Simple Algorithm for Simple Timed Games*

Y. Abdeddaïm[1], E. Asarin[2], and M. Sighireanu[2]

[1] Department of Embedded Systems, ESIEE Paris, University of Paris-Est
Cité Descartes, BP 99, 93162 Noisy-le-Grand Cedex, France
[2] LIAFA, University Paris Diderot and CNRS, 75205 Paris 13, France.
y.abdeddaim@esiee.fr, {asarin,sighirea}@liafa.jussieu.fr

Abstract. We propose a subclass of timed game automata (TGA), called Task TGA, representing networks of communicating tasks where the system can choose when to start the task and the environment can choose the duration of the task. We search to solve finite-horizon reachability games on Task TGA by building strategies in the form of Simple Temporal Networks with Uncertainty (STNU). Such strategies have the advantage of being very succinct due to the partial order reduction of independent tasks. We show that the existence of such strategies is an NP-complete problem. A practical consequence of this result is a fully forward algorithm for building STNU strategies. Potential applications of this work are planning and scheduling under temporal uncertainty.

1 Introduction

Timed Game Automata (TGA) model has been introduced in [21] in order to represent open timed systems, and is nowadays extensively studied both from a theoretical and applied viewpoints. As usual in the game theory, main problems for timed games are (1) search for winning strategies that allow to the protagonist player (or the controller) to reach his or her aims whatever the opponent player (or the environment) does, and (2) search for winning states (from which such a strategy exists). A winning strategy can be interpreted as a control program that respects its specification (or maximizes some value function) for any actions of the environment, and for this reason game-solving for TGA is often referred to as controller synthesis. Such a synthesis finds applications to, e.g., industrial plants [24], robotics [1], or multi-media documents [17].

A large class of applications of timed games, relevant to this work, is scheduling and planning under temporal uncertainty (see [3, 12]). For this kind of applications, the protagonist masters all the behaviour of the system, except the durations of its actions. The aim is to ensure a good functioning of the system for any timing (decided by the opponent/environment player in some predefined bounds). Thus a strategy is a control program, schedule or plan robust to timing variations in the controlled process. During the last decade, the planning community has developed several techniques for timed planning problems.

* This work has been supported by the French ANR project AMAES.

These techniques (e.g., [18]) are based on constraint programming and give good performances in practice.

On the other hand, during first years of timed games, a major difficulty was related to the fast state explosion and non-scalability of algorithms and tools. Theoretically, this is not surprising since the upper bound complexity of solving reachability games on TGA has been proved EXPTIME [16]. But the main reason of these bad performances was that most algorithms explored (in a backward way [21, 2], or in a mixed one [24]) almost all the huge symbolic state space of the timed game automaton. Four years ago, an important practical progress in timed strategy synthesis has been attained by Cassez et al. [9], who adapted to the timed case a forward on-the-fly game-solving algorithm from [19], and implemented the resulting algorithm in the tool Uppaal-Tiga [7]. This tool has a performance comparable to reachability analysis of timed automata, and thus makes timed game solving only as difficult as timed verification. However, in many practical cases, even this solution is not always satisfactory. One issue is still the state explosion preventing the tool from finding a strategy. Another issue is a big size and complexity of the strategy synthesized. It is often too heavy to be deployed on the controller. For distributed systems, the controller has to be centralized because the strategy obtained is difficult to distribute. Partial order based methods [8, 20] could help, but as far as we know, they have not yet been applied to timed games.

In this paper, we give a formal explanation of the good results obtained by the planning community by (1) identifying a game model for the timed planning problems and by (2) showing that the complexity of solving such games becomes NP-complete if we search to build strategies of a special form. As a consequence, we obtain a fully forward algorithm for solving timed games by building compact and easy to distribute strategies.

More precisely, we consider games played on a special kind of TGA, so-called *task timed game automata* (TTGA). Such an automaton is a parallel composition of several TGA called tasks. In every task, the controller executes a (possibly infinite) sequence of steps. At each step, the controller decides when to launch some actions whose durations are chosen by the environment and it waits the end of actions launched before starting the next step. Figure 1 gives an example of a TTGA with five tasks. The task **Move** repeats three steps infinitely, at each step the controller launches one action using the controllable transition (solid arrow); the duration of this action is chosen by the environment in the interval given on the uncontrollable transition (dashed arrow). Notice that, in our games, the environment (the opponent player) has no discrete choice, it determines only some durations. The controller has only the choice of the task to execute, but tasks cannot do discrete choices. The aim of the controller is to reach some set of goal states before some deadline (the horizon of the game), and never leave the set of safe states until this goal.

For such games, we will consider strategies of a certain form. First, we disallow "conditional moves", i.e., the discrete choices of the controller are fixed, and the only way that it reacts to the choices of the environment is by adapting

the time at which it launches its actions. This requirement limits somewhat the power of the controller. For example, we are not able to obtain optimality like in [2] because this needs conditional schedulers. Second, instead of a memory-less strategy saying for each state what to do (like in almost all the papers/tools on timed games), we represent a strategy by a compact data structure, called STNU [25], and a computational procedure saying for every game history what are possible next moves for the protagonist. In [1] we show that such a strategy is, by its nature, a more complex object than the usual "state feedback" (or memory-less), but it is in most cases smaller wrt the number of transitions needed to reach the goal, more permissive wrt the number of runs included, and easier to distribute in a multi-component timed system (although some parts of the execution shall be centralized).

Related work. A data structure similar to STNU, called event zones, has been used in [20] for the verification of timed automata. STNU extend event zones with uncontrollable edges. The IxTeT planner [14] uses the STNU data structure and an A^*-like search algorithms to obtain STNU plans from constraint-based specification. Our work gives a TGA model sufficient and yet more expressive for modeling pure[1] timed planning problems considered by IxTeT. Moreover, the completeness of the algorithm used by IxTeT, even for pure timed planning problems, has been a conjecture. We provide here a proof of this conjecture. Finally, our work fulfills the study of the relation between STNU and TGA started by Vidal in [26]. In his work, Vidal shows that TGA are more expressive than STNU and provides a semantics for the execution of STNU strategies in terms of reachability games on TGA.

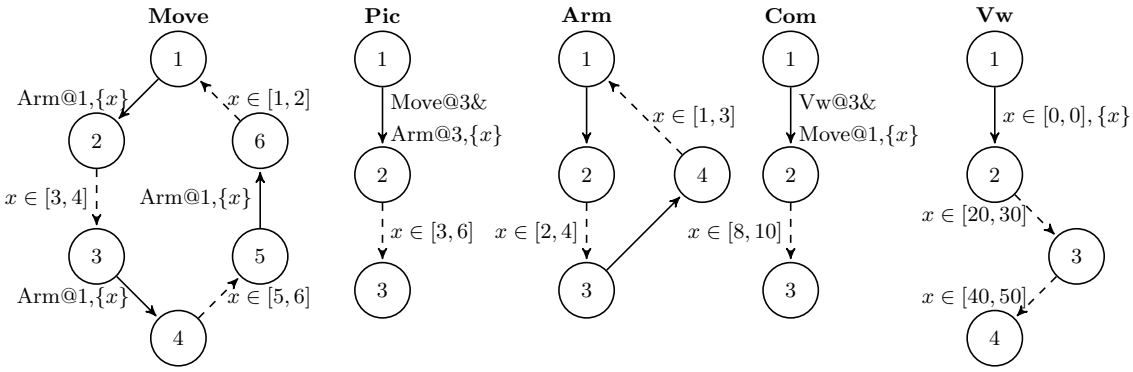

Fig. 1. TTGA network for the Explore rover.

[1] IxTeT models are in fact hybrid since they may relate timed variables (clocks) and discrete variables.

2 Task Timed Game Automata

A TTGA is the parallel composition of a finite set of special TGA called tasks. In each task, the transitions form a sequence with a (possibly empty) final cycle. The sequence of transitions is built from subsequences where a controllable transition is followed by zero or more uncontrollable transitions. These subsequences model a step of the controller starting zero or more tasks with uncontrollable durations but with totally ordered finishing times. When executing a controllable transition, a task may synchronize with its peers only by inspecting their location.

First, let us recall the classical definition of TGA. Let X be a finite set of clocks with values in $\mathbb{R}_{\geq 0}$. We note by $\mathcal{B}(X)$ the set of *rectangular constraints* on variable in X which are possibly empty conjuncts of atomic constraints of the form $x \sim k$ where $k \in \mathbb{N}$, $\sim \in \{<, \leq, \geq, >\}$, and $x \in X$. A constraint in $\mathcal{B}(X)$ defines an interval in $\mathbb{R}_{\geq 0}$ for any $x \in X$.

