Computational Complexity of Planning with Discrete Time and Continuous State Variables

Chitta Baral[1] **and Vladik Kreinovich**[2]
[1]Department of Computer Science and Engineering
Arizona State University, Tempe, Arizona 85287-5406, chitta@asu.edu
[2]Department of Computer Science, University of Texas at El Paso
El Paso, Texas 79968, vladik@cs.utep.edu

Abstract

Traditionally, most planning research in AI was concentrated on systems whose state can be characterized by discrete-valued fluents. In many practical applications, however, we want to control systems (like robots) whose state can only be described if we used continuous variables (like coordinates). Planning for such systems corresponds, crudely speaking, to Level 2 of the planing language PDDL2.1. In this paper, we analyze the computational complexity of such planning problems.

Introduction

Until recently, most planning research in AI was done for *discrete* systems, i.e., for systems in which at any given moment of time $t = 0, 1, 2, \ldots$, the state is characterized by a finite number of fluents each of which can only take a finite number of possible values. Typically, these fluents are propositional, i.e., each fluent can take only two values "true" and "false".

In many practical systems, however, the set of possible states is continuous. For example, a mobile robot can be in any point on a floor. In physics and engineering, a traditional way to characterize a state of such a system is to measure the values of several related quantities. For example, to characterize the position of a robot on a floor, we can measure the values of the two coordinates x and y, or, alternatively, use ultrasonic sensors to measure the distances r_1 and r_2 from the robot to two fixed sensor locations. In other words, instead of discrete fluents, we characterize the state of a system by *continuous* (numerical) fluents, i.e., fluents whose values can be arbitrary real numbers.

Since numerical fluents are practically important, it is important to formulate and solve planning problems for such systems. This is not easy. Even for systems with discrete fluents, planning problems are often computationally difficult; see, e.g., (Baral et al. 2000), and by allowing continuous fluents, we add even more complexity to the problem. It is therefore desirable, before we start developing general algorithms for solving such problems, to gauge their computational complexity. In this paper, we will undertake this analysis.

Planning with Discrete Time and Continuous State Variables: A Formal Description

In order to describe planning for such systems, it is necessary to extend the known planning AI languages by adding numerical fluents. One of such proposed expansions – the language PDDL – is, in our opinion, becoming a standard for such analysis. The latest version of this language PDDL2.1 (PDDL 2002) not only allows numerical fluents, but also many other features such as continuous time (as opposed to discrete time), etc. The more features we add, the more complex the corresponding problems become. Eventually, it is desirable to analyze the computational complexity of all possible planning problems corresponding to adding different features. We are far from this objective. In this short paper, we start this process by analyzing only one generic class of planning problems: problems in which we allow numerical fluents but keep the time discrete.

This class, roughly speaking, corresponds to Level 2 of PDDL2.1 language. Let us formulate problems from this class in precise terms. Since in this paper, we only consider planning problems with continuous variables and discrete time (and we will not consider more complex levels of planning), we will be able to use notations which are slightly simplified in comparison with general PDDL1.1 ones.

Let n denote the *dimension* of a system, i.e., the number of numerical fluents that characterize a state. Then, a state can be characterized by a tuple $s = (s_1, \ldots, s_n)$ of n real values, where s_i is the numerical value of i-th variable.

Changes in states are described by actions. There is a finite list of action types ("names") $a_1, \ldots, a_k$.

To specify an action of a given action type, we must specify the parameters of this action. For example, for a robot, a natural action type is "move forward". To specify an action of this type, we must describe with what exactly speed we want this robot to move. This speed is then the parameter that characterized an action.

For each type of action, we know the number n_k of parameters that characterize actions of this type. The corresponding parameters will be denoted by $p_1, \ldots, p_{n_k}$. To complete the description of a system, we must describe how its state changes when we apply an action. In other words, we must be able, given a state $s(t) = (s_1(t), \ldots, s_n(t))$ of the system at a given moment of time t, the type of the action

a_k, and the parameters $p_1, \ldots, p_{n_k}$ of this action, to describe the state of the system $s(t+1) = (s_1(t+1), \ldots, s_n(t+1))$ at the next moment of time. This change is described by the formulas

$$s_i(t+1) = F_{ki}(s_1(t), \ldots, s_n(t), p_1, \ldots, p_{n_k}), \qquad (1)$$

where, for each k and i, F_{ki} is an algebraic expression with $n + n_k$ variables, an expression that is formed from arithmetic operations $(+, -, \cdot, /)$, $\min$, $\max$, and if-then statements (in terms of PDDL2.1, these formulas represent *assignment statements*).

