
TIMED TRANSITION AUTOMATA AS NUMERICAL PLANNING DOMAIN

Alfredo Milani, Silvia Suriani

Abstract: A general technique for transforming a timed finite state automaton into an equivalent automated planning domain based on a numerical parameter model is introduced. Timed transition automata have many applications in control systems and agents models; they are used to describe sequential processes, where actions are labelling by automaton transitions subject to temporal constraints. The language of timed words accepted by a timed automaton, the possible sequences of system or agent behaviour, can be described in term of an appropriate planning domain encapsulating the timed actions patterns and constraints. The time words recognition problem is then posed as a planning problem where the goal is to reach a final state by a sequence of actions, which corresponds to the timed symbols labeling the automaton transitions. The transformation is proved to be correct and complete and it is space/time linear on the automaton size. Experimental results shows that the performance of the planning domain obtained by transformation is scalable for real world applications. A major advantage of the planning based approach, beside of the solving the parsing problem, is to represent in a single automated reasoning framework problems of plan recognitions, plan synthesis and plan optimisation.

Keywords: Timed Transition Automata, Automated Planning, Domain

ACM Classification Keywords: F.1.1 Models of Computation I.2.8 Problem Solving, Control Methods, and Search

Conference: The paper is selected from Sixth International Conference on Information Research and Applications – i.Tech 2008, Varna, Bulgaria, June-July 2008

Introduction

Timed transition automata, introduced in [Alur and Dill, 1994], are an extension of finite state automata where the notion of time has been introduced. Transitions take place in specific instants of the time, they can subject to time constraints based on absolute time and/or clocks. Timed automata are very useful to describe the behaviour of systems where the transition or system activities are characterised by prevailing temporal aspects. Many applications of TTA have been developed for control systems and agent models [Ceri et al, 2005]

A main drawback of the automata based model is that they focus on a single aspect of the sequential process, i.e. the problem of recognizing a pattern of actions. On the other hand a planning [Blum and Furst, 1997] based approach to sequential process modeling would allow to manage general issues such as goal attainment problems (i.e. the problem of finding sequence of actions which reach a given state), optimization problems (sequences of actions which minimise/maximise some given cost function) in a single framework.

In the following paragraphs it will be shown how the timed transition automata model can be modeled in the framework of numerical parameters planning model, where more general planning and optimisation problems can be posed. Experimental results both for the timed word recognition problem and the general planning problems are also discussed.

Timed Automata

A *Timed Transition Automata* (TTA) [Alur and Dill, 1994] is a finite state machine which is able to recognise timed words, i.e. a sequence of pairs made by symbols over a given alphabet Σ and time values. The pairs in the sequence can be seen as a sequence of logs records, describing user events or system operations annotated with the time in which they occurred. In a TTA it is possible to constrain a certain action to be executed, i.e. a certain transition to occur, only when some time conditions are met. In the following an action-time pair is also referred to as a *token*.

Let us recall more formally some basic concepts related to *Timed Transition Automata*.

Def. Timed word. Given a finite alphabet Σ , a timed word on Σ , is a finite sequence of pairs or tokens $[(a_0, \tau_0) \dots (a_k, \tau_k)]$ where $a_i \in \Sigma^*$, $\tau_i \in \mathfrak{R}$ for $i \in [0, k]$ with $\tau_i \leq \tau_{i+1}$ $i \in [0, k-1]$

Def. Timed Language. A *timed language* over an alphabet Σ is a subset of timed words on Σ .

Def. Time Transition Automata. A *Timed Transition Automata (TTA)* is a tuple $(\Sigma, S, s_0, C, E, F)$ where Σ is finite alphabet, S is a finite set of state, $s_0 \in S$ is an initial state, C is a finite set of clocks, $F \subseteq S$ is a set of final acceptance states, $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ defines the transition table for the automata.

Each transition $e \in E$ is a 5-ple $e = \langle s, s', a, \Lambda, \delta \rangle$ representing a transition from state s to state s' on input symbol a which can occur at a certain time τ when clock constraint δ is verified by the current values of clocks; the transition also resets to 0 the subset $\Lambda \subseteq C$ of clocks.

