Extending Timed Automaton and Real-Time Logic to Many-Valued Reasoning*

Ana Fernández Vilas, José J. Pazos Arias, and Rebeca P. Díaz Redondo

Departamento de Ingeniería Telemática
Universidad de Vigo. 36200, Vigo, Spain
{avilas,jose,rebeca}@det.uvigo.es
http://www-gris.det.uvigo.es

Abstract. Past decade has witnessed a great advance in the field of dense-time formal methods for the specification, synthesis and analysis of real time systems. In this context timed automata and real-time temporal logics provide a simple, and yet general, way to model and specify the behavior of these systems. At the same time, iterative and incremental development has been massively adopted in professional practice. In order to get closer to this current trend, timed formal methods should be adapted to such lifecycle structures, getting over their traditional role of verifying that a model meets a set of fixed requirements. We advocate the suitability of many-valued reasoning in order to achieve this goal; in the scope of knowledge representation, many-valued reasoning is suitable to deal with both uncertainty and disagreement which are pervasive and desirable in an iterative and incremental design process. In this respect, this paper introduces SCTL/MUS-T methodology as an extension of timed automata and temporal logic theories to many-valued reasoning in real-time.

1 Introduction

Timed formal methods for the specification, analysis and verification of real-time systems has been around for a long time. Nowadays, there is a broad consensus [1–3] that dense-time approaches are more expressive and suitable for composition and refinement since a quantum of time is not needed to be fixed a priori. In addition to flexibility, decision procedures for dense-time are increasingly efficient every day.

The dense formal approach to real-time systems has been carried out extending a large amount of formalisms studied at length, which differ on their methods, aims, and abstraction-levels: temporal logics, first order logics, state-machines, process algebras, synchronous languages, etc. In the context of timed formal methods in general, and specifically of specification of real time systems in terms of timed process algebras and real-time logics, timed automata provide a simple, and yet general, way to model the behavior of real-time systems.

* This work was partially supported by the Xunta de Galicia Basic Research Project PGIDT01PX132203PR.

W. Damm and E.-R. Olderog (Eds.): FTRTFT 2002, LNCS 2469, pp. 185–204, 2002.

While timed automata can be considered the standard timed state-machines, for what concerns properties, the situation seems not to be so clear: a variety of logics have been applied to requirements specification [4]. As far as analysis is concerned, formal techniques assuming dense time have their early origin in [5]. In this work, *Alur et al* introduced the first model checking algorithm for timed automata with real-valued clocks. The main idea behind this algorithm is the construction of an abstract finite state-space, called region graph, from the dense state-space of a timed automaton. Since obtaining an accurate finite model of the system was revealed as possible, many works have extended this solution to other kinds of analysis.

Shortly after theoretical results appeared, verification tools were developed mainly in academic area (KRONOS[1] followed by UPPAAL[2]). Both tools are based on timed automaton to a large extent, although they differ in their property specification languages. Real case studies, like the ones tackled by UPPAAL and KRONOS, have highlighted timed models and methods are ready for practitioners' usage. However, these timed theoretical foundations come up against industry reluctance about the adoption of formal methods in the software process; technology-transfer problems, which had been observed in conventional untimed systems, appear once again. In order to get closer to professional practice, formal methods are in urgent need of being incorporated into the software process. In this respect, iterative and incremental development is a major milestone.

Formalizing Incremental Design by Many-Valued Reasoning. Despite the origins of many-valued reasoning can be traced back to antiquity, its application to computer science is almost exceptional. For instance, many-valued reasoning and specifically many-valued logics are known to support the explicit modeling of uncertainty and disagreement by allowing additional truth values outside bivalence principle [6].

In spite of being a common practice in software engineering, and its proven reduction in time to market, integration of iterative and incremental development with formal methods is still immature. Research has not provided the formal bases and methodologies enabling this paradigm. As a tool for knowledge representation, we postulate that many-valued reasoning fits in appropriately with an incremental and iterative specification process. In incremental specification the system gradually takes form as more is learned about the problem, that is, both uncertainty and disagreement are pervasive and desirable in this process.

Many-valued reasoning, in its role of knowledge formalizer, has been studied in SCTL/MUS [7] methodology in order to support a formal lifecycle which is iterative and incremental in the specification process. On these bases, real-time SCTL/MUS (SCTL/MUS-T) is introduced in this paper, extending timed automaton (by defining MUS-T graphs) and real-time temporal logic (by defining SCTL-T temporal logic) to many-valued reasoning as a means of formalizing the incremental and iterative nature of the process.

[1] http://www-verimag.imag.fr/TEMPORISE/kronos/
[2] http://www.docs.uu.se/docs/rtmv/uppaal/

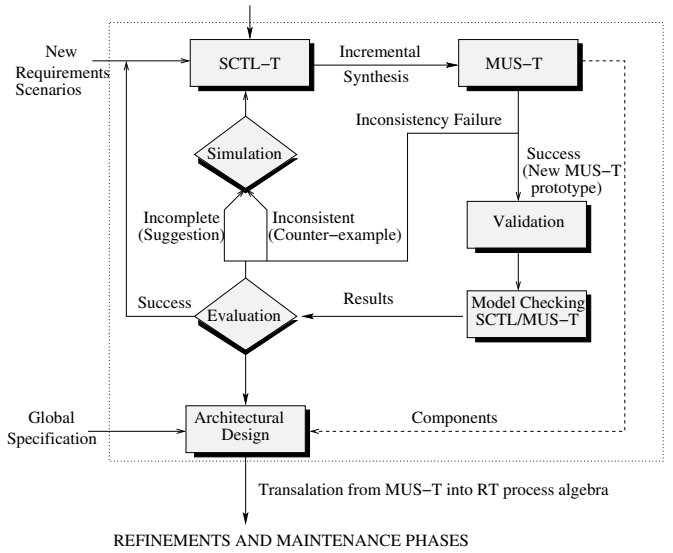


Fig. 1. *Initial* phase of the SCTL/MUS-T formalized lifecycle

The formalized lifecycle model consists, at a very abstract level, of three macro-phases: *initial*, *refinements* and *maintenance*. At the *initial* macro-phase (figure 1) the system is iteratively designed; a model-oriented specification (MUS-T graph, section 2) is obtained by incrementally incorporating timing scenarios identified on the target solution. At every iteration, the user identifies not only requirements, but typical scenarios of the system that is formally specified by means of a many-valued real-time logic (SCTL-T, section 3). New scenarios, which lead to a growth in the system functionality, are **synthesized** (if feasible) in the system modeled as a MUS-T graph. Also, requirements are **model checked** in the current model, the one in the current cycle, in order to decide: if the system already satisfies the requirements; if it is not able to provide, in a future design state, these requirements from the current design (inconsistency); or, if the system does not satisfy the requirements, but it is able to do it (incompleteness). Model checking and synthesis methods are introduced in section 4. Finally, in section 5, we illustrate SCTL/MUS-T methodology by means of examples extracted from the steam-boiler case study in [8].

Despite this paper focuses on the *initial* phase, the picture is completed as follows. Outcome of the *initial* phase is a set of MUS-T components which are transferred to a *refinement* phase (by translation into a real-time process algebra) where architecture decisions are incorporated and a more detailed design is successively reached, also iterative and incrementally. Finally, after delivering the system, *maintenance* turns into another development phase (*initial* + *refinements*) whose starting point is the previously developed system which is operating.

2 MUS-T: Model for Incomplete Real-Time Systems

For modeling real-time systems we define MUS-T graphs (**T**imed **M**odel of **U**nspecified **S**tates), timed extension of MUS models proposed in [7]. This extension defines a dense-time model similar to the timed automaton in [5], but event-driven. However, whereas atomic propositions on a location of a timed automaton are assertions being true or false, in order to support incompleteness and consistency-checking in an incremental process, timed events in a location of a MUS-T model can be characterized as possible (true), non-possible (false) or non-specified (unspecified).

