Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence

Temporally Expressive Planning Based on Answer Set Programming with Constraints

Forrest Sheng Bao and **Yuanlin Zhang**
Department of Computer Science, Texas Tech University, Lubbock, Texas 79409
Emails: forrest.bao@ gmail.com, y.zhang@ ttu.edu

Abstract

Recently, a new language $\mathcal{AC}(\mathcal{C})$ was proposed to integrate answer set programming (ASP) and constraint logic programming (CLP). In this paper, we show that temporally expressive planning problems in PDDL2.1 can be translated into $\mathcal{AC}(\mathcal{C})$ and solved using $\mathcal{AC}(\mathcal{C})$ solvers. Compared with existing approaches, the new approach puts less restrictions on the planning problems and is easy to extend with new features like PDDL axioms. It can also leverage the inference engine for $\mathcal{AC}(\mathcal{C})$ which has the potential to exploit the best reasoning mechanisms developed in the ASP, SAT and CP communities. More details, including the proof to the correctness of the translation, are provided online at https://sites.google.com/site/pddl2acc/

Introduction

In 2003, as an extension to the PDDL language, PDDL2.1 introduced durative actions to represent temporal information of a domain (Fox and Long 2003). Several solvers had been developed for PDDL2.1. However, those solvers can not find plans for *temporally expressive problems* (Cushing et al. 2007). A planning problem is temporally expressive if any plan of the problem requires concurrent actions.

To address the weakness of existing solvers, Coles and colleagues developed new systems COLIN (Coles et al. 2009) and POPF (Coles et al. 2010) for temporally expressive problems. Huang et al. (Huang, Chen, and Zhang 2009) developed a SAT based approach, and Hu (Hu 2007) proposed a constraint satisfaction problem based planner.

As Coles et al.'s systems are specialized for planning problems, it cannot fully exploit the existing inference engines (e.g., SAT and constraint solvers). It is also hard to extend this approach to handle new features, e.g., PDDL axioms (Thiébaux, Hoffmann, and Nebel 2005). Huang et al.'s method needs the discretization of time which may lead to large problem instances and therefore the planner may be very slow. This approach cannot take advantage of effective constraint reasoning algorithms either. Both Huang et al.'s and Hu's approaches cannot solve planning problems with continuous durative actions. It is also unclear how Hu's work can be extended to support PDDL axioms.

Our approach offers a full treatment of planning problems in PDDL2.1, based on a general purpose language $\mathcal{AC}(\mathcal{C})$ (Mellarkod, Gelfond, and Zhang 2008), which integrates answer set programming (ASP) and constraint logic programming (CLP). It is well known that ASP provides elegant descriptions for planning problems while CLP can effectively represent and reason with (temporal) constraints. In this paper we will show that drawbacks of existing approaches can be resolved by translating PDDL2.1 planning problems into $\mathcal{AC}(\mathcal{C})$ and finding plans by $\mathcal{AC}(\mathcal{C})$ solvers. More details about this work is at http://narnia.cs.ttu.edu/drupal/node/4

The $\mathcal{AC}(\mathcal{C})$ Language

An $\mathcal{AC}(\mathcal{C})$ program consists of rules of the form

$$l_0 \leftarrow l_1, \ldots, l_m, \ \textit{not}\ l_{m+1}, \ldots, \ \textit{not}\ l_n.$$

where each l_i ($0 \leq i \leq n$) is a literal. A literal can be atoms or arithmetic constraints or their negations. Each variable or constant belongs to a unique sort, either regular (logical) or constraint (arithmetic). $\mathcal{AC}(\mathcal{C})$ also introduces mixed predicate, whose parameters are from both regular and constraint sorts.

The *not* is called default negation or negation as failure, which is a "keyword" from ASP (Gelfond 2008). *not* l means that there is no reason to believe that literal l is true.

Planning Problems in PDDL2.1

For introducing our PDDL2.1 to $\mathcal{AC}(\mathcal{C})$ translation later, we give an informal version of PDDL2.1 here.

A planning problem consists of a set A of actions, the initial condition $Init$, and a goal requirements $Goal$. $Init$ is a set of propositional letters and arithmetic constraints. $Goal$ is a *formula*, consisting of propositional letters and arithmetic constraints connected by logic connectives negation ($\neg$), conjunction ($\wedge$) and disjunction ($\vee$). Arithmetic constraints are of the form $exp1 \circ exp2$ where $\circ \in \{=, \geq, \leq\}$ and $exp1$ and $exp2$ are arithmetic expressions.

An action can be durative or simple (i.e., instant). A simple action has *conditions* and *effects*. The conditions of a simple action is a formula, while effects is a set of literals and assignments to variables. An assignment is of the form $lv \diamond exp$ where lv is a variable, $\diamond \in \{:=, \texttt{+=}, \texttt{-=}, \texttt{*=}, \texttt{/=}\}$ and exp is an arithmetic expression.

A durative action has conditions at the start and/or end of the action, effects at the start and/or end of the action, and sometimes invariants and continuous effects. Invariants are formulas that must hold over the duration. Continuous effects are of the form $lv \diamond exp \cdot \Delta t$ where $\diamond \in \{\texttt{+=}, \texttt{-=}\}$ and Δt is a real number representing the elapse of an action.

A plan for a planning problem is a set of action-time pair in the form (t, a) or $(t, a[d])$, where a is an action, t is a real

number for time, and d is a rational value specifying the duration if the action is durative. A plan is valid if each action's conditions (including invariants) are satisfied, no mutex actions happen simultaneously, and the $Goal$ is satisfied after the last action.