Clocks are used to express more easily time constraints such as durations relative to sub patterns in the transition diagram. Clocks are usually initialised to 0 and they are updated as time advances.

Given a set X of clocks, the set of clock constraints $\Phi(X)$ includes all the simple constraints conjunctions and negations defined by $\delta := x \leq c \mid c \leq x \mid \neg \delta \mid \delta \wedge \delta$ where $x \in X$ is a clock and c is a rational constant.

Def. Run of Timed Transition Automata. A *run* of a timed transition automata records a sequence of legal state transitions and the value of all the clocks when state transitions take place, starting from the initial state s_0 .

Def. Timed Language. The language $L(A)$ accepted by an automaton $A = (\Sigma, S, s_0, C, E, F)$ is the set of all timed words which correspond to consistent runs of the automaton starting with the state s_0 and ending with a final state $s_f \in F$, i.e. a timed word $w = [(a_i, \tau_i)]$ with $i \in [0, k]$ is also $w \in L(A)$ if exists a run from s_0 with each transition $\langle s, s', a_i, \lambda, \delta \rangle$ taking place at time instant τ_i and the final transition being $\langle s_{f-1}, s_f, a_k, \lambda, \delta \rangle$ for a state $s_f \in F$.

A *domain automaton* can then be defined for representing the legal transitions or, equivalently, the legal sequences of actions which can occur in the system or the agent process to be modelled.

For example, an automaton can be used to describe and recognize the behaviour of a user of an e-learning platform. Assume for instance that the user can perform 7 main operations or activities: *login*, *lesson*, *quiz*, *assignment*, *chat*, *view*, *logout*, and some additional operations: *main menu* which allows to abandon an activity and go back to the main menu; *submit/abandon* which respectively allow to submit the answers of a quiz, or to abandon it without answering. The activities are not all available at the same time, but they are subject to time and precedence constraints. Possible user behaviours are represented by TTAs in the fig.1 and fig.2, with transition labelled by symbols *login*, *logout*, *main*, *clock*, *submit*, *chat* and *view* (dashed loops indicate the idle action, i.e. the action of remaining in the current state).

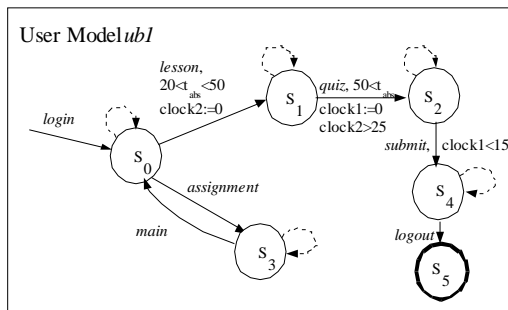


Fig.1 User behavior TTA model *Ub1*

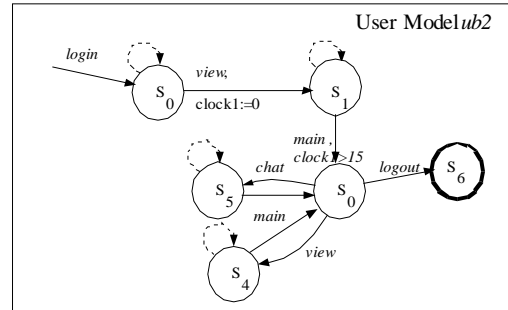


Fig.2 User behavior TTA model *Ub2*

User Model *ub1*. In model *ub1* the user, after entering the e-learning platform (*login* action), can repeat the *assignment* activity many times, but, in order to reach the final state S_5 , he has to attend the *lesson* until the end for at least 25 minutes ($\text{clock2} > 25$) and after that he has to *submit* the answer to the quiz.

User Model *ub2*. Behaviour model *ub2*, instead, describes a user which chooses *view* for at least 15 minutes as first activity, and then he/she can alternate *view/chat* without temporal constraints before *logout*.

Let consider, for example, the following timed words, where each element consists of a pair *time stamps* and *symbols*.