Syntax. Before defining MUS-T graphs, some definitions concerning clocks and valuations are in order. A clock is a simple real-valued variable drawn from the finite set $\mathcal{C}$. A clock constraint $\psi \in \Psi(\mathcal{C})$ is a boolean combination of atomic formulas of the from $x \prec c$ or $x - y \prec c$, where x, y are clocks, c is an integer constant, and $\prec$ is taken from $\{\leq, \geq, <, >, =\}$. A clock valuation γ is a point in $\Re_+^{\#\mathcal{C}}$. If γ is a clock valuation, $\gamma+\tau$ for some $\tau \in \Re_+$ stands for the clock valuation obtained by adding τ to the value of every clock. Given a clock constraint ψ, $\psi(\gamma) = 1$ iff γ satisfies the clock constraint ψ, otherwise $\psi(\gamma) = 0$. Finally, let $\lambda \subseteq \mathcal{C}$ be a set of clocks, then γ_λ is the clock valuation obtained from γ by setting every clock in λ to 0. A MUS-T graph $\mathcal{M}$ is a 6-tuple $\langle s_0, S, T, I, A, \mathcal{C} \rangle$ over $\mathcal{L}_3$, where:

- $\mathcal{L}_3 = \{0, \frac{1}{2}, 1\}$ is the truth set which establishes the specification condition –possible (1), non-possible (0) or unspecified ($\frac{1}{2}$)– of the transitions in the graph.
- A is a finite set of events.
- $\mathcal{C}$ is a finite set of real-valued clocks.
- S is a finite set of locations, including two fictitious locations referred to as unspecified (s_u) and non-possible drain (s_{np}).
- s_0 is the initial location.
- $T \subseteq S \times A \times \Psi(\mathcal{C}) \times 2^{\mathcal{C}} \times S$ is the transition relation; and $CS : T \to \mathcal{L}_3$ is a total function which assigns a truth value (specification condition) to every transition.
- I_p and I_{np}: $S \to \Psi(\mathcal{C})$ are functions which assign to every location $s \in S$ a possible invariant $I_p(s)$, establishing when the progress of the time is possible in the location; and a non-possible invariant $I_{np}(s)$, establishing temporal contexts in which time cannot progress.

Every transition $t \in T$ with $CS(t) = 1$ defines a possible transition $(s, < a, g_p(s,a), \lambda >, s')$ with source location s and target location s'. The timed event $< a, g_p(s,a), \lambda >$ specifies an event $a \in A$, a possible guard $g_p(s,a) \in \Psi(\mathcal{C})$ and a subset $\lambda \subseteq \mathcal{C}$ of clocks to reset in the transition. In the same way, every $t \in T$ with $CS(t) = \frac{1}{2}$ defines a partially-unspecified transition $(s, < a, g_u^P(s,a), \lambda >, s')$ with source location s, target location s' and an unspecified guarded event $< a, g_u^P(s,a), \lambda >$. Finally, transitions $t \in T$ with $CS(t) = 0$ define non-possible

transitions $(s, < a, g_{np}(s,a), \{\} >, s_{np})$, with the non-possible drain s_{np} as fictitious target location.

Specification conditions partition the valuation space $\Re_+^{\#\mathcal{C}}$ in three subsets (possible, non-possible and unspecified). We enforce that partition to be complete and disjoint. For completeness, the totally-unspecified guard g_u^T is defined:

$$g_u^T(s,a) = \neg(\bigvee_i g_{p_i}(s,a)) \wedge \neg(\bigvee_j g_{u_j}^P(s,a)) \wedge \neg(g_{np}(s,a))$$

In this way, given a location s and an event a, valuation subsets defined by g_p, g_{np}, g_u^P, and g_u^T cover $\Re_+^{\#\mathcal{C}}$. $g_u^T(s,a)$ implicitly defines a totally-unspecified transition $(s, < a, g_u^T(s,a), \{\} >, s_u)$. Also, the unspecified invariant, I_u, is defined as $I_u(s) = \neg I_p(s) \wedge \neg I_{np}(s)$; given a location s, the subsets defined by I_p, I_{np} and I_u cover $\Re_+^{\#\mathcal{C}}$.

The drain-locations (s_u, s_{np}) are the fictitious target locations of non-transitions in a MUS-T graph, i.e., non-possible transitions and totally-unspecified transitions. However, their nature is definitively different, the drain-location s_{np} is zero-evolution, and the drain-location s_u is maximal-evolution. That is, $\forall\ a \in A$, $g_u^T(s_u, a) = true$ whereas $g_{np}(s_{np}, a) = true$. Similarly, $I_u(s_u) = true$ and $I_{np}(s_{np}) = true$.

Semantics. The semantics for a MUS-T graph is given in terms of a many-valued labeled transition system, which is often called dense due to the time domain. Formally, every MUS-T graph induces a dense many-valued transition system $\mathcal{S}_\mathcal{M} = \langle (s_0, \gamma^0), \mathcal{ST} = \{S \times \Re_+^{\#C}\}, \mathcal{T}, A, \mathcal{C} \rangle$, over the truth set $\mathcal{L}_3$, where $\mathcal{ST}$ is the set of timed states, i.e., pairs of the form (s, γ) with $s \in S$ and $\gamma \in \Re_+^{\#\mathcal{C}}$; (s_0, γ^0) is the timed initial state with γ_0 assigning 0 to every clock; and $\mathcal{T} \subseteq \mathcal{ST} \times (A \cup \Re_+) \times \mathcal{L}_3 \times \mathcal{ST}$ is the transition relation. The transition relation $\mathcal{T}$ identifies a source and a target timed state $\in \mathcal{ST}$; a label $a \in A$ (discrete transition) or $\tau \in \Re_+$ (temporal transition); and a specification condition $\in \mathcal{L}_3$.

In general, dense time models assume orthogonality between discrete and dense changes. An execution of a timed model is a sequence of steps which alternate dense temporal transitions (incrementing the valuation an arbitrary amount of time); and discrete transitions (from one location to another). Preserving this orthogonal assumption but following the many-valued nature of a MUS-T graph, transitions $\in \mathcal{T}$ are characterized as possible (1), unspecified ($\frac{1}{2}$) or non-possible ones (0). Clock constraints on locations (invariants) and transitions (guards) implicitly define the total function $\mathcal{CS} : \mathcal{T} \to \mathcal{L}_3$, assigning a specification condition c_s to every transition $t \in \mathcal{T}$. Intuitively, the model can progress in a temporal way in a location s, with $c_s = 1$, as long as $I_p(s)$ is satisfied; with $c_s = 0$, as long as $I_{np}(s)$ is satisfied in some point; otherwise with $c_s = \frac{1}{2}$. Guards on transitions are enabling conditions, i.e. a transition $t \in T$ of a MUS-T graph can only be taken, with specification condition $c_s = CS(t)$, provided that clock constraints defining its guard are satisfied. Also, when a transition $\in T$ occurs, all clocks in the set of clocks labeling it are reset to 0. In this way, a MUS-T graph is a many-valued timed automaton.

3 SCTL-T

We will specify requirements and scenarios of incomplete real-time systems using the temporal logic SCTL-T (**T**imed **S**imple **C**ausal **T**emporal **L**ogic), a dense real-time extension of the causal and many-valued temporal logic SCTL [7]. SCTL-T formulas match the causal pattern **Premise** Æ ⊗ **Consequence**. This generic causal formula establishes a causing condition (Premise); a temporal operator which determines the applicability of the cause (⊗); a condition which is the effect (Consequence); and a quantifier which determines the degree of satisfaction of the consequence on the applicability set (Æ). We now present the syntax and semantics of SCTL-T in three stages. The first stage introduces the quasi-boolean algebra L_6 (section 3.1), the second one defines syntax and semantics of propositional and temporal operators in SCTL-T (section 3.2); and the third one deals with recursion (section 3.3). Finally, in section 3.4 we introduce an imperative version of SCTL-T which is synthesis-oriented.

3.1 MPU: Algebra of Middle Point Uncertainty

Apart from being causal, SCTL-T is many-valued. SCTL-T semantics is given over the partially ordered set $(\mathcal{L}_6, \leq)$ of truth values, where $\mathcal{L}_6 = \{0, \frac{1}{4}, \widehat{\frac{1}{2}}, \frac{1}{2}, \frac{3}{4}, 1\}$ (Hasse diagram of figure 2). Every two elements $a, b \in \mathcal{L}_6$ have a least upper bound, $a \vee b = \sup\{a, b\}$, and a greatest lower bound $a \wedge b = \inf\{a, b\}$. Besides, the unary operation $\neg$ is defined by horizontal symmetry.

The 4-tuple $L_6 = (\mathcal{L}_6, \wedge, \vee, \neg)$ has the structure of a quasi-boolean algebra called algebra of **M**iddle **P**oint **U**ncertainty (MPU, see [7] for details). That is, the reduced algebra $(\mathcal{L}_6, \vee, \wedge)$ is a distributive lattice and the unary operation $\neg$ satisfies De Morgan, involution and antimonotonic properties. For the quasi-boolean algebra L_6, we define the causal operation, $\rightarrow$, as follows:

$$a, b \in \mathcal{L}_6, \ \rightarrow (a, b) = \left(a \vee \widehat{\frac{1}{2}}\right) \wedge \left(\left(\neg\, a \wedge \widehat{\frac{1}{2}}\right) \vee b\right)$$

that is, causal connective is interpreted as usual implication when $a \geq \frac{1}{2}$, otherwise the causal formula is non-applicable ($\widehat{\frac{1}{2}}$). $\rightarrow$ is monotonic, like $\wedge$ and $\vee$, and holds distributive properties and $\neg(a \rightarrow b) = a \rightarrow \neg\, b$.