If we know the expressions describing the dynamics of the system, and we know the initial state $s(0) = (s_1(0), \ldots, s_n(0))$ of the system, then we can use the formula (1) to determine the states $s(1), s(2), \ldots$, of the system at all consequent moments of time.

The *objective* of the plan is characterized by a finite list of equalities $F = 0$ and inequalities $F \geq 0$ and/or $F > 0$, where $F(s_1, \ldots, s_n)$ is an algebraic expression (of the same type as used in the description of the system's dynamics).

For example, if we want to reach a certain place, then the objective is that the distance F from the robot's location to this place be able to 0. If we want to move into a certain circular area of radius 1, then the objective is that the distance d between the robot's location and the center of this area should be less than 1, i.e., that $F > 0$, where $F \stackrel{\text{def}}{=} 1 - d$.

The planning problem can be therefore formulated as follows: given the expressions (1) that describe the dynamics of the system, the initial state $s(0) = (s_1(0), \ldots, s_n(0))$ of the system, the objective, and the upper bound for the time T, we want to find a sequence of actions (i.e., action types and corresponding parameters) that would lead us to a state satisfying the given objective.

In principle, we could formulate more general and more complex objectives, what we formulated is a direct analogue of a typical formulation of a planning problem for discrete states, when the objective is to attain a given value of one of the discrete fluents (or, sometimes, given values of several discrete fluents). The reason why we restrict ourselves to simple objectives is that, as we have mentioned earlier, we want to analyze the complexity change caused by continuity of fluents and we do not want therefore to drastically change other features of the planning problem.

Before we start analyzing the complexity of the above planning problems, we need to make one last comment. From the purely mathematical viewpoint, it is OK to consider all possible real numbers as values of the numerical fluents s_i and of the action parameters p_i, but inside the computer, usually, only rational numbers are represented. A national number is a fraction m/n, so we can represent it by presenting a binary expansion for m and a binary expansion for n. Specifically, if we have binary-rational numbers, then n is simply a power of 2, so this representation corresponds to fixed point real numbers (as opposed to sometimes used floating point real numbers). We will therefore assume that the values s_i and p_i are thus represented rational numbers.

First Result: Even When We Know a Plan, Computing the Final State May Require Exponential Time

Before we start analyzing the complexity of planning, i.e., the complexity of *finding* a plan that satisfies the desired objective, let us first consider the computational complexity of a much simpler problem: of finding out what is the result of applying a given plan (i.e., a given sequence of actions) to a given initial state.

For traditional discrete control, with discrete number of states, each transition from $s(t)$ to $s(t+1)$ requires a fixed amount of time. A natural way of computing $s(T)$ based on our knowledge of $s(0)$ is to apply this transition T times: from $s(0)$ to $s(1)$, from $s(1)$ to $s(2)$, etc. Thus, we can compute $s(T)$ by using computation time that grows linearly with T.

At first glance, it may seem that for problems with continuous state variables, we should get the same computational complexity. Alas, a rather simple example shows that the results can be much worse:

Theorem 1. *There exists a system for which computing $s(T)$ for a given T requires computation time that grows as 2^T.*

For the convenience of the readers, all the proofs are placed in the special Proofs section at the end of this paper.

Second Result: There Exists a Planning Algorithm

Since even computing the state $s(T)$ may require exponential time, there is no hope that solving the planning problem can be done faster. This conclusion is not so bad because even for discrete systems, the corresponding planning problem is NP-hard (Baral et al. 2000; Bylander 1994; Erol at al. 1995; Liberatore 1997) – which means, most probably, that the actual solution of this problem may require exponential time for some cases. So, at first glance, it may seem that the situation is not much worse than before.