Translation from PDDL2.1 to $\mathcal{AC}(\mathcal{C})$

In this section, we discuss the key ideas in translating a PDDL2.1 program into an $\mathcal{AC}(\mathcal{C})$ one. A concept *step* is used to track the execution of actions in a plan. For every time instant that an action occurs, there are two consecutive steps, one even and one add, associated to it. The even step is for action happening while the odd step is for invariant checking. Note that though the concept step is used, time is not discretized. Mixed literal $at(S, T)$ is true if step S happens at time T.

For every propositional letter p involved in a planning problem, we use the literal $p(S)$ to represent its truth value in step S. We index all variables [1] and denote the i-th variable invovled in a planning problem as x_i. Mixed literal $m(i, S, X_i)$ is true if the value of x_i is $X_i \in \mathbb{R}$ in step S.

As mentioned earlier, $Init$ is a set of propositional letters and arithmetic constraints. For every propositional letter p in $Init$, we introduce the fact rule $p(0)$. We also have the closed-world assumption rule $\neg p(0) \leftarrow$ *not* $p(0)$.

For every arithmetic constraint $exp1 \circ exp2$ in $Init$, we introduce the rule

$$\leftarrow \neg exp1' \circ exp2', m(c_1, 0, X_{c_1}), \ldots, m(c_k, 0, X_{c_k}).$$

where $exp1'$ (and similarly $exp2'$) is the result of substituing every variable x_{c_i} in $exp1'$ by its value X_{c_i}.

$Goal$ and conditions of simple actions are formulas. To translate formulas, for every non-literal subformula, we introduce a new propositional letter, and use a set of rules to represent the logical connectives. For example, the formula $f = p \wedge (x_2 - x_1 > 3)$ is translated into (S is a step):

$p'(S) \leftarrow p(S).$
$p''(S) \leftarrow m(1, S, X_1), m(2, S, X_2), X_2 - X_1 > 3.$
$f(S) \leftarrow p'(S), p''(S).$

At any step S, a simple action a cannot occur if its condition c, a formula, is not satisfied:

$$\leftarrow occur(a, S),\ \textit{not}\ c(S). \tag{1}$$

where $c(S)$ is a new atom defined by a set of rules for translating c as we just discussed.

For every literal l in the effects of a simple action a, we have $l(S + 1) \leftarrow occurs(a, S)$.

For a variable assignment $x_i \diamond exp$ in the effects of a simple action a, we consider the case that a is the only action that changes x_i's value and $\diamond$ is += as an example. A new mixed literal $contribution(i, a, S, Y)$ is introduced and it holds if the contribution from action a to the variabel x_i at step S is Y. The rules below ensure that Y equals to exp at step S if action a occurs at step S, and Y is 0 otherwise:
$\leftarrow contribution(i, a, S, Y), occurs(a, S), Y \neq exp'.$
$\leftarrow contribution(i, a, S, Y),$ *not* $occurs(a, S), Y \neq 0.$
where exp' is defined as before.

The value of x_i is updated by:

$$\leftarrow m(i, S + 1, Z), m(i, S, X_i), \\ contribution(i, a, S, Y), Z \neq X_i + Y.$$

All literals and variable values follow the inertial law.

$$p(S + 1) \leftarrow p(S),\ \textit{not}\ \neg p(S + 1).$$
$$\neg p(S + 1) \leftarrow \neg p(S),\ \textit{not}\ p(S + 1).$$
$$\leftarrow m(I, S + 1, Y), m(I, S, Z), Y \neq Z.$$

A durative action can be replaced by two simple actions representing its start and end. These two simple actions can be translated using methods we just discussed.

For invariants, we require them to hold at all steps over the duration of an durative action similarly to Eq. 1. If a durative action has continous effects, we also require the invariants to hold between all step pairs over its duration.

For continuous effects, we introduce a new predicate and use the method similiar to how we handle variable assignments for simple actions earlier.

In plan generation, every simple action (including those induced from durative actions) can occur or not in each even step. Non-zero-separation for mutex actions is also enforced.

Conclusion

In this paper, we show key ideas in using $\mathcal{AC}(\mathcal{C})$ to solve temporally expressive planning problems represented in PDDL2.1. For PDDL extended with axioms, since PDDL axiom is simply logic programs, our translation has the ability to support PDDL axioms as well.

Acknowledgment

Yuanlin Zhang's contribution in this work was partially supported by NSF grant IIS-1018031. Forrest S. Bao's contribution in this work was partially supported by ARMWorks, Inc. (Seattle, WA) via Texas Tech University.

References

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2009. Temporal planning in domains with linear processes. In *IJCAI*, 1671–1676.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *ICAPS*, 42–49.

Cushing, W.; Kambhampati, S.; Weld, M.; and Weld, D. 2007. When is temporal planning really temporal? In *Proceedings of the 20th international joint conference on Artifical intelligence (IJCAI)*, 1852–1859.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)* 20:61–124.

Gelfond, M. 2008. *Answer sets*. Handbook of Knowledge Representation. Elsevier. 285–316.

Hu, Y. 2007. Temporally-expressive planning as constraint satisfaction problems. In *Proceedings of 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 192–199.

Huang, R.; Chen, Y.; and Zhang, W. 2009. An optimal temporally expressive planner: Initial results and application to P2P network optimization. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 178–185.

Mellarkod, V. S.; Gelfond, M.; and Zhang, Y. 2008. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence* 53(1-4):251–287.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artificial Intelligence* 168(1):38–69.

[1] called Primitive Numerical Expression in (Fox and Long 2003)