$\tau\varepsilon$2asp : Implementing $\mathcal{TE}$ via Answer Set Programming

Hai Wan[1], Yu Ma[2], Zhanhao Xiao[2], and Yuping Shen[3]

[1] Software School, Sun Yat-Sen University, Guangzhou, China, 510275
[2] School of Information Science and Technology, Sun Yat-Sen University, Guangzhou, China, 510275
[3] Institution of Logic and Cognition, Sun Yat-Sen University, Guangzhou, China, 510275

Abstract. This paper studies computational issues related to the problem of reasoning about action and change in timed domains by translating it into answer set programming paradigm. Based on this idea, we implement a new action and change reasoning solver - $\tau\varepsilon$2asp with a polynomial translation without increasing its complexity for checking satisfiability and entailment in $\mathcal{TE}$ and report some experimental results.

1 Introduction

In order to extend the study of reasoning about action and change [1–3] to handle *timed domains*, the so-called timed action language $\mathcal{A}_{\mathcal{T}}$ [4], narrative-based action logic AL^2_{TC} [5], timed action language $\mathcal{TE}$ [6] are proposed. This paper focuses on providing a computational realization of solutions to reasoning about problems described by $\mathcal{TE}$, regarded as an extension of action language $\mathcal{E}$ [7]. Reasoning about action and change in *timed domains*, is not only considered as toy problems, e.g., *timed Yale Shooting*, but also studied as real-world applications, e.g., *timed automaton*[8], *assembly plant*[8], and *rail road crossing control*[9]. Compared with $\mathcal{A}_{\mathcal{T}}$ and AL^2_{TC}, $\mathcal{TE}$ overcomes semantics defect of $\mathcal{A}_{\mathcal{T}}$ and can be implemented by *satisfiability modulo theory*(SMT) [10] solvers.

This paper studies a link between timed action language $\mathcal{TE}$ and declarative logic programming approach of *answer set programming* (ASP) [11], intending to implement reasoning about action and change in *timed domains* by translating $\mathcal{TE}$ into ASP [12], a promising approach implemented by a number of sophisticated solvers [13–16]. A study on encodings of reasoning problems in language $\mathcal{E}$ was developed by exploiting the relation between $\mathcal{E}$ and ASP[7].

This translation is not only theoretically interesting but also of practical relevance. Based on the translation, we implement a new action and change reasoning solver, called $\tau\varepsilon$2asp. We report some experimental results, which demonstrate that the performance of $\tau\varepsilon$2asp is rather satisfactory.

The paper is organized as follows. Section 2 recalls some basic notions and definitions in timed action language $\mathcal{TE}$. Section 3 presents the translation from $\mathcal{TE}$ to answer set programming. Section 4 explains the implementation of $\tau\varepsilon$2asp and reports some experiments. Finally, Section 5 concludes the paper.

P. Anthony, M. Ishizuka, and D. Lukose (Eds.): PRICAI 2012, LNAI 7458, pp. 820–825, 2012.

2 Preliminaries

We consider Shen's *timed action language* $\mathcal{TE}$[6], in which a $\mathcal{TE}$ theory should be composed of a 5-tuple and propositions in $\mathcal{TE}$ are defined as follows:

- C-proposition is either of the form: A **initiates** F **resets** λ **if** C **when** Ψ, or A **terminates** F **resets** λ **if** C **when** Ψ, means action A can make fluent F hold (or not hold) and reset a set of clocks C;
- H-proposition: A **happens-at** T means action A happens at time T;
- T-proposition: L **holds-at** T means fluent L does (or doesn't) hold at T.

C-propositions introduce a new kind of action effect: clock resetting, and action precondition: clock constraint. Resetting a clock is to start counting its ticks and execute an action satisfying related clock constraints besides fluent preconditions. H- and t-propositions remain the same w.r.t. $\mathcal{E}$ [17]. Model of $\mathcal{TE}$ is defined by Definition 2 in [6]. The following is an example [4–6]:

Example 1. **Timed Yale Shooting in $\mathcal{TE}$**
Let $\mathcal{TE} = \langle \mathbf{N}, \{Load, Shoot\}, \{Loaded, Alive\}, \{x\}, \mathcal{B}(\{x\})\rangle$,a $\mathcal{TE}$ theory D_{TYS} of timed Yale Shooting consists of the following propositions:

- *Shoot* **terminates** *Alive* **if** $\{Loaded\}$ **when** $\{x < 5\}$
- *Load* **initiates** *Loaded* **resets** $\{x\}$
- *Alive* **holds-at** 0, $\neg$*Loaded* **holds-at** 0
- *Load* **happens-at** 1, *Shoot* **happens-at** 3

Based on [6], we implement the translation from $\mathcal{TE}$ to SMT and can reason by using SMT solver $Z3$ [1], which tops solvers of kind.

