Using Timed Game Automata to Synthesize Execution Strategies for Simple Temporal Networks with Uncertainty

Alessandro Cimatti[1], **Luke Hunsberger**[2], **Andrea Micheli**[1,3] and **Marco Roveri**[1]
cimatti@fbk.eu, hunsberg@cs.vassar.edu, amicheli@fbk.eu, roveri@fbk.eu
[1]Fondazione Bruno Kessler – Trento, Italy
[2]Vassar College – Poughkeepsie, NY, USA
[3]University of Trento – Trento, Italy

Abstract

A Simple Temporal Network with Uncertainty (STNU) is a structure for representing and reasoning about temporal constraints in domains where some temporal durations are not controlled by the executor. The most important property of an STNU is whether it is dynamically controllable (DC); that is, whether there exists a strategy for executing the controllable time-points that guarantees that all constraints will be satisfied no matter how the uncontrollable durations turn out.

This paper provides a novel mapping from STNUs to Timed Game Automata (TGAs) that: (1) explicates the deep theoretical relationships between STNUs and TGAs; and (2) enables the memoryless strategies generated from the TGA to be transformed into equivalent STNU execution strategies that reduce the real-time computational burden for the executor. The paper formally proves that the STNU-to-TGA encoding properly captures the execution semantics of STNUs.

1 Introduction

Temporal Networks are commonly used to represent temporal constraints among activities (Dechter, Meiri, and Pearl 1991; Vidal and Fargier 1999; Tsamardinos and Pollack 2003). Each activity is associated with two time-points, representing the starting and ending times of the activity. Many kinds of temporal networks have been identified depending on the nature of the constraints. In a Simple Temporal Network, each constraint is a binary difference constraint. In a Disjunctive Temporal Network, arbitrary Boolean combinations of such constraints are allowed. In a Temporal Network with Uncertainty (TNU), the durations of some activities are not controlled by the executor. The ending times for those activities are represented by *uncontrollable* time-points, while the starts are represented by *free* time-points. For TNUs, three types of *controllability* problems have been addressed: strong, weak and dynamic. Strong controllability requires the existence of an assignment to all of the free time-points that satisfies all of the constraints in the network for every possible combination of durations for the uncontrollable activities. Weak controllability assumes that the executor knows the durations of all uncontrollable activities in advance, enabling the scheduling of activities to be a function of those durations. Dynamic controllability is similar to weak, except that each scheduling decision is constrained to depend only on past events.

Because it captures the real-time constraints that apply to most application domains, this paper focuses on dynamic controllability (DC) for Simple Temporal Networks with Uncertainty (STNUs). Polynomial algorithms for solving the DC-checking problem have been presented (Morris and Muscettola 2005; Morris 2006). Polynomial algorithms for managing the execution of such networks have also been presented (Hunsberger 2010; 2013a). However, the fastest of these requires substantial run-time computations to propagate constraints in response to observed executions. The problem of synthesizing dynamic strategies that can be simply and compactly executed is currently an open problem.

This paper introduces a new approach to synthesizing execution strategies for STNUs that is based on Timed Game Automata (TGA) (Maler, Pnueli, and Sifakis 1995). A TGA is a timed automata (Henzinger, Manna, and Pnueli 1991) in which two opposing players are able to take transitions subject to specified timing constraints. The goal of a player can be, for example, to reach a certain state or to avoid certain states. Several practical algorithms have been developed for constructively synthesizing winning strategies for TGAs. We remark that TGAs are much more general than STNUs.

This paper makes the following contributions. First, it presents a novel, linear mapping from STNUs to TGAs that: (1) explicates the deep theoretical relationships between STNUs and TGAs; and (2) enables the memoryless *counter-strategies* synthesized for TGAs to be transformed into equivalent STNU execution strategies with the aim of reducing the real-time computational burden for the executor. Second, it proves that each dynamically controllable STNU maps onto a TGA for which there is a winning counter-strategy; and that any winning counter-strategy for that TGA is equivalent to a winning strategy for the original STNU. Finally, the paper tests the feasibility of the approach whereby an input STNU is mapped onto a TGA, a winning counter-strategy is synthesized by the UPPAAL-TIGA tool (Cassez et al. 2005), and then that strategy is converted into executable C code that is compiled and efficiently executed.

Related work. Researchers have explored a variety of techniques for solving temporal problems in the face of uncertainty, but only one is close to the approach presented this

paper. Vidal (2000) uses TGAs for checking the dynamic controllability of a variant of STNUs called *Contingent Temporal Constraint Networks* (CTCNs). His algorithm incrementally constructs a TGA, interleaving checks for winning TGA strategies along the way. The most significant drawback is that the resulting TGA has exponential size, compared to the linear-sized TGAs in our approach.

TGAs have also been used to validate timeline-based plans (Cesta et al. 2011; Orlandini et al. 2011). Each plan is encoded as a TGA that includes an uncontrollable observer that plays the role of the environment. The observer checks the plan controllability and synthesizes a controller. The encoding used by Orlandini et al. (2011) deviates from the standard DC definition by allowing a free time-point to be scheduled *immediately* after an uncontrollable.

Abdeddaim et al. (2009) use STNUs to represent strategies for a subclass of TGA—the exact opposite of our approach. In their work, an executor needs to be able to solve the DC-checking problem (e.g., using an on-line algorithm) to generate a TGA strategy. Finally, Cimatti et al. (2012b; 2012a) focus on strong and weak controllability, while also allowing disjunctive constraints.

Paper structure. Section 2 presents a novel characterization of the execution semantics for STNUs. Section 3 presents our encoding of STNUs into TGAs. Section 4 provides formal proofs of correctness. Section 5 presents an empirical evaluation of our approach. Finally, Section 6 draws some conclusions and discusses future work.