Truth values $\in \mathcal{L}_6$ have its origins in the MUS-T specification condition $\in \mathcal{L}_3$ and the causal operation, $\rightarrow$, defined above. A truth value expresses the capability of a MUS-T graph to satisfy a causal formula now or in future evolutions (see the table in figure 2). As being incremental, evolution means the loss of some unspecification in the model, that is, temporal or discrete transitions unspecified in the semantic graph turn into specified ones (possible or non-possible).

The partial order defined in MPU is and order relation related to the degree of satisfaction. So 0 is the least truth degree, whereas 1 is the greatest. The values $\widehat{\frac{1}{2}}$ and $\frac{1}{2}$ are middle points in this order: $\widehat{\frac{1}{2}}$ is far from the two ends (0 and 1), but it cannot get to them, whereas $\frac{1}{2}$ is near both, and it can get to either. This is the reason why it is called algebra of Middle Point Uncertainty.

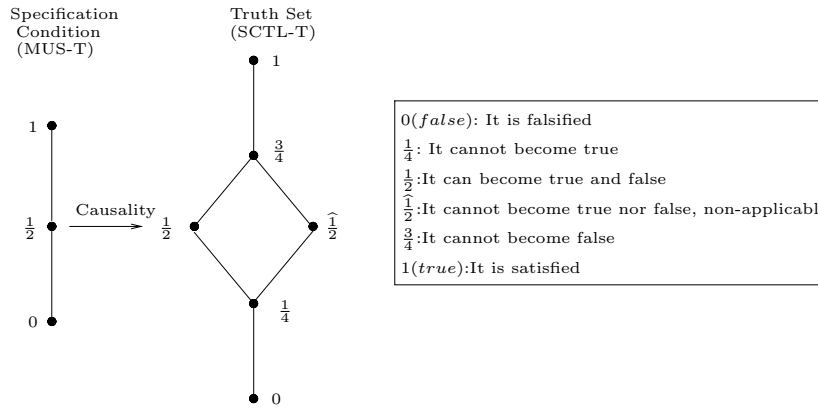


Fig. 2. SCTL-T Truth values

3.2 SCTl-T: Expressing Requirements

A SCTL-T declaration $I_i := \phi$ establishes an identifier I_i, which is taken from a set $\mathcal{I}$, and a SCTL-T formula $\phi \in \Phi_{SCTL-T}$. Φ_{SCTL-T} is given by the following grammar:

$$\langle \Phi_{SCTL-T} \rangle ::= \phi_C \forall \otimes \phi_C \mid \phi_C \exists \otimes \phi_C$$
$$\langle \phi_C \rangle ::= \psi \in \Psi \mid a \in A \mid \theta \in \Theta \mid y.\phi_C \mid \phi_C \wedge \phi_C \mid \phi_C \vee \phi_C \mid \neg \phi_C \mid \Phi_{SCTL-T}$$
$$\langle \otimes \rangle ::= \Rightarrow \mid \Rightarrow_+ \mid \Rightarrow_- \mid \Rightarrow \bigcirc \mid \Rightarrow \odot$$

where $\{\forall, \exists\}$ are the usual path quantifiers adapted to many-valued reasoning; $\otimes$, the set of temporal operators; $\theta \in \Theta ::= \{true \mid false \mid \emptyset\}$, a propositional constant of state; $a \in A$, an event in the alphabet of a MUS-T graph; $y.$, the reset quantifier which binds and resets a specification clock $y \in \mathcal{E}$; and $\mathcal{E}$, the set of specification clocks. Finally, $\psi \in \Psi$ is the set of clock constraints over model clocks $\in \mathcal{C}$ and specification clocks $\in \mathcal{E}$.

Applicability Set and Quantified Causality. Temporal operators are used to reason about transition successors and predecessors of a given timed state. A temporal operator $\in \otimes$ fixes the qualitative order between the timed state in which the premise is formulated, and the timed states in the scope of the consequence (applicability set). For every temporal operator in the syntax, an applicability set, $\perp$, is defined in table 1. Thus, $\Rightarrow \bigcirc$ and $\Rightarrow \odot$ are discrete temporal operators (successor and predecessor respectively); $\Rightarrow$, $\Rightarrow_+$ and $\Rightarrow_-$ are dense temporal operators (present, future and past respectively).

Whereas temporal operators determine the qualitative order between the premise state and the consequence states (applicability set), quantification determines the degree of satisfaction required in the consequence states. Thus, $\{\forall, \exists\}$ quantifiers provide existential and universal quantification over the applicability set. In quantified causality (Æ $\otimes$), the truth degree of the quantified

Table 1. Applicability set for temporal operators $\in \otimes$. P is the premise of the causal formula. If the temporal operator is $\Rightarrow \bigcirc$ only events in the simplest premise in P, applicability events (AE), are considered.

$\perp((\Rightarrow \bigcirc, P), (s,\gamma)) = \{(s',\gamma',c_s) \mid (s,\gamma) \overset{a,c_s}{\multimap} (s',\gamma') \text{ and } a \in AE\}$
$\perp((\Rightarrow \bigodot, P), (s,\gamma)) = \{(s',\gamma',c_s) \mid (s',\gamma') \overset{a,c_s}{\multimap} (s,\gamma)\}$
$\perp((\Rightarrow_+, P), (s,\gamma)) = \{(s,\gamma',c_s) \mid (s,\gamma) \overset{\tau,c_s}{\multimap} (s,\gamma')\}$
$\perp((\Rightarrow_-, P), (s,\gamma)) = \{(s,\gamma',c_s) \mid (s,\gamma') \overset{\tau,c_s}{\multimap} (s,\gamma)\}$
$\perp((\Rightarrow, P), (s,\gamma)) = \{(s,\gamma,1)\}$

consequence depends on the specification condition of the transition that makes a timed state accessible in the causal formula, referred to as accessibility condition (c_s in table 1).

Informally, an existential quantification over a consequence ϕ is satisfied (truth value 1) iff it exists an applicability state $s_a = (s,\gamma,c_s)$ in which $\vdash(\phi, s_a) = 1$, and the accessibility condition c_s is 1. Thus, every future evolution of the model will satisfy the existential quantification. On the contrary, the existential quantification is not satisfied (truth value 0) iff the degree of satisfaction in every state of the applicability set is 0, whatever specification condition. On the other hand, an universal quantification over a consequence ϕ is satisfied (truth value 1) iff all timed states s_a in the applicability set with accessibility condition $\in \{\frac{1}{2}, 1\}$ satisfy the consequence ($\vdash(\phi, s_a) = 1$). It is not satisfied iff it exists an applicability timed state with $c_s = 1$ which does not satisfy the consequence ($\vdash(\phi, s_a) = 0$). See the following section for formal definition.

Semantics. Given a MUS-T graph $\mathcal{M}$, SCTL-T formulas are interpreted with respect to the many-valued dense graph $\mathcal{S}_{\mathcal{M}^\varepsilon}$ induced by $\mathcal{M}^\mathcal{E}$, which is $\mathcal{M}$ with $\mathcal{C}$ extended by all clocks $\mathcal{E}$ mentioned in the SCTL-T requirement. The satisfaction relation $\vdash$ is defined as:

$$\vdash : \left(\Phi_{SCTL-T} \times (S \times \Re_+^{\#\mathcal{C}\cup\mathcal{E}} \times \mathcal{L}_3)\right) \longrightarrow \mathcal{L}_6$$

where $\vdash(\phi, (s,\gamma,c_s))$ assigns a truth value $\in \mathcal{L}_6$ to the formula ϕ evaluated in a state (s,γ) with accessibility condition c_s. We start defining SCTL-T by giving the semantics of propositional constants. If (s,γ,c_s) is a timed state with accessibility condition c_s, the satisfaction relation, $\vdash$, is as follows:

i) $\psi \in \Psi, \vdash(\psi, (s,\gamma,c_s)) = \psi(\gamma)$.

ii) $a \in A, \vdash(a, (s,\gamma,c_s)) = c_s{}' \mid (s,\gamma) \overset{a,c_s{}'}{\multimap} (s',\gamma')$.

iii) $\theta \in \Theta, \vdash(true, (s,\gamma,c_s)) = c_s$, i. e., $true$ is satisfied in a state (s,γ,c_s) iff this state is reached by a possible (discrete or temporal) transition ($c_s = 1$). On the contrary, $\vdash(false, (s,\gamma,c_s)) = \neg\, c_s$, i. e., $false$ is satisfied iff the state is reached by a non-possible transition ($c_s = 0$). Finally, $\emptyset$ is always satisfied ($\vdash(\emptyset, (s,\gamma,c_s)) = 1$).