Definition 1. *[21] A Timed Game Automata (TGA) is a tuple* $A = (L, \ell_0, X, E, \texttt{Inv})$ *where* L *is a finite set of locations,* $\ell_0 \in L$ *is the initial location,* X *is a finite set of real-valued clocks,* $E \subseteq L \times \mathcal{B}(X) \times \{c, u\} \times 2^X \times L$ *is a finite set of controllable (label* c*) and uncontrollable (label* u*) transitions,* $\texttt{Inv} : L \to \mathcal{B}(X)$ *associates to each location its invariant.*

For simplicity, this classical definition does not include discrete variables. However, TGA can be extended with discrete variables whose values can be tested and assigned in transitions. Also, TGA can be composed in parallel to form networks of TGA. The synchronization between parallel TGA can be done using shared clocks or discrete variables.

Then, a TTGA is a network of TGA where (1) each TGA has a special form, called task, (2) the clocks are not shared between parallel tasks, and (3) the synchronization is done only using location constraints. A *location constraint* $i@[m, n]$ requires that the task i is in any location between locations m and n. Let I be a finite set of tasks identifiers. We denote by $\mathcal{I}(I)$ the set of possibly empty conjunctions of locations constraints over the set of tasks I. Wlog, we consider that all location constraints in $\mathcal{I}(I)$ are well formed, i.e., they refer to existing locations in these tasks.

Definition 2. *A* task timed game automaton *(TTGA) is a finite set of tasks,* $A = ||_{i \in I} T_i$*, each task* T_i *being a tuple* $(L_i, \ell_i^0, x_i, E_i)$ *where* L_i *is a finite set of* locations*,* $\ell_i^0 \in L_i$ *is the* initial location*,* x_i *is the* local clock*, and* $E_i : L_i \rightharpoonup \mathcal{I}(I \backslash \{i\}) \times \mathcal{B}(x_i) \times \{c, u\} \times L_i$ *is a finite set of transitions satisfying the following constraints:*

(i) (sequence fully connected) at most one location has no successor by E_i*, i.e.,* $|L_i| - 1 \leq |Dom(E_i)|$*,*

(ii) (no synchronization for the environment) for any transition $\ell \mapsto (g_d, g_x, u, \ell') \in E_i$ *the constraint* g_d *is empty (true),*

(iii) (no time blocking for the environment) for any two consecutive transitions $\ell \mapsto (g_d, g_x, a, \ell')$ *and* $\ell' \mapsto (g'_d, g'_x, a', \ell'')$ *s.t.* $a = u$, g_x *defines an interval in* $\mathbb{R}_{\geq 0}$ *which precedes the interval defined by* g'_x.

Compared to TGA, tasks have only one clock and a transition relation given by a partial function. Moreover, it follows from (i) that a task is either a finite sequence (when $|L_i| - 1 = |Dom(E_i)|$) or a sequence with a final cycle (when $|L_i| = |Dom(E_i)|$). For this reason, we will denote locations $\ell \in L_i$ by a natural number representing the size of the shortest path between ℓ_i^0 and ℓ; by convention, we denote ℓ_i^0 by 1. The transitions of tasks have a (discrete) location constraint, but no set of clocks to be reset. Instead, this set is implicitly determined by the kind of transitions, since x_i is reset iff the transition is controllable. Another implicit definition in tasks is the invariant labeling for locations, `Inv`. For any $\ell \in L_i$, the invariant of ℓ is implicitly $x_i \leq M$, where M is the upper bound of the interval defined by the timed guard g_x in the transition $\ell \mapsto (g_d, g_x, a, \ell')$ in E_i (or empty if such transition does not exists). With this implicit definition for `Inv`, the constraint (iii) ensures that when the environment executes an uncontrollable transition, it can not be blocked by the invariant of the target location. Moreover, uncontrollable transitions are executed in a strict sequence and a controllable transition is executed after the termination of all previous uncontrollable transitions. This constraint has an important consequence: the cycle of a task shall contain at least one controllable transition to reset the clock. Wlog, we will suppose that cycles always start with a controllable transition. Also, like for TGA, we consider only tasks with non-Zeno cycles.

Since TTGA is a subclass of TGA, we omit the definition of its semantics (see [4] for details). We only recall some notions needed to define game semantics. A *run* of a TGA is a sequence of alternating time and discrete transitions also represented by a timed words $(a_1, \tau_1) \dots (a_m, \tau_m)$ where $a_1, \dots, a_m$ are discrete transitions and $\tau_1, \dots, \tau_m$ are the real values corresponding to the global time at which each discrete transition is executed. A *normal run* is a timed word such that when $i \leq j$ then $\tau_i \leq \tau_j$. We denote by $\rho[0]$, $\rho[i]$, and $last(\rho)$ the first state, the i^{th} state (i.e., after the i^{th} discrete action), and resp. the last state[2] of a run ρ. $\texttt{Runs}(A, s)$ denotes the set of normal runs of A starting in state s.

Example 1. Our running example is an instance of the *Explore* system inspired from a Mars rover [5]. The rover explores an initially unknown environment and it can (a) move with the cameras pointing forward; (b) move the cameras (fixed to an arm); (c) take pictures (while still) with the cameras pointing downward, and (d) communicate (while still). The mission of the rover is to navigate in order to take pictures of predefined locations, to communicate with an orbiter during predefined visibility windows, and to return to its initial location before 14 hours. There is a lot of temporal uncertainties, especially in the duration of moves and communications. A TTGA model of this rover is given on Figure 1. Task Move models navigation between three predefined locations reached in

[2] A state of an automaton is given by the locations of components and valuations of clocks.

locations 1, 3, resp. 5. Task Pic models tacking a picture at the second location. Task Arm models arm moving between two positions: forward (location 1) and downward (location 3). Task Com models the communication with an orbiter when the rover is at the first location (the base) and the visibility window is active. Task Vw models the visibility window whose start and end are controlled by the environment and it is active in location 3.

3 Simple Games

The games we consider on TTGA belong to a special subclass of *reachability games*. We first provide general definitions about reachability games [15].

A *reachability game structure* is a tuple $G = (\mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{S})$ where $\mathcal{A}$ is an automaton with edges labeled by controllable and uncontrollable actions, an *initial state* $\mathcal{I}$ in $\mathcal{A}$, a *goal* set of states $\mathcal{G}$, and a *safe* set of states $\mathcal{S}$. The game starts in the initial state $\mathcal{I}$ and, in every state, the controller and the environment choose between waiting or taking a transition they control. The state evolves according to these choices. If the current state is not in $\mathcal{S}$, then the environment wins. If the current state is in $\mathcal{G}$, then the controller wins. Solving a game G consists in finding a *strategy* f such that the automaton $\mathcal{A}$ starting from $\mathcal{I}$ and supervised by f satisfies at any point the constraints in $\mathcal{S}$ and reaches $\mathcal{G}$. A special class of reachability games are *finite horizon games* where the game stops after some finite *horizon* B. In TCTL, this means that $\mathcal{A}$ supervised by f satisfies the formula $\mathcal{I} \wedge \mathtt{A}[\mathcal{S}\ \mathtt{U}_{\leq B}\ (\mathcal{S} \wedge \mathcal{G})]$.

In this work, we consider a finite horizon reachability game defined as follows:

Definition 3. *A* simple game *is a finite horizon game* $G = (\mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{S})$ *such that:*
– $\mathcal{A}$ *is a TTGA and* $\mathcal{I}$ *is its initial state,*
– $\mathcal{G}$ *is specified using a (non empty) conjunction of location constraints* $i@[m,n]$ *saying that any location of task* i *between* m *and* n *is part of the goal,*
– $\mathcal{S}$ *is specified using a conjunction of constraints of the form:*
∘ $i@[m,n] \Longrightarrow j@[k,\ell]$ (embedding) *which requires that if task* i *is in a location in* $[m,n]$ *then task* j *is in a location in* $[k,\ell]$*, and*
∘ $i@[m,n] \sharp j@[k,\ell]$ (mutex) *which requires mutual exclusion between locations in* $[m,n]$ *of task* i *and locations in* $[k,\ell]$ *of task* j*.*

Wlog, we consider that safety and goal constraints are minimal, i.e., all redundant conjuncts are eliminated.