Seq1: [(login,0), (assignment,10), (main,12), (lesson,22), (main,23), (lesson,24), (quiz,51), (submit,65), (logout,70)]

Seq2 : [(login,0), (view,3), (main,19), (chat,20) (main,25), (29, chat,29), (main,35), (view,37), (main,40), (logout,41)]

Seq3: [(logon,0), (view,5), (main,30), (logout,32)]

It is easy to see that sequence *Seq1* is an example of user behaviour which is recognised by TTA *ub1*, while sequence *Seq2* and *Seq3* are recognized by *ub2*.

Numerical Parameters Planning Model

In the following we recall some basic notions about the numerical parameters planning model which is used to implement the TTA recognition process. The plan synthesis problem consists in finding a sequence of domain actions which, if executed, transforms a given initial state in a goals state. Planning systems have been widely used to model domain where one or more deliberative actors can modify the state of the world executing a set of predefined available actions. The numerical parameters extension enriches the classical Boolean planning model with the management of numerical resources and goals, moreover effects can depend on numerical continuous parameters of the action instance [Suriani, 2007]. The semantics of the model is based on three finite sets: B , N , and P , respectively representing *logical fluents*, *numerical fluents* and *numerical parameters*. Numerical fluents and numerical parameters are defined in bounded real interval domains.

Definition (State) A *state* is a pair of assignments $s=(s_B, s_N)$ where $s_B: B \rightarrow \{\text{true}, \text{false}\}$ assigns truth values to logical fluents, and $s_N: N \rightarrow \mathfrak{R}$ assigns real values to numerical fluents. S_B denotes the set of all possible logical assignments and S_N the set of all possible numerical assignments; finally S denotes the set of all possible states.

Definition (Operators) An operator is defined by a triple $o=(X, \pi, \varepsilon)$ where: $X \subseteq P$ are the numerical parameters of o , π are the preconditions of o and ε are its effects.

Preconditions π are conjunctions of *literals* (i.e. b or $\neg b$, where $b \in B$ is a logical fluent) and *numerical constraints* of the form $f_{N \cup X} \otimes 0$, where f is a linear function of numerical fluents/parameters and $\otimes \in \{<, \leq, =, \neq, \geq, >\}$. Effects ε are conjunctions of *literals* and *numerical effects* (i.e. assignments of numerical fluents of the form $u := g_{N \cup X}$ where $u \in N$, g is a linear function of numerical fluents/parameters). Let O denote the set of all operators.

Definition (Action Instance) An action instance is defined by a pair (o, σ) where $o=(X, \pi, \varepsilon)$ is an operator and σ a parameter assignment $\sigma: X \rightarrow \mathfrak{R}$. An action instance (o, σ) is said to be executable in a state $s=(s_B, s_N)$ if logical and numerical conditions hold in s and numerical effects are consistent with the domain bounds.

Definition (Action Execution) If an action instance (o, σ) is executable in a state $s=(s_B, s_N)$, the result of its execution is a state $s'=\gamma(s, (o, \sigma)) = (s'_B, s'_N)$, where:

- for each logical fluent $b \in B$ 1) $s'_B(b) = \text{true}$ if $b \in \varepsilon$; 2) $s'_B(b) = \text{false}$ if $\neg b \in \varepsilon$ 3) $s'_B(b) = s_B(b)$ otherwise
- for each numerical fluent $u \in N$ 1) $s'_N(u) = g_{N \cup X}(s_N(u))$ if $u := g_{N \cup X} \in \varepsilon$ 2) $s'_N(u) = s_N(u)$ otherwise.

Definition (Numerical Parameterized Planning Problem) A numerical parameterized planning problem is a tuple $\Sigma = (B, N, P, S, O, s_0, G)$ where B, N, P, S, O represent boolean fluents, numerical fluents, numerical parameters, states and operators, and

- $s_0 = (s_0^B, s_0^N)$ is the initial state;

- G is a conjunction of literals and numerical constraints defined over $B \cup N$ representing the goal.

Note that goals are defined over (B, N) , i.e. goals cannot contain any parameter symbols.

Definition (Solution Plan) A plan, i.e. a sequence of action instances $((o_0, \sigma_0) \dots, (o_k, \sigma_k))$, is a solution plan for a planning problem $\Sigma = (B, N, P, S, O, s_0, G)$ if the sequence is executable and the goal G holds in the final state.