2 STNUs and Dynamic Controllability

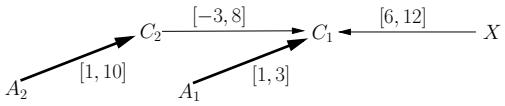
An STNU is a data structure for representing and reasoning about temporal knowledge in domains where some time-points are controlled by the executor (or agent) while others are controlled by the environment.[1]

Definition 1. *An STNU is a tuple $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ where: $\mathcal{T}$ is a set of real-valued variables called* time-points, *that is partitioned into the sets, $\mathcal{T}_f$ and $\mathcal{T}_u$, of* free *and* uncontrollable *time-points; $\mathcal{C}$ is a set of binary constraints, each of the form, $Y - X \leq \delta$, for some $X, Y \in \mathcal{T}$ and $\delta \in \mathbb{R}$; and $\mathcal{L}$ is a set of* contingent links, *each of the form, (A, ℓ, u, C), where $A \in \mathcal{T}, C \in \mathcal{T}_u$, and $0 < \ell < u < \infty$.*

An expression, $Y - X \in [a, b]$, abbreviates the pair of constraints, $Y - X \leq b$ and $X - Y \leq -a$. A contingent link, (A, ℓ, u, C), represents a temporal interval from A to C whose duration is uncontrollable, but bounded by $C - A \in [\ell, u]$. A is called the *activation* time-point; C is called the *contingent* time-point. Fig. 1 shows an STNU that will be used as a running example. In the figure, contingent links are indicated by thicker arrows. Note that the terminus of any contingent link is an uncontrollable time-point.

Execution Semantics for STNUs

In the literature, the execution semantics for STNUs is expressed in terms of *dynamic execution strategies* (Morris, Muscettola, and Vidal 2001). For an STNU, $(\mathcal{T}, \mathcal{C}, \mathcal{L})$, the agent seeks a strategy for executing the *free* time-points in $\mathcal{T}_f \subseteq \mathcal{T}$ such that all constraints in $\mathcal{C}$ will necessarily be satisfied *no matter what durations the environment "chooses" for the contingent links in $\mathcal{L}$—within their specified bounds.* The decisions that constitute such a strategy can depend only on execution events that occurred in the past; however, the strategy can be dynamic in that it may react—*after a positive delay*—to observations of contingent time-points executing.



$$\begin{array}{lcl} \mathcal{T}_f & = & \{A_1, A_2, X\};\ \mathcal{T}_u = \{C_1, C_2\} \\ \mathcal{C} & = & \{C_1 - C_2 \in [-3, 8],\ C_1 - X \in [6, 12]\} \\ \mathcal{L} & = & \{(A_1, 1, 3, C_1), (A_2, 1, 10, C_2)\} \end{array}$$

Figure 1: An STNU and its graphical representation.

An agent's execution strategy can be compactly defined in terms of *real-time execution decisions* (RTEDs), where each RTED has one of two forms: `wait` or (T, χ) (Hunsberger 2009). A `wait` decision can be glossed as "wait until some contingent time-point happens to execute." A (T, χ) decision can be glossed as "if nothing happens *before* time T (i.e., if no contingent time-point happens to execute before time T), then I shall execute the (free) time-points in the set χ at time T." The *outcomes* for an RTED specify the range of execution events that could happen *next.* For example, a contingent time-point might happen to execute sometime before time T, in which case, the agent could react by adopting a new decision; or a contingent time-point C might happen to execute precisely *at* time T, in which case the time-points in χ would be executed simultaneously with C at time T.

In the case of the STNU in Fig. 1, the agent seeks a strategy for executing the free time-points, A_1, A_2 and X, that will guarantee that the constraints among C_2, C_1 and X are satisfied, no matter what durations the environment happens to "choose" for the contingent links, $(A_1, 1, 3, C_1)$ and $(A_2, 1, 10, C_2)$. For example, the agent might decide to execute A_2 at time 0, and X at time 1, and then wait. Should the environment happen to "choose" a duration of 5 for the contingent link, $(A_2, 1, 10, C_2)$, the agent would observe, at time 5, the execution of C_2. The agent might then react—after some positive delay—by, for example, deciding to execute A_1 at time 7. Later, the agent might observe the environment choosing to execute C_1 at 9. In this example, after all time-points have executed, $C_1 - C_2 = 9 - 5 = 4 \in [-3, 8]$ and $C_1 - X = 9 - 1 = 8 \in [6, 12]$; thus, all constraints in $\mathcal{C}$ are satisfied and the agent has succeeded. It can be checked that this STNU is *dynamically controllable* (i.e., there exists a strategy for the agent that ensures success no matter how the environment behaves).

Although the execution semantics described above makes reference to the agent and the environment, execution strategies are only defined for the agent; the strategies available to the environment are only implicitly described by the sets of possible outcomes of the agent's decisions. Thus, the semantics is effectively a description of a one-player game where

[1] The agent and environment are not part of the formal STNU semantics; they are used here for expository convenience.

the outcomes of the agent's decisions are non-deterministic. This section introduces a novel formulation of the execution semantics for STNUs as a two-player game between Agnes (the agent) and Vera (the environment), where Agnes controls the execution of free time-points and Vera controls the contingent durations. Agnes seeks an execution strategy that will ensure the satisfaction of all constraints in $\mathcal{C}$ no matter what durations Vera chooses; Vera seeks a strategy that will ensure that at least one constraint in $\mathcal{C}$ is *unsatisfied* no matter what Agnes does. As will be seen, this formulation highlights an important asymmetry in the execution semantics: Agnes is *not* able to react instantaneously to observations of contingent time-points executing, but Vera *is* able to react instantaneously to executions of free time-points.

A *partial schedule* represents the current state of the game (i.e., the set of time-points that have executed so far).[2]

Definition 2 (Partial Schedule)**.** *A partial schedule for an STNU, $(\mathcal{T}, \mathcal{C}, \mathcal{L})$, is a set, ψ, of variable assignments to time-points in $\mathcal{T}$. $TPs(\psi) \subset \mathcal{T}$ denotes the set of time-points appearing in ψ; $Vals(\psi) \subset \mathbb{R}$ denotes the set of values appearing in ψ; for any $X \in TPs(\psi)$, $\psi(X)$ denotes the value assigned to X; and $\texttt{now}_\psi = \max\{v \mid v \in Vals(\psi)\}$ is the time of the latest execution event in ψ. (If $\psi = \emptyset$, let $\texttt{now}_\psi = 0$.) Time-points in $TPs(\psi)$ are said to be* executed. *A partial schedule is called* respectful *if its assignments do not violate the bounds on any contingent link.*

Given a partial schedule ψ, Agnes must decide what to do next. She has two options: (1) wait for Vera to (eventually) do something; or (2) conditionally commit to executing a set of free time-points at some time, $T_f > \texttt{now}_\psi$. For example, given $\psi = \{(A_2, 0), (X, 1)\}$, for which $\texttt{now}_\psi = 1$, Agnes could decide to wait until Vera eventually executes C_2. Alternatively, she could decide that "if nothing happens before time 7, I shall execute A_1 at time 7." The decisions available to Agnes are called *real-time execution decisions* (RTEDs).

