Q}$.
In the formalism $\mathcal{F}$, exactly one of those conditions must be true for each o and v. If the third holds, we say that o has an ***assigning effect****, otherwise an* ***additive effect*** *on v.*

For calculating the value of a numerical state variable v at the end of plan execution, we can ignore all plan steps that precede an operator with an assigning effect on v. We say that v becomes *active* in that plan after the last assignment to it, or with the first planning step, if it is never assigned a new value. Plan steps that cause some state variable to become active are called *activating steps*. The sequence of activating steps is called the *activation sequence*. It can comprise no more than $|V_N|$ operators, because each numerical state variable is activated at most once.

The key idea of our algorithm is to "guess" the activation sequence and flesh out the details of the plan by inserting subplans between steps of the activation sequence, and before the first and after the last activating step.

We will use a "guess" operation in the description of the algorithm a number of times. The algorithm makes a number of guesses that can be bounded by a computable function in the size of the instance, and in each case, the set of possible guesses is finite and can be generated systematically. Thus, the algorithm can be made deterministic by systematically exploring the space of possible guesses. If the task is solvable, there is a sequence of guesses that leads to success. If it is unsolvable, all possible guesses lead to failure.

Algorithm 22 ***Cycle counting, part 1***
First, guess a natural number $k \in \{0, \ldots, |V_N|\}$ and an activation sequence $aseq = (o_1, \ldots, o_k) \in Ops^k$. Verify that $aseq$ is a valid activation sequence by checking that each o_i has an assigning effect on some numerical state variable that is not assigned to by any operator o_j for $j > i$.

For each $v \in V_N$, the ***activation time*** *of v is defined as $atime(v) = \max\{\, i \in \{1, \ldots, k\} \mid o_i$ assigns to $v \,\}$, or 0 if this set is empty. The* ***activation value*** *$aval(v)$ of v is defined as the value that v is assigned by $o_{atime(v)}$, or the initial value of v, if $atime(v) = 0$.*

Next, guess a sequence $(\alpha_1, \ldots, \alpha_k)$ of propositional states such that each o_i is applicable in states with propositional part α_i. Calculate the propositional states α'_i that result from applying o_i in a state with propositional part α_i. We assume that each o_i is applied in a state with propositional part α_i. We define $\alpha'_0 = \alpha_I$, the propositional part of the initial state, and guess another propositional state α_{k+1} satisfying all propositional goal conditions, assuming that this is the propositional part of some reachable goal state.

What remains to be done is finding (possibly empty) operator sequences π_i for $i \in \{0, \ldots, k\}$ such that each π_i is applicable in the propositional state α'_i and results in propositional state α_{i+1} and such that $\pi_0 o_1 \pi_1 o_2 \ldots \pi_n$ also satisfies the numerical parts of the goal. We call such a sequence π_i the i-th *episode* of the plan.

Episodes cannot contain arbitrary operators: In episode i, operators that have assigning effects for any numerical state variable with activation time less than or equal to i are forbidden, because these state variables are active in that part of the plan.

If we make sure that these operators are ruled out, we can safely ignore assigning effects within an episode: All state variables that are assigned a value will be assigned a different value later, when they become active. So all relevant numerical effects are additive, which is important because addition is commutative and associative: We do not need to know the episode exactly, we only need the information which operators are executed and how often they are executed.

Each operator sequence linking propositional nodes α and α' can be partitioned, by iteratively removing propositional cycles, into a propositionally acyclic path and a number of propositional cycles. If for each episode we know its acyclic path and the number of times each propositional cycle in it is traversed, we can calculate the values of all numerical state variables in the goal.

The correct acyclic paths π_i^{ac} can be guessed, as there is only a finite number of candidates. We can also guess the sets C_i of propositional cycles that are traversed at least once in the episode, as there is only a finite number of possible choices here, too. As noted above, for both π_i^{ac} and C_i, we must restrict ourselves to sequences containing no operators with assigning effects that are illegal for the i-th episode.

Not every choice for C_i is feasible: We need to check for each chosen cycle that it touches a propositional node traversed by π_i^{ac}, or a propositional node on some other feasible cycle. This can be done with a fixpoint reachability test as described in the following part of the algorithm.