We now proceed by defining the semantics of generic formulas by structural induction. Let ϕ, ϕ' be SCTL-T formulas, the satisfaction relation is as follows:

iv) $\vdash (y.\phi, (s, \gamma, c_s)) \equiv \vdash (\phi, (s, \gamma_{\{y\}}, c_s))$

v) $\vdash (\neg\phi, (s, \gamma, c_s)) \equiv \neg(\vdash (\phi, (s, \gamma, c_s)))$

vi) $\vdash (\phi \vee \phi', (s, \gamma, c_s)) \equiv \vdash (\phi, (s, \gamma, c_s)) \vee \vdash (\phi', (s, \gamma, c_s))$

vii) $\vdash (\phi \wedge \phi', (s, \gamma, c_s)) \equiv \vdash (\phi, (s, \gamma, c_s)) \wedge \vdash (\phi', (s, \gamma, c_s))$

viii) $\vdash (\phi \forall\otimes \phi', (s, \gamma, c_s)) \equiv \rightarrow \left(\vdash (\phi, (s, \gamma, c_s)), \bigwedge_i \neg\, {c_{s_i}}' \vee \vdash (\phi', ({s_i}', {\gamma_i}', {c_{s_i}}')) \right)$
where $({s_i}', {\gamma_i}', {c_{s_i}}') \in \perp ((\otimes, \phi), (s, \gamma))$

ix) $\vdash (\phi \exists\otimes \phi', (s, \gamma, c_s)) \equiv \rightarrow \left(\vdash (\phi, (s, \gamma, c_s)), \bigvee_i {c_{s_i}}' \wedge \vdash (\phi', ({s_i}', {\gamma_i}', {c_{s_i}}')) \right)$
where $({s_i}', {\gamma_i}', {c_{s_i}}') \in \perp ((\otimes, \phi), (s, \gamma))$

3.3 SCTL-T: Fixpoints As Recursion

As in [9], a SCTL-T requirement is a set of mutually recursive declarations of the form $\{I_0 := \phi_0, \cdots, I_n := \phi_n\}$, where SCTL-T syntax is extended with instantiation and recursive operators:

$$\langle \phi_C \rangle ::= I_i \mid \ll I_i \gg \mid [\![I_i]\!] \text{ , with } I_i \in \mathcal{I}$$

An identifier I_i in the right-hand side of a declaration instantiates the formula ϕ_i in a declaration of the form $I_i = \phi_i$. Recursive operators $\ll I_i \gg$ and $[\![I_i]\!]$ are understood respectively as the least and greatest fixed points of the predicate transformer defined by the recursion. In this way, usual operators in branching logics can be expressed in SCTL-T but with a many-valued nature. For a practical specification of scenarios and requirements without explicit fixpoint constructions we provide a set of such constructions as predefined macros[3]. Additionally, we enforce requirements to be alternation free in order to simplify and make fully local model checking and synthesis algorithms.

Preliminary Definitions. Before defining semantics of fixpoints, some definitions concerning many-valued fixpoints are in order. One way to provide semantics of a state-based temporal logic (SCTL-T, for instance) is to map formulas to set of states, but in many-valued reasoning a mapping from formulas to partitions of states is needed. Let $\mathcal{M}$ be a MUS-T graph. We define the set $\mathcal{P}^6$ of all 6-cardinality partitions of its state-space, $\mathcal{ST}$, where every $P \in \mathcal{P}^6$ is:

$$P = \{\langle P \rangle_0, \langle P \rangle_{\frac{1}{4}}, \langle P \rangle_{\widehat{\frac{1}{2}}}, \langle P \rangle_{\frac{1}{2}}, \langle P \rangle_{\frac{3}{4}}, \langle P \rangle_1\}$$

[3] In the examples in this paper (section 5) macro-symbols stand for the usual meanings, that is, $\mathcal{G}$, *it is always Going to be the case*; $\mathcal{F}$, *at least once in the Future*; $\mathcal{H}$, *it Has always been the case*; $\mathcal{P}$, *at least once in the Past*; $\mathcal{U}$, *Until*; $\mathcal{S}$, *Since*; $\mathcal{U}_w$, *Weak Until*; and $\mathcal{S}_w$, *Weak Since*

For every P, $P' \in \mathcal{P}^6$, union ($\cup_{\mathcal{P}}$), intersection ($\cap_{\mathcal{P}}$), complement ($\_^{\mathcal{P}}$) and inclusion ($\subseteq_{\mathcal{P}}$) are defined as:

$$s \in \langle P \cup_{\mathcal{P}} P' \rangle_j \text{ iff } j = \sup\{i, i' \mid s \in \langle P \rangle_i \text{ and } s \in \langle P' \rangle_{i'}\}$$
$$s \in \langle P \cap_{\mathcal{P}} P' \rangle_j \text{ iff } j = \inf\{i, i' \mid s \in \langle P \rangle_i \text{ and } s \in \langle P' \rangle_{i'}\}$$
$$s \in \langle \overline{P}^{\mathcal{P}} \rangle_j \text{ iff } s \in \langle P \rangle_{\neg j}$$
$$P \subseteq_{\mathcal{P}} P' \text{ iff } \langle P \rangle_i \subseteq \bigcup_{i \leq j \leq 1} \langle P' \rangle_j \ \forall\ i$$

where $\subseteq$ and $\cup$ stand for the set inclusion and union in $\mathcal{ST}$; $\leq$ is the partial order relation in $\mathcal{L}_6$; inf and sup are, respectively, the greatest lower bound and least upper bound in this partial order; and $\neg$ is the complement in $\mathcal{L}_6$.

The set of all 6-cardinality partitions $\mathcal{P}^6$ forms a lattice under the inclusion $\subseteq_{\mathcal{P}}$ defined above, where for every $P, P' \in \mathcal{P}^6$, the least upper bound is $P \cup_{\mathcal{P}} P'$; and the greatest lower bound is $P \cap_{\mathcal{P}} P'$. Besides, $P_\perp = \{\mathcal{ST}, \{\} \cdots \{\}\}$, is the least element in the lattice; and $P_\top = \{\{\} \cdots \{\}, \mathcal{ST}\}$, is the greatest element in the lattice. Each element P of the lattice can also be thought as a predicate on $\mathcal{ST}$, where the predicate is viewed as being i for exactly the states $\in \langle P \rangle_i$.

Semantics. A SCTL-T declaration $I_j := \phi_j$ which includes a recursive operator $\ll I_i \gg$ (or $[\![I_i]\!]$) with $i \leq j$, defines the predicate transformer $I_i = f(I_i)$. Let $f : \mathcal{P}^6 \to \mathcal{P}^6$ be such a predicate transformer; then f is monotonic provided that $P \subseteq_{\mathcal{P}} P'$ implies $f(P) \subseteq_{\mathcal{P}} f(P')$. Since every logical connective, except negation, is monotonic, and all the negations can be pushed down to the atomic propositions using De Morgan's laws and dualities, every predicate transformer defined by a SCTL-T recursion is monotonic, guaranteeing the existence of the fixpoints [10]. As identifiers do not appear free in SCTL-T formulas, but bound by recursive operators, enforcing an even number of negations in the scope of recursive operators is not needed. So, SCTL-T semantics is extended as follows:

x) $\vdash (I_i, (s, \gamma, c_s)) \equiv \vdash (\phi_i, (s, \gamma, c_s))$, where ϕ_i is such that $I_i := \phi_i$.
xi) $\vdash (\ll I_i \gg, (s, \gamma, c_s)) \equiv j \in \mathcal{L}_6 \mid (s, \gamma) \in < \bigcap_{\mathcal{P}} \{P | f(P) \subseteq P\} >_j$
xii) $\vdash ([\![I_i]\!], (s, \gamma, c_s)) \equiv j \in \mathcal{L}_6 \mid (s, \gamma) \in < \bigcup_{\mathcal{P}} \{P | f(P) \supseteq P\} >_j$

3.4 SCTL-T: Expressing Scenarios

Using scenarios a designer proposes a situation and decides what behavior would be appropriate to that context. We formalize scenarios defining a synthesis-oriented version of SCTL-T (SCTL-T^s). To be precise, we formalize scenarios with the simple "declarative past causes imperative future" idea in [11]. The general idea behind imperative future is to rewrite each formula to be synthesized into a set of SCTL-T^s formulas of the form:

$$\textit{declarative past}\ \{\forall\}\{\Rrightarrow, \Rrightarrow_+, \Rrightarrow \bigcirc\}^4\ \textit{imperative future}$$

[4] Temporal operators overloaded for synthesis

and then treat such formulas as scenarios showing how the future can be constructed given the past constructed so far. Formally, a synthesis rule $SR \in \mathcal{SR}$ is a causal formula where the premise specifies the synthesis context and the consequence specifies the new non-strict future behavior:

$$\langle \mathcal{SR} \rangle ::= ini \otimes_s \phi_{fs} | \phi_p \otimes_s \phi_{fs}$$
$$\langle \otimes_s \rangle ::= \Rrightarrow \mid \forall \Rrightarrow \bigcirc \mid \forall \Rrightarrow_+$$
$$\langle \phi_{fs} \rangle ::= \psi \mid a \mid \neg\, a \mid \theta \mid y.\phi_{fs} \mid a\{y\} \mid \phi_{fs} \wedge \phi_{fs} \mid \phi_{fs} \forall \otimes_f \phi_{fs}$$

where ϕ_p is a non-strict past formula $\in \Phi_{SCTL-T}$, $\otimes_f$ is the set of non-strict future temporal operators[5]; and $\otimes_s$ are the overloaded versions of non-strict future temporal operators for imperative specification. Finally in order to allow the explicit management of model clocks from the logic, we introduce an imperative form of bind, $a\{x\}$, which adds a clock x to $\mathcal{C}$ (if it does not exists) and resets the clock in a discrete transition labeled by a.