The sequence of actions is executable when (o_0, σ_0) is executable in s_0 and each action instance (o_i, σ_i) is executable in $s_i = \gamma(s_{i-1}, (o_{i-1}, \sigma_{i-1}))$ for each $i = 1, \dots, k$. The goal G holds in the final state $s_{k+1} = \gamma(s_k, (o_k, \sigma_k))$, if $\forall g \in G$ when g is a literal $g=b$ ($g=-b$) then $s_B^k(b)=\text{true}$ ($s_B^k(b)=\text{false}$), or when g is a numerical constraint $f_{N \cup X}$ then $f_{N \cup X} \otimes 0$ holds in s_{k+1} .

Timed Transition Automata and Equivalent Planning Domain

Since automated planning models encode state transitions, the basic idea of our approach has been to use actions to encode TTA state transitions. The parsing process of a given TTA can be embedded by an appropriate planning domain, where each planning action corresponds to parse a transition in the TTA model, (i.e. corresponds to a legal TTA transition), timed words represent a plan to a final state in the equivalent planning domain.

The current state of TTA is simulated by asserting/negating appropriate fluents. Each planning action representing a TTA transition $\langle s, s', a_i, \lambda, \delta \rangle$ is executable only if the current simulated state is "s", and if the pair to be parsed is (a_i, τ) where time stamp τ verifies the time constraints δ . The planner can then be used to verify if a timed word, corresponds to a path from the initial TTA state to a final TTA state.

The Planning Domain Problem

Given the TTA $(\Sigma, S, s_0, C, E, F)$, and given a timed word $w = [(a_i, \tau)]$ it is possible to define a planning domain problem (B, N, P, O, s_0, G) for timed word recognition problem, where:

- $B = \{ \text{curr_state}(s_i), \text{final}(s_j), \text{success}, \text{curr_tk}(l_i), \text{next}(l_i, l_j), \text{tk}(l_i, a_i, t_i, d_i) \}$ is the set of logical fluents where s_i and l_i refer to TTA states and *Tokens*;
- $N = \{ t_{\text{abs}}, t_{\lambda} \}$ is the set of numerical fluents;
- $P = \{ t_{\theta} \}$ is the set of numerical parameters;
- $O = \{ A_{\langle s, s', a, \lambda, \delta \rangle}, A_{\langle s, s, a, \lambda, \delta \rangle}, \text{Idle}_S, A_f \} \forall s \in S, \forall f \in F, \forall \langle s, s', a, \lambda, \delta \rangle \in E$ is the set of the operators;
- $G = G_B \cup G_{\delta}$ with $G_B = \{ \text{success} \}$ is the set of literals defined over B and $G_{\delta} = \{ \}$ is the set of numerical constraints defined over N .

Each pair (a, t) symbol/time of the timed word is parsed to a 4-pla (l, a, t, d) , said *token*, where l is a sequential identifier, a is the symbol encoding the performed action, t is the time stamp of the starting time and d is the time interval between the action and the next one.

Fluents and TTA States

Given a TTA $(\Sigma, S, s_0, C, E, F)$ some *logical* and *numerical fluents* are introduced to represent states, tokens, current state, current token and tokens sequence, i.e. timed words.

Logical Fluents.

$\text{curr_state}(s_i) \forall s_i \in S$ represents the current state. Note that curr_state fluents are used to represent the situation in which the TTA is currently in the state s_i , the domain actions must guarantee that at most one $\text{curr_state}(s_i)$ can be true at the same time.

$\text{tk}(l_i, a_i, t_i, d_i) \forall (l_i, a_i, t_i, d_i) \in \text{Token}$ is introduced to represent the token information, l_i is the token id (i.e. sequential identifier), a_i the *action*, t_i the *time*, d_i is the *duration* i.e. the time before the next token.

