Timed Planning

Ajay Bansal
Department of Computer Science,
Georgetown University,
Washington, DC 20057.
Email: bansal@cs.georgetown.edu

Neda Saeedloei and **Gopal Gupta**
Department of Computer Science,
The University of Texas at Dallas,
Richardson, TX 75080.
Email: {nxs048000, gupta}@utdallas.edu

Abstract

Planning has been at the forefront of research in the areas of Artificial Intelligence and cognitive science. High-level action description languages (e.g., language $\mathcal{A}$) have been used to specify, verify and diagnose plans. Timed Planning is planning under real-time constraints. To specify timed planning problems, an extension of the action description language $\mathcal{A}$ with real-time stop-watches, called $\mathcal{A}_T$ has been used. In this paper, we show how timed planning domains (described in $\mathcal{A}_T$) can be easily and elegantly encoded as answer set programs extended with constraints over reals.

1. Introduction

Planning has been an active area of research since the early days of AI and cognitive science. In planning, a domain description *D* is given along with a set of observations about the initial state *O* and a collection of fluent literals $G = \{g_1, \ldots, g_l\}$, which is referred to as a goal. The problem is to find a sequence of actions $a_1, \ldots, a_n$ such that $\forall i,\ 1 \le i \le l$, *D* entails g_i from initial state *O*, after actions $a_1, \ldots, a_n$. The sequence of actions $a_1, \ldots, a_n$ is called a plan for goal *G* w.r.t. *(D,O)* (Baral 2003).

Timed Planning is planning under real-time constraints. For real-time domains, the occurrence of an action is as important as the time at which the action occurs. One needs to be able to reason about time in a quantitative manner, as the system may have real-time constraints that must be satisfied for an action to occur. In the field of logic programming, action description languages (e.g., language $\mathcal{A}$ - realized via Answer Set Programming (ASP)) have been used to represent and reason about actions and change (Gelfond and Lifschitz 1993). These are (high-level) languages that can be used to specify, verify and diagnose plans.They are widely used for planning with domain specific constraints To encode timed planning problems, an extension of the action description language $\mathcal{A}$ with real-time stop-watches, called $\mathcal{A}_T$ has been proposed (Simon, Mallya, and Gupta 2005). However, the implementation of $\mathcal{A}_T$ is quite ad hoc. In this paper, we show how timed planning domains (described in $\mathcal{A}_T$) can be elegantly encoded as answer set programs extended with constraints over reals.

2. A Real-time Action Description Lang.: $\mathcal{A}_T$

The action description language $\mathcal{A}$ was proposed by Gelfond and Lifschitz to represent actions and change using logic programs (Gelfond and Lifschitz 1993). *Real-time systems* are computing systems in domains where response within a hard time bound is critical for success. The real-time action description language $\mathcal{A}_T$, which extends language $\mathcal{A}$ by augmenting action with real-time constraints, is presented in (Simon, Mallya, and Gupta 2005). $\mathcal{A}_T$ defines a complete real-time action α by pairing its name with a list of clock constraints associated with it. In $\mathcal{A}_T$, this is written as

$$A \textbf{ at } T_1, \ldots, T_n$$

where $T_1 \ldots, T_n$ $(n \ge 0)$ are clock constraints of the form $C \le E$, $C \ge E$, $C < E$, and $C > E$, where C and E are clock names or a clock name plus or minus a real valued constant. When $n = 0$ the **at** clause can be dropped.

With the ability to explicitly state when an action occurs, value propositions can be easily extended to include it. Given fluent expressions $F_1, \ldots, F_m$ $(m > 0)$ and real-time actions $\alpha_1, \ldots, \alpha_n$ $(n \ge 0)$, a real-time value proposition can be written as:

$$F_1, \ldots, F_m \textbf{ after } \alpha_1; \ldots; \alpha_n$$

When the sequence of actions is empty $(n = 0)$, a real-time value proposition is written as

$$\textbf{initially } F_1, \ldots, F_m$$

The real-time effect propositions (sometimes referred to as action rules) in $\mathcal{A}_T$ are written as

$$A \textbf{ causes } F_1, \ldots, F_m \textbf{ resets } C_1, \ldots C_n$$
$$\textbf{when } T_1, \ldots, T_k \textbf{ if } P_1, \ldots, P_i$$

for action name A, fluent expressions $F_1 \ldots, F_m$, $P_1, \ldots, P_i$ $(m, i \ge 0)$, clock names $C_1, \ldots, C_n$ $(n \ge 0)$, and clock constraints $T_1, \ldots, T_k$ $(k \ge 0)$, where $m + n + k + i > 0$. As usual, when m, n, k, or i is zero the keywords **causes**, **resets**, **when**, or **if** respectively, can be dropped. The **resets** clause specifies clocks that are to be reset, assuming the **when** clause and fluent preconditions are satisfied. Clocks continue to advance, if they are not reset. A special action **wait** that denotes the action of waiting for time to elapse is also provided in $\mathcal{A}_T$. This action acts as a wild-card that matches all other action names, and thus provides the ability to encode the passing of time in the current state of the system.