Definition 3 (RTED, for Agnes)**.** *Let ψ be a respectful partial schedule. An RTED for Agnes has one of two forms:* `wait` *or (T_f, χ_f). A* `wait` *decision is applicable if at least one contingent time-point, C, is* active *in ψ (i.e., C's activation time-point has already been executed, but C has not). A (T_f, χ_f) decision (i.e., "If nothing happens* before *time T_f, execute the time-points in χ_f at time T_f") is applicable if $T_f > \texttt{now}_\psi$ and χ_f is a non-empty subset of unexecuted free time-points (i.e., $\chi_f \neq \emptyset$ and $\chi_f \cap TPs(\psi) = \emptyset$).*

The kinds of decisions available to Vera are different in two important respects. First, Vera's version of an RTED—called an RTED*—allows a decision of the form, "if nothing happens before *or at* time T_u, then I shall execute the contingent time-points in the set $\chi_u \subseteq \mathcal{T}_u$ at time T_u." Note that when time T_u arrives, should Vera observe Agnes executing any time-points *at time T_u,* Vera has the option of *instantaneously changing her mind.* Second, in such cases, Vera may *instantaneously react* by executing some other contingent time-points *at time T_u*. Such decisions are called *instantaneous reactions.* For example, suppose Vera had decided that "if nothing happens before *or at* time 7, then I shall execute C_2 at time 7", but when time 7 arrived, she observed Agnes executing some time-point(s). Vera could withdraw her decision to execute C_2 and instantaneously react by deciding to execute some other contingent time-point(s) *at time* 7.

[2]Defns. 2 and 3 are drawn from Hunsberger (2009).

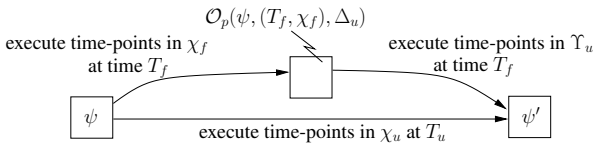


Figure 2: The outcome of Agnes' and Vera's decisions.

Definition 4 (RTED*, for Vera)**.** *Let ψ be a respectful partial schedule. A* before-or-at RTED *(RTED*) has one of two forms:* `wait` *or (T_u, χ_u). A* `wait` *decision is only applicable if no contingent time-points are currently activate in ψ. A (T_u, χ_u) decision (i.e., "If nothing happens* before-or-at *time T_u, I shall execute the time-points in χ_u at time T_u") is applicable only if $T_u > \texttt{now}_\psi$, and χ_u is a non-empty subset of currently-activated contingent time-points.*

Definition 5 (Instantaneous reaction, for Vera)**.** *Let ψ be a respectful partial schedule in which at least one contingent time-point is activated and whose execution window includes $\texttt{now}_\psi$. An* instantaneous reaction *is a decision (by Vera) to execute a set of such time-points at time $\texttt{now}_\psi$.*

To accommodate Vera's ability to react instantaneously, the outcome for a pair of decisions—one by Agnes, one by Vera—is defined in two stages: partial and full.

Definition 6 (Partial Outcome)**.** *Let ψ be a respectful partial schedule; let Δ_f be an RTED for Agnes; and let Δ_u be an RTED* for Vera. The* partial outcome, *$\mathcal{O}_p(\psi, \Delta_f, \Delta_u)$, is defined as follows.*[3]

(1a) $\mathcal{O}_p(\psi, \texttt{wait}, (T_u, \chi_u)) = \psi \cup \{(C, T_u) \mid C \in \chi_u\}$.
(1b) $\mathcal{O}_p(\psi, (T_f, \chi_f), (T_u, \chi_u)) = \psi \cup \{(C, T_u) \mid C \in \chi_u\}$, *if* $T_u < T_f$.
(2a) $\mathcal{O}_p(\psi, (T_f, \chi_f), \texttt{wait}) = \psi \cup \{(X, T_f) \mid X \in \chi_f\}$.
(2b) $\mathcal{O}_p(\psi, (T_f, \chi_f), (T_u, \chi_u)) = \psi \cup \{(X, T_f) \mid X \in \chi_f\}$, *if* $T_f \leq T_u$.

Note that in cases (1a) and (1b), the partial outcome includes only the execution of the contingent time-points in χ_u at time T_u. Cases (2a) and (2b) are analogous, in that the partial outcome includes only the execution of the free time-points in χ_f at time T_f, except that Vera is also able to instantaneously react by executing one or more contingent time-points, *also at time T_f,* as described below.

Definition 7 (Full Outcome)**.** *Let $\psi_p = \mathcal{O}_p(\psi, \Delta_f, \Delta_u)$ be a partial outcome, as described above; and let Υ_u be a set of contingent time-points that constitute an instantaneous reaction to ψ_p. The* full outcome, *$\mathcal{O}(\psi, \Delta_f, \Delta_u, \Upsilon_u)$, is the same as ψ_p, except that in cases (2a) and (2b), the schedule is augmented to include the execution of the time-points in Υ_u at time T_f.*

The possible paths from a partial schedule ψ to the full outcome $\psi' = \mathcal{O}(\psi, \Delta_f, \Delta_u, \Upsilon_u)$, are illustrated in Fig. 2.

[3]Note that it is impossible for a `wait` decision to be simultaneously applicable for Agnes and Vera.