$\text{curr_tk}(l_i) \forall (l_i, a_i, t_i, d_i) \in \text{Token}$ represents the current token; similarly to $\text{curr_state}(s_i)$, only one $\text{curr_tk}(l_i)$ can be true at the same time. The sequential order of the tokens is represented by the fluents $\text{next}(l_i, l_j)$, where l_i is the successor of l_j in the sequence. A special fluent $\text{curr_tk}(init)$ represents the initial situation when no tokens are have been parsed yet; conversely, a special fluent $\text{curr_tk}(end)$ is used to mark the end of the token sequence. Moreover two fluents $\text{next}(init, l_1)$ and $\text{next}(l_k, end)$ are also added accordingly.

A set of fluents $final_{s_j}$ for each final state $s_j \in F$ and a single logical fluent $success$ are also used to specify disjunctive goals.

Numerical Fluents. A numerical fluent t_{abs} is defined to represent the absolute time as it evolves while actions are executed. A numerical fluent t_c is also introduced for each clock $c \in C$.

Initial State

The initial state of the planning problem represents the initial state of the timed automaton and the value of the clocks and of the absolute time are initially set to 0.

$$curr_state(s_0) = T \quad curr_state(s_j) = \perp \quad \forall s_i \in S, i \neq 0 \quad s_i \text{ is false in } I \quad t_{abs} = 0, t_d = 0 \quad \forall d \in \delta$$

moreover it is also needed to represent the state of the parsing process:

$$curr_tk(init) = T \quad curr_tk(end) = \perp \quad curr_tk(l_i) = \perp \quad \forall (l_i, a_i, t_i, d_i) \in Token \quad final_{s_j} = T \quad \forall s_j \in F \quad success = \perp$$

the latter two are needed to indicate which are the final states and the fact that the parsing is not yet successful.

TTA transitions and tokens

Appropriate actions $A_{\langle s, s', a, \lambda, \delta \rangle}$, $A_{\langle s, s, a, \lambda, \delta \rangle}$, $Idle_{s_j}$ and A_{s_j} are introduced in the planning domain in order to represent respectively *transitions*, *self-referencing transitions*, *idle states* and the *final disjunctive goal*.

Transitions and Self-referencing Transitions

For each transition $e \in E$, $e = \langle s, s', a, \Lambda, \delta \rangle$ of the automata where $s \neq s'$, a planning operator denoted by $A_{\langle s, s', a, \lambda, \delta \rangle}$ or equivalently by A_e is introduced as follows,

$$Pre(A_e) = \{ curr_state(s) \wedge curr_tk(l_1) \wedge next(l_1, l_2) \wedge tk(l_1, a, t, d) \wedge \delta \wedge t_{abs} = t \}$$

$$NumPar(A_e) = \{ \}$$

$$Eff(A_e) = \{ \neg curr_state(s) \wedge \neg curr_tk(l_1) \wedge curr_state(s') \wedge curr_tk(l_2) \wedge (t_\lambda := 0, \forall \lambda \in \Lambda) (t_\lambda := t_\lambda + d, \forall s.t. \lambda \in C \text{ and } \lambda \notin \Lambda) \wedge (t_{abs} := t_{abs} + d) \}$$

where d is the duration of the action whilst l_1 and l_2 are sequential identifiers. The time constraints δ are numerical constraints on the numerical fluents corresponding to the clocks and/or the absolute time; the constraint $t_{abs} = t$ establishes that the transition in TTA takes place at the time t specified by the token.

As shown in figure 3, the basic idea is to constraint the introduced action operator A_e to behave equivalently to transition e :

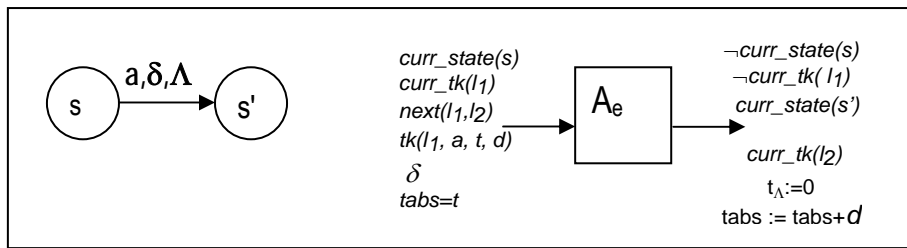


Fig.3 The transition $e = \langle s, s', \delta, \Lambda \rangle$ and the corresponding action A_e

Also note that the current state and the current token are updated accordingly to the state transition table and to the tokens order, while absolute time is updated with action duration d and clocks are either updated or reset to 0.