Example 2. $\mathcal{G}$ for the *Explore* example is specified by the constraint:
$\gamma = \text{Move@1} \wedge \text{Pic@3} \wedge \text{Com@3}$
specifying that the rover has to take the picture, to communicate with the orbiter, and to return at its initial location. The horizon is $B = 14$ and the set $\mathcal{S}$ is given by the constraint δ below. In δ, the first two conjuncts forces the rover to take the picture (Pic@2) when the arm is downward (Arm@3) and the rover is still at the second location (Move@3); the next conjuncts ask that rover is moving (Move at locations 2, 4, or 6) with the arm oriented forwardly (Arm@1).

$$\delta = (\text{Pic@2} \Longrightarrow \text{Arm@3}) \wedge (\text{Pic@2} \Longrightarrow \text{Move@3}) \wedge \bigwedge_{\ell \in \{2,4,6\}} (\text{Move@}\ell \Longrightarrow \text{Arm@1}) \wedge (\text{Com@2} \Longrightarrow \text{Move@1}) \wedge (\text{Com@2} \Longrightarrow \text{Vw@3})$$

4 Strategies and STNU

We first recall some definitions concerning strategies. A *strategy* for a controller playing a game G is a relation f between $\texttt{Runs}(\mathcal{A}, \mathcal{I})$ and the set of controllable transitions in $\mathcal{A}$ extended with a special symbol λ. Its semantics is given by the following three rules: (1) if $(\rho, \lambda) \in f$, the controller may wait in the last state of ρ, (2) if $(\rho, e) \in f$, the controller may take the controllable transition e, (3) if ρ is not related by f, the controller has no way to win for ρ with the strategy f. Given a strategy f, we define the *plays* of f, *plays*(f), to be the set of normal runs that are possible when the controller follows the strategy f. Given a reachability game structure G, a strategy f is a *winning strategy* for the game G if for all $\rho \in$ *plays*(f) such that $\rho[0] \in \mathcal{I}$, there exists a position $i \geq 0$ such that $\rho[i] \in \mathcal{G}$, and for all positions $0 \leq j \leq i$, $\rho[j] \in \mathcal{S}$.

4.1 STNU

For finite horizon games, the winning strategies have a finite representation (in absence of Zeno runs). A compact way to represent such strategies is the Simple Temporal Network with Uncertainties (STNU). We present shortly STNU, further details can be found in [25, 23].

An STNU is a weighted oriented graph in which edges are divided into two classes: controllable (or requirement) edges and uncontrollable (or contingent) edges. The weights on edges are non-empty intervals in $\mathbb{R}$. The nodes in the graph represent (discrete) events, called *time-points*. The edges correspond to (interval) constraints on the durations between events. The time-points which are target of an uncontrollable edge are controlled by the environment, subject to the limits imposed by the interval on the edge. All other time-points are controlled by the controller, whose goal is to satisfy the bounds on the controllable edges.

Definition 4. *An STNU Z is a 5-tuple $\langle N, E, C, l, u \rangle$, where N is a set of nodes, E is a set of oriented edges, C is a subset of E containing controllable edges, and $l : E \to \mathbb{R} \cup \{-\infty\}$ and $u : E \to \mathbb{R} \cup \{+\infty\}$ are functions mapping edges into extended real numbers that are the lower and upper bounds of the interval of possible durations. Each uncontrollable edge $e \in E \backslash C$ is required to satisfy $0 \leq l(e) < u(e) < \infty$. Multiple uncontrollable edges with the same finishing points are not allowed.*

Each STNU is associated with a distance graph [11] derived from the upper and lower bound constraints. An STNU is consistent iff the distance graph does not contain a negative cycle, and this can be determined by a single-source shortest path propagation such as in the Bellman-Ford algorithm [10].

Choosing one of the allowed durations for each edge in an STNU Z corresponds to a *schedule* of time-points and gives a distance graph which can be checked for consistency. Then schedules of Z represent finite normal runs over the events in Z. Choosing one of the allowed durations for each uncontrollable edge in an STNU Z determines a family of distance graphs called *projections* of Z. Each projection determines a set of finite runs where uncontrollable time-points are executed always at the same moments. An *execution strategy* f for an STNU Z is a partial mapping from projections of Z to schedules for Z such that for any choice p of execution time for uncontrollable time-points, f assigns an execution time for *all* time-points in Z such that (1) if the time-point is uncontrollable, the execution time is equal to one chosen in p, and (2) if the time-point is controllable, the execution time satisfies the constraints on edges in Z. An execution strategy is said *viable* if it produces a consistent schedule for any projection of Z. So an STNU represents several execution strategies.

Various types of execution strategies have been defined in [25]. We consider here the *dynamic execution strategies* which assign a time to each controllable time-point that may depend on the execution time of uncontrollable edges in the past, but not on those in the future (or present). An STNU is *dynamically controllable* (DC) if it represents a dynamic execution strategy. For this reason, we call in the following an *STNU strategy* an STNU having the DC property.

Definition 5. *An STNU strategy Z is winning if any schedule of Z is a winning run.*

In [23, 22] is shown that the DC property is tractable, and that a polynomial ($\mathcal{O}(n^5)$ with $n = |N|$) algorithm exists defined as follows:

`procedure` Π_{DC} (STNU Z) `returns` Z' or $\perp$

The algorithm applies iteratively on Z a set of rewriting rules including the ones in the shortest path algorithm [10]. These rules introduce controllable edges and tighten the bounds of controllable edges in the input STNU. The algorithm returns the rewritten STNU Z' if it is DC, or an empty STNU $\perp$ otherwise.

Figure 2 shows four STNU with the same nodes and edges, but with different labels and properties (uncontrollable edges are represented by dashed arrows).

4.2 Interfacing STNU with simple games

In order to be a strategy for a simple game G, an STNU shall speak about the transitions of the TTGA A done before reaching some goal state in $\mathcal{G}$. We formalize here this relation by defining (1) the interface of a game G and (2) the satisfaction relation between an STNU and a game interface.

Intuitively, the interface of a simple game is an STNU containing a time-point for any transition of the game that can take place until the horizon of the game is reached; these time-points are related with edges labeled by the timing constraints in the TTGA of the game. For tasks with cycles, a transition may appear several times until the game end. Due to the special form of tasks,

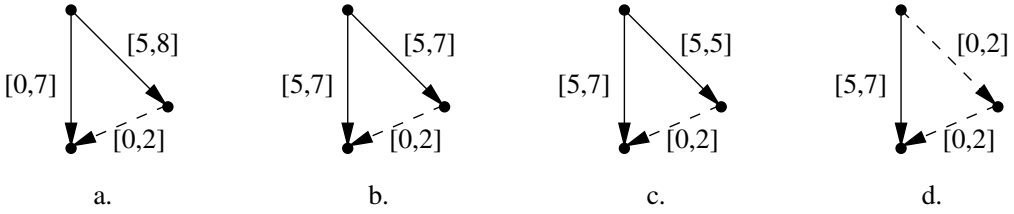

Fig. 2. STNU (a) DC but not reduced, (b) reduced wrt shortest path, (c) reduced wrt DC, (d) not DC.

we can unfold these tasks and compute (see [4] for details) for each transition m of a task i its maximal number of occurrence in the horizon B, denoted by $\mu(i)(m) \geq 1$. Thus, we can identify each transition e that can be executed during the game horizon by a triple (i, m, u) where $i \in I$ is the identifier of the task owning e, $m \in L_i$ is the source location of e, and $u \in [1, \mu(i)(m)]$ is the occurrence number computed for e. In the following, we denote by $prec(i, m, u)$ the transition preceding (i, m, u) in the unfolding of task i. Similarly, $prec_c(i, m, u)$ denotes the last controllable transition before (i, m, u) in the unfolding of task i. If (i, m, u) is the first controllable transition in the unfolding, both notations return $(i, 0, 1)$.

The *interface of* a simple game G with horizon B is an STNU $Z_G = \langle N_G, E_G, C_G, l_G, u_G \rangle$ and a labeling of time-points $\sigma_G : I \times \mathbb{N} \times \mathbb{N} \to N_G$ defined as follows. N_G contains two special time-points t_0 and t_G corresponding respectively to the initial moment and to the goal moment. Moreover, for each task $i \in I$ and each transition $m \in E_i$, N_G contains a number of time-points given by $\mu(i)(m)$. σ_G labels time-points in N_G with transition occurrences such that $\sigma_G(i, m, u)$ is defined if $1 \leq u \leq \mu(i)(m)$ and $\sigma_G(i, 0, 1) = t_0$ for any $i \in I$. E_G relates nodes in N_G wrt timing constraints and ordering of transitions in A. More precisely, the node corresponding to some occurrence u of a transition $e = m \mapsto (g_d, g_x, a, n)$ is related by E_G to the node corresponding to the last controllable transition preceding e (and which resets the local clock) in order to express the duration between the execution of these transitions (events) given by the timed guard g_x. The edge $(\sigma_G(prec_c(i, m, u)), \sigma_G(i, m, u))$ is labeled with the interval defined by g_x and it is in C_G iff $a = c$. The precedence relation on transitions of each task is represented by a controllable transition labeled by $[0, \infty)$. To model the finite horizon constraint of the game, C_G (and E_G) contains a controllable edge (t_0, t_G) labeled by $[0, B]$.

Consider the game given in Example 2. Figure 3 provides the interface of this game. The time-points are put in clusters recalling the task owning the transitions represented by the time-points. Controllable edges without labeling intervals are implicitly labeled by $[0, +\infty)$.