$\psi = \{(A_2, 0), (X, 1)\}$; $\Delta_f = (7, \{A_1\})$; $\Delta_u = (6, \{C_2\})$.
$\psi' = \{(A_2, 0), (X, 1), (C_2, 6)\}$; Υ_u irrelevant.
$\psi = \{(A_2, 0), (X, 1)\}$; $\Delta_f = (7, \{A_1\})$; $\Delta_u = (8, \{C_2\})$.
$\psi' = \{(A_2, 0), (X, 1), (A_1, 7), (C_2, 7)\}$, where $\Upsilon_u = \{C_2\}$.
$\psi = \{(A_2, 0), (X, 1)\}$; $\Delta_f = (7, \{A_1\})$; $\Delta_u = (8, \{C_2\})$.
$\psi' = \{(A_2, 0), (X, 1), (A_1, 7)\}$, where $\Upsilon_u = \emptyset$.

Table 1: Full outcomes, ψ', for sample decisions.

Note that $\texttt{now}_{\psi'}$ is either T_f or T_u, depending on which path is taken. Note, too, that the full outcome, ψ', is typically a partial schedule, except at the very end when all of the time-points have been executed. Some partial execution sequences that might arise in the case of the sample STNU are illustrated in Table 1, where in each case, $\psi' = \mathcal{O}(\psi, \Delta_f, \Delta_u, \Upsilon_u)$.

Definition 8 (Execution Strategies)**.** *An* RTED-based strategy *(for Agnes) is a mapping from respectful partial schedules to RTEDs. An* RTED*-based strategy *(for Vera) is a pair of mappings, (f_1, f_2), where f_1 is a mapping from respectful partial schedules to RTED*s; and f_2 is a mapping from respectful partial schedules to instantaneous reactions.*

Definition 9 (Outcomes of Strategies)**.** *Let ψ be a respectful partial schedule; R an RTED-based strategy; and $R^\star = (f_1, f_2)$ an RTED*-based strategy. The one-step outcome is: $\mathcal{O}^1(\psi, R, R^\star) = \mathcal{O}(\psi, R(\psi), f_1(\psi), f_2(\psi_p))$, where $\psi_p = \mathcal{O}_p(\psi, R(\psi), f_1(\psi))$. The terminal outcome, $\mathcal{O}^*(\psi, R, R^\star)$, is the complete schedule that terminates the sequence recursively defined by: $\psi_0 = \emptyset$ and $\psi_{i+1} = \mathcal{O}^1(\psi_i, R, R^\star)$.*

The constraints on the decisions generated by $R^\star$—namely, that Vera must observe the bounds on the contingent durations—ensure that each ψ_i in the sequence will be respectful, given that $\psi_0 = \emptyset$ is trivially respectful.

Given the above execution semantics for STNUs, the definition of dynamic controllability is straightforward.

Definition 10 (Dynamic Controllability)**.** *An STNU, $(\mathcal{T}, \mathcal{C}, \mathcal{L})$, is* dynamically controllable *if there exists an RTED-based strategy R, such that for* all *RTED*-based strategies $R^\star$, the variable assignments in the complete schedule, $\mathcal{O}^*(\psi_0, R, R^\star)$, satisfy all of the constraints in $\mathcal{C}$.*

Theorem 1. *Defn. 10 is equivalent to the definition of dynamic controllability given by Hunsberger (2009).*

- *The proof is omitted due to space limitations.*

3 From STNUs to Timed Game Automata

A Timed Game Automaton (TGA) is a formalism used to model a game between two players—the controller and the environment—where transitions are subject to various kinds of temporal constraints (Maler, Pnueli, and Sifakis 1995). This section presents a novel encoding of STNUs into Timed Game Automata that preserves the property of dynamic controllability across the models. It first introduces relevant background and then formally describes the encoding.

Timed Game Automata

A *finite automaton* (Lewis and Papadimitriou 1998) comprises a finite set of states (or locations) and a finite set of

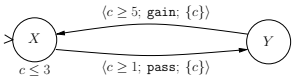


Figure 3: A sample timed automaton.

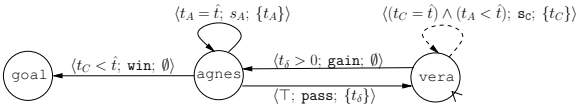


Figure 4: A sample Timed Game Automaton.

labeled transitions (or actions). One of the states is called the initial (or starting) state; a distinguished subset of states comprise the *final* (or *accepting*) states. Each labeled transition specifies a legal move from one state to another.

A *Timed Automaton* (TA) (Alur and Dill 1994) augments a finite automaton to include real-valued *clocks.* Each transition in a TA may include temporal constraints, called *guards,* that disable the transition if the current clock values do not satisfy those constraints. Each transition may also include *clock resets* that cause specified clocks to be reset to 0 whenever the transition is taken. Finally, each location may include an *invariant*—that is, a constraint specifying the conditions under which the automaton may stay in that location.

Fig. 3 shows a sample TA. The TA has one clock, c. The $>$ symbol indicates that X is the initial location. X's invariant is $c \leq 3$. Each transition has a label, $\langle G;\ N;\ R\rangle$, where G is the guard, N is a name for the transition, and R is the set of clocks it resets. A *run* starts at X, with $c = 0$. X's invariant, $c \leq 3$, and the guard, $c \geq 1$, on the `pass` transition, together ensure that the TA must take the transition to Y at some time when $1 \leq c \leq 3$. When taken, that transition resets c to 0. The `gain` transition can be taken back to X at any time at which $c \geq 5$. If taken, it will again reset c to 0. However, since Y has no invariant, the TA could remain at Y forever.

Definition 11 (Timed Automaton)**.** *A* Timed Automaton *(TA) is a tuple, $A = (L, l_0, Act, \mathcal{X}, E, Inv)$, such that: L is a finite set of locations, $l_0 \in L$ is the initial location, Act is a set of actions, $\mathcal{X}$ is a finite set of real-valued clocks, $E \subseteq L \times \mathcal{H}_k^\cap(\mathcal{X}) \times Act \times 2^{\mathcal{X}} \times L$ is a finite set of transitions, and $Inv : L \rightarrow \mathcal{H}_k^\cap(\mathcal{X})$ associates an* invariant *to each location. Elements in $\mathcal{H}_k^\cap(\mathcal{X})$ are conjunctions of constraints of the form, $x \bowtie k$ or $y - x \bowtie k$, where $x, y \in \mathcal{X}$, k is an integer, and $\bowtie$ is one of $<, \leq, =, >$ or $\geq$.*

A *Timed Game Automaton* (TGA) in turn generalizes a TA by partitioning the set of transitions into *controllable* and *uncontrollable* transitions. A TGA can be used to model a two-player game between an *agent* and the *environment,* where the agent controls the *controllable* transitions, and the environment controls the *uncontrollable* transitions.