A special case is when a transition specifies the same starting and target state, i.e. the corresponding node in the automaton graph contains a self reference loop.

For each transition $e \in E$ of type $e = \langle s, s, a, \Lambda, \delta \rangle$ it is introduced an action $A_{\langle s, s, a, \Lambda, \delta \rangle}$ whose definition differs from the previous one only in the effects, i.e. the negation of current state and the update to the new state, $\neg curr_state(s) \wedge curr_state(s)$, are omitted from the action effects since they would lead to inconsistency.

Note that the execution of an action of type $A_{\langle s, s', a, \lambda, \delta \rangle}$ or $A_{\langle s, s, a, \lambda, \delta \rangle}$ corresponds to parse a token record as required by the precondition $tk(l, a, t, d)$.

Parsing starts from the only one action executable in the initial state, where $curr_state(init)$ is true, and it follows the order encoded by the *next* predicates.

Idling state.

If the TTA model admits idling in a state, i.e. remaining in a state while performing no action, then a special *idle operator* $Idle_{S_i}$ is added for each state $S_i \in S$ of the TTA in order to model the time flow. The possibility of being idle allows to have gaps in the logs temporal sequence. The idle operators have a quite simple structure since in order to be executed, they do not require either tokens to exist, or time/clock constraints to be verified. On the other hand idle operators contain an additional *numerical parameter* t_e which represents the elapsed time

$$Pre(Idle_{S_i}) = \{ curr_state(S_i) \}$$

$$NumPar(Idle_{S_i}) = \{ t_e \}$$

$$Eff(Idle_{S_i}) = \{ (t_{abs} := t_{abs} + t_e) \wedge ((t_\lambda := t_\lambda + t_e, \forall \lambda \in C)) \}$$

Note that the numerical parameter t_e represents the idling interval and it is used to update the absolute time as well as all the clocks. Numerical parameters are values which are chosen by the planner in order to instantiate the action instance.

TTA Final States & Planning Goals

The TTA recognizes a timed word when it reaches one of the possible final states after parsing all the tokens. These conditions are specified by a disjunctive goal: $curr_tk(end) \wedge (\vee curr_state(S_i) \forall S_i \in F)$

A well known technique [Nebel, 2000] has been used to specify disjunctive goals in a conjunctive planner, a set of dummy actions representing the disjunctive goal is introduced as following:

- for each final state, $\forall S_i \in F$, a dummy operator A_{S_i} is added to the set of domain operator O such that:

$$Pre(A_{S_i}) = \{ curr_state(S_i), final(S_i), curr_tk(end) \}, \quad Eff(A_{S_i}) = \{ success \}$$

where *success* is a logical fluent representing the end of the user behaviour recognition process.

The fluent *success* will represent the problem goal. *Success* is true in a state when at least one of the possible action A_{S_i} with $S_i \in F$ has been executed, i.e. a final state has been reached (see preconditions $curr_state(S_i)$, $final(S_i)$) when parsing the last token (precondition $curr_tk(end)$).

Experiments

The TTA planning rules described in the previous paragraph show that the transformation space complexity is linear in the size of the planning domain. On the other hand it is not possible to provide a theoretical estimate for plan synthesis time, since it strongly depends on the planner implementation which can employ very efficient strategies especially for the logical fluents. In order to obtain a general estimate of the effectiveness of the approach we have held systematic experimental tests using PNP (Parametric Numerical Planner), the tests are based on *ub1* and *ub2* domains.

PNP has been implemented in C language and performs the graph construction phase and the encoding phase, while the solution of the MILP system is performed by using ILOG CPLEX. The Numerical Parameter Planning model has been implemented using a technique of mixed integer linear programming (MIP) encodings [Wolfman and Weld, 1999]. The algorithm built a planning graph [Kautz and Selman, 1998] with logical fluents and operators ignoring the numerical aspects of the problem, then, the planning graph is encoded as a MIP extended to handle numerical fluents and parameterized actions [Vossen et al., 2001, Van de Briel et al., 2005]. A standard MIP solver, ILOG CPLEX is then used to solve the planning problem. The tests have been executed on Intel Pentium IV 3.00GHz with 1GB of RAM running the operating system Linux.