The interface is the "largest" STNU for G wrt the time-points represented and the constraints on edges; it can not represent a strategy for G because neither goal nor safety constraints are satisfied. However, the interface is used as a sanity check for candidate STNU strategy: a candidate strategy shall contain

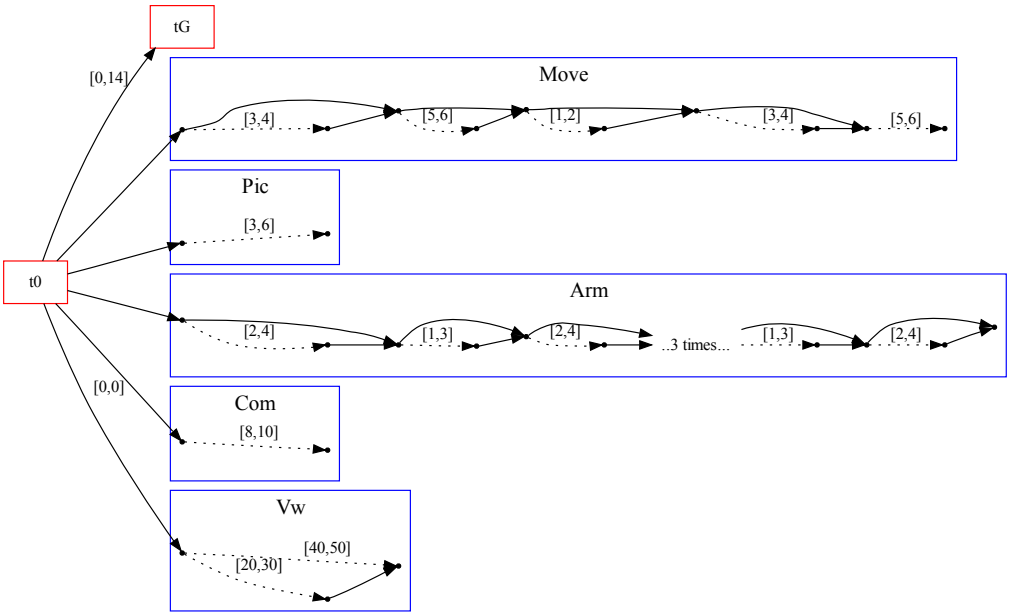

Fig. 3. Interface for the Explorer game with horizon $B = 14$.

a "consistent" subset of time-points and edges in the interface of G. Formally, an STNU $Z = \langle N, E, C, l, u\rangle$ is said to *satisfy the interface* (Z_G, σ_G) of a game G, denoted by $Z \models G$, iff (1) N is a subset of N_G containing t_0 and t_G such that if the time-point labeled by σ_G with (i, m, u) is in N then all its predecessors in the unfolding are also present, and (2) edges defined in E_G between nodes included in N are also defined in E with the same kind (controllable or uncontrollable) and the same labeling intervals for uncontrollable edges; for controllable edges, the labeling interval in Z shall be included in the corresponding one in Z_G. Intuitively, Z shall include exactly the same constraints as Z_G on common uncontrollable edges but may squeeze the constraints on controllable edges.

5 Computing STNU Strategies

We consider the following two problems for simple games G with horizon B given in unary:

G**-Solve:** Decide if a winning strategy exists for G.
G**-Solve-STNU:** Decide if a winning STNU strategy exists for G.

Our first result is that, despite the simplicity of the game considered, the problem of finding strategies for such games is still NP-hard.

Theorem 1. *The G-Solve problem is NP-hard.*

Proof. The proof consists in reducing the CliqueCover problem [13] to the synthesis of a strategy for simple games. An instance of the CliqueCover problem is an integer k and an undirected graph $P = (V, E)$ with n vertices ($|V| = n$) numbered 1 to n. The CliqueCover problem is to decide whether a graph can be partitioned into k cliques. We explain how to reduce this problem to G_k-Solve problem, where G_k is a simple game. The TTGA of G_k is built as follows: for each vertex $i \in [1, n]$ of P, we define a task indexed by i with the form: $1 \xrightarrow{x_i \geq 0, c, \{x_i\}} 2 \xrightarrow{0 \leq x_i < 1, u, \{\}} 3$. The set $\mathcal{S}$ of G_k is specified by $\bigwedge_{(i,j) \notin E} i@2 \sharp j@2$, i.e., for each pair of vertices (i, j) such that there is no edge in E between i and j we ask mutual exclusion between the locations 2 of the corresponding tasks. The set $\mathcal{G}$ of G_k is specified by $\wedge_i i@3$, i.e., all tasks shall reach location 3. The horizon of G_k is fixed to k. It follows (details in [4]) that P can be covered by k cliques or less iff the instance of the G_k-Solve problem has a strategy to win. Moreover, an STNU strategy can be built for G_k.

A direct consequence of the proof above is:

Corollary 1. *The G-Solve-STNU problem is NP-hard.*

The upper bound of complexity for G-Solve is given by the upper bound of solving reachability game problems for TGA, i.e., EXPTIME in [16]. This tight upper bound is obtained for general TGA with safety games and state based (memory less) strategies.

The following theorem says that this upper bound decreases for G-Solve-STNU problems to NP.

Theorem 2. *The G-Solve-STNU problem is NP-complete.*

Proof idea. We show that it exists a correct and complete polynomial test to check that an STNU strategy Z satisfying the interface of a simple game G is a winning STNU strategy. Then, the theorem follows since for a horizon B given in unary, the description of a candidate STNU strategy is polynomial in the number of transitions in G and in the horizon B (Lemma 5).

The polynomial test is given on Figure 4. First (line 1), Z is filled with the missing time-points and edges in the interface of G thus obtaining a new STNU $\overline{Z}$. Second (line 2), the DC algorithm is applied on $\overline{Z}$ to test its dynamic controllability and to compute its reduced form. Third (line 6), $\widehat{Z}$ is tested for satisfaction of goal and safety constraints in G. This test is done by searching in $\widehat{Z}$ a set of edges (called *shapes*) for each goal constraint in $\mathcal{G}$, for each discrete guard on controllable transitions of the TTGA of G, and for each safety constraint in $\mathcal{S}$.

The technical point in the proof is the definition of shapes such that the test is correct and complete. A shape is a positive boolean composition of precedence relations between two time-points. A time-point t precedes the time-point t' in an STNU Z, denoted by $Z \vdash t \prec t'$, iff Z contains an edge from t to t' labeled

Require: STNU Z s.t. $Z \models G$.
Ensure: Returns true if Z is a winning strategy for G.
1: $\overline{Z} \leftarrow Z \cup Z_G$
2: $\widehat{Z} \leftarrow \Pi_{DC}(\overline{Z})$
3: **if** $\widehat{Z} = \bot$ **then**
4: **return** false
5: **end if**
6: **return** $\widehat{Z} \models shape_{\mathcal{G}oal}$ and $\widehat{Z} \models shape_{\mathcal{S}afe}$

Fig. 4. Test for winning STNU strategy.

by an interval included in $(0, +\infty)$. For example, the shape used to test that the goal constraint $i@[m, n]$ is satisfied by $\widehat{Z}$ is $prec(i, m, u) \prec t_G \wedge t_G \prec (i, n, u)$ (see Figure 7). The test succeeds if such shape exists in $\widehat{Z}$ for some u. To test

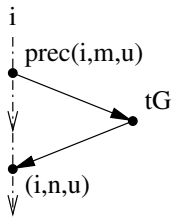

Fig. 5. Shape for goal constraint $i@[m, n]$.

safety constraint and discrete guards on transitions, the shapes shall be carefully defined in order to obtain completeness. For example, the shape for a constraint $i@[m, n] \Longrightarrow j@[k, l]$ is the disjunction of three cases which are not disjoint (see Figure 8). Intuitively, the constraint is satisfied if for any occurrence u of transitions in task i leading to location m (time-point $prec(i, m, u)$) and leaving the location n (time-point (i, n, u)) there exists an occurrence u' of the transition in task j leading to location k (time-point $prec(j, k, u')$) and leaving the location l (time-point (j, l, u')) such that one of the following cases holds:
(i) the safety constraint is entirely satisfied since $prec(j, k, u')$ precedes $prec(i, m, u)$ and (i, n, u) precedes (j, l, u'),
(ii) the safety constraint is satisfied at the beginning of the interval $[m, n]$ for i but nothing is asked for the end because the goal (time-point t_G) is reached before the end of the interval $j@[k, l]$; indeed, the game semantics asks that safety constraints are satisfied until the goal is reached but not beyond,
(iii) the safety constraint is not satisfied because the considered time-points are after the goal.

The correctness proof follows easily from the definition of shapes. The completeness is based on the convexity of the STNU strategies and the fact that the shapes defined can not exclude correct but convex STNU. The full proof is given in [4].