Incremental synthesis is based on the following principle: changes in the model refer to specifying transitions which are unspecified in the semantic graph (from $\frac{1}{2}$ up to 1, or from $\frac{1}{2}$ down to 0). Any other change in the specification condition reflects an inconsistency failure which should identify a set of scenarios in conflict.

Regarding the synthesis algorithm, unfortunately, tableau construction for dense real-time logics allowing punctuality is undecidable [4]. Incremental synthesis applies a bounded model construction algorithm similar to the one in [12]. In the bounded synthesis approach, given a SCTL-T requirement and a source MUS-T model, a satisfying target model is synthesized (if feasible) within given bounds on the number of clocks in $\mathcal{C}$ and constants in clock constraints. The synthesis algorithm proceeds as an imperative model checking algorithm, that is, selecting synthesis contexts (states in the model satisfying the declarative part) and then adding, in these contexts, the new future behavior specified by the imperative part (enforcing a truth value 1). That is, given a synthesis rule $SR := \phi_p \otimes_s \phi_{fs}$, if $\phi_p = ini$, (s_0, γ^0) is the single synthesis context, otherwise (s, γ) is a synthesis contexts iff $\vdash (\phi_p, (s, \gamma)) = 1$ (applicable rule in this context).

Synthesis rules are global-scope (invariants), so it is necessary to reuse locations in order to avoid MUS-T graphs indefinitely growing. In brief, reuse criteria are defined as preserving simulations and bisimulations, at different demanding levels, between the non-reusing model and the reusing one.

The incremental synthesis of a SCTL-T scenario in a MUS-T graph can be viewed as a form of the more traditional controller synthesis problem where all specified transitions (possible and non-possible) in a MUS-T graph are considered to be uncontrollable, meanwhile all unspecified transitions are considered to be controllable. In this sense, the timed version of the controller synthesis problem in [13] could be applied to the synthesis of SCTL-T scenarios by handling the unspecified transitions, that is, turning unspecified transitions into possible or non-possible ones. However, a MUS-T graph is really a multi-valued graph

[5] For readability we provide $\forall \Rightarrow \overset{\forall \rightarrow}{\bigcirc} \equiv \forall \Rightarrow \bigcirc \forall \Rightarrow_+$, and its overloaded $\forall \Rrightarrow \overset{\forall \rightarrow}{\bigcirc}$.

with a potential arbitrary number of clocks in which only the clocks in the set $\mathcal{C}$ are specified, so the controller synthesis solution cannot be applied.

4 Many-Valued Reasoning

SCTL/MUS-T is a dense-time methodology, so automation of model checking and synthesis involves obtaining an exact abstraction of the infinite and dense state-space. In this respect, two many-valued strong time abstracting bisimulations (FSTaB and BSTaB) are defined which preserve truth values in SCTL-T. To be precise, FSTaB preserves non-strict future SCTL-T formulas without clock constraints and BSTaB preserves strict past SCTL-T formulas without clock constraints. Complete SCTL-T preservation can be obtained using a technique similar to the one used in [5], that is, extending the MUS-T graph with the set of specification clocks and considering an equivalence relation that also distinguishes the clock constraints in the formula.

Many-Valued Strong Time Abstracting Bisimulations. Consider a MUS-T graph $\mathcal{M}$, and a subset $A^S \subseteq A$ of preserving events. A **F**orward **S**trong **T**ime-**a**bstracting **B**isimulation (FSTaB) with preserving events A^S abstracts away the exact amount of time elapsed in a time transition and discrete events not in the subset A^S. This is done by replacing all labels $\tau \in \Re_+$ by the label $\epsilon \notin (A \cup \Re_+)$, and all labels $a \in A - A^S$ by the label $d \notin (A \cup \Re_+)$

A binary relation $\cong_S$ on the states $\mathcal{ST}$ ($\cong_S \subseteq \mathcal{ST} \times \mathcal{ST}$) is a FSTaB with preserving events A^S, iff for all states $st_1 \cong_S st_2$, where $st_1 = (s_1, \gamma_1)$ and $s_2^t = (s_2, \gamma_2)$, the following conditions hold:

$$\forall\ st_1{}' \mid st_1 \xrightarrow{a,l}\!\!\circ\ st_1{}' \in \mathcal{T}, \exists\ st_2{}' \mid st_2 \xrightarrow{a,l}\!\!\circ\ st_2{}' \text{ and } st_1{}' ; \cong_S st_2{}';$$

$$\forall\ st_1{}' \mid st_1 \xrightarrow{\epsilon,l}\!\!\circ\ st_1{}' \in \mathcal{T}, \exists\ st_2{}' \mid st_2 \xrightarrow{\epsilon,l}\!\!\circ\ st_2{}' \text{ and } st_1{}' \cong_S st_2{}';$$

$$\forall\ st_1{}' \mid st_1 \xrightarrow{d,l}\!\!\circ\ st_1{}' \in \mathcal{T}, \exists\ st_2{}' \mid st_2 \xrightarrow{d,l}\!\!\circ\ st_2{}' \text{ and } st_1{}' \cong_S st_2{}';$$

the above conditions also hold if the roles s_1 and s_2 are reversed.

In the same way, a binary relation $\cong_S$ on the states $\mathcal{ST}$ is a **B**ackward **T**ime-**a**bstracting **B**isimulation, BSTaB, with preserving events A^S, iff for all states $st_1 \cong_S st_2$ the same conditions hold backward. Finally, let ψ a clock constraint or a reset predicate $x = 0$ for some clock x, a binary relation $\cong$ preserves ψ iff for all states $st_1 \cong_S st_2$, $\psi(st_1) = \psi(st_2)$.

Lemma 1. *Let $\mathcal{M}$ be a MUS-T graph and $\cong_S$ be a FSTaB with preserving events A^S on $\mathcal{M}$. For any non-strict future formula $\phi \in \Phi_{SCTL-T}$ with events $\in A^S$ and any pair of states $st \cong_S st'$, $\vdash (\phi, (st, c_s)) = \vdash (\phi, (st', c_s))$.*

Lemma 2. *Let $\mathcal{M}$ be a MUS-T graph and $\phi \in \Phi_{SCTL-T}$ with events A_ϕ, a set of specification clocks $\mathcal{E}_\phi$, and a set of clock constraints Ψ_ϕ. Let $\mathcal{M}^{\mathcal{E}}$ be the*

MUS-T graph $\mathcal{M}$ extended with the clocks in $\mathcal{E}_\phi$ and $\cong_S$ a F-BSTaB on $\mathcal{M}^{\mathcal{E}}$ with preserving events $A^S = A_\phi$. If $\cong_S$ preserves reset predicates $y_i = 0 \ \forall \ y_i \in \mathcal{E}_\phi$; and clock constraints in the formula $\psi_i \in \Psi_\phi$, then $\cong_S$ preserves the truth value of ϕ. That is, for any pair of states $st_1 \cong_S st_2$, $\vdash (\phi, (st_1, c_s)) = \vdash (\phi, (st_2, c_s))$.

For shortness, the reader is referred to [8] for proofs of the above lemmas. Finally, the region equivalence in [5] is a F-BSTaB with the above characteristics. Since this equivalence induces a finite partition of the state-space, the quotient of a MUS-T graph with respect to this F-BSTaB bisimulation is finite.

SCTL-T Model Checking and Synthesis. Model checking and synthesis of SCTL-T formulas use the STaBs defined above to compute a finite quotient of a given MUS-T graph. In this respect, minimization algorithms, computing the minimal quotient, have been adapted to timing reasoning [14] in order to overcome the state-explosion problem. Minimization algorithms are based on the facts that a forward bisimulation induces a pre-stable partition and vice-versa, and, in the same way, a backward bisimulation induces a post-stable partition and vice-versa. Minimization is done by partition refinement, that is, the coarser pre(post)-stable partition of the state-space $\mathcal{ST}$ is computed starting from an initial partition and successively refining it until it becomes pre(post)-stable.