The tests has been divided into three classes: positive and negative cases for user behaviour recognition, and planning problems in e-learning domain. Negative cases has been tested for different causes of recognition failure: a) logical failure i.e. action sequences not allowed by the TTA describing the user behaviour and b) numerical failures, action time stamps which violates the numerical time constraint of the TTA. The scalability of the approach has been tested with different users histories, i.e. log sequences of increasing length.

Finally an e-learning planning domain has been modelled to verify the flexibility and expressivity; since the problem does not require to parse any tokens, the fluents of type *tk*, *curr_tk*, and *next* have been removed from the action descriptions, dummy actions and goals.

Tokens	Ub1			Ub2		
	Time	Nodes	Var	Time	Nodes	Var
5	0,03	32	174	0,03	34	165
9	0,04	47	345	0,04	49	364
21	0,1	83	1083	0,09	85	1234
33	0,28	119	2253	0,2	121	2536
41	0,49	143	3273	0,34	145	3644
53	1,09	179	5163	0,63	181	5666
61	1,92	203	6663	1,01	205	7254
73	3,56	239	9273	1,74	241	9996
81	5,03	263	11253	2,26	265	12064
93	6,98	299	14583	3,49	301	15526
101	8,23	322	16711	4,55	325	18074

Table 1. Positive Recognition Test for *Ub1*

Tokens	Ub1		Ub1	
	Log	Num	Log	Num
5	0,03	0,06	0,03	0,09
9	0,03	0,06	0,03	0,09
21	0,03	0,07	0,04	0,11
33	0,06	0,12	0,06	0,18
41	0,09	0,19	0,10	0,29
53	0,18	0,38	0,20	0,58
61	0,26	0,54	0,28	0,82
73	0,47	1,00	0,53	1,53
81	0,68	1,43	0,75	2,18
93	1,11	2,30	1,19	3,49
101	1,50	3,09	1,59	4,68

Table 2. Negative Recognition Test for *Ub1*

The results obtained are completely satisfactory for the three classes of tests. In particular positive user behaviour recognition is quite efficient to be used in real time applications, since the top sequence size of 97 log records are fairly more than the typical user sessions, which consist of less than ten actions, the time performance for twenty actions is worst case not greater than 0.1 seconds. Negative tests on user behaviour recognition were even more efficient than positive tests, in particular it must be noted that negative test of type a), i.e. where the sequence violates a logical constraint, can be detected very efficiently in the early plangraph construction phase, and the error detection time is proportional to the length of the correct prefix. Negative tests of type b), i.e. where the timed actions violate the numerical constraints, require the execution of both phases of plangraph construction and LP solving; these tests show an execution time which is slightly minor than the correspondent positive test.

The last class of tests, i.e. e-learning planning problems, is not plotted since the time results are all extremely fast, always below 0.04 seconds for all the posed problems. It would be interesting to investigate in a future extension a task planning approach similar to [Baiocchi et al., 1997] where task goals and logical goals can be mixed.

Conclusion

A general method for transforming a *Timed Transition Automata* (TTA) into an equivalent planning domain has been introduced. The main idea of the proposed approach is to build a planning domain model to encode the state transitions of TTA representing behaviours, where each planning action corresponds to parse a time-symbol token, and the sequence of tokens in the time word is given as initial state. The timed word recognition problem is then transformed into the planning problem of finding a parsing plan for the sequence of tokens. The formal TTA to plan transformation is proved to be correct, and it is built in the framework of a numerical parameters planning model, which extends the classical boolean planning models with the management of numerical resources and goals and effects can depend on numerical continuous parameters of the action instance.

One of the relevant advantage in using a planning approach to user task modeling is that behaviour recognition, reachability and plan optimisation problems can be modeled in a unique framework.

Systematic experiments with PNP, a general purposes parametric numerical planner implementation, show that the approach is effective and scalable for real world application such as user behaviour recognition.