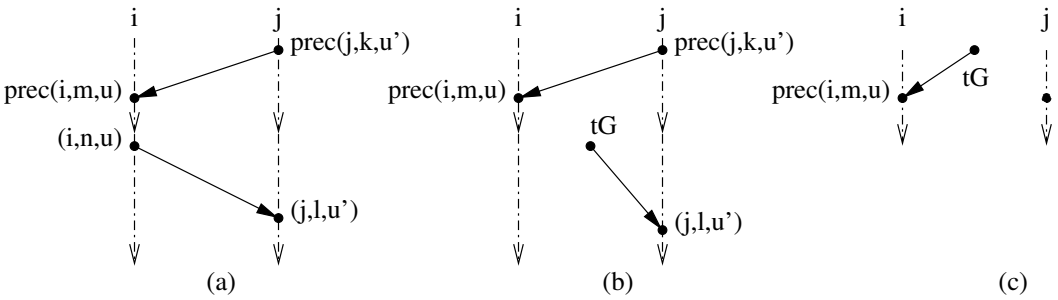

Fig. 6. Shape for safety constraint $i@[m,n] \Longrightarrow j@[k,l]$.

To finish the proof, we compute the complexity of the proposed algorithm. If the horizon B is given in unary, the number of points in the interface of G is polynomial, so the first two steps of the algorithm (lines 1–2) are also polynomial (DC is polynomial in the number of time-points in $\tilde{Z}$). Testing shapes (line 5) is also polynomial. Indeed, to test the satisfaction of some constraint (in guards, goal, or safety), one has to find a set of edges of constant size (the shape) in the STNU. But the number of edges in STNU is quadratic wrt the number of nodes, so we obtain the following result:

Lemma 1. *Given an STNU Z satisfying the interface of a game G, the problem of deciding if Z is a winning strategy for G is in PTIME.*

Now, let us compute the size of a representation for STNU that are candidate strategies for a game G. If the TTGA of G has n transitions, the maximal number of time-points in the interface of G is $n \times B + 2$, limit reached when each transition takes one time unit. Then, the maximal number of edges that has to be specified for the STNU candidate is $O(n^2 \times B^2)$. Each of these edges is labeled by an interval given by two integer numbers. However, these numbers are limited by the horizon B.

Lemma 2. *The size of describing a candidate STNU strategy for a game G is $O(n^2 \times B^3)$ where n is the number of transitions in the network N and B is the horizon of the game.*

6 Algorithm for solving simple games

The proof of Theorem 2 gives also the sufficient conditions for an STNU to be a winning strategy for a simple game G: (1) it has to satisfy the interface of G, (2) it has to be DC, and (3) its reduced form by Π_{DC} has to implement some set of precedence relations defined by the shapes for guards, goal, and safety constraints. The last condition defines a combinatorial space $\mathcal{W}$ for building winning STNU strategies: for each time-point in the interface of G and for each constraint in G involving the transition represented by the time-point, a choice of precedence constraints has to be made.

Then, a simple and fully forward algorithm for building winning STNU strategy is a backtracking algorithm in the combinatorial space $\mathcal{W}$. The algorithm implements each precedence constraint by a controllable edge labeled by a maximal interval (i.e., $(0, +\infty)$) and may use Π_{DC} as a selection (cut off) test for the partially built candidates.

Such an algorithm does not find all winning strategies of G but only "standard" ones, i.e., STNU strategies that contain all the time-points of the interface and have maximal intervals on controllable transitions. This property of solutions built by our algorithm is due to the fact that the algorithm for testing DC computes the maximal timing constraints on controllable links. The proof of correctness and completeness of this algorithm is given in [4].

7 Conclusion

We defined a class of timed games for which searching winning strategies in STNU form is NP-complete. This result gives a formal interpretation of results obtained by the planning and scheduling community with constraint based techniques. Moreover, it shows the existence of a fully forward algorithm for solving games on TGA. This algorithm builds STNU strategies which are finite memory strategies with no discrete choices but including several orderings between independent actions of the controller. Such strategies have to be compared with ones built using classical tools on TGA in order to obtain qualitative results about their size (i.e., number of transitions), volume [6] (i.e., number of runs), etc.

References

1. Y. Abdeddaïm, E. Asarin, M. Gallien, F. Ingrand, C. Lessire, and M. Sighireanu. Planning Robust Temporal Plans A Comparison Between CBTP and TGA Approaches. In *ICAPS*. AAAI, 2007.
2. Y. Abdeddaïm, E. Asarin, and O. Maler. On optimal scheduling under uncertainty. In *TACAS*, volume 2619 of *LNCS*, pages 240–253. Springer-Verlag, 2003.
3. Y. Abdeddaïm, E. Asarin, and O. Maler. Scheduling with timed automata. *Theor. Comput. Sci.*, 354(2), 2006.
4. Y. Abdeddaïm, E. Asarin, and M. Sighireanu. Simple algorithm for sumple timed games. HAL preprint hal-XXXX, 2009.
5. M. Ai-Chang, J. Bresina, L. Charest, A. Jónsson, J. Hsu, B. Kanefsky, P. Maldague, P. Morris, K. Rajan, and J. Yglesias. MAPGEN: Mixed initiative planning and scheduling for the Mars 03 MER mission. In *ISAIRAS*, 2003.
6. E. Asarin and A. Degorre. Volume and entropy of regular timed languages. HAL preprint hal-00369812, 2009.
7. G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime. Uppaal-Tiga: Time for playing games! In *CAV*, volume 4590 of *LNCS*. Springer-Verlag, 2007.
8. J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *CONCUR*, volume 1466 of *LNCS*, pages 485–500. Springer-Verlag, 1998.

9. F. Cassez, A. David, E. Fleury, K. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, volume 3653 of *LNCS*. Springer-Verlag, 2005.
10. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, 1997.
11. R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Network. *Artificial Intelligence*, 49(1-3):61–95, 1991.
12. A. Fehnker. *Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. PhD thesis, KUNijmegen, 2002.
13. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
14. M. Ghallab and H. Laruelle. Representation and control in IxTeT, a temporal planner. In *AIPS*, 1994.
15. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *LNCS*. Springer-Verlag, 2002.
16. T. Henzinger and P. Kopke. Discrete-time control for rectangular hybrid automata. *Theor. Comput. Sci.*, 221:369–392, 1999.
17. N. Layaïda, L. Sabry-Ismaïl, and C. Roisin. Dealing with uncertain durations in synchronized multimedia presentations. *Multimedia Tools Appl.*, 18(3):213–231, 2002.
18. Solange Lemai and F. Ingrand. Interleaving temporal planning and execution in robotics domains. In *AAAI*, San Jose, CA, USA, 2004.
19. X. Liu and S. Smolka. Simple linear-time algorithms for minimal fixed points (extended abstract). In *ICALP*, volume 1443 of *LNCS*. Springer-Verlag, 1998.
20. D. Lugiez, P. Niebert, and S. Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theor. Comput. Sci.*, 345(1):27–59, 2005.
21. O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS*, volume 900 of *LNCS*. Springer-Verlag, 1995.
22. P. Morris. A structural characterization of temporal dynamic controllability. In *CP*, volume 4204 of *LNCS*. Springer-Verlag, 2006.
23. P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *IJCAI*, 2001.
24. S. Tripakis and K. Altisen. On-the-fly controller synthesis for discrete and dense-time systems. In *FM*, volume 1706 of *LNCS*. Springer-Verlag, 1999.
25. T. Vidal. A unified dynamic approach for dealing with temporal uncertainty and conditional planning. In *ICAPS*, Breckenridge, CO, USA, 2000.
26. T. Vidal and H. Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *JETAI*, 11(1), 1999.

A Semantics of TTGA

A *state* of a task T_i is a triple $(\ell, s, v) \in L \times \mathbb{N}^I \times \mathbb{R}^X_{\geq 0}$ such that $s[i] = \ell$; (ℓ, s) is called the discrete part of the state. From a state (ℓ, s, v) such that $v \models tc[\ell]_\prec$, a task can either let time progress or do a discrete transition and reach a new state. This is defined by the transition relation $\longrightarrow$ built as follows:

- For $a = c$, $(\ell, s, v) \xrightarrow{a} (\ell', s', v')$ iff there exists a transition $(\ell, g_d, g_x, a, \{x_i\}, \ell') \in E_i$ such that $s \models g_d$, $v \models g_x$, $s' = s[i@\ell']$, $v' = v[\{x_i\}]$, and $v' \models tc(\ell')_\prec$ (i.e., the upper bound of constraints in $tc(\ell')_\prec$).
- For $a = u$, $(\ell, s, v) \xrightarrow{a} (\ell', s', v)$ iff there exists a transition $(\ell, true, g_x, a, \{x_i\}, \ell') \in E_i$ such that $v \models g_x$, $s' = s[i@\ell']$, and $v \models tc(\ell')_\prec$.
- For $\delta \in \mathbb{R}_{\geq 0}$, $(\ell, s, v) \xrightarrow{\delta} (\ell, s, v')$ if $v' = v + \delta$ then $v', v \in [\![tc[\ell]_\prec]\!]$.