5 Case Study: A Fragment

In [8] it is showed a complete example which applies the SCTL/MUS-T methodology to the steam-boiler case study [15]. In brief, the steam-boiler has a water boiler tank, a pump, and a number of sensor one of which measures water level. The entire physical system operates under guidance of a controller. The controller must keep the water level between extreme values M_1 and M_2 at all times. It should also try to keep the water level between normal operating levels N_1 and N_2 as much as possible. The controller operates in five modes (initialization, normal, rescue, degraded and emergency stop) following a cycle and a priori does not terminate; this cycle takes place every 5 seconds. The reader is referred to the complete case study in [8] for details of modes which are not included in this paper; also, only design for non-failure environment (accurate measure and non-failure pump) is included. In normal model, the controller makes its decision to turn on or turn off the pump based on the current water level; no action is taken if this level lies in the range $[N_1, N_2]$. The controller works periodically with each cycle consisting of three tasks: sensor regulation, report computation (optional behavior) and actuating the pump.

Controller Requirements. Events in the controller of the steam-boiler are identified as synchronization events (in a non-failure environment) and internal events (controller tasks). The set of synchronization events is $\{p_{on}, p_{off}, na, n1_\downarrow, n2_\uparrow, n_\leftrightarrow, m1_\downarrow, m2_\uparrow\}$ where p_{on}, p_{off} and na stand for opening, closing and non-actuating the pump; $n_\leftrightarrow$ ($> N1$ and $< N2$), $m1_\downarrow$ ($< M1$), $m2_\uparrow$ ($> M2$),

Table 2. Controller requirements

R_1	$x.BS \;\; \forall \Rightarrow \bigcirc \;\; (\emptyset \Rightarrow \mathcal{AF}[ES \wedge x \leq 3 \wedge x \geq 1])$
R_2	$x.BR \;\; \forall \Rightarrow \bigcirc \;\; (\emptyset \Rightarrow \mathcal{AF}[ER \wedge x = 3])$
R_3	$x.act \Rightarrow \mathcal{AF}[x = 5 \Rightarrow act]$
R_4	$x.act \Rightarrow \mathcal{EF}[act \wedge x = 5]$
R_5	$x.act \;\; \forall \Rightarrow \bigcirc \;\; \mathcal{AG}[(x < 5 \Rightarrow \neg act) \vee (x \geq 5)]$
R_6	$(n1_{\downarrow} \vee n2_{\uparrow} \vee n_{\leftrightarrow}) \;\; \forall \Rightarrow \bigcirc \;\; (\emptyset \Rightarrow \mathcal{AF}[ES \Rightarrow \mathcal{AF}[act]])$
R_7	$(n1_{\downarrow} \vee n2_{\uparrow} \vee n_{\leftrightarrow}) \;\; \forall \Rightarrow \bigcirc \;\; (\emptyset \Rightarrow \mathcal{EF}[ES \;\; \forall \Rightarrow \bigcirc \;\; (BR \Rightarrow \mathcal{AF}[act])])$

$n1_{\downarrow}$ ($< N1$) and $n2_{\uparrow}$ ($> N2$) stand for the measures of the water sensor. Internal events are identified as BS, ES (start and completion of sensor regulation); BR, ER (start and completion of report computation) and act (entering actuation task). Identified controller requirements (in table 2) matches the following informal specification:

- **Task Completion** (R_1, R_2): Once regulation starts it is always completed within no less than 1 and no more than 3 seconds. Besides, once report computation starts it takes 3 seconds.
- **Actuation Cycle** (R_3, R_4, R_5): Actuation only occurs once every 5 seconds. Actuation is not possible if such delay from the last one has not elapsed.
- **Cycle Sequence** (R_6, R_7): Once water level has been read, provided that the boiler does not reach any extreme limit ($M1$ or $M2$), a new cycle starts which always includes regulation and, possibly, report computation.

Controller Scenarios. The design process is based on identifying timing scenarios of the system and specifying them as SCTL-T^s synthesis rules (table 3):

- **Init** (S_{INI}): At the beginning the controller enters the regulation task. This is the only task at this moment.
- **Regulation** (S_{REG}): Sensor regulation takes no less than 1 and no more than 3 seconds.
- **Entering Actuation** (S_{EA}): Actuation phase starts (act) 5 seconds after the last actuation or the beginning of regulation.
- **Actuation** (S_{ACT}, S_{AM}): In the actuation phase controller checks the water sensor ($n1_{\downarrow}$, $n2_{\uparrow}$, $m1_{\downarrow}$, $m2_{\uparrow}$, $n_{\leftrightarrow}$), this checking is assumed to be instantaneous. On the basis of the water level, controller decides to open (p_{on}), close (p_{off}) or not to actuate (na) the pump (Actuation Mode); also actuation is assumed to be instantaneous.
- **Actuation Completion and New Cycle** (S_{NC}): Once controller has actuated, provided that extreme values are not reached, a new cycle starts.
- **Report Computation** –optional behavior– (S_{RC}): Report computation, if it occurs, comes after completion of regulation and takes 3 seconds.

Applying SCTL/MUS-T. The methodology is articulated as subsequent iterations consisting of formalizing the specification which captures the controller

Table 3. Controller scenarios

S^0_{INI}	$ini\ \ \forall\!\!\Rightarrow\!\!\bigcirc^{\forall\rightarrow}\ \ BS \wedge \neg(ER \vee BR \vee ES)$
S^1_{INI}	$ini \Rrightarrow (\emptyset\ \ \forall\Rightarrow_+\ \ false)$
S^0_{REG}	$BS\{x_s\}\ \ \forall\!\!\Rightarrow\!\!\bigcirc^{\forall\rightarrow}\ \ ((x_s \leq 3 \Rightarrow true)\ \wedge (x_s > 3 \Rightarrow false))$
S^1_{REG}	$BS\{x_s\}\ \ \forall\!\!\Rightarrow\!\!\bigcirc^{\forall\rightarrow}\ \ ((x_s \geq 1 \wedge x_s \leq 3) \Rightarrow ES)$
S^0_{EA}	$x.\emptyset \Rightarrow \mathcal{EP}[BS \wedge x = 5] \Rrightarrow act$
S^1_{EA}	$x.\emptyset \Rightarrow \mathcal{EP}[BS \wedge x < 5] \Rrightarrow \neg act$
S^2_{EA}	$(x.\emptyset\ \ \exists\Rightarrow\!\!\bigodot\ (\emptyset \Rightarrow \mathcal{EP}[act \wedge x < 5])) \Rrightarrow \neg act$
S^3_{EA}	$x.\emptyset \Rightarrow \mathcal{AP}[act \wedge x = 5] \Rrightarrow act$
S^0_{ACT}	$act\ \ \forall\!\!\Rightarrow\!\!\bigcirc\ \ (\emptyset\ \ \forall\Rightarrow_+\ \ false)$
S^1_{ACT}	$act\ \ \forall\!\!\Rightarrow\!\!\bigcirc\ \ n1_\downarrow \wedge n2_\uparrow \wedge m1_\downarrow \wedge m2_\uparrow \wedge n_\leftrightarrow$
S^2_{ACT}	$(\emptyset\ \ \forall\Rightarrow\!\!\bigodot\ \ \neg act) \Rrightarrow \neg(n1_\downarrow \vee n2_\uparrow \vee m1_\downarrow \vee m2_\uparrow \vee n_\leftrightarrow)$
S^0_{AM}	$(n1_\downarrow \wedge n2_\uparrow \wedge m1_\downarrow \wedge m2_\uparrow \wedge n_\leftrightarrow) \Rrightarrow \neg p_{on} \wedge \neg p_{off}$
S^1_{AM}	$n1_\downarrow\ \ \forall\!\!\Rightarrow\!\!\bigcirc\ \ (na \wedge p_{on} \wedge \neg p_{off})$
S^2_{AM}	$n2_\uparrow\ \ \forall\!\!\Rightarrow\!\!\bigcirc\ \ (na \wedge p_{off} \wedge \neg p_{on})$
S^3_{AM}	$n_\leftrightarrow\ \ \forall\!\!\Rightarrow\!\!\bigcirc\ \ na \wedge \neg p_{on} \wedge \neg p_{off}$
S^4_{AM}	$\neg(n1_\downarrow\ \wedge\ n2_\uparrow)\ \ \forall\!\!\Rightarrow\!\!\bigcirc^{\forall\rightarrow}\ \ \neg p_{on} \wedge \neg p_{off}$
S^5_{AM}	$(\emptyset\ \ \exists\Rightarrow\!\!\bigodot\ \ act)\ \ \forall\!\!\Rightarrow\!\!\bigcirc\ \ (\neg act \wedge (\emptyset\ \ \forall\Rightarrow_+\ \ false))$
S_{NC}	$p_{on} \vee p_{off} \vee na\ \ \forall\!\!\Rightarrow\!\!\bigcirc\ \ BS$
S^0_{RC}	$(\emptyset\ \ \exists\Rightarrow\!\!\bigodot\ \ ES) \vee BR\{x_r\}\ \ \forall\!\!\Rightarrow\!\!\bigcirc^{\forall\rightarrow}\ \ (x_r = 3 \Rightarrow ER)$
S^1_{RC}	$(\emptyset\ \ \exists\Rightarrow\!\!\bigodot\ \ ES) \vee BR\}\ \ \forall\!\!\Rightarrow\!\!\bigcirc^{\forall\rightarrow}\ \ (x_r \leq 3 \Rightarrow true) \wedge (x_r > 3 \Rightarrow false)$

requirements; incremental design of the controller by means of formalizing typical behaviors as scenarios; and verification and validation of design decisions conforming to the requirements specification. The advantages which many-valued reasoning provides comes from the many-valued results obtained from model checking and incremental synthesis:

- Incremental synthesis results: A SCTL-T scenario is successfully synthesized provided that its imperative part is $\geq \frac{1}{2}$. Otherwise the imperative part is $\widehat{\frac{1}{2}}$, non-applicable, or $< \widehat{\frac{1}{2}}$, so the synthesis is not feasible. In the last case the scenarios in conflict are supplied allowing the designer to inspect what scenarios are error-prone, a misunderstanding of the system is uncovered.
- Model checking results: If a SCTL-T requirement is not satisfied (< 1) the designer is guided as follows. In case of inconsistency failure, $< \widehat{\frac{1}{2}}$, counterexamples are computed. By animating the counterexamples the designer inspects which scenarios, or maybe the requirement, are error-prone. In case of incompleteness failure ($\geq \frac{1}{2}$) completion suggestions are computed, then the model, extended with supplied suggestions, is animated in order to explore what alternative conforms to wishes and new scenarios are discovered.

On the basis of the specified requirements and scenarios, appendix A outlines a part of the SCTL/MUS-T process following the lifecycle in figure 1.

6 Conclusions

With respect to other formal approaches proposed in the literature, our lifecycle model is based on an iterative and incremental structure in which real-time characteristics are considered from the beginning of the process. One advantage of this approach is early detection of timing failures; considering timing requirements comparatively late in the process, as performance requirements, would cause ad hoc changes to the system. Also, as the system gradually takes form as more is learned about the problem, alternative solutions can be explored. Regarding the specification process, scenarios of using the system are often easier to identify at the beginning than generic requirements that can be made clear only after a deeper knowledge about the system has been gained. Despite this paper only includes some parts of the steam-boiler case study, the complete process in [8] reveals that specially at early phases: requirements often change guided by verification results, new scenarios are discovered throughout the process; and misunderstanding of the system is uncovered by conflicts in requirements or scenarios. So, reaching a good design entails a lot of interaction with designers.

Loose or partial specifications has previously been pursued in [16] by using the real-time process calculi TMS (**T**imed **M**odal **S**pecifications). TMS allow partial specifications by introducing two modalities in transitions, *may* and *must*. Intuitively, the more *must* transitions and the fewer *may* transitions a TM specification has, the finer or more specified it is. Despite this characterization supports refinements in and incremental process, consistency checking is not implicit since *not-must* transitions are not considered. Besides, the far or the close a TM specification is from satisfying a property-oriented specification is not measured by means of many-valued reasoning.

Finally, we are working on distributing methodology for collaborative specification [17] since development of complex systems usually involves many stakeholders, each with their own perspectives on the system. In this context, a MUS-T model turns into a MUS-T view provided by every single stakeholder. Inconsistencies will relate not just MUS-T models in successive cycles of a single lifecycle, but MUS-T views from parallel lifecycles in the distributed process.

References

1. Alur, R.: Techniques for Automatic Verification of Real-Time Systems. PhD thesis, Department of Computer Science, Stanford University (1991)
2. Gollu, A., Puri, A., Varaiya, P.: Discretization of Timed Automata. In: Decision and Control, 33rd IEEE Conference. (1994) 957–958
3. Asarin, E., Maler, O., Pnueli, A.: On Discretization of Delays in Timed Automata and Digital Circuits. In: Concurrency Theory, 9th International Conference (CONCUR'98). Volume 1466 of LNCS. Springer Verlag (1998) 470–484
4. Alur, R., Henzinger, T.A.: Logics and Models of Real Time: A Survey. In: Real Time: Theory and Practice, REX Workshop. Volume 600 of LNCS. Springer Verlag (1992) 74–106
5. Alur, R., Courcoubetis, C., Dill, D.: Model Checking in Dense Real-time. Information and Computation **104** (1993) 2–34

6. Chechik, M., Devereux, B., Easterbrook, S.M., Lai, A., Petrovykh, V.: Efficient Multiple-Valued Model-Checking Using Lattice Representations. In: Concurrency Theory, 12th International Conference (CONCUR'01). (2001) 21–24
7. Pazos Arias, J.J., García Duque, J.: SCTL-MUS: A Formal Methodology for Software Development of Distributed Systems. A Case Study. Formal Aspects of Computing **13** (2001) 50–91
8. Fernández Vilas, A.: Tratamiento Formal de Sistemas con Requisitos de Tiempo Real Críticos. PhD thesis, Dept. Ingeniería Telemática, Universidad de Vigo (2002)
9. Sokolsky, O.V., Smolka, S.A.: Local Model Checking for Real-Time Systems. In: Computer Aided Verification, 7th International Conference (CAV'95). Volume 939 of LNCS., Springer Verlag (1995) 211–224
10. Tarski, A.: A Lattice-Theoretical Fixpoint Theorem and its Applications. Pacific Journal of Mathematics **5** (1955) 285–309
11. Barringer, H., Fisher, M., Gabbay, D., Gough, G., Owens, R.: METATEM: An Introduction. Formal Aspects of Computing **7** (1995) 533–549
12. Laroussinie, F., Larsen, K.G., Weise, C.: From Timed Automata to Logic - And Back. In: Mathematical Foundations of Computer Science, 20th International Symposium (MFCS'95). Volume 969 of LNCS., Springer Verlag (1995) 529–539
13. Wong Toi, H., Hoffmann, G.: The Control of Dense Real Time Discrete Event Systems. In: Decision and Control, 30th IEEE Conference. (1991) 1527–1528
14. Alur, R., Courcoubetis, C., Halbwachs, N., Dill, D.L., Wong Toi, H.: Minimization of Timed Transition Systems. In: Concurrency Theory, 3rd International Conference (CONCUR'92). Volume 630 of LNCS., Spinger Verlag (1992) 340–354
15. Abrial, J.R., Borger, E., Langmaack, H.: Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control. Volume 1165 of LNCS. Springer Verlag (1996)
16. Cerans, K., Godskesen, J.C., Larsen, K.G.: Timed Modal Specifications – Theory and Tools. In: Computer Aided Verification, 5th International Conference (CAV'93). Volume 697 of LNCS., Springer Verlag (1993) 253–267
17. García Duque, J., Pazos Arias, J.J.: Reasoning over Inconsistent Viewpoints: How levels of agreement can evolve? In: Living With Inconsistency, 2nd International Workshop (ICSE'01). (2001)

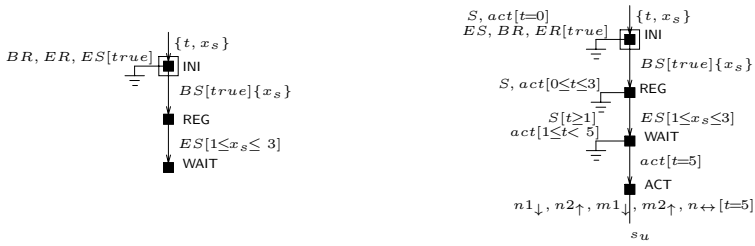
A Controller Synthesis

In this appendix we outline part of the SCTL/MUS-T design process in [8]. At the beginning, init (S_{INI}) and regulation rules (S_{REG}) are applied, synthesized MUS-T prototype is showed in figure 3(a)[6]. Then, context of entering actuation is identified. However, rule S^0_{EA}, which establishes the context of the first actuation, is never applicable; to be precise S^0_{EA} is such that its declarative part is 0 in every instance of locations INI and REG, i.e., rule will not be applicable now nor future evolutions; and $\in \{0, \frac{1}{2}\}$ in instances of location WAIT.

