Object models with temporal constraints

Alessandro Cimatti cimatti@fbk.eu   Marco Roveri roveri@fbk.eu   Angelo Susi susi@fbk.eu   Stefano Tonetta [*] tonettas@fbk.eu

Fondazione Bruno Kessler - IRST
Via Sommarive 18, 38050 Povo (TN) Italy

Abstract

Flaws in requirements often have a negative impact on the subsequent development phases. In this paper, we propose a novel formalism for the formal representation and validation of requirements. The formalism allows us to represent and reason about object models and their temporal evolution. The key ingredients are class diagrams to represent the objects in the scenarios, fragments of first order logic to deal with the relationships between their attributes and with rich data, and elements of temporal logic operators to deal with the dynamic evolution of the scenario. Formal validation is carried out by means of satisfiability checking, for which we propose a novel procedure based on the reduction to checking the language non-emptiness of a Fair Transition System.

1. Introduction

A majority of the problems encountered in advanced development phases are caused by flaws in the requirements. For this reason, the development of techniques and tools for the formal analysis of requirements is an important goal for safety-critical applications and/or software.

A key factor is clearly the choice of the formal language used to formalize the requirements. This choice is always argument of debates and implies trade-offs between expressiveness, decidability and complexity. The difficulties lie in the fact that the requirements for such applications often involve several dimensions: on the one side, the ability to express the relationships among the objects having an active role in the target application, involving rich data types; on another side, static constraints over their attributes must be combined with constraints on their temporal evolution.

In this paper, we address this problem by making two contributions. First, we propose a novel framework where elements from several formalisms are combined, in order to enable for a natural specification. Second, we propose an automatic procedure for satisfiability, leveraging recent advances in verification.

We characterize the objects and their attributes using class diagrams, that induce the signature of our logic. The logic allows for quantifiers over objects and first-order state formulas. Temporal constraints are expressed by means of temporal operators, resulting in a fragment of first order temporal logic [22, 23]. This logic can also be seen as a "standard" temporal logic whose atoms are constraints in a first-order theory. We formally define the syntax and the semantics of the language. We see this combination as improving over known formalisms in several ways. With respect to OCL [25] and Alloy [19], we provide the ability to express temporal constraints in an adequate manner, while compared to propositional temporal logic [15] we provide the ability to deal with richer data and objects.

In terms of verification, we provide a satisfiability procedure, which is a major building block for requirements validation, based on the following steps. First, we create an equi-satisfiable formula ϕ' whose variables range over finite domains; second, we create a fair transition system whose accepted language is non-empty iff ϕ' is satisfiable; finally, we check for the language accepted by the fair transition being non-empty. Well-known decision procedures [24, 5, 27] for equalities and uninterpreted functions can be used to automatize the finite instantiation of objects. Emerging symbolic model checking techniques, based on the use of SMT solvers and abstraction [20, 6], can be used to efficiently check for the non-emptiness of the language of the fair transition system.

The paper is structured as follows. In Section 2 we present a motivating scenario. In Section 3 we formalize the notion of class diagrams, and in Section 4 we define static constraints over them. In Section 5 we introduce temporal constraints. In Section 6 we present our approach to satisfiability. In Section 7 we compare our approach with the state of the art, and in Section 8 we draw conclusions and discuss directions for future work.

[*]Supported by the Provincia Autonoma di Trento (project ANACONDA).

2. Running example: informal description

Let us consider a simple railway system describing the interoperability between trains and track-side. Each *track* consists of a sequence of *sections*. Every section is delimited by an initial and a final position. At any time, on each track, a number of *trains* are running. Each train has a *position*, a current section and a *Movement Authority* (MA) consisting of a first and an end section, that delimit the part of the track where the train can move. All positions are given with regard to the start of the track. The track has also a final position, called destination.