3 From $\mathcal{TE}$ to Answer Set Programming

3.1 Translation Algorithm

In order to construct translation from $\mathcal{TE}$ to ASP, we define a $\mathcal{TE}$ theory D with five sets, $\alpha, \beta, Hp, Tp, Cp$, and a N-time sequence, where α and β is a set composed of all fluent and clocks in D respectively and Hp, Tp, Cp is a set composed of all h-, t-, or c-propositions in D respectively.

The translation from $\mathcal{TE}$ to ASP can be accomplished by algorithm 1. We can get the facts corresponding to this ASP program, named as Δ. To specify the time sequence, a series of propositions denoting time(*i.e.* $time(0), time(1), \ldots,$ $time(N-1)$) should be taken into Δ. Note that we add $time(N-1)$ except $time(N)$ into Δ to terminate the reasoning at time N. For every fluent F in D would be expressed in the form of proposition like "$fluent(F)$." in Δ_α. Similarly, for every clock C in D would be translated to "$clock(C)$." in Δ_β.

Assuming that h-propositions $Hp_1, Hp_2, ..., Hp_p$ comprise of set Hp and an h-proposition Hp_i is in the form like "a_i **happens-at** t_i". When translated into

[1] $Z3$-3.2 at `http://research.microsoft.com/en-us/um/redmond/projects/z3/`

Algorithm 1. Translating $\mathcal{TE}$ to answer set programming

input : A $\mathcal{TE}$ theory $D = \{\alpha, \beta, Hp, Tp, Cp\}$ with a N-time sequence
output: An ASP program Δ

$\Delta \leftarrow time(0..N-1).$
forall the $1 \leq i \leq l$ **do** // fluent
  $\Delta_\alpha \leftarrow \Delta_\alpha \cup \{fluent(\alpha_i).\}$
forall the $1 \leq i \leq m$ **do** // clocks
  $\Delta_\beta \leftarrow \Delta_\beta \cup \{clock(\beta_i).\}$
forall the $1 \leq i \leq p$ **do** // H-propositions
  $\Delta_H \leftarrow \Delta_H \cup \{a_i(t_i).\}$
forall the $1 \leq i \leq q$ **do** // T-propositions
  if $f_i = \alpha_j$ **then**
    $\Delta_T \leftarrow \Delta_T \cup \{fluent(f_i, t_i).\}$
  if $f_i = \neg\alpha_j$ **then**
    $\Delta_T \leftarrow \Delta_T \cup \{-fluent(f_i, t_i).\}$
forall the $1 \leq i \leq r$ **do** // C-propositions
  if *initiates* fl_i **then**
    $\Delta_C \leftarrow \Delta_C \cup \{ac_i(T), Literal(C_i), Number(\Psi_i) \to ini(fl_i, T+1).\}$
  if *terminates* fl_i **then**
    $\Delta_C \leftarrow \Delta_C \cup \{ac_i(T), Literal(C_i), Number(\Psi_i) \to tmn(fl_i, T+1).\}$
  if *resets* λ_i **then**
    forall the $1 \leq j \leq s$ **do**
      $\Delta_C \leftarrow \Delta_C \cup \{ac_i(T), Literal(C_i), Number(\Psi_i) \to rst(\lambda_{ij}, T).\}$
return $\Delta \leftarrow \Delta \cup \Delta_\alpha \cup \Delta_\beta \cup \Delta_H \cup \Delta_T \cup \Delta_C$

ASP, Hp_i is switched to a proposition of "$a_i(t_i)$." standing for an action occurrence a_i at time t_i. A t-proposition like "F **holds-at** T" means that the positive literal of F at time T is true, while "$fluent(F, T)$." should be included by the translated ASP program. Correspondingly, "$\neg F$ **holds-at** T" means the negative one is true and "$-fluent(F, T)$" should be included. Like set Hp, set Tp is made up of t-propositions $Tp_1, Tp_2, ..., Tp_q$ and a t-proposition Tp_i is presented as "f_i **holds-at** t_i". The translation can generate according to the proposition with the different literal of f_i, positive and negative literal. In order for translation, we introduce two binary functions in syntax: *Literal* and *Number*. *Literal* translates a set of fluent literals into a series of atoms and *Number* translates a set of temporal constraints into atoms of body in Δ_C.