Thus, the semantics of a task T_i is a labeled transition system (LTS) $S_{T_i} = (Q, q_0, \longrightarrow)$ where $Q = L \times \mathbb{N}^I \times \mathbb{R}^X_{\geq 0}$, $q_0 = (\ell^0_i, s_{\mathbf{1}}, \mathbf{0})$, and the set of labels is $\{c, u\} \cup \mathbb{R}_{\geq 0}$.

The semantics of TTGA is defined similarly. A state of the TTGA is a tuple $(\boldsymbol{\ell}, s, v)$ such that for any task i $s[i] = \boldsymbol{\ell}[i]$ and $v \models \wedge_{i \in I} tc[\boldsymbol{\ell}[i]]_\prec$. The transition relation $\longrightarrow$ is defined as above for discrete transitions. For timed transitions, all timed constraints of transitions starting from $\boldsymbol{\ell}$ shall be verified: if $\delta \in \mathbb{R}_{\geq 0}$, $(\boldsymbol{\ell}, s, v) \xrightarrow{\delta} (\boldsymbol{\ell}, s, v')$ if $v' = v + \delta$, and $v', v \in [\![\wedge_{i \in I} tc[\boldsymbol{\ell}[i]]_\prec]\!]$.

A *run* of a TTGA is a sequence of alternating time and discrete transitions in its LTS. Runs can also be represented by timed words $(a_1, \tau_1) \dots (a_m, \tau_m)$ where $a_1, \dots, a_m \in \texttt{Act}$ and $\tau_1, \dots, \tau_m$ are the real values corresponding to the time at which each discrete action is executed. Then, a *normal run* has a timed word such that when $i \leq j$ then $\tau_i \leq \tau_j$. We denote by $last(\rho)$ the last state of a run ρ. We denote by $\texttt{Runs}(N, (\boldsymbol{\ell}, s, v))$ the set of normal runs of a network N starting in state $(\boldsymbol{\ell}, s, v)$. A *maximal run* is a run where time is not blocked by the time constraints, i.e., either it is an infinite run or it ends in a state which location has no discrete successor by any E_i or in a state such that $s \not\models guard(\ell)$ for any ℓ of the state. The constraint (iv) of the definition gives a necessary syntactic condition on the timed constraints of transitions in order to obtain complete runs. Indeed, it implies that for any two successive transitions of a task T_i, $\ell \xrightarrow{\psi_1, \varphi_1, a_1} \ell' \xrightarrow{\psi_2, \varphi_2, a_2} \ell''$ such that $a_1 \neq c$ we have that if, e.g., $\varphi_1 = x_i \in [m_1, M_1]$ and $\varphi_2 = x_i \in [m_2, M_2]$ then $M_1 < m_2 \leq M_2$. Then, by firing the first transition we are sure that the state invariant of ℓ' is satisfied.

B Computing Unfolding of Tasks

Due to the form of TTGA, the minimum execution time of each TTGA is given by the following formula:

$$\delta(T_i)(\omega) = \delta_1(T_i) + \omega \times \delta_*(T_i)$$

where $\delta_1(T_i)$ is the minimum time taken to execute the first time all transitions of the task T_i, ω is the number of iterations of the loop of T_i, and $\delta_*(T_i)$ is the minimum time taken to execute the next iterations of the loop of T_i. All these times are computed only from the timed constraints on the transitions of T_i and the computation does not consider the dependences implied by the discrete guards of transitions.

The minimum execution time for the first execution of all transitions of T_i is given by:

$$\begin{aligned}
\delta_1(T_i) &= \Sigma_{t\ ctrl.}\delta_1(t) + \widehat{\delta_1}(T_i) \\
\delta_1(t) &= max(\{l[t]\} \cup \{l[t-1] \mid (t-1)\ unctrl\}) \\
\widehat{\delta_1}(T_i) &= \begin{cases} 0, & E_i(|L_i|-1)\ ctrl. \\ l[|L_i|-1], & E_i(|L_i|-1)\ unctrl. \end{cases}
\end{aligned}$$

where $l[t]$ is the lower bound (stronger constraint $c \leq x_i$) in the timed guard of transition t. Intuitively, the minimum time for the first execution of T_i is computed from the lower bounds given by the time constraints on the controllable and uncontrollable transitions of T_i. Recall that transitions are indexed by their source locations, so $t-1$ represent the transition before t in the TTGA. Also, since controllable transitions reset the local clock, the minimal execution time of the task is the sum of minimal times elapsed between two controllable transitions plus the residue due to the en of the task which may be not a controllable transition. The time elapsed between two successive controllable transitions t_1 and t_2 is given either by the maximum between the lower bound of t_2 and the lower bound of the uncontrollable transition just before t_2, i.e., $t_2 - 1$ if it is uncontrollable.

The minimum execution time of the second and next iterations of the loop of T_i is computed similarly by tacking the lower bounds of time constraints on the controllable transitions of the loop. However, when considering the predecessor of a controllable transition in the loop t, we have to stay in the loop, i.e., $t-1$ has to be in the loop. Then, for the first controllable transition of the loop we have to consider the last uncontrollable transition in the loop (given by $|L_i|$) if it exists:

$$\begin{aligned}
\delta_*(T_i) &= \Sigma_{t \text{ ctrl. in loop of } T_i}\delta_*(t) \\
&\quad + \widehat{\delta_*}(T_i) \\
\delta_*(t) &= max(\ \{l[t]\} \cup \\
&\qquad \{l[t-1] \mid t-1 \text{ is unctrl. in the loop}\} \\
\widehat{\delta_*}(T_i) &= \begin{cases} 0, & |L_i| \text{ is ctrl.} \\ l[|L_i|], & |L_i| \text{ is unctrl.} \end{cases}
\end{aligned}$$

We define the mapping $\mu : I \to \mathbb{N} \to \mathbb{N}$ which gives for each transition j ($j \geq 1$) of the task i the maximum number of unfolding in order to cover the horizon B. Moreover, $\mu(i)(0)$ returns the maximum number of complete

executions of the loop of T_i to cover B.

$$\mu(i)(0) = \begin{cases} min\{\omega \mid \delta(T_i) \leq B\} & \text{if } \delta_*(T_i) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\mu(i)(j) = 1 + \begin{cases} 0 & \text{if } j \notin \text{loop of } T_i \\ \mu(i)(0) & \text{if } j \in \text{loop of } T_i \\ & \text{but } \delta(\mu)(i)(j) > B \text{ and} \\ & \quad \delta(\mu)(i)(j-1) > B \\ \mu(i)(0) + 1 & \text{if } j \in \text{loop of } T_i \\ & \text{and } \delta(\mu)(i)(j) \leq B \text{ or} \\ & \quad \delta(\mu)(i)(j-1) \leq B \end{cases}$$

where

$$\begin{aligned} \delta(\mu)(i)(j) = {} & \delta_1(T_i) + \mu(i)(0) \times \delta_*(T_i) \\ & + \Sigma_{t<j \text{ ctrl. in loop of } T_i} \delta_*(t) + l[j] \end{aligned}$$

Example 3. The computation of unfoldings for the Explorer rover (see Figure 1) is given on Table 1. The table pictured in this figure gives for each task and for each transition (named by its source location) the number of time-points represented for this transition in the game interface.

			Transitions (j)					
Task (i)	$\mu(0)(i)$	$\delta(T_i)(\omega)$	1	2	3	4	5	6
`Move`	0	$9 + 9\omega$	2	2	2	2	1	1
`Pic`	0	3	1	1	–	–	–	–
`Arm`	3	$3 + 3\omega$	5	5	5	4	–	–
`Com`	0	8	1	1	–	–	–	–
`Vw`	0	40	1	1	1	–	–	–

Table 1. Unfolding computation for Explore example.

C Proof of Theorem 1

Let k be an integer and P be an undirected graph $P = (V, E)$ with n vertices ($|V| = n$) numbered 1 to n. Let G_k be the game built at Theorem 1. We show that P can be covered by k cliques or less iff the G_k-Solve problem has a strategy to win.

($\Rightarrow$) If P has at most k cliques, let $V = \cup_{0 \leq \ell \leq k-1} V_\ell$ with V_ℓ the set of ℓ^{th} clique. By definition of G_k, for any ℓ, all tasks indexed by vertices in V_ℓ can be in their location 2 at the same time. Then, the first transition of these tasks can be executed at the same time without invalidating the safety property of G_k and they have to leave location 2 after one time unit at most. A winning strategy

for G_k is to take at moment t the controllable transition of the tasks indexed by the vertices in V_t. By construction of G_k, all tasks will end in location 3 before k time units and the safety constraints are satisfied.