However, there is a subtle point here. In the discrete case, we might worry whether there is a *feasible* algorithm for solving the planning problem, but we knew for sure that there exists *an* algorithm for checking whether there is plan. Indeed, since at any given moment of time, we had only finitely many possible actions, we could therefore enumerate all possible plans of length T and thus, use exhaustive search to check whether such a plan exists (and what is this plan).

For continuous systems, we have infinitely many possible actions – corresponding to infinitely many possible rational values of parameters characterizing action of each type. Therefore, we cannot simply try all possible plans. In other words, there is no longer a simple answer to the question of whether there is *an* algorithm for solving the planning problem. Good news is that, although we no longer have a *simple* algorithm, we do have *an* algorithm:

Theorem 2. *There exists an algorithm for solving the planning problem with continuous state variables.*

In other words, there exists an algorithm that, given the dynamics of the system, an initial state, time bound T, and an objective, checks whether there exists a plan that satisfies this objective, and if the answer if "yes", produces this plan.

As the readers can see from the proof of Theorem 2, this algorithm uses the Tarski-Seidenberg result about the decidability of the first order theory of real numbers. All known algorithms for solving that problem require, in the worst case, exponential time. However, this worst-case exponential time complexity does not mean that the planning problem is hopeless.

Indeed, as we have mentioned, the planning problem for the traditional discrete-state case is NP-hard, but there are efficient algorithms that solve many useful real-life planning problems. Similarly, there exist efficient algorithms that solve many practical problems by reducing them to the first order theory of real numbers. In particular, there exist applications to transportation problems (Loos et al. 1993), to control system design (Abdallah et al. 1996), etc.

Additional Problem for Planning with Continuous States: Partial Knowledge and Interval Computations

In the above text, we assumed that we have a complete knowledge of the initial state, i.e., that we know the *exact* values of the numerical fluents $s_i(0)$ that describe the initial state of the system. The main reason why we made this assumption is that this is a typical assumption in traditional planning problems, in which states are characterized by discrete fluents.

For discrete fluents, this assumption is very realistic. For example, if our description of the state of a system includes knowing the truth value of a fluent "the light is on", then it is natural to assume that we know whether the light is on or not. If we originally had a partial knowledge, i.e., if we did not know the values of some fluents, they we can make a finite number of observations and learn the values of all of them.

For planning problems with continuous states, however, the assumption of complete knowledge is much less realistic. Indeed, we get the values of the numerical fluents from measurements, but measurements are never 100% accurate. We may may make very precise measurements, but there is always a possibility of some measurement error, as a result of which the measured value $\widetilde{x}$ of the measured quantity may somewhat differ from the actual (unknown) value x of this quantity.

Usually, the manufacturer of the measuring instrument provides us with a guaranteed upper bound Δ on the measurement error $\Delta x \stackrel{\rm def}{=} \widetilde{x} - x$. In other words, we are guaranteed that $|\Delta x| = |\widetilde{x} - x| \le \Delta$. Therefore, when we get the measurement result $\widetilde{x}$, the only conclusion that we can make about the actual value x of the measured quantity is that x is within the interval $\mathbf{x} \stackrel{\rm def}{=} [x^-, x^+]$, where $x^- \stackrel{\rm def}{=} \widetilde{x} - \Delta$ and $x^+ \stackrel{\rm def}{=} \widetilde{x} + \Delta$.

As a result, for each of the numerical fluents $s_i(0)$ that describe the initial state of the system, we only know an *interval* $[s_i^-(0), s_i^+(0)]$ of possible values – i.e., we have only *partial* information about the initial state of the system. The actual initial state $s(0)$ can be any tuple $(s_1(0), \ldots, s_n(0))$ for which $s_1(0) \in [s_1^-(0), s_1^+(0)]$, …, $s_n(0) \in [s_n^-(0), s_n^+(0)]$.

We must be able to solve a planning problem under this partial information. In other words, we must describe a sequence of actions such that for each possible state $s(0)$, the resulting state satisfies the required objectives.