3.2 Reasoning Rules

Besides Δ obtained from Algorithm 1, we need 6 reasoning rules, denoted by Γ. Because rules in Γ construct the basic rules of reasoning about $\mathcal{TE}$ via ASP, Γ can't change with the change of $\mathcal{TE}$ theory.

Rule 1: generating initial complete knowledge.

$$1\{fluent(F, 0), -fluent(F, 0)\}1 \leftarrow fluent(F). \tag{1}$$

Rule 2: specifying initial clock value.

$$clock(C, 0, 0) \leftarrow clock(C). \tag{2}$$

Rule 3: eliminating inconsistent fluent literals.

$$\leftarrow fluent(F,T), -fluent(F,T). \tag{3}$$

Rule 4: specifying fluent persistent.

$$fluent(F,T+1) \leftarrow fluent(F,T), -tmn(F,T+1), time(T). \tag{4}$$

$$-fluent(F,T+1) \leftarrow -fluent(F,T), -ini(F,T+1), time(T). \tag{5}$$

$$-ini(F,T+1) \leftarrow not\ ini(F,T+1), time(T), fluent(F). \tag{6}$$

Rule 5: describing initiation and termination effects.

$$fluent(F,T+1) \leftarrow ini(F,T+1). \tag{7}$$

$$-fluent(F,T+1) \leftarrow tmn(F,T+1). \tag{8}$$

Rule 6: describing resetting effects.

$$clock(C,XX,T+1) \leftarrow XX := X+1, clock(C,X,T), -rst(C,T), time(T). \tag{9}$$

$$clock(C,1,T+1) \leftarrow rst(F,T), time(T). \tag{10}$$

3.3 Temporal Constraints

In addition, an ASP program of rules for defining temporal constraints is necessary, which is called by Λ.

$$eq(C,n,T) \leftarrow X == n, clock(C,X,T). \tag{11}$$

$$eq(C1,C2,n,T) \leftarrow X1 - X2 == n, clock(C1,X1,T), clock(C2,X2,T). \tag{12}$$

Because temporal constraints are different with the change of theory, Λ change accordingly. To reduce the reasoning space, all temporal constraints in theory D are substituted into Λ and those not in D should be removed from Λ.

So far the translation from a $\mathcal{TE}$ theory to ASP is accomplished and we can obtain ASP program Σ consisting of Δ,Λ,Γ. Observe that Δ, Λ and Δ can be translated in polynomial time, so the process is polynomial.

Theorem 1. *Let D be a $\mathcal{TE}$ theory whose time is complete and Δ be its translated program by Algorithm 1, Λ be its program of temporal constraints rules, Γ be the ASP program of reasoning rules and an ASP program Σ consisting of Δ,Λ,Γ. Then D has a model $< H, K >$ if and only if Σ has an answer set M s.t. every fluent F, every clock C and time T in D,*

1. *$H(F,T) = \top$ if and only if $fluent(F,T) \in M$;*
2. *$H(F,T) = \bot$ if and only if $-fluent(F,T) \in M$;*
3. *$K(T)(x) = x_T$ if and only if $clock(x, x_T, T) \in M$.*

4 Implementation and Application

4.1 Implementation

A new solver is implemented for $\mathcal{TE}$, called $\tau\varepsilon$2asp (Fig.1). An input $\mathcal{TE}$ theory is firstly translated to an ASP program by the *translator* in $\tau\varepsilon$2asp. Then, an ASP solver *Clasp*[2] is called to compute answer sets, which will be interpreted to the original $\mathcal{TE}$ theory by a *convertor*. $\tau\varepsilon$2asp is written in C, running on a machine with 2 processors($Intel(R)\ Core(TM)2\ Duo\ CPU\ T7600$) under Ubuntu 10.04.

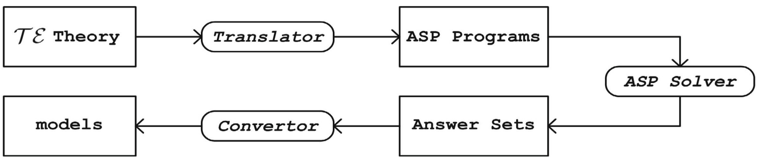

Fig. 1. Outline of $\tau\varepsilon$2asp

4.2 Experimental Results

Table 1 presents some experimental results of $\tau\varepsilon$2asp and compare it with SMT solver $Z3$ and timed automaton tool *HyTech*[3], which record computing time of series timed Yale Shooting problem in seconds, taking the average of 50 runs.