Definition 12 (Timed Game Automaton)**.** *A Timed Game Automaton (TGA) is a Timed Automaton whose set of actions, Act, is partitioned into controllable (Act_c) and uncontrollable (Act_u) actions.*

Fig. 4 shows a TGA with three locations: `agnes`, `vera` and `goal`, where `vera` is the initial location. It has four

clocks: $t_A, t_C, \hat{t}$ and t_δ. The solid arrows represent controllable transitions; the dashed arrow represents the one uncontrollable transition. For example, the transition from agnes to itself has the label, $\langle t_A = \hat{t};\ \mathtt{s_A};\ \{t_A\}\rangle$, which specifies that it can only be taken if t_A and $\hat{t}$ have the same value; and that this transition resets t_A to 0. Consider the following possible run of this TGA. It begins at the initial location vera, with all clocks set to 0. Five units of time later, when all clocks read 5, the agent takes the gain transition to agnes. (The guard is satisfied; and no clocks get reset.) Then, at time 6, the agent takes the s_A transition, which causes t_A to be reset to 0. Then, at time 7, the agent takes the pass transition back to vera, which resets t_δ back to 0. At this point, $t_\delta = 0; t_A = 1$; and $t_C = \hat{t} = 7$. Thus, the environment can take the s_C transition from vera to itself, resetting t_C to 0. Then, at time 10, the agent takes the gain transition back to agnes, and at 11 the win transition to the goal state.

In what follows, the common practice of labeling certain locations as *urgent* is used. An urgent location is one in which players are prevented from *waiting*. An urgent location ℓ is equivalent to: (1) introducing a new clock c that is reset by every transition entering ℓ; and (2) conjoining a new invariant, $c \leq 0$, to ℓ.

For any TGA, different kinds of games can be modeled (Cassez et al. 2005). In a reachability game, the controller (or agent) seeks to move the TGA into one of the *winning locations* within a finite amount of time. In the avoidance game, the controller seeks to prevent the TGA from entering a certain set of locations. This paper focuses on *memoryless* strategies, since they have been shown to be sufficient for reachability and avoidance games (Maler, Pnueli, and Sifakis 1995; Cassez et al. 2005). Intuitively, a memoryless strategy associates a state of the system to either an action to be executed or a special symbol λ that stands for "wait" (i.e., do nothing; wait until something changes).

Definition 13. *For a TGA,* $(L, l_0, Act, \mathcal{X}, E, Inv)$*, a* memoryless strategy *is a mapping* $f : L \times \mathbb{R}^{\mathcal{X}}_{>=0} \to Act_c \cup \lambda$.

Further details on the semantics for TGAs are available from Maler et al. (1995).

Encoding an STNU into a TGA

This section presents an encoding of STNUs into TGAs. Given an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$, the goal is to provide a TGA $T_\mathcal{S} = (L, l_0, Act, \mathcal{X}, E, Inv)$, and a winning condition ϕ, such that the STNU $\mathcal{S}$ is dynamically controllable if and only if the TGA $T_\mathcal{S}$ admits a *counter-strategy* for ϕ. An important—and unexpected—part of this encoding is that *uncontrollable* TGA transitions are used to model the execution of the *free* time-points in $\mathcal{S}$, and *controllable* TGA transitions are used to model the execution of the *uncontrollable* time-points in $\mathcal{S}$. Thus, the traditional use of TGAs where the environment is associated with uncontrollable transitions has been inverted. (That is why a *counter-strategy* is sought.) The underlying reason is that according to the STNU semantics, when both players attempt to make transitions at the same time, Agnes must play before Vera, whereas in the TGA semantics, the uncontrollable transition would go first.

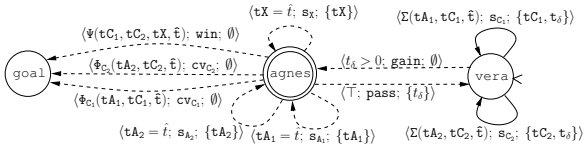


Figure 5: Encoding the STNU from Fig. 1 into a TGA. Solid arrows represent controllable transitions (for Vera); dashed arrows uncontrollable transitions (for Agnes). The doubly-circled agnes location is *urgent;* the initial location is vera.

The set of locations is: $L \doteq \{\mathtt{agnes}, \mathtt{vera}, \mathtt{goal}\}$, where agnes is marked *urgent.* Note that L has only three locations, regardless of the number of time-points in the STNU. Intuitively, agnes represents a state in which Agnes can execute one or more free time-points; vera represents a state in which Vera can execute one or more contingent time-points; and goal represents a state in which all of the constraints have been satisfied and the game is over. The initial location of the TGA is vera (i.e., $l_0 \doteq \mathtt{vera}$).

The set of clocks is: $\mathcal{X} \doteq \{\hat{\mathtt{t}}, \mathtt{t}_\delta\} \cup \{\mathtt{tX} \mid X \in \mathcal{T}\}$. All clocks start at 0. The clock $\hat{\mathtt{t}}$ is never reset; it simply measures global time. The clock $\mathtt{t}_\delta$ is used to ensure that there will always be a positive delay between the execution of any contingent time-point (by Vera) and any *reaction* by Agnes. (This is crucial for capturing the STNU semantics.) Finally, for each time-point $X \in \mathcal{T}$, there is a corresponding clock tX. That clock is reset at most once each run, at the instant X is executed. It follows that any time-point X has been executed if and only if $\mathtt{tX} < \hat{\mathtt{t}}$. (Since the initial state is vera, no time-point can be executed at 0.) Also, after being executed, the execution time for X is forever equal to $\hat{\mathtt{t}} - \mathtt{tX}$.