Planning under interval uncertainty can be viewed as a particular case of *interval computations*, i.e., computations in which the input is only known with interval uncertainty (see, e.g., (Kearfott et al. 1996); see also (Trejo et al. 2000)). Interval computations have been used, together with more traditional AI techniques, to produce a robot which won 1st place at the AAAI'97 robot competition (Baral and Son 1997a; Baral et al. 1998; Morales et al. 1998). For the latest applications in robotics see, e.g., (Jaulin et al. 2001).

The partial information does not necessarily consist only of intervals. In addition to knowing the upper bound on the measurement errors, we sometimes have some information about the probabilities of different values of measurement errors. In some cases, we have a full information about the probability distributions; these cases are covered by POMDP planning models; see, e.g., (Heffner et al. 1998; Kaelbling, et al. 1998) and references therein. In some cases, we only have a partial information about these probabilities; see, e.g., (Trejo et al. 2000).

In this paper, we only consider the simplest case of pure interval uncertainty.

Planning under Partial (Interval) Knowledge: There Still Exists a Planning Algorithm

The first natural question is: with this additional complication, is there still an algorithm for solving the corresponding general planning problem, or does this problem become algorithmically undecidable? The answer is "yes":

Theorem 3. *There exists an algorithm for solving the planning problem with continuous state variables under partial (interval) knowledge.*

In other words, there exists an algorithm that, given the dynamics of the system, an interval information about the initial state, time bound T, and an objective, checks whether there exists a plan that satisfies this objective, and if the answer if "yes", produces this plan.

If We Take Interval Uncertainty Into Consideration, Then Even Predicting the Next State $s(1)$ Is NP-Hard

Now that we know that the algorithm exists, a natural question is: how difficult is this problem? how long will an algorithm take?

Similarly to the case of complete knowledge, let us start our complexity analysis with the simplest problem: we have information about the initial state $s(0)$, we know a plan – i.e., a sequence of actions (action types and corresponding parameter values), and we want to predict the state $s(T)$ at moment T. For the case of complete knowledge, when we assumed that we know the exact values of the numerical fluents, we can apply straightforward computations (that follow formula (1) on each step) and get the resulting state $s(T)$. In Theorem 1, we have shown that while one step – i.e., transition from $s(0)$ to $s(1)$ – is easy to implement, the overall computational complexity grows exponentially when T grows.

It turns out that when we take the (inevitable) interval uncertainty into consideration, then even the problem of predicting $s(1)$ becomes difficult. Specifically, since we do not know the exact values of $s_i(0)$, we can have many possible states $s(0)$. When we apply the the formula (1) to these states, we get different states $s(1)$.

So, the problem of predicting the next state can be formulated as follows: We know the intervals $[s_i^-(0), s_i^+(0)]$ of possible values of the numerical fluents $s_i(0)$, we know the expressions (1), and we want to know, for every i from 1 to n, the interval of possible values of $s_i(1)$.

In some cases, this problem is easy to solve. For example, a robot moving along the straight line can be characterized by its position s_1. The command to move with a velocity p_1 leads to a new state $s_1 + p_1 \cdot \Delta t$, where Δt is the physical time (in seconds) elapsed between the two consequent moments of time. In this case, $s_1(1) = s_1(0) + p_1 \cdot \Delta t$, so if we know the interval $[s_1^-(0), s_1^+(0)]$ of possible values of the initial state, we can easily predict the interval of possible values of $s_1(1)$ as

$$[s_1^-(1), s_1^+(1)] = [s_1^-(0) + p_1 \cdot \Delta t, s_1^+(0) + p_1 \cdot \Delta t].$$

In this simple case, the problem is easy to solve, but in general, this problem is NP-hard:

Theorem 4. *For systems with continuous states, if we take interval uncertainty into consideration, then the following problem becomes NP-hard: given:*

- *the information about the initial state $s(0)$, i.e., the intervals $[s_i^-(0), s_i^+(0)]$ of possible values of $s_i(0)$,*
- *the dynamics (1), and*
- *the action,*

to predict the next state $s(1)$, i.e., the intervals $[s_i^-(1), s_i^+(1)]$ of possible values of $s_i(1)$.

This theorem was, in effect, proven in (Gaganov 1985; Kreinovich et al. 1997; Vavasis 1991).