($\Leftarrow$) If the game G_k has a winning strategy, this strategy shall start all tasks in the interval $[0, k]$. Since each task takes at most one time unit, then there are at most k groups of tasks executed simultaneously (i.e., in their location 1) during $[0, k]$. By construction of G, these tasks correspond to vertices related by an edge in P, so P has at most k cliques.

Show that an STNU strategy exists.

D Proof of Theorem 2

The proof gives a polynomial test to decide if an STNU strategy Z satisfying the interface of a simple game G is a winning strategy for G (Lemma 3 and 4 below). Then, the theorem follows since for a horizon B given in unary, the description of potential STNU candidate for a winning strategy is polynomial in the size of the game and the bound B (Lemma 5 below).

The algorithm testing when an STNU is a winning strategy for a simple game is given on Figure 4.

The shapes used in the test algorithm are defined from the goal $\mathcal{G}$, the guards in the network N, and the safety constraints $\mathcal{S}$.

An *atomic shape* is a precedence constraint between two time-points, denoted by $t \prec t'$, saying that the time-point t precedes time-point t'. An STNU Z containing the time-points t and t' satisfies the atomic shape $t \prec t'$, denoted by $Z \vdash t \prec t'$, iff Z contains a link from t to t' whose label is an interval with strictly positive lower bound (i.e., an interval in $(0, +\infty)$). Atomic shape constraints are combined using conjunctions, disjunctions and quantification over time-points to form shape constraints.

More precisely, the time-points in shape constraints belong to STNU satisfying the interface of G, so they are labeled (due to σ_G^{-1}) by triples (i, m, u) with $i \in I$ the index of the task, $m \geq 1$ the order number of the state source of the transition, and $u \geq 1$ the occurrence of the transition. Quantification in shapes is done over the occurrence numbers u in time-points labels.

Shapes (without quantification) can be represented graphically as shown on Figures 7 and 8. Time-points are labeled by their tuple (tasks index, transition index, occurrence number). Dashed lines are used to show time-points belonging to the same task.

Formally, the syntax of *shape constraints* is:

$$\text{Atomic shape} \ni a ::= (i, m, u) \sim t_G \mid (i, m, u) \sim (j, k, l)$$
$$\text{Shape} \ni s ::= a \mid s \wedge s \mid s \vee s \mid \exists u.\ s \mid \forall u.\ s$$

where $i, j \in I$, $m, k \geq 1$ are index of transitions in T_i resp. T_j, $u, l \geq 1$ are occurrences of these transitions, and $\sim \in \{\prec, \succ\}$.

For example, the shape constraint of Figure 7 is $prec(i,m,u) \prec t_G \wedge (i,n,u) \succ t_G$ where we use $prec(i,m,u)$ to represent the time-point for the transition preceding the transition (i,m,u) on T_i.

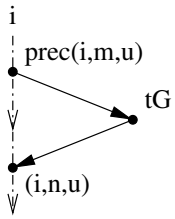

Fig. 7. Shape for $i@[m,n] \in \mathcal{G}$.

The shape constraints used in the algorithm to test the goal are built from the goal constraints as follows:

$$shape_{\mathcal{G}oal} ::= \bigwedge_{i@[m,n]\in\mathcal{G}} \exists u.\ prec(i,m,u) \prec t_G \wedge t_G \prec (i,n,u)$$

which means that for each constraint in the goal, $\gamma = i@[m,n]$, there exists an occurrence number for the transitions preceding transition (i,m) (which leads to the m^{th} state of T_i) and the transition (j,n) (which goes out from the n^{th} state of T_i) such that the goal is reached at t_G between these two moments.

Shapes used to test safety are generated from safety constraints as follows:

$$shape_{\mathcal{S}afe} ::= \bigwedge_{i\in I} \left(\bigwedge_{act[(i,m)]\in \mathtt{Act}_c} shape_{(i,m)} \right) \wedge \bigwedge_{\delta\in\mathcal{S}} shape_{\delta}$$

where the first set of conjuncts gives shape constraints from the guards of controllable transitions in each TTGA, while the second set represents shapes from the safety constraints $\mathcal{S}$ of G. In defining these shapes, a special treatment shall be considered for points succeeding the goal time-point t_G because the safety constraints shall be satisfied until reaching the goal but they are not mandatory afterwards.

Consider now a controllable transition (i,m) and some of its occurrence u. Then, the safety constraint corresponding to the guard of (i,m) may be ignored if time-point corresponding to (i,m,u) is after the goal. Otherwise, for each indexed interval constraint $j@[k,l]$ in the guard of (i,m), the time-point (i,m,u) shall be between some occurrence u' of the transition leading to the state k of T_j (i.e., $prec(j,k,u')$) and the transition going out from the state l. Then, $shape_{(i,m)}$ is defined by:

$$\begin{array}{l} \forall u.\ t_G \prec (i,m,u) \vee \\ \bigwedge_{j@[k,l]\in guard[(i,m)]} \big(\exists u'.\quad prec(j,k,u') \prec (i,m,u) \\ \qquad\qquad\qquad \wedge\ (i,m,u) \prec (j,l,u')\big) \end{array}$$

The shapes for safety constraints in $i@[m,n] \Longrightarrow j@[k,l]$ are generalization of shapes for guards:

$$\begin{array}{l}\forall u.\ t_G \prec prec(i,m,u) \vee \\ \quad \exists u'.\ prec(j,k,u') \prec prec(i,m,u) \wedge \\ \qquad \big((i,n,u) \prec (j,l,u')\ \vee\ t_G \prec (j,l,u')\big)\end{array}$$

Intuitively, the i^{th} TTGA is between states m and n while the j^{th} TTGA is between states k and l if for any occurrence u of transitions leading to state m ($prec(i,m,u)$) and the one leaving the state n ((i,n,u)) there exists an occurrence u' of transitions of T_j leading to state k ($prec(j,k,u')$) and leaving the state l ((j,l,u')) such that one of the following cases is true:

- the safety constraint is entirely satisfied by the u occurrence of the execution of T_i between states m and n and by the u' occurrence of the execution of T_j between states k and l,
- the safety constraint is satisfied at the beginning of these executions but not at the end of the intervals due to the fact that the goal is reached before execution of T_j between k and l ends, or
- the safety constraint is not satisfied because the considered occurrences are after the goal.

Figure 8 gives an illustration of each case above. Note that the above cases are not disjoint and this is an important fact for the proof of completeness of our test procedure (Lemma 1).

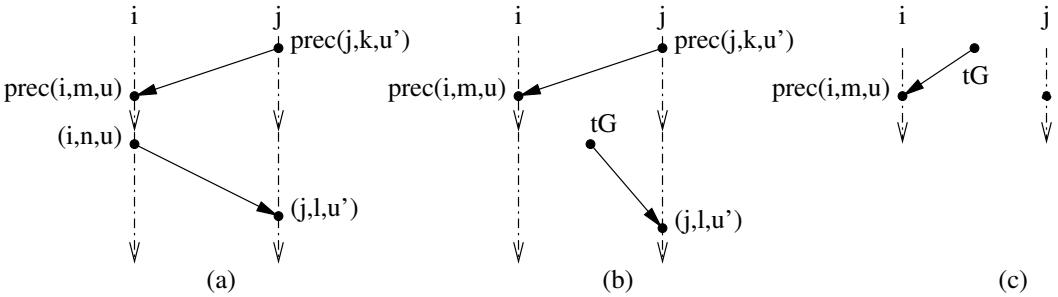

Fig. 8. Shape for safety $i@[m,n] \Longrightarrow j@[k,l]$ of $\mathcal{S}$.

Finally, the shapes for mutual exclusion safety constraints $i@[m,n] \sharp j@[k,l]$ are defined as follows:

$$\begin{array}{l}\big(\forall u.\ t_G \prec prec(i,m,u) \\ \quad \vee \exists u'.\quad (j,l,u'-1) \prec prec(i,m,u) \\ \qquad \wedge\ (i,n,u) \prec prec(j,k,u')\big) \\ \wedge\ \big(\forall u.\ t_G \prec prec(j,k,u) \\ \quad \vee \exists u'.\quad (i,n,u'-1) \prec prec(j,k,u) \\ \qquad \wedge\ (j,l,u) \prec prec(i,m,u')\big)\end{array}$$

where for any $i \in I$ and any state m in T_i, $(i, m, 0)$ represents t_0. Intuitively, the safety constraint $i@[m, n] \sharp j@[k, l]$ is satisfied if for each occurrence u of an execution between states m and n in T_i (and k and l in T_j) one of the following conditions holds:

- this occurrence is after the goal, i.e., the u occurrence of the transition leading to the m (resp. k) is after the goal, or
- the execution takes place strictly between occurrences $u' - 1$ and u' of execution of T_j from state k to l (resp. T_i from state m to n) for some $u' \geq 1$.