The sets of controllable and uncontrollable actions are defined as follows. First, the controllable actions (for Vera) consist of one action for each contingent time-point in $\mathcal{S}$, as follows: $Act_c \doteq \{s_X \mid X \in \mathcal{T}_u\}$. Each action in this set represents the execution of the corresponding time-point. The uncontrollable actions (for Agnes) include more options: $Act_u \doteq \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3$, where: $\mathcal{A}_1 = \{s_X \mid X \in \mathcal{T}_f\}$; $\mathcal{A}_2 = \{cv_C \mid (A, \ell, u, C) \in \mathcal{L}\}$; and $\mathcal{A}_3 = \{\mathtt{gain}, \mathtt{pass}, \mathtt{win}\}$. $\mathcal{A}_1$ contains one execution action for each free time-point. $\mathcal{A}_2$ contains one action for each contingent link; these actions are only enabled if Vera violates the bounds on her contingent links. gain and pass model the interplay between the execution of time-points by Agnes and Vera; win is used at the end when all time-points have been executed and all constraints have been satisfied.

The transition relation, E, for the TGA encoding of an STNU is demonstrated in Fig. 5, using the sample STNU from Fig. 1. For each free time-point X, there is a transition from agnes to agnes labeled by $\langle \mathtt{tX} = \hat{\mathtt{t}};\ \mathtt{s_X};\ \{\mathtt{tX}\}\rangle$, which represents the execution of X by Agnes. The *guard,* $\mathtt{tX} = \hat{\mathtt{t}}$ (i.e., X not yet executed), ensures that this transition will be taken at most once per run. The set, $\{\mathtt{tX}\}$, stipulates that the clock tX will be reset by this transition, signalling that X has been executed. Similarly, for each contingent link, (A, ℓ, u, C), there is a transition from vera to vera labeled by $\langle \Sigma(\mathtt{tC}, \mathtt{tA}, \hat{\mathtt{t}});\ \mathtt{s_C};\ \{\mathtt{tC}, \mathtt{t}_\delta\}\rangle$,

which represents the execution of C by Vera. The guard, $\Sigma(\mathtt{tC}, \mathtt{tA}, \hat{\mathtt{t}}) \doteq (\mathtt{tA} < \hat{\mathtt{t}}) \wedge (\mathtt{tC} = \hat{\mathtt{t}}) \wedge (\mathtt{tA} \geq \ell) \wedge (\mathtt{tA} \leq u)$, ensures that this transition can only be taken when the link is currently activated and its duration would fall within $[\ell, u]$. In addition, for each contingent link, (A, ℓ, u, C), there is a transition from `agnes` to `goal` labeled by $\langle \Phi_C(\mathtt{tA}, \mathtt{tC}, \hat{\mathtt{t}});\ \mathtt{cv_C};\ \emptyset \rangle$, enabling Agnes to move to `goal` should Vera ever violate the bounds on that link by failing to execute C. The guard is: $\Phi_C(\mathtt{tA}, \mathtt{tC}, \hat{\mathtt{t}}) \doteq (\mathtt{tA} < \hat{\mathtt{t}}) \wedge (\mathtt{tA} > u) \wedge (\mathtt{tC} = \hat{\mathtt{t}})$. Next, if $\vec{t}$ is the vector of clocks $\mathtt{tX}$ such that $X \in \mathcal{T}$, the transition from `agnes` to `goal` labeled by $\langle \Psi(\vec{t}, \hat{\mathtt{t}});\ \mathtt{win};\ \emptyset \rangle$ signals the end of the game. $\Psi(\vec{t}, \hat{\mathtt{t}})$ models that all time-points have been executed and all constraints are satisfied: $\Psi(\vec{t}, \hat{\mathtt{t}}) \doteq \bigwedge_{x \in \mathcal{T}} (\mathtt{tX} < \hat{\mathtt{t}}) \wedge \bigwedge_{Y - X \leq k} (\mathtt{tX} - \mathtt{tY} \leq k)$. Last, to model the interplay between the players, there are two more transitions. The transition from `vera` to `agnes` labeled by $\langle \mathtt{t}_\delta > 0;\ \mathtt{gain};\ \emptyset \rangle$ enables Agnes to gain control for the purpose of executing some free time-points—but only after some positive delay since Vera last executed a contingent time-point. The transition from `agnes` to `vera` labeled by $\langle \top;\ \mathtt{pass};\ \{\mathtt{t}_\delta\} \rangle$ enables Agnes to immediately pass back to `vera`, once she has finished executing her chosen time-points. Crucially, no time elapses from the instant Agnes leaves `vera` to the instant she returns, because `agnes` is an urgent state. From Vera's perspective, the winning condition ϕ of the (safety) game is to avoid the `goal` state. A counter-strategy for Agnes foils Vera by ensuring that `goal` can be reached.

4 Correctness

This section presents theoretical results that confirm the correctness of the encoding, and the correspondence between strategies for the STNU and its TGA counterpart.

Theorem 2. *Let $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be any STNU; and let Θ be the encoding of $\mathcal{S}$ as a TGA, as described in Section 3. Then Θ correctly captures the execution semantics for $\mathcal{S}$ in the sense that any sequence of partial schedules that can be generated for $\mathcal{S}$ according to the execution semantics for STNUs corresponds to a run for Θ that can be generated by following its transitions according to the TGA semantics.*

Proof. The following invariant is proved by induction. Each respectful partial schedule ψ that can be generated for $\mathcal{S}$ corresponds to a state of Θ in which the location is `vera`, $\mathtt{t}_\delta = 0$, $\mathtt{now}_\psi = \hat{\mathtt{t}}$, for each executed time-point X, $\psi(X) = \hat{\mathtt{t}} - \mathtt{tX}$, and for each unexecuted time-point Y, $\psi(Y) = \hat{\mathtt{t}}$. For the base case, the initial partial schedule, $\psi_0 = \emptyset$, corresponds to the initial state of Θ in which the location is `vera`, all clocks are at zero, and all time-points are unexecuted. Note that ψ_0 is trivially respectful.