Algorithm 22 (continued) ***Cycle counting, part 2***
For all $i \in \{0, \ldots, k\}$, guess a propositionally acyclic path π_i^{ac} from α'_i to α_{i+1} and a set of propositional cycles C_i.

Verify that π_i^{ac} and the cycles in C_i contain no operators with assigning effects for numerical state variables with activation time i or less. Check that the choice for C_i is valid as follows:

First, label all cycles in C_i which pass through propositional nodes on π_i^{ac} as feasible. Then, label all cycles in C_i passing through propositional nodes on some feasible cycle as feasible. Iterate this step until a fixpoint is reached, and reject the choice of C_i if some cycle in C_i has not been labeled.

The only important piece of information which is still lacking is the *number of times* each cycle in C_i is executed in episode i. We cannot use a systematic enumeration technique here, because any positive integer would be a valid choice and thus, choices cannot be explored exhaustively. For this reason, we introduce a variable x_i^π for each cycle $\pi \in C_i$, representing the number of times this cycle is traversed in the i-th episode. The planning task is solvable if and only if there is a function which maps each such variable x_i^π to a positive integer such that the final values of the numerical state variables, which can be computed from the guessed information and the values of the x_i^π variables,

match the requirements of the goal.

Algorithm 22 (continued) ***Cycle counting, part 3***
Write down the following linear equations in the rational variables $goal_v$ for $v \in V_N$ and integer variables x_i^π for $i \in \{0, \ldots, k\}$ and $\pi \in C_i$:[8]

$$goal_v = aval(v) + \sum_{i \in \{atime(v)+1,\ldots,k\}} incr(o_i, v) + \sum_{i \in \{atime(v),\ldots,k\}} \sum_{o \in \pi_i^{ac}} incr(o, v) + \sum_{i \in \{atime(v),\ldots,k\}} \sum_{\pi \in C_i} \sum_{o \in \pi} incr(o, v) \cdot x_i^\pi$$

Additionally, write down the following formulas:
For each $i \in \{0, \ldots, k\}$, $\pi \in C_i$: $x_i^\pi \geq 1$.
*For goal conditions v_1 **relop** v_2:* $goal_{v_1}$ **relop** $goal_{v_2}$.
*For goal conditions v **relop** c:* $goal_v$ **relop** c.
Taking these together, we obtain a mixed integer program in the variables $goal_v$ and x_i^π that has a solution if and only if the planning task is solvable. A mixed integer program solver can be used to generate such a solution or prove that none exists (Bixby et al. *2000).*

Putting this algorithm and the preceding results together, we obtain the following corollary.

Corollary 23 ***Some decidability results***
PLANEX *is decidable for these planning formalisms:*
$(\mathcal{C}_p, \mathcal{C}_\emptyset, \mathcal{E}_{\pm c}^{=c})$ *(Theorem 18 and Algorithm 22)*
$(\mathcal{C}_=, \mathcal{C}_\emptyset, \mathcal{E}_{\pm c}^{=c})$ *(Algorithm 22)*
$(\mathcal{C}_p, \mathcal{C}_p, \mathcal{E}_{+c}^{=c})$ *(Theorem 19, Theorem 18 and Algorithm 22)*
$(\mathcal{C}_=, \mathcal{C}_p, \mathcal{E}_{+c}^{=c})$ *(Theorem 19 and Algorithm 22)*

Combining these four results with the two trivial results stated at the beginning of the section and the generalization/specialization relationships in our framework, we see that plan existence is decidable for all planning formalisms that were not covered in the previous section. This completes our analysis.

Discussion

Let us summarize the results of our analysis. We have defined and investigated a family of decision problems related to planning with numerical state variables. Many of these were undecidable, due to the fact that the state space is infinite. The results are repeated in Figure 3.