Direct Interval Computations – A Continuous State Analogue of 0-Approximation

0-Approximation: Main Idea Reformulated

We have just shown that for systems with continuous states, if we take into consideration the (inevitable) interval uncertainty, then even predicting the next state $s(1)$ – i.e., to be more precise, predicting the exact intervals of possible values of the numerical fluents – becomes NP-hard. In other words, unless P=NP, we cannot have a feasible algorithm for exactly computing these intervals.

This difficulty is similar to the difficulty of more traditional discrete-state planning under partial knowledge. For discrete states, there is a useful feasible approach for solving such problems – the approach of 0-approximation proposed in (Baral and Son 1997).

To explain how we want to apply the idea of 0-approximation to our case, let us reformulate the 0-approximation algorithm in interval terms (in this reformulation, we follow (Trejo et al. 2000). In the discrete case, in the case of complete knowledge, for each fluent f, the truth value is either "true" or "false". We can reformulate "true and "false" in numerical terms: "true" means that the probability of this fluent being true is 1, and "false" means that the probability for this fluent to be true is 0. In case of partial knowledge, we also have the possibility that about some fluents, we do not know whether they are true or not. In this case, the probability of this fluent to be true can take any value from the interval $[0, 1]$. So, in the case of partial knowledge, for each fluent $f \in \mathcal{F}$, the initial probability interval $\mathbf{p}(f)$ is equal to $[0, 0]$, $[1, 1]$, or $[0, 1]$.

In terms of these intervals, the 0-approximation algorithm can be described as follows: To check whether a plan $\alpha = [a_1, \ldots, a_n]$ is successful, for each moment of time $t = 1, \ldots, n$, and for each fluent $f \in \mathcal{F}$, we estimate the interval $\mathbf{p}(f, s_t)$ of possible values of this fluent's probability in the state $s_t = res(a_t, res(a_{t-1}, \ldots, res(a_1, s) \ldots))$. To be more precise, for each t and f, we compute the *enclosure* $\widetilde{\mathbf{p}}(f, s_t) \supseteq \mathbf{p}(f, s_t)$. We start with the known values $\widetilde{\mathbf{p}}(f, s_0) = \mathbf{p}(f, s_0)$; after the estimates $\widetilde{\mathbf{p}}(f, s_t)$ are found for a certain t, we compute the estimates for s_{t+1} as follows: Let $V^+(a, s_t)$ denote the set of all fluent literals F for which the rule base D contains a rule "a causes F if $F_1, \ldots, F_m$" for which all the conditions F_i are definitely true (have probability intervals $\widetilde{\mathbf{p}}(F_i, s_t) = [1, 1]$). Let $V^-(a, s_t)$ denote the set of all fluent literals F for which D contains a rule "a causes F if $F_1, \ldots, F_m$" for which all the conditions F_i may be true (have probability intervals $\widetilde{\mathbf{p}}(F_i, s_t) \neq [0, 0]$). Then, for every fluent $f \in \mathcal{F}$:

- We assign $\widetilde{\mathbf{p}}(f, s_{t+1}) := [1, 1]$ if

$f \in V^+(a_{t+1}, s_t) \vee (\widetilde{\mathbf{p}}(f, s_t) = [1, 1] \& \neg f \notin V^-(a_{t+1}, s_t))$.

- We assign $\widetilde{\mathbf{p}}(f, s_{t+1}) := [0, 0]$ if

$\neg f \in V^+(a_{t+1}, s_t) \vee (\widetilde{\mathbf{p}}(f, s_t) = [0, 0] \& f \notin V^-(a_{t+1}, s_t))$.

- In all other cases, we take $\widetilde{\mathbf{p}}(f, s_{t+1}) := [0, 1]$.

It is proven that this algorithm indeed produces an enclosure and thus, if we get $\widetilde{\mathbf{p}}(f, s_n) = [1, 1]$ at the final state, we are thus guaranteed that this plan works.

Continuous Analogue of 0-Approximation: Direct Interval Computations

In mathematical terms, the interval $\mathbf{s}_i(1)$ is a range of the function F_{ki} when the inputs $s_i(0)$ take values from the intervals $\mathbf{s}_i(0)$. So, in these terms, what we are looking for is good algorithms for computing a range of a given function $f(x_1, \ldots, x_n)$ on given intervals $\mathbf{x}_i = [x_i^-, x_i^+]$.