Like for inclusion constraints, these two cases are also overlapping.

The following lemma states the correctness and the completeness of our algorithm.

Lemma 3. *An STNU Z satisfying the interface of a game G is a winning strategy iff the algorithm* ***IsWinningStrategy*** *returns true.*

Proof. ($\Leftarrow$) Let Z be an STNU such that $Z \vdash G$ and the test algorithm returns true. The algorithm builds from Z an STNU which is DC and satisfies all the goal and safety constraints. By definition of winning strategies, the result follows.

($\Rightarrow$) The proof of completeness uses the convexity of the STNU strategies and the fact that the shape tested cannot exclude correct but convex STNU. Let Z be a winning strategy satisfying the interface of G. Then Z is already DC and satisfies the goal and safety constraints.We suppose that Z is in the reduced form of DC.

The algorithm first completes Z to obtain the full interface of G. However, the time-points added are only appended to the last time-point of each task so they can not influence the links in Z. Then, the DC algorithm applied to $\tilde{Z}$ only introduces more controllable links involving the newly introduced time-points, but does not change the original part of Z.

Then, since Z is contained in $\widehat{Z}$ and it is a winning strategy, it means that each schedule of Z determines a play which goes through a goal configuration. Since (1) STNU can not represent union of goal states and (2) the constraints on states in $\mathcal{G}$ are disjoint imply that only one possibility to satisfy state constraints in $\mathcal{G}$ is present in Z. This configuration can be used to satisfy the shape for the goal.

All plays of Z already satisfy the safety constraints (guards and constraints in $\mathcal{S}$). Since the added points in $\tilde{Z}$ are all after the goal configuration, they trivially satisfy the safety shapes since all these shapes are trivial for points after t_G. For time-points before t_G, we claim that these time-points satisfy exactly one of the disjunction of the shapes in $shape_{\mathcal{S}afe}$. The idea is to show that (1) the shapes considered correspond to disjoint set of plays and (2) the plays of Z cannot be in two such shapes due to convexity of Z.

Figure 9 gives an intuition of the point (1) for the shapes corresponding to inclusion safety constraints $i@[m, n] \Rightarrow j@[k, l]$. The plan represents in x axis the beginning of executions ($prec(i, m, u)$ and $prec(j, k, u')$) and in the y axis the end of theses executions ((i, n, u) and (j, l, u')). Then all considered

points shall be in the second half of the first quadrant. Points A, B, C and D represent occurrences of the executions in T_j respectively, i.e., the the first, second, third, and fourth occurrences of $prec(j, k, u')$ and (j, l, u'). In fact, these points are DBM zones in Z, but they can be abstracted to points dues to the following property: the consecutive occurrences of executions of T_j are disjoint and moreover, $(j, l, u') \prec prec(j, k, u + 1)$. The zones attached to points A–C and G correspond to executions on T_i allowed by the shape for safety constraint $i@[m, n] \Rightarrow j@[k, l]$. For points A and B situated before the goal (x and y are less than t_G), the executions on T_i shall satisfy $prec(j, k, u') \prec prec(i, m, u)$ and $(i, n, u) \prec (j, l, u')$. For point C, although the beginning of execution of T_j is before t_G, this execution ends after t_G, so the executions allowed for T_i satisfy $prec(j, k, u') \prec prec(i, m, u)$. Finally, all points greater than t_G correspond to valid executions. It is important to see that we have only convex regions and any DBM covering these regions is fully included in one of them.

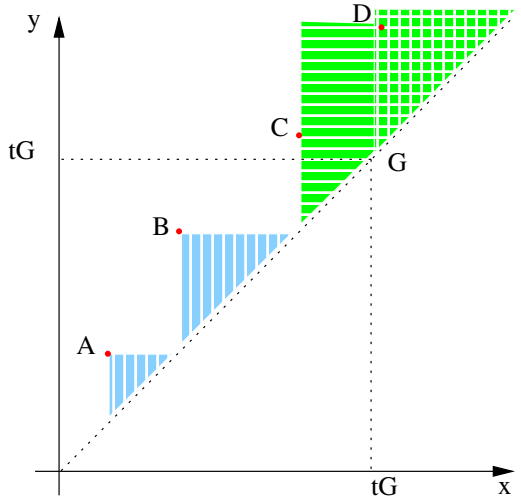

Fig. 9. Proof of completeness.

If Z is winning, it shall satisfy every safety constraint δ of the form $i@[m, n] \implies j@[k, l]$, that is for every moment t, if i is in t between states m and n, then either t is after t_G or task j is between states k and l. The intervals in which j is between states k and l are disjoint (they are the hypotenuse of the triangle determined by the point attached to this point) because they correspond to different occurrences of this interval. Then, to satisfy the safety constraint δ, the hypotenuse for intervals $i@[m, n]$ shall be contained in the hypotenuse of triangles for points A–C or after t_G. These zone are those defined by the safety shape for this constraint.

To finish the proof, we compute the complexity of the proposed algorithm. If B (the game horizon) is given in unary, the number of points in the interface

of G is polynomial, so the first two steps of the algorithm (lines 1–2) are also polynomial (DC is polynomial in the number of time-points in $\tilde{Z}$). Testing shapes (lines 5 and 7) is also polynomial. Indeed, for each constraint (in guards, goal or safety), to test its satisfaction one has to find a (small) set of links in the graph defined by the STNU. The number of links in this graph is quadratic wrt the number of vertices. We obtain then the following lemma which is a first step in establishing the complexity of our problem.

Lemma 4. *Given an STNU Z satisfying the interface of a game G, the problem of deciding if Z is a winning strategy for G is in PTIME.*

Now, let us compute the size of a representation for STNU that are candidate strategies for a game G. If the TTGA of G has n transitions, the maximal number of time-points in the interface of G is $n \times B + 2$, limit reached when each transition takes one time unit. Then, the maximal number of edges that has to be specified for the STNU candidate is $O(n^2 \times B^2)$. Each of these edges is labeled by an interval given by two integer numbers which are limited by the horizon B. Then, we obtain the following result which finishes the proof of NP completeness.

Lemma 5. *The size needed for describing a candidate STNU strategy for a game G is $O(n^2 \times B^3)$ where n is the number of transitions in the TTGA of G and B is the horizon of the game given in unary.*

E Properties of the simple algorithm

Proposition 1. *All controllable edges in a SWS for a game G are maximal wrt the game interface and constraints.*

An STNU Z is included in some SWS iff (1) Z contains all time-points of the interface of G, (2) Z it is minimized using DC (so consistent), and (3) all controllable edges in Z are labelled by intervals included in the interval labeling the same edge in SWS.

Corollary 2. *An STNU Z satisfying the interface of G is a winning strategy iff $\widehat{Z}$ is included in some standard solution for the game G.*

where $\widehat{Z}$ is built as shown in Figure 4, i.e., by adding to Z all time-points and edges in the interface of G and then applying DC.

Proof. $\Rightarrow$ If Z is a winning strategy for G, then it satisfies the goal and safety constraints in G and for any choice of the environment. $\widehat{Z}$ is obtained from Z by adding time-points which are not useful for satisfying safety and goal (since these time-points are related only with the last time-points of Z on each task...). Moreover, the DC algorithm does not change the controllability property. From the completeness result of Lemma 3, it results that in order to satisfy the goal and safety constraints of G, Z (and so $\widehat{Z}$) shall include some combination of safety and goal shapes. This combination gives the SWS to be chosen. But SWS is the maximal STNU with interface of G implementing the shapes and being controllable, so $\widehat{Z}$ shall be included in this SWS.

$\Leftarrow$ By the correctness of result of Lemma 3.

A direct consequence of the above property is given in the following corollary and it will be used to prove the relative completeness of our algorithm for solving simple games:

Corollary 3. *The G-Solve-STNU has a solution iff there exists a "standard" solution for G.*

Let $\mathcal{B}$ the set of standard winning strategies (SWS) built for the game G. From the definition of SWS, the elements of $\mathcal{B}$ are STNU containing all points of the interface of G, implementing different shapes for each goal and safety constraints, and being DC. The maximal size of $\mathcal{B}$ is given by the maximal number of choices for the implementation of goal and safety constraint for each occurrence of transitions/states in the interface of G. For example, if $i@[m,n] \in \mathcal{G}$ and $\mu(i)(m)$ is the number of occurrences of the m^{th} transition on task i, the number of choices to implement this (part of the) goal is given by $\mu(i)(m)$. For a safety constraint $i@[m,n] \Longrightarrow j@[k,l]$, the number of choices is given by $(3\mu(j)(k))^{\mu(i)(m)}$. In general the maximal size of $\mathcal{B}$ is exponential in the size (number of points) of the interface of G.