By animating the actual prototype, we can check that a time delay 5 can only be reached by unspecified parts of the model: both by unspecified discrete

[6] In the figures representing MUS-T models only partially unspecified events (dashed lines) are showed. Transitions which are not showed are totally unspecified transition with target location s_u.

transitions in locations {INI, REG}; or by unspecified temporal transitions in location REG. As enabling discrete transitions entails falsifying regulation completion (R_1), the designer expectations meet enabling temporal transitions in the location WAIT. A new scenario $S^2_{REG} := ES\ \forall\Rightarrow\bigcirc\ (\emptyset\ \forall\Rightarrow_+\ true)$ is incorporated. In this way, S^0_{EA} turns into applicable in the location WAIT.



(a)

I_{np}(INI) : $false$
I_p(REG) : $x_s \leq 3$, I_{np}(REG) = $x_s > 3$
I_u(WAIT) : $true$

(b)

I_p(WAIT) : $x_s \geq 1$
I_{np}(ACT) : $t \geq 5$

Fig. 3. 3(a): Rules $\{S_{INI}, S_{REG}\}$. 3(b): Rules $\{S^0_{EA}, S^1_{EA}\}$, S^2_{EA} is a conflicting rule. S stands for the set $\{n1_\downarrow, n2_\uparrow, m1_\downarrow, m2_\uparrow, n_\leftrightarrow\}$.

Then, entering actuation (S_{EA}) scenarios are synthesized (figure 3(b)). Synthesizing event *act* as possible makes S^2_{EA} applicable and $\neg act$ is synthesized in every discrete successor of location ACT. However, in these successors S^0_{EA} is also applicable and a conflict between S^0_{EA} and S^2_{EA} is discovered. Conflict arises from the fact that S^0_{EA} enables consecutive events *act* within 0 delay if a 5 time delay exists from event BS. So, S^0_{EA} is refined as:

$$S^0_{EA} := (x.\emptyset \Rightarrow \mathcal{EP}[BS \wedge x = 5]) \wedge \mathcal{A}[\neg act\ \mathcal{S}\ BS] \Rrightarrow act$$

By applying rules $\{S_{ACT}, S_{AM}\}$ the model in figure 4(a) is obtained. In the synthesis an WLUM [7] criterion has been selected for the rule S^4_{AM}; the remainder rules does not match up any criterion. Once rules have been synthesized, controller requirements are verified over the current prototype. Relevant results are the following ones:

– **Task Completion**: $\vdash R_1 \in \{\widehat{\frac{1}{2}}, \frac{1}{2}\}$, so it is not falsified in the current prototype. That is, once regulation starts there are not executions in which regulation does not finish ($\neg ES$) within the specified interval. However a totally-unspecified model would provide the same ($\frac{1}{2}$). The minimal model

[7] Examples only use WLUM (**W**ithout **L**oss of **U**nspecification in the **M**odel) criterion. Intuitively, WLUM criterion selects existing locations in the model in which specified behavior in a scenario can be reached without making changes.

obtained by turning unspecified behaviors into non-possible ones results in truth values $\in \{\widehat{\frac{1}{2}}, 1\}$, that is, the model satisfies the requirement or the requirement is non-applicable. Finally, R_2 requirement is always $\frac{1}{2}$ both in the actual model and in the minimal one, since it is an optional behavior.

– **Actuation cycle**: As the weakest requirement, $\vdash R_4 \in \{\widehat{\frac{1}{2}}, \frac{1}{2}\}$, that is, although R_4 is not falsified, an actuation cycle within 5 seconds is not ensured.

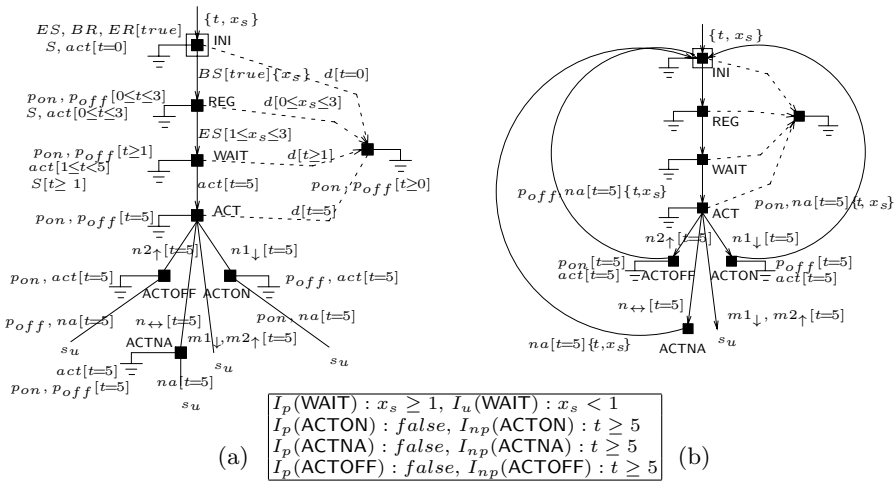


Fig. 4. 4(a): Rules $\{S_{ACT}, S_{AM}\}$. d stands for a generic discrete transition. 4(b): Rule S_{NC} with WLUM criterion + implicit reset.

On the basis of the last result, cyclic behavior should be synthesized by means of S_{NC} rule. New prototype is showed in figure 4(b) where an WLUM criterion with implicit reset has been selected. By validating this model, we discover instances of location WAIT being potential deadlock. Avoiding this potential deadlock can be reached by restricting stop invariant or enabling discrete evolution for $t > 5$. Once again, controller requirements are verified:

– **Task Completion**: Same results.

– **Actuation cycle**: Although $\vdash R_4 \in \{\widehat{\frac{1}{2}}, \frac{1}{2}, 1\}$, it is the weakest requirement in this set. By verifying R_3 requirement in the minimal model, we obtain truth values $\in \{0, \widehat{\frac{1}{2}}\}$. Animating the counterexample, we can discover executions, staying in WAIT infinitely, that contain a temporal delay greater than 5 without an *act* possible-event. As we saw above, WAIT is a potential deadlock. Restricting the time model can stay in this location is needed: $S^3_{ACT} := act\ \forall \Rrightarrow_+\ false$; and, consequently, $I_p(\mathsf{WAIT}) = x_s \geq 1 \wedge t \leq 5$. In this way, $\vdash R_3 \in \{\widehat{\frac{1}{2}}, 1\}$ in the minimal model, i.e., requirement is satisfied or is non-applicable ($\widehat{\frac{1}{2}}$).

– **Cycle Sequence**: Both requirements result in truth values $\in \{\widehat{\frac{1}{2}}, \frac{1}{2}\}$. However, in the minimal model, R_7 requirement is falsified in instances in which it is applicable ($\neq \widehat{\frac{1}{2}}$), that is, there is not any execution in which report computation is carried out.

Applying S_{RC} rule, the controller fragment in figure 5(a) is obtained where report computation is still optional (unspecified transition). S_{RC} synthesis makes S^3_{EA} rule applicable and $\{S_{ACT}, S_{AM}\}$ as well (figure 5(a), WLUM criterion selected). However, in this model, R_2 requirement (report completion) is impossible to satisfy (truth value $\frac{1}{4}$) in some instances. Misscompletion arises from the fact that actuation urgency (every 5 seconds) disables ER. So, it is necessary to refine report computation, that is, BR only must be enabled when the system has enough time both to complete this task (3 seconds), and to enter actuation task (5 seconds from the last actuation):

$$S^2_{RC} := (x.\emptyset \Rightarrow \mathcal{AP}[act \wedge x > 2]) \Rrightarrow \neg BR$$

Besides all events are forbidden until report computation is completed:

$$S^3_{RC} := (\emptyset \ \forall\Rightarrow\!\odot\ BR) \ \overset{\forall\rightarrow}{\forall\Rrightarrow\!\bigcirc} \ (x_r < 3 \ \forall\Rightarrow\!\bigcirc \ false)$$

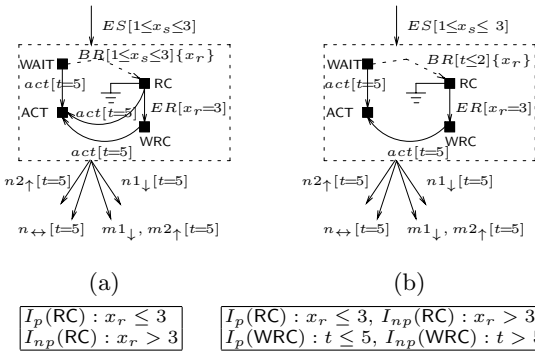


Fig. 5. MUS-T fragment obtained by rules S_{RC}.

In this way the controller fragment in figure 5(b) is obtained in which degree of satisfaction of R_2 is $\frac{3}{4}$. That is, although report computation is still optional, it is impossible to falsify in future evolutions. Besides R_2 will be satisfied (1) turning BR transition into a possible one.