Similarly to the (above reformulated) 0-approximation, we can produce a similar feasible algorithm for providing *enclosures* for the intervals $\mathbf{s}_i(1)$, i.e., intervals $\mathbf{S}_i(1)$ that are guaranteed to contain the desired intervals $\mathbf{s}_i(1)$. If we can produce an enclosure, then, by using the same algorithm, we can produce enclosures $\mathbf{S}_i(2)$ for the intervals $\mathbf{s}_i(2)$, etc., until we get enclosures $\mathbf{S}_i(T)$ for the intervals $\mathbf{s}_i(T)$. If all the states $s(T)$ for which, for every i, $s_i(T)$ is within the enclosure $\mathbf{S}_i(T)$, satisfy the desired objective, then we are thus guaranteed that all possible final state satisfied our objective, and thus, that the tested plan is indeed a solution to our planning problem.

Of course, since the computed interval is an enclosure, it can happen that while all the states within the intervals $\mathbf{s}_i(T)$ satisfy our objectives, some additional states within the larger intervals $\mathbf{S}_i(T)$ do not satisfy the objectives, and thus, our analogue of 0-approximation will declare a correct action plan to be wrong. This is the same drawback that we have with the original 0-approximation. Since the problem of computing the exact intervals for $s_i(T)$ is NP-hard, we cannot have a perfect plan checking feasible algorithms, we have to live with some drawbacks. Missing a good plan is not very good, but at least we are guaranteed that when the generalized 0-approximation proclaims the plan to be good, it is actually good.

This analogue of 0-approximation is actually known since the 1950s, when it was first used by Ramon E. Moore from Lockheed and Stanford to plan a spaceflight to the Moon (Moore 1959). It is called *direct interval computations*. It started with the observation that for simple arithmetic operations $f(x_1, x_2) = x_1 + x_2$, $x_1 - x_2$, etc., the range can be computed explicitly; e.g.:

$$[x_1^-, x_1^+] + [x_2^-, x_2^+] = [x_1^- + x_2^-, x_1^+ + x_2^+];$$

$$[x_1^-, x_1^+] - [x_2^-, x_2^+] = [x_1^- - x_2^+, x_1^+ - x_2^-];$$

$$[x_1^-, x_1^+] \cdot [x_2^-, x_2^+] = [\min(x_1^- \cdot x_2^-, x_1^- \cdot x_2^+, x_1^+ \cdot x_2^-, x_1^+ \cdot x_2^+),$$

$$\max(x_1^- \cdot x_2^-, x_1^- \cdot x_2^+, x_1^+ \cdot x_2^-, x_1^+ \cdot x_2^+)].$$

The corresponding expressions are called formulas of *interval arithmetic*.

It turns out that, similarly to 0-approximation, we can use these expressions to get reasonable enclosures for arbitrary functions f. Indeed, when the computer computes the function f, it *parses* the function, i.e., it represents the computation as a sequence of elementary arithmetic operations. It can proven, by induction, that if we start with intervals and replace each arithmetic operation with the corresponding operation of interval arithmetic, at the end, we get an enclosure for f. For example, if $f(x) = x \cdot (1 - x)$, represent f as a sequence of two elementary operations:

- $r := 1 - x$ (r denotes the 1st intermediate result);
- $y := x \cdot r$.

In the interval version, perform the following computations: $\mathbf{r} := 1 - \mathbf{x}$; $\mathbf{y} := \mathbf{x} \cdot \mathbf{r}$.

In particular, when $\mathbf{x} = [0, 1]$, compute the intervals $\mathbf{r} := [1, 1] - [0, 1] = [0, 1]$, and

$$\mathbf{y} := [0, 1] \cdot [0, 1] = [\min(0 \cdot 0, 0 \cdot 1, 1 \cdot 0, 1 \cdot 1),$$

$$\max(0 \cdot 0, 0 \cdot 1, 1 \cdot 0, 1 \cdot 1)] = [0, 1].$$

The interval $[0, 1]$ is indeed an enclosure of the actual range $[0, 0.25]$.