Future works will regard the development of special purpose plan search techniques targeted on the timed word parsing problem, and extending the proposed model with task constraints [Baiocchi et al., 1998].

Bibliography

- [Alur and Dill, 1994] R. Alur, D. Dill, "A theory of timed automata", *Theoretical Computer Science*, vol. 126, num. 2, p. 183-235, 1994.
- [Berendt et al., 2000] B. Berendt, M. Spiliopoulou, "Analysis of navigation behaviour in web sites integrating multiple information systems", *The VLDB Journal*, 9, Springer-Verlag, 2000, pp. 56–75.
- [Ceri et al, 2005] S. Ceri, F. Daniel, V. Demaldé, F. M. Facca, "An Approach to User-Behavior-Aware Web Applications", ICWE 5 Proceedings, Sydney, Australia, Springer, 2005
- [Masseglia et al., 2005] F. Masseglia, P. Poncelet, M. Teisseire, A. Marascu, "Web Usage Mining: Extracting Unexpected Periods from Web Logs", *TDM 2 - ICDM'05 Proceedings*, Houston, USA, 2005.
- [Mühlenbrock, 2005] M. Mühlenbrock, "Automatic Action Analysis in an Interactive Learning Environment", *AIED-2005 Proceedings*, Amsterdam, NL, pp. 73-80.
- [Teltzrow et al., 2003] M. Teltzrow, B. Berendt, "Web-Usage-Based Success Metrics for Multi-Channel Businesses", *WebKDD 2003 9th ACM SIGKDD Proceedings*, Washington DC, USA, 2003.
- [Baiocchi et al., 1998] Marco Baiocchi, Stefano Marcugini, Alfredo Milani: Encoding Planning Constraints into Partial Order Planners. *KR98 Proceeding, 6th Int. Conf. on Principles of Knowledge Representation and Reasoning*, pp.608-616, Morgan Kaufmann 1998, ISBN 1-55860-554-1
- [Baiocchi et al., 1997] Marco Baiocchi Stefano Marcugini, Alfredo Milani: Task Planning and Partial Order Planning: A Domain Transformation Approach. in *Lecture Notes in Computer Science, Vol.1348*, pp.52-63, Springer-Verlag, Berlin, Germany, 1997, ISBN 3-540-64912-8
- [Nebel, 2000] Bernhard Nebel: On the Compilability and Expressive Power of Propositional Planning Formalisms. *J. Artif. Intell. Res. (JAIR)* 12: 271-315 (2000)
- [Blum and Furst, 1997] Blum, A., and Furst, M. Fast planning graph analysis. *Artificial Intelligence* 90, 1-2 (1997), 279-298
- [Kautz and Selman, 1992] Kautz, H., and Selman, B. Planning as satisfiability. In *10th European Conference on Artificial Intelligence (ECAI)* (1992), B. Neumann, Ed., Wiley & Sons, pp. 360-363.
- [Kautz and Selman, 1998] Kautz, H., and Selman, B. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the AIPS-98 Workshop on Planning as Combinatorial Search* (1998), pp. 58-60
- [Suriani, 2007] Suriani, S. Numerical Parameters in Automated Planning. *PhD Thesis - Department of Mathematics and Computer Science - University of Perugia.2007.*
- [Wolfman and Weld, 1999] Wolfman, S., and Weld, D. The LPSAT engine and its application to resource planning. In *Proc. of IJCAI-99* (1999)
- [Vossen et al., 2001] Vossen, T., Ball, M. Lotem, A. and Nau, D. Applying integer programming to AI planning, *Knowledge Engineering Review* 16:85–100, 2001.
- [Van de Briel et al., 2005] Van de Briel, M. and Kambhampati, S. Optiplan: Unifying ip-based and graph-based planning. *Journal of Artificial Intelligence Research* 24 (2005), 919-931

Authors' Information

Alfredo Milani –Department of Mathematics and Informatics, University of Perugia, Via Vanvitelli 1, 06100 Perugia, Italy, e-mail: milani@dipmat.unipg.it

Silvia Suriani –Department of Mathematics and Informatics, University of Perugia, Via Vanvitelli 1, 06100 Perugia, Italy, e-mail: suriani@dipmat.unipg.it