Type of effects	Decidability status
$\mathcal{E}_\emptyset, \mathcal{E}^{=c}$	Always decidable
$\mathcal{E}_{+1}, \mathcal{E}_{+c}, \mathcal{E}_{+1}^{=c}, \mathcal{E}_{+c}^{=c}$	Decidable iff $\mathcal{G} \neq \mathcal{C}_{p+}$, $\mathcal{P} \notin \{\mathcal{C}_=, \mathcal{C}_{p+}\}$
$\mathcal{E}_{\pm 1}, \mathcal{E}_{\pm c}, \mathcal{E}_{\pm 1}^{=c}, \mathcal{E}_{\pm c}^{=c}$	Decidable iff $\mathcal{G} \neq \mathcal{C}_{p+}$, $\mathcal{P} = \mathcal{C}_\emptyset$
$\mathcal{E}_p, \mathcal{E}_{p+}$	Decidable iff $\mathcal{G} = \mathcal{P} = \mathcal{C}_\emptyset$

Figure 3: Results for the variants of PLANEX-$(\mathcal{G}, \mathcal{P}, \mathcal{E})$ in the hierarchy.

[8] We use the notation $o \in \pi$ to traverse the operators in an operator sequence. This is *not* supposed to mean that π is viewed as a set: Operators that appear multiple times in π *must* be considered multiple times.

Some of the undecidability results, as mentioned before, even apply to very restricted special cases, such as a constant number of operators, no operator preconditions, no propositional state variables, and only two numerical state variables. Additionally, as can be verified from the proofs provided, all undecidability results still hold if the set of relational operators is limited to $\{=, \neq\}$ (rather than the full set $\{=, \neq, <, \leq, \geq, >\}$).

Our classification approach for planning formalisms is based on the idea of restricting the type of calculations that can be performed by the planner. There are other possible restrictions to facilitate planning. One of them is to assign lower and upper bounds to each state variable. Operators that try to set the value of a numerical state variable to some number which is beyond these bounds can either be disallowed, or increases and decreases can be "capped". If in addition to this, differences in the values of state variables cannot be arbitrarily small (e. g., if the values must be integers), this immediately leads to a finite state space and hence decidability.[9]

On a more practical note, what is the impact of undecidability? Even the most general decision problem in our framework has the property of *semi-decidability*, because it is possible to systematically enumerate all operator sequences, say in lexicographical order, and check for each of them whether it solves the task or not. Such an algorithm would succeed in generating a plan for all solvable tasks. For tasks that do not have a solution, however, it would not terminate, and the undecidability results we have proved show that there is no algorithm that can possibly recognize *all* unsolvable planning tasks.

However, even without numerical state variables, many planning systems do not make an effort to always detect unsatisfiable goals. This has practical reasons, one of them being that this kind of result is usually hard to find without completely exploring the state space. Systematic planning systems, like **Graphplan** (Blum & Furst 1997) or the symbolic breadth-first exploration mode of **Mips** (Edelkamp & Helmert 2001), can detect unsolvable tasks with simple termination criteria, but sacrifice efficiency in doing so, compared to local search approaches that are by design better suited for finding plans than for proving their non-existence.

We do not think that there is a general answer to the question how important it is to be able to reliably detect failure in planning, but it is evident that the decidability status of the underlying decision problem is of high relevance to that discussion: If undecidable formalisms are employed, not being able to reliably detect unsolvable tasks is a necessary property of *any* planning algorithm.

Related Work

We are not aware of any work in the AI planning literature that is directly concerned with the decidability issues that arise when numbers are introduced into a planning for-

[9] In our framework, bounds lead to decidability of PLANEX for all numerical effect function sets except $\mathcal{E}_p$ and $\mathcal{E}_{p+}$. For $\mathcal{E}_p$ and $\mathcal{E}_{p+}$, bounds do not make a difference. Unfortunately, we do not have the space to prove this here.

malism. Some interesting decidability and undecidability results for general cases of STRIPS-style planning can be found, e. g. in work by Erol et al. (1995), covering problems that arise when function symbols and/or an infinite number of constant symbols are introduced into the formalism (which would correspond to an infinite number of propositional variables in the terminology of this paper). Other related problems, such as reachability for hybrid automata, are investigated in the model checking literature.