The enclosure obtained by using the above simple idea is often too wide. One of the main objectives of interval computations is to make this enclosure narrower; see, e.g., (Jaulin et al. 2001; Kearfott et al. 1996).

Proofs

Proof of Theorem 1

In our example, we will consider the effects of repeating a single action a on a 1-dimensional system. The state of a system is characterized by a single variable s_1. Its initial value is $s_1(0) = 2$, and the dynamics is described by the expression $s_1(t+1) = s_1(t)^2$. For this system, as one can easily prove by induction, $s_1(T) = 2^{2^T}$. If we want to produce $s_1(T)$, we must generate all the bits in the binary expansion of $s_1(T)$. In binary notation, this integer is represented by 1 and 2^T zeros. Thus, we need to generate at least $2^T + 1$ zeros, i.e., we need at least $2^T + 1$ computational steps. The theorem is proven.

Proof of Theorem 2

In our description of systems with continuous state variables, we used expressions obtained from variables by using arithmetic operations, $\min$, $\max$, and if-then constructions. These functions are a particular case of the so-called *semi-algebraic* functions (see, e.g., (Arnold 1983)).

A set $S \subseteq \mathbb{R}^q$ is called *semi-algebraic* if it is a finite union of subsets, each of which is defined by a finite system of polynomial equations $P_r(x_1, \ldots, x_q) = 0$ and inequalities of the types $P_s(x_1, \ldots, x_q) > 0$ and $P_t(x_1, \ldots, x_q) \geq 0$ – for some polynomials P_i with integer coefficients. A function $f : \mathbb{R}^n \to \mathbb{R}$ is called *semi-algebraic* if its graph $\{(x, f(x))\}$ is a semi-algebraic set.

According to the famous Tarski-Seidenberg theorem (Seidenberg 1954; Tarski 1951) (see also (Arnold 1983)), every relation that is obtained from a semi-algebraic relation by adding quantifiers $\forall x$, $\exists x$ (that run over all real numbers x), is still semi-algebraic. For a closed formula, there is an algorithm for checking whether this formula is true or not. For a formula with variables, there is an algorithm that, for each formula for which a formula $\exists x_1 \ldots \exists x_m\, F$ with semi-algebraic F is true, produces a set of values $x_1, \ldots, x_n$ for which F is indeed true.

The condition that there exists a plan which satisfies a given objective can be formulated in these terms, with quantifiers over parameters of actions characterizing this plan. Specifically, for each sequence of action types, the possibility to have a plan consisting of actions of these types can be described as

$$\exists p_1(0) \ldots \exists p_{n(1)}(0) \exists s_1(1) \ldots \exists s_n(1)$$

$$\ldots$$

$$\exists p_1(T-1) \ldots \exists p_{n(T)}(T-1) \exists s_1(T) \ldots \exists s_n(T)\, G,$$

where $p_i(t)$ are parameters of actions applied at time t, $s_i(t)$ are numerical fluents describing the state at time t, and the formula G describes that each transition is going on according to the dynamics (1), and that the resulting state satisfies the objective. Since the functions F_{ki} describing the dynamics and the expressions used in describing the objective are all semi-algebraic, the formula G is semi-algebraic too. Thus, Tarski-Seidenberg algorithm can be used to solve our planning problem. The theorem is proven.

Proof of Theorem 3

Theorem 3 follows from the same Tarski-Seidenberg result. The only difference is that the corresponding first order formula becomes somewhat more complex; it now looks like this:

$$\exists s_1(0) \ldots \exists s_n(0)$$
$$\exists p_1(0) \ldots \exists p_{n(1)}(0) \exists s_1(1) \ldots \exists s_n(1)$$
$$\ldots$$
$$\exists p_1(T-1) \ldots \exists p_{n(T)}(T-1) \exists s_1(T) \ldots \exists s_n(T)\, G',$$

where variables $p_i(t)$ and $s_i(t)$ have the same meaning as in the proof of Theorem 2, and the main difference with the formula (2) is that the new formula G' describes not only that each transition is going on according to the dynamics (1), and that the resulting state satisfies the objective, but also that the original values $s_i(0)$ are within the given intervals $[s_i^-(0), s_i^+(0]$. Similarly to Theorem 2, the formula G' is semi-algebraic and therefore, Tarski-Seidenberg algorithm can be used to solve our planning problem. The theorem is proven.