Now, suppose that ψ is a respectful partial schedule that can be generated according to the execution semantics for STNUs, and that satisfies the hypothesized invariant. Let θ be the corresponding state of the TGA. Since $\mathtt{t}_\delta = 0$, the only transitions that are immediately enabled are the loops whereby contingent time-points are executed. These transitions, if taken, correspond to the instantaneous reaction decisions for Vera, in which a set Υ_u of one or more contingent time-points can be executed simultaneously. However, suppose that Vera does not make any such transitions at $\mathtt{t}_\delta = 0$. Once $\mathtt{t}_\delta > 0$, both Agnes and Vera have transitions that they could make at any time. For example, Vera might decide to execute one or more contingent time-points when $\mathtt{t}_\delta = 3$. That would correspond to an RTED*-based decision, (T_u, χ_u), where $T_u = \mathtt{now}_\psi + 3$ and χ_u contains the time-points to be executed. Since each transition by Vera resets $\mathtt{t}_\delta$ to 0, Agnes is unable to interrupt Vera's simultaneous execution of contingent time-points. The resulting outcomes are equivalent to the partial schedules that arise in Cases (1a) and (1b) of Defn. 6. The guards on Vera's transitions, which enforce the durational bounds for the contingent links, ensure that the resulting partial schedule is respectful. Also, when Vera's sequence of "simultaneous" transitions complete, $\hat{\mathtt{t}}$ equals the time of the most recent execution (i.e., $\mathtt{now}_\psi + 3$). In addition, for each newly executed time-point, C, the clock $\mathtt{tC}$ is set to 0, ensuring that $\hat{\mathtt{t}} - \mathtt{tC}$ equals the execution time of C. Since both clocks will never again be reset, this difference remains fixed forever.

On the other hand, suppose that Agnes decided to execute the time-points in χ_f at an earlier time, say, $\mathtt{now}_\psi + 2$. This would correspond to her making the transition to the `agnes` location and instantaneously executing the time-points in χ_f at that time and, then, immediately returning to the `vera` location. Since `agnes` is an *urgent* state, the global clock equals $\mathtt{now}_\psi + 2$ when the return transition is made. This sequence of transitions corresponds to the *partial* outcomes in Cases (2a) and (2b) in Defn. 6, where Agnes' decision is (T_f, χ_f), where $T_f = \mathtt{now}_\psi + 2$. Furthermore, if Vera chooses to instantaneously execute some contingent time-points at that same time, $\mathtt{now}_\psi + 2$, that will correspond to an instantaneous reaction, as specified in Defn. 5.

Finally, if at time $\mathtt{now}_\psi$, Agnes and Vera both decided to execute some time-points at time $\mathtt{now}_\psi + 1$, then the STNU semantics ensures that Agnes' time-points will be executed, and that Vera will be able to instantaneously react, if she chooses. This corresponds to Agnes' transition having *priority* over Vera's transition. Agnes transitions to the `agnes` state, executes her time-points, and returns to the `vera` state, with the global clock ending up at $\mathtt{now}_\psi + 1$.

Since, in all cases, the resulting state of the TGA satisfies the desired invariant property, the result is proven. □

Theorem 3. *Let $\mathcal{S}$ be any STNU; let Θ be the encoding of $\mathcal{S}$; and let σ be a winning TGA counter-strategy for Agnes. Then there is an equivalent RTED-based strategy for Agnes that will ensure the satisfaction of all constraints in $\mathcal{S}$ no matter how the contingent durations turn out.*

Proof. Let $\mathcal{S}, \Theta$ and σ be as described in the statement above. Therefore, $\sigma : L \times \mathbb{R}^{\mathcal{X}}_{>=0} \rightarrow Act_u \cup \{\lambda\}$, where Act_u is the set of uncontrollable actions (for Agnes).

Suppose the TGA has just entered the state, $(\mathtt{vera}, v)$, where v represents the vector of clock values. As has already been noted, for any time-point X and associated clock $\mathtt{tX}$: (1) before X executes, $\mathtt{tX} = \hat{\mathtt{t}}$; and (2) after X executes, $\mathtt{tX} < \hat{\mathtt{t}}$ and the fixed difference, $\hat{\mathtt{t}} - \mathtt{tX}$, equals the time at

which X executed. Thus, the vector of clock values specifies a partial schedule, ψ.

Now, suppose that $\texttt{now}_\psi < \hat{\texttt{t}}$ (i.e., that some positive time has elapsed since the last execution event in ψ). The only way that could have happened is if the state $(\texttt{vera}, v)$ had been preceded by one or more useless loops (i.e., loops using only the `gain` and `pass` transitions to go back and forth between `vera` and `agnes` without executing any time-points). Let $(\texttt{vera}, v')$ be the state immediately preceding the first such useless loop. Then for some positive ϵ, $v = v' + \epsilon$ (i.e., the clock values in v are ϵ units larger than their corresponding values in v'). And by construction, $\texttt{now}_\psi = v'(\hat{\texttt{t}})$.

Next, let D be the minimum time that can elapse from v before the strategy σ recommends a non-trivial transition to the `agnes` location. That is: $D = \min\{d \mid \sigma(\texttt{vera}, v' + d) \neq \lambda, \sigma(\texttt{agnes}, v' + d) \neq \texttt{pass}\}$. Let $v_0 = v' + D$. The unique sequence of *execution* transitions at the `agnes` location is: $\tau_1 = \sigma(\texttt{agnes}, v_0)$, $\tau_2 = \sigma(\texttt{agnes}, v_1), \tau_3 = \sigma(\texttt{agnes}, v_2), \ldots$, where each v_{i+1} is the same as v_i, except that the clock for the just-executed time-point is 0 in v_{i+1}. This sequence must terminate, since there are only finitely many time-points, and each can be executed only once. If τ_m is the last execution transition, it follows that $\texttt{pass} = \sigma(\texttt{agnes}, v_m)$. That transition leads back to the state, $(\texttt{vera}, v_m)$, where v_m is the same as v', except that the clocks for the time-points executed by the transitions, $\tau_1, \ldots, \tau_m$, are all zero in v_m.

Next, let $T_f = v_0(\hat{\texttt{t}})$ be the global time at which σ recommends its first non-trivial transition to `agnes`; and let χ_f be the set of time-points that correspond to the execution transitions, $\tau_1, \ldots, \tau_m$. Then (T_f, χ_f) is an RTED for ψ that corresponds to what the strategy σ recommends at $(\texttt{vera}, v')$. Note that Vera may decide to instantaneously react by executing some contingent time-points also at time T_f, an outcome that is sanctioned by the execution semantics for STNUs. Finally, it may happen that Vera decides to intervene *before* time T_f arrives, by executing one or more contingent time-points and effectively generating a new partial schedule, ψ^*. In that case, the same procedure could be applied to ψ^* to generate an appropriate RTED. Since the guard on the transition from `vera` to `agnes` requires a positive time delay, that RTED is properly prohibited from any kind of instantaneous reaction (by Agnes).