3. $\mathcal{A}_T$ = ASP + CLP(R)

$\mathcal{A}_T$ programs can easily and elegantly be encoded as answer set programs extended with constraints over reals. Given an answer set program, constraints over reals can be embedded within goals in the body of the rules, similar to how Horn clause logic programs are extended with constraints. As the body of these rules are evaluated in a goal-directed manner, these constraints are posted in the constraint store as they are encountered. Posting a constraint that is not entailed by the constraint store will result in a failure which will cause backtracking. Constraints over reals in logic programming are realized in the CLP(R) system (Jaffar and Lassez 1987), which we have incorporated in our goal-directed implementation of Answer Set Programming (ASP).

3.1 Example: Fragile Object Domain

Now we present a domain with real-time constraints on its actions (Gelfond and Lifschitz 1993). The real-time Fragile Object domain, extends the original example with the notion that a dropped object can be caught before it hits the ground. We assume the object takes 1 second to hit the ground. The assumption that units are in seconds is merely a convention we use in this example. In the language $\mathcal{A}_T$ all clocks are variables in the real number domain, i.e., they can take any arbitrary real values. Figure 1 depicts the real-time Fragile Object Domain.

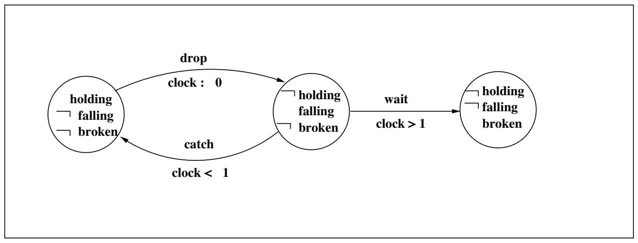

Figure 1: Real-time Fragile Object Domain

The language $\mathcal{A}_T$ describes many possible worlds. In one of these worlds **initially** $Holding, \neg Falling, \neg Broken$ is true, and therefore $Broken$ **after** $Drop$; **wait at** $Clock = 2$ also holds as the object is dropped and then allowed to fall to the ground. In that same world, if one takes too long to catch the object, then the object still shatters on the ground. Hence in the aforementioned world $Broken$ **after** $Drop$; $Catch$ **at** $Clock = 2$ is also true. However, if the object is dropped and then is successfully caught, say at half a second after dropping (i.e., before it hits the ground), then the object is not broken by the sequence of events, i.e., $\neg Broken$ **after** $Drop$; $Catch$ **at** $Clock = 0.5$ is true. Other possible worlds include the object starting out already in a falling state, while another world could even have the object already broken. Table 1 shows the encoding of the real-time Fragile Object Domain in language $\mathcal{A}_T$.

Table 1: Fragile Object Domain in language $\mathcal{A}_T$

$Drop$ **causes** $\neg Holding, Falling$
  resets $Clock$ **if** $Holding, \neg Falling$

$Catch$ **causes** $Holding, \neg Falling, \neg Broken$
  when $Clock \leq 1$ **if** $\neg Holding, Falling$

wait causes $Broken, \neg Falling$
  when $Clock > 1$ **if** $\neg Holding, Falling$

3.2 General Procedure

Now we briefly present the general procedure for encoding an $\mathcal{A}_T$ program in our integrated framework. The general $\mathcal{A}_T$ command:

$$A \textbf{ causes } F_1, \ldots, F_l, \neg F_{l+1}, \ldots, \neg F_m \\ \textbf{resets } C_1, \ldots, C_n \textbf{ when } T_1, \ldots, T_k \\ \textbf{if } P_1, \ldots, P_u, \neg P_{u+1}, \ldots, \neg P_v$$

is encoded in our framework as:

for each F_i ($i = 1 \ldots l$), we define
  holds(F_i, res(A, S)) :-
    holds(P_1, S), . . ., holds(P_u, S),
    not_holds(P_{u+1}, S), . . ., not_holds(P_v, S),
    T_1, . . ., T_n, $C_1 > 0$, . . ., $C_n > 0$,
    $NewC_1 > C_1$, . . ., $NewC_n > C_n$.

for each F_j ($j = l + 1 \ldots m$), we define
  not_holds(F_j, res(A, S)) :-
    holds(P_1, S), . . ., holds(P_u, S),
    not_holds(P_{u+1}, S), . . ., not_holds(P_v, S),
    T_1, . . ., T_n, $C_1 > 0$, . . ., $C_n > 0$,
    $NewC_1 > C_1$, . . ., $NewC_n > C_n$.

For the *fluents* that are true initially, we define the *holds* and *not_holds* clauses with the second argument as $s0$.

4. Conclusions

We presented how real-time domains and thus timed planning problems can easily and elegantly be encoded in our integrated framework, that combines the power of CLP and ASP in one system. The ability to do non-monotonic reasoning (ASP) in presence of time constraints (CLP) in a single system, is needed to realize Timed Planning.

References

Baral, C. 2003. *Knowledge Representation - Reasoning and Declarative Problem Solving.* Cambridge Univ. Press.

Gelfond, M., and Lifschitz, V. 1993. Representing action and change by logic programs. *Journal of Logic Programming* 17(2/3&4):301–321.

Simon, L.; Mallya, A.; and Gupta, G. 2005. Design and implementation of A_T: A real-time action description language. In *International Workshop on Logic-based Program Synthesis and Transformation.* Springer Verlag.

Jaffar, J., and Lassez, J. L. 1987. Constraint Logic Programming. In *POPL*, 111–119.