Within this context, we consider the following requirements:

R#1 Every point of the track is covered by some section.
R#2 The position of a train is always within the current section.
R#3 The MAs of two trains on a track do not overlap.
R#4 Every train must not go beyond the end section of its MA.
R#5 A train enters the track from the start of the track.
R#6 A train exits the track only if it reaches the destination of the track.
R#7 If not all trains are blocked forever, then eventually some trains will reach the end of the current section.

The following is an expected invariant of the system:

P#1 Two trains are never in the same position.

The following is an expected liveness property:

P#2 If not all trains are blocked forever, then eventually some trains will reach the destination of the track.

3. Class Diagrams

Our work considers as fundamental modelling concepts classes of objects. We use the notation of UML2 class diagrams[1]. In particular, we focus on the concepts of classes, primitive types and attributes of the classes. Attributes have a type and a multiplicity. If the multiplicity is different from 1..1, the attributes are collections of elements. The multiplicity defines the range of the size of such collections.

Definition 1 *A class diagram consists of:*

Primitive Types *A finite set of primitive types* $PT = \{\tau_1, \ldots, \tau_h\}$.

Classes *A finite set of classes* $C = \{c_1, \ldots, c_n\}$.

[1]A description of the concepts can be found in the OMG UML2 metamodel specification documents [1].

Attributes *For each class* $c \in C$*, a finite set of attributes* $c.A = \{c.a_1, \ldots, c.a_m\}$. *Every attribute* $c.a$ *has*

- *a type* $c.a.Type \in C \cup PT$;
- *a multiplicity* $c.a.Mult$ *that defines a bounded integer range* $n..m$, *with* $0 \leq n \leq m$; *we write* $\min(c.a.Mult)$ *for* n, *and* $\max(c.a.Mult)$ *for* m.

In the following, we assume a class diagram D is given. For the sake of simplicity, we limit the description to a core subset of class diagrams. However, our techniques can be straightforwardly applied to encompass other elements (e.g. associations, aggregations, inheritance). We also assume that the attributes are *sequences* of elements, but we can handle similarly the other UML collection types such as *sets* and *bags*. The main restriction, from the point of view of expressiveness, comes from requiring that multiplicity on the attributes is bounded. This restriction is essential, because it allows us to deal with guarded quantifications. On the other hand, this limitation appears to be acceptable in many practical cases. Note that this does not mean we assume a restricted number of objects: in our running example, only a bounded number of trains are allowed to run on a track at a time, but infinitely many different trains may pass over the track.

3.1 Running example

The class diagram of Figure 1 defines the ontology of our case study. Four concepts are represented via classes: the Track, the Train, the Movement Authority (MA) and the Section.

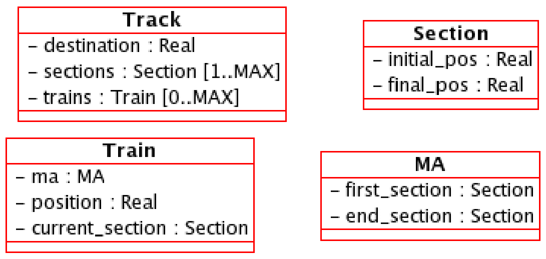

Figure 1. A class diagram

We consider the primitive type Real. The classes have their properties described via attributes. In the class Train, the attribute *position* has type Real, while the attribute *ma*, representing the assigned movement authority (for a given Train), has type MA. In the class Track, the attributes *trains* and *sections* are associated with multiplicities $[0 \ldots \text{MAX}]$ and $[1 \ldots \text{MAX}]$, respectively.