This procedure provides a mapping from any $(\texttt{vera}, v)$ state that is reachable following the winning strategy σ. In addition, the sequences of partial schedules generated by following the RTEDs correspond to runs that can be produced by σ. Thus, the complete schedules generated by the RTEDs are guaranteed to satisfy all STNU constraints assuming Vera observes the bounds on all contingent links. □

5 Implementation and Experiments

The encoding presented in this paper has a clear theoretical importance, but it can also be used to synthesize executable strategies for a dynamically controllable STNU. In previous approaches (Hunsberger 2010; 2013a), Agnes is required to reason at run-time to decide the next action to execute. With our STNU-to-TGA encoding, we can produce a strategy for the TGA using classical techniques and transform it into an executable form. In principle, it is also possible produce a hardware circuit that implements the strategy. This is important for domains such as spacecraft control, where the platform is subject to stringent safety constraints and resource bounds that preclude the real-time use of $O(N^3)$ algorithms. For such domains, the ability to synthesize a fixed but provably correct, simple-to-execute strategy is essential. In addition, avoiding the reasoner in the executor allows for the use of existing techniques for the validation and certification of the executor itself.

We show the feasibility of this approach by implementing a proto-type toolset[4] that allows for the production of a dynamic strategy compiled into executable code starting from an STNU. The flow of the toolset is as follows. First, an encoder takes an STNU and produces a TGA in the format used by the UPPAAL-TIGA tool (Cassez et al. 2005). UPPAAL-TIGA is able to solve the TGA, and returns a counter-strategy for Agnes iff the STNU is dynamically controllable. Then, a compiler takes the TGA counter-strategy and produces a function in C code.

We ran our toolset on two sets of benchmarks for a total of 1028 STNU instances. The first set was taken from Cimatti et al. (2012b). The second is composed of 14 non-DC "magic-loop" instances as described by Hunsberger (2013b) and 14 DC "near-magic-loop" instances. We used an Intel Xeon E31270 @3.40GHz workstation setting the timeout to 400 seconds and a memory limit of 4 GB. We produced an executable strategy for 370 instances and proved that 49 instances were not dynamically controllable. The other instances reached the timeout limit.

We also designed an experiment to evaluate the speed of the generated strategies. We implemented a program that simulates the execution of an STNU by randomly generating durations for the contingent links and applying a given strategy. We measured the execution time of all the 370 generated strategies running each of them on 100 randomly generated situations. The results show that the execution times are negligible, always below 0.0003 seconds.

We highlight that the TGA check is the bottleneck of the entire toolset and causes all the observed timeouts. Nevertheless, the presented proto-typical toolset shows the feasibility of strategy synthesis using our TGA encoding.

6 Conclusion and Future Work

This paper presented a novel approach to synthesizing execution strategies for STNUs. The approach is based on a reduction to Timed Game Automata that we prove correct and complete. Our encoding is linear in the size of the STNU and allows the construction of strategies that can be directly executed without run-time reasoning. In the future, we plan to extend this work by customizing the synthesis technique in order to scale up the efficiency. We also plan to generalize the approach to deal with disjunctive temporal networks, for which no sound-and-complete dynamic controllability checking algorithm has yet been presented in the literature.

[4]The toolset and all the benchmarks are available at: `https://es.fbk.eu/people/roveri/tests/aaai14`.

References

Abdeddaim, Y.; Asarin, E.; and Sighireanu, M. 2009. Simple algorithm for simple timed games. In *TIME*, 99–106.

Alur, R., and Dill, D. L. 1994. A theory of timed automata. *Theoretical Computer Science* 126(2):183–235.

Cassez, F.; David, A.; Fleury, E.; Larsen, K. G.; and Lime, D. 2005. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, 66–80.

Cesta, A.; Fratini, S.; Orlandini, A.; and Finzi, A. 2011. Flexible plan verification: Feasibility results. *Fundamenta Informaticae* 107(2–3):111–137.

Cimatti, A.; Micheli, A.; and Roveri, M. 2012a. Solving temporal problems using SMT: strong controllability. In *CP*, 248–264.

Cimatti, A.; Micheli, A.; and Roveri, M. 2012b. Solving temporal problems using SMT: weak controllability. In *AAAI*, 448–454.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Henzinger, T. A.; Manna, Z.; and Pnueli, A. 1991. Timed transition systems. In *REX Workshop, LNCS 600*, 226–251.

Hunsberger, L. 2009. Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In *TIME*, 155–162.

Hunsberger, L. 2010. A fast incremental algorithm for managing the execution of dynamically controllable temporal networks. In *TIME*, 121–128.

Hunsberger, L. 2013a. A faster execution algorithm for dynamically controllable stnus. In *TIME*, 26–33.

Hunsberger, L. 2013b. Magic loops in simple temporal networks with uncertainty. In *ICAART*, 157–170.

Lewis, H. R., and Papadimitriou, C. H. 1998. *Elements of the Theorey of Computation*. Prentice-Hall, Inc., 2 edition.

Maler, O.; Pnueli, A.; and Sifakis, J. 1995. On the synthesis of discrete controllers for timed systems. In *STACS*, 229–242.

Morris, P. H., and Muscettola, N. 2005. Temporal dynamic controllability revisited. In *AAAI*, 1193–1198.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *IJCAI*, 494–499.

Morris, P. 2006. A structural characterization of temporal dynamic controllability. In *CP*, 375–389.

Orlandini, A.; Finzi, A.; Cesta, A.; and Fratini, S. 2011. Tga-based controllers for flexible plan execution. In *KI*, number 7006 in LNAI. Springer-Verlag. 233–245.

Tsamardinos, I., and Pollack, M. E. 2003. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence* 151:43–49.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence* 11(1):23–45.

Vidal, T. 2000. Controllability characterization and checking in contingent temporal constraint networks. In *KR*, 559–570.