Table 1. Experimental results of timed Yale Shooting problem

Load	*Shoot*	*Bound*	$\tau\varepsilon$2asp	*Z3*	*HyTech*
1	3	8	0.00538	0.03348	0.02668
5	10	15	0.00716	0.04372	0.02142
20	24	30	0.00822	0.05134	0.02822
60	65	80	0.01590	0.11776	0.02108
120	123	150	0.03472	0.23290	0.02732
250	255	300	0.09556	3.85892	0.02076

Because *HyTech* solver can only find traces, a comparator is necessary to confirm whether the trace is consistent with the action sequence. Therefore the data in column *HyTech* in Table 1 is the time spent in finding traces to each state, omitting comparison time which is not a polynomial. Because of different computation mechanism, *HyTech* has little change with the increasing scale of the problem, while time via $\tau\varepsilon$2asp and SMT solver $Z3$ increases radically with the expansion of problem's scale. To sum up, considering the comparison time of Hytech, $\tau\varepsilon$2asp solver is the best.

5 Conclusions and Future Work

This paper contributes the study of reasoning about action and change in timed domains $\mathcal{TE}$. Theoretically, $\mathcal{TE}$ can be translated into ASP via a polynomial algorithm and a series of rules without increasing its computation complexity.

[2] Clasp-2.0.5 at `http://www.cs.uni-potsdam.de/clasp/`

[3] *HyTech*-1.0.4 at `http://embedded.eecs.berkeley.edu/research/hytech/`

Practically, a new solver $\tau\varepsilon$2asp is developed. In addition, a series of experiments about the comparison among $\tau\varepsilon$2asp, SMT solver ($Z3$) and a timed automaton tool (*HyTech*) have been done. It is a pleasure that $\tau\varepsilon$2asp performs much better than $Z3$ and *HyTech*. For future work, it is an important task to extend $\mathcal{TE}$ with ramification and qualification and improve $\tau\varepsilon$2asp. Last but not least, to explore more applications of $\mathcal{TE}$ by using $\tau\varepsilon$2asp is a job of significance.

Acknowledgments. This research has been partially supported by *doctoral program foundation of institutions of higher education of China* under Grant 20110171120041 and under Grant GD10YZX03, WYM10114.

References

1. Shoham, Y.: Reasoning about action and change. MIT Press (1987)
2. Harmelen, F.V., Lifschitz, V., Porter, B.: Handbook of knowledge representation. Elsevier Science (2008)
3. Reiter, R.: Knowledge in action: logical foundations for specifying and implementing dynamical systems. MIT Press (2001)
4. Simon, L., Mallya, A., Gupta, G.: Design and Implementation of A_T: a Real-Time Action Description Language. In: Hill, P.M. (ed.) LOPSTR 2005. LNCS, vol. 3901, pp. 44–60. Springer, Heidelberg (2006)
5. Cabalar, P., Otero, R.P., Pose, S.G.: Temporal constraint networks in action. In: ECAI, pp. 543–547 (2000)
6. Shen, Y.P., Dang, G.R., Zhao, X.S.: Reasoning about action and change in timed domains. In: NMR (2010)
7. Dimopoulos, Y., Kakas, A.C., Michael, L.: Reasoning About Actions and Change in Answer Set Programming. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS (LNAI), vol. 2923, pp. 61–73. Springer, Heidelberg (2003)
8. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press (2000)
9. Alur, R.: Timed Automata. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 8–22. Springer, Heidelberg (1999)
10. Nieuwenhuis, R.: SAT Modulo Theories: Enhancing SAT with Special-Purpose Algorithms. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, p. 1. Springer, Heidelberg (2009)
11. Lifschitz, V.: Action languages, answer sets and planning. The Logic Programming Paradigm: a 25 Year Perspective 25, 357–373 (1999)
12. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP, pp. 1070–1080 (1988)
13. Gebser, M., Kaufmann, B., et al.: Conflict-driven answer set solving. In: IJCAI, pp. 386–392 (2007)
14. Niemelä, I., Simons, P.: Smodels Implementation of the Stable Model and Well-Founded Semantics for Normal Logic Programs. In: Fuhrbach, U., Dix, J., Nerode, A. (eds.) LPNMR 1997. LNCS, vol. 1265, pp. 421–430. Springer, Heidelberg (1997)
15. Leone, N., Pfeifer, G., et al.: The dlv system for knowledge representation and reasoning. ACM Trans. Comput. Log. 7(3), 499–562 (2006)
16. Lierler, Y., Maratea, M.: Cmodels-2: SAT-based Answer Set Solver Enhanced to Non-tight Programs. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS (LNAI), vol. 2923, pp. 346–350. Springer, Heidelberg (2003)
17. Kakas, A.C., Miller, R.: A simple declarative language for describing narratives with actions. Journal of Logic Programming 31(1-3), 157–200 (1997)