4. Static constraints

4.1 Syntax

Given a class diagram D, we define a language of static constraints on D. We assume to have an underlying first-order signature Σ. The symbols in Σ can express relations over the domains of the types in *PT*. For example, Σ can contain constants and functions over Booleans, Reals and Integers. Finally, we assume a given set of variables V_τ for each type $\tau \in C \cup PT$. The language L_D is defined as follows:

Definition 2 (L_D Terms)

Variables *Any variable $v \in V_\tau$ is a term of type τ.*

Constants *Any constant in Σ of type τ is a term of type τ.*

Functions *If $t_1, \ldots, t_n$ are terms of type $\tau_1, \ldots, \tau_n$ resp., and f is a function symbol in Σ of type $\tau_1 \rightarrow \ldots \rightarrow \tau_n \rightarrow \tau$, then $f(t_1, \ldots, t_n)$ is a term of type τ.*

Simple Attributes *For any term t of class type $c \in C$, for any attribute $a \in c.A$, if $c.a.Type = \tau$ and $c.a.Mult = 1..1$, then $t.a$ is a term of type τ.*

Multiple Attributes *For any term t of class type $c \in C$, for any attribute $a \in c.A$, if $c.a.Type = \tau$ and $c.a.Mult = n..m$ with $n..m \neq 1..1$, then $t.a$ is a term of type COLL_τ and $t.a.size$ is a term of type $n..m$.*

The distinction between simple and multiple attributes respects the UML standard [1]. For example, if v is a variable of class c and a is an Integer attribute in $c.A$, then if $a.Mult = 1..1$ then $v.a$ is an Integer, otherwise $v.a$ is a collection of Integers.

Definition 3 (L_D Formulas)

Relations *If $t_1, \ldots, t_n$ are terms of type $\tau_1, \ldots, \tau_n$ and R is an n-ary relation in Σ over $\tau_1, \ldots, \tau_n$, then $R(t_1, \ldots, t_n)$ is an L_D formula.*

Comparisons *If t_1 and t_2 are both terms of type $\tau \in C$, then $t_1 = t_2$ is an L_D formula.*

Collection membership *If t_1 and t_2 are L_D terms of type τ and COLL_τ respectively, then $t_1 \in t_2$ is an L_D formula.*

Boolean Combinations *If φ_1 and φ_2 are L_D formulas, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$ are L_D formulas.*

Quantifiers *If φ is an L_D formula, v is a variable of type $\tau \in C$, and t is a term of type COLL_τ, then $\forall v \in t.\varphi$ is an L_D formula.*

We denote with L_Σ the language L_D in the case there are no attribute symbols.

4.1.1 Abbreviations

We use the following standard abbreviations:

- $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$;
- $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$;
- $\exists x \in t.(\varphi) \equiv \neg\forall x \in t.(\neg\varphi)$;
- $t_1 \notin t_2 \equiv \neg(t_1 \in t_2)$.

When we specify the constraints of a particular class $c \in C$, we assume there exists a variable THIS of type c, and we abbreviate $\mathrm{THIS}.a$ with a for all attributes $a \in c.A$.

4.2 Semantics

A class diagram D with static constraints is interpreted over object models. An object model M consists of a universe $\mathcal{U}$ and of an interpretation $\mathcal{I}$ of the symbols in Σ and of the attribute symbols of D. In particular, every type $\tau \in C \cup PT$ is associated with a non-empty subset of the universe $U_\tau \subseteq \mathcal{U}$, called the domain. For every class $c \in C$, for every attribute $a \in c.A$, $\mathcal{I}(a)$ is a function from U_c to $U_{a.Type}^{a.Mult}$ where $U_{a.Type}^{a.Mult}$ denotes the set of tuples over $U_{a.Type}$ whose size ranges over $a.Mult$.

We consider a first-order Σ-theory $\mathcal{T}$ that constrains the interpretation of the primitive symbol in Σ. We consider only models M that satisfy $\mathcal{T}$ ($M \models \mathcal{T}$).

Given a term t, an object model M, and an assignment μ to the free variables of t, we define the interpretation $[\![t]\!]_{\langle M,\mu\rangle}$ as follows:

Definition 4 (Interpretation of L_D Terms)

Variables $[\![v]\!]_{\langle M,\mu\rangle} = \mu(v)$.

Constants $[\![c]\!]_{\langle M,\mu\rangle} = \mathcal{I}(c)$.