Researchers such as Bylander (1994) or Bäckström and Nebel (1995) have studied hierarchies of planning formalisms similar to the ones in our paper with regard to complexity. In this context, work on compilation schemes for translating between different planning formalisms should also be mentioned (Nebel 1999).

As for algorithms for planning with numerical state variables, a multitude of approaches have been proposed. Only recently, Do and Kambhampati have presented a planning system using an action representation which they describe as "influenced by the PDDL+ language proposal", and which is at least as general as the most general planning formalism considered in this paper (2001).

Haslum and Geffner have recently presented an optimal planning system capable of dealing with some of the features we have investigated in this paper (2001). Apart from their model of time and concurrency, which has no counterpart in our formalism, their planning formalism most closely resembles $(\mathcal{C}_\emptyset, \mathcal{C}_c, \mathcal{E}_{\pm c})$, which is undecidable. However, they say that they have implemented the algorithm "with the restriction that consumable resources are monotonically decreasing", which relates to the decidable $(\mathcal{C}_\emptyset, \mathcal{C}_c, \mathcal{E}_{+c})$.

Outlook

As is usually the case in research, in the process of answering a question we also leave some open ends, issues that might or even should be addressed in the future. In this paper, we have only cared about decidability – can the decision problem be solved? We were able to say "yes" in a few non-trivial cases. For these, a natural next question would be "How efficiently can we solve it?", i. e. computational complexity should be analyzed.

We also provided some translations between different planning formalisms, or translations to some "normal form" (e. g., only using integers) within the same formalism. This is an area that could be expanded, proving domain compilation results like the ones obtained for propositional planning by Nebel (1999). Here, we only required our translations to be computable, but for practical applications in planning systems, a polynomial translation is much more useful.

When more PDDL2.1 planning domains involving numbers become available, it will be interesting to investigate to what extent – and how easily – they can be represented in the decidable planning formalisms we have studied.

Finally, recalling our introductory comments, the motivation for conducting this analysis was the advent of the PDDL2.1 planning formalism. As we mentioned in the beginning, PDDL2.1 is not just planning with numbers, and although we gave good reasons why numerical state variables were foremost on our agenda, there are some interesting questions around the new features of PDDL2.1 in the area of concurrent planning that have not been covered yet.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

Bixby, R.; Fenelon, M.; Gu, Z.; Rothberg, E.; and Wunderling, R. 2000. MIP: Theory and practice – Closing the gap. In Powell, M. J. D., and Scholtes, S., eds., *System Modelling and Optimization: Methods, Theory and Applications*, volume 174 of *International Federation for Information Processing*. Kluwer Academic Press. 19–49.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1–2):281–300.

Boolos, G. S., and Jeffrey, R. C. 1989. *Computability and Logic*. Cambridge University Press.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1–2):165–204.

Cesta, A., and Borrajo, D., eds. 2001. *Pre-proceedings of the Sixth European Conference on Planning (ECP'01)*.

Cutland, N. J. 1980. *Computability — An Introduction to Recursive Function Theory*. Cambridge University Press.

Do, M. B., and Kambhampati, S. 2001. Sapa: A domain-independent heuristic metric temporal planner. In Cesta and Borrajo (2001), 109–120.

Doherty, P., and Kvarnström, J. 2001. TALplanner: A temporal logic based planner. *AI Magazine* 22(3):95–102.

Edelkamp, S., and Helmert, M. 2001. The model checking integrated planning system (MIPS). *AI Magazine* 22(3):67–71.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence* 76(1–2):65–88.

Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Available at http://www.dur.ac.uk/d.p.long/competition.html.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In Cesta and Borrajo (2001), 121–132.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Matiyasevich, Y., and Senizerguez, G. 1996. Decision problems for semi-Thue systems with a few rules. In *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science (LICS '96)*, 523–531.

Nebel, B. 1999. What is the expressive power of disjunctive preconditions? In Fox, M., and Biundo, S., eds., *Recent Advances in AI Planning. 5th European Conference on Planning (ECP'99)*, volume 1809 of *Lecture Notes in Artificial Intelligence*, 294–307. New York: Springer-Verlag.