Acknowledgments

This work was supported in part by NASA grants NCC5-209 and NCC 2-1232, by the Air Force Office of Scientific Research grants F49620-95-1-0518 and F49620-00-1-0365, and by NSF grants CDA-9522207, ERA-0112968 and 9710940 Mexico/Conacyt.

References

C. Abdallah, P. Dorato, R. Liska, S. Steinberg, and W. Yang, "Applications of quantifier elimination theory to control system design", *4th IEEE Mediterranean Symposium on Control and Automation*, 1996.

V. I. Arnold, *Geometrical methods in the theory of ordinary differential equations*, Springer-Verlag, N.Y., 1983.

C. Baral *et al.*, "From theory to practice: The UTEP robot in AAAI 96 and AAAI 97 robot contests", *Proc. 2nd International Conference on Autonomous Agents (Agents'98)*, 1998, pp. 32-38.

C. Baral, V. Kreinovich, and R. Trejo, "Computational complexity of planning and approximate planning in presence of incompleteness", *Artificial Intelligence*, 2000, Vol. 122, pp. 241–267.

C. Baral and T. Son, "Approximate reasoning about actions in presence of sensing and incomplete information", In: *Proc. of International Logic Programming Symposium (ILPS'97)*, 1997, pp. 387–401.

C. Baral and T. Son, "Regular and special sensing in robot control – relation with action theories", *Proc. AAAI 97 Workshop on Robots, Softbots, and Immobots – Theories of Action, Planning and Control*, 1997a.

T. Bylander, "The computational complexity of propositional STRIPS planning", *Artificial Intelligence*, 1994, Vol. 69, pp. 161–204.

K. Erol, D. Nau, and V.S. Subrahmanian, "Complexity, decidability and undecidability results for domain-independent planning", *Artificial Intelligence*, 1995, Vol. 76, No. 1–2, pp. 75–88.

A. A. Gaganov, "Computational complexity of the range of the polynomial in several variables", *Cybernetics*, 1985, pp. 418–421.

H. Geffner and B. Bonet, "High-Level Planning and Control with Incomplete Information Using POMDPs", *Proc. AIPS-98 Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, 1998.

L. Jaulin, M. Keiffer, O. Didrit, and E. Walter, *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics*, Springer-Verlag, London, 2001.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains", *Artificial Intelligence*, 1998, Vol. 101, pp. 99–134.

R. B. Kearfott and V. Kreinovich (eds.), *Applications of Interval Computations*, Kluwer, Dordrecht, 1996.

V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1997.

P. Liberatore, "The complexity of the language $\mathcal{A}$", *Electronic Transactions on Artificial Intelligence*, 1997, Vol. 1, pp. 13–28 (http://www.ep.liu.se/ej/etai/1997/02).

R. Loos and V. Weispfenning, "Applying linear quantifier elimination", *Computer Journal*, 1993, Vol. 36, No. 5, pp. 450–462.

R. E. Moore, *Automatic error analysis in digital computation*, Lockheed Missiles and Space Co. Technical Report LMSD-48421, Palo Alto, CA, 1959.

D. Morales and Tran Cao Son, "Interval Methods in Robot Navigation", *Reliable Computing*, 1998, Vol. 4, No. 1, pp. 55–61.

PDDL website
http://www.dur.ac.uk/d.p.long/IPC/pddl.html

A. Seidenberg, "A new decision method for elementary algebra", *Annals of Math.*, 1954, Vol. 60, pp. 365–374.

R. Trejo, V. Kreinovich, and C. Baral, "Towards Feasible Approach to Plan Checking Under Probabilistic Uncertainty: Interval Methods", *Proc. AAAI'2000*, Austin, TX, July 30–August 3, 2000, pp. 545–550.

A. Tarski, *A Decision Method for Elementary Algebra and Geometry*, University of California Press, Berkeley, 1951.

S. A. Vavasis, *Nonlinear optimization: complexity issues*, Oxford University Press, N.Y., 1991.