Functions $[\![f(t_1, \ldots, t_n)]\!]_{\langle M,\mu\rangle} = \mathcal{I}(f)([\![t_1]\!]_{\langle M,\mu\rangle} \cdots [\![t_n]\!]_{\langle M,\mu\rangle})$.

Simple Attributes *If $c.a.Mult = 1..1$, and $\mathcal{I}(a)([\![t]\!]_{\langle M,\mu\rangle}) = \langle q\rangle$ for some $q \in U_{c.a.Type}$, then $[\![t.a]\!]_{\langle M,\mu\rangle} = q$.*

Multiple Attributes *If $c.a.Mult = n..m$, then*
$[\![t.a]\!]_{\langle M,\mu\rangle} = \mathcal{I}(a)([\![t]\!]_{\langle M,\mu\rangle})$
$[\![t.a.size]\!]_{\langle M,\mu\rangle} = |\mathcal{I}(a)([\![t]\!]_{\langle M,\mu\rangle})|$.

Given a formula ϕ, an object model M, and an assignment μ to the free variables of ϕ, we define the relation $\langle M,\mu\rangle \models \phi$ as follows:

Definition 5 (Interpretation of L_D Formulas)

Relations $\langle M,\mu\rangle \models R(t_1, \ldots, t_n)$ *iff* $\mathcal{I}(R)([\![t_1]\!]_{\langle M,\mu\rangle} \cdots [\![t_n]\!]_{\langle M,\mu\rangle})$ *holds.*

Comparisons $\langle M, \mu\rangle \models t_1 = t_2$ *iff* $[\![t_1]\!]_{\langle M,\mu\rangle} = [\![t_2]\!]_{\langle M,\mu\rangle}$.

Collection membership $\langle \omega, \mu\rangle \models t_1 \in t_2$ *iff, for some* i, $1 \leq i \leq |[\![t_2]\!]_{\langle \omega,\mu\rangle}|$, $[\![t_1]\!]_{\langle \omega,\mu\rangle}$ *is the* i*-th element of* $[\![t_2]\!]_{\langle \omega,\mu\rangle}$.

Boolean Combinations
$\langle M, \mu\rangle \models \neg\varphi_1$ *iff* $\langle M, \mu\rangle \not\models \varphi_1$,
$\langle M, \mu\rangle \models \varphi_1 \wedge \varphi_2$ *iff* $\langle M, \mu\rangle \models \varphi_1$ *and* $\langle M, \mu\rangle \models \varphi_2$.

Quantifiers $\langle M, \mu\rangle \models \forall v \in t.\varphi$ *iff, for all* $q \in [\![t]\!]_{\langle M,\mu\rangle}$, $\langle M, \mu[q/v]\rangle \models \varphi$.

4.3 Running example

Some of the requirements of the running example can be expressed as static constraints on the class $Track$:

R#1 $(\exists s \in sections.(s.initial\_pos = 0)) \wedge$
$(\forall s \in sections.(s.final\_pos = destination \vee$
$\exists n \in sections.(n.initial\_pos \leq s.final\_pos \wedge$
$n.final\_pos > s.final\_pos)))$

R#2 $\forall t \in trains.(t.position \geq$
$t.current\_section.initial\_pos \wedge$
$t.position \leq t.current\_section.final\_pos)$

R#3 $\forall t_1, t_2 \in trains.((t_1 \neq t_2) \rightarrow$
$(t_1.ma.first\_section.initial\_pos >$
$t_2.ma.end\_section.final\_pos \vee$
$t_2.ma.first\_section.initial\_pos >$
$t_1.ma.end\_section.final\_pos))$

5. Temporal Constraints

5.1 Syntax

We consider a fragment of the First-Order Temporal Logic of Manna and Pnueli ([23]), and we combine temporal operators with regular expressions in order to get ω-regular expressiveness [21]. As before, we consider a signature Σ over the primitive types in *PT*, and a set of variables V_τ for each type $\tau \in PT \cup C$. We define the temporal language TL_D as follows.

Definition 6 (**TL_D temporal terms**) *A TL_D temporal term of type τ either is an L_D term of type τ or is an expression* $\text{NEXT}(t)$*, where t is a L_D term of type τ.*

Definition 7 (**TL_D transition expressions**)

Relations *If $t_1, \ldots, t_n$ are temporal terms of type $\tau_1, \ldots, \tau_n$, R is an n-ary relation in $\Sigma \cup \Sigma_U$ over $\tau_1, \ldots, \tau_n$, then $R(t_1, \ldots, t_n)$ is a TL_D transition expression.*

Comparisons *If t_1 and t_2 are temporal terms of the same class type $\tau \in C$, then $t_1 = t_2$ is a TL_D transition expression.*

Collection membership *If t_1 and t_2 are temporal terms of type τ and* COLL_τ *respectively, then $t_1 \in t_2$ is a TL_D transition expression.*

Boolean Combinations *If φ_1 and φ_2 are transition expressions, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$ are TL_D transition expressions.*

Quantifiers *If φ is a TL_D transition expression, v is a variable of type $\tau \in C$, and t is a temporal term of type* COLL_τ*, then $\forall t \in o.\varphi$ is a TL_D transition expression.*

Definition 8 (**TL_D regular expressions**)

Transition expressions *Any transition expression is a TL_D regular expression.*

Empty word *ϵ is a TL_D regular expression.*

Regular Operators *If r_1 and r_2 are TL_D regular expressions, then r_1*, $r_1;r_2$, $r_1 : r_2$, $r_1|r_2$, $r_1\&\&r_2$ are TL_D regular expressions.*

Definition 9 (**TL_D formulas**)

Transition expressions *Any transition expression is a TL_D formula.*

Boolean Combinations *If φ_1 and φ_2 are TL_D formulas, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$ are TL_D formulas.*

Temporal Operators *If φ_1 and φ_2 are TL_D formulas, then $\mathbf{X}\ \varphi_1$, $\varphi_1\ \mathbf{U}\ \varphi_2$ are TL_D formulas.*

Suffix Operators *If r is a TL_D regular expression and φ is a TL_D linear temporal formula, then $\{r\}\varphi$ is a TL_D formula.*

We denote with TL_Σ the language TL_D in the case there are no attribute symbols.

5.1.1 Abbreviations

Besides the abbreviations defined in Section 4.1.1, we use the following:

- $\mathbf{F}\ \varphi \equiv \top\ \mathbf{U}\ \varphi$;
- $\mathbf{G}\ \varphi \equiv \neg\mathbf{F}\ \neg\varphi$;
- $r \mapsto \varphi \equiv \neg\{r\}\neg\varphi$.

In the following we use $\varphi[\varphi_2/\varphi_1]$ to denote the TL_D formula obtained by substituting any occurrence of TL_D sub-formula φ_1 of φ with the TL_D formula φ_2.

5.2 Semantics

TL_D formulas are interpreted over sequences of object models. Let ω be an infinite sequence of such models. We denote with ω^i the $i+1$-th element of the sequence, with $\omega^{i..}$ the suffix sequence $\omega^i, \omega^{i+1}, \ldots$.

As before, we assume to have first-order Σ-theory $\mathcal{T}$ that constrains the interpretation of some symbols in Σ. Note that the $\mathcal{T}$ is time independent. For this reason, symbols such as the operations over the primitive types are rigid, i.e., their interpretation does not change over time. On the contrary the value of attributes and other uninterpreted symbols is time-dependent.

Definition 10 (TL_D temporal terms)

Term $[\![t]\!]_{\langle\omega,\mu\rangle} = [\![t]\!]_{\langle\omega^0,\mu\rangle}$.

Next Term $[\![\textsc{next}(t)]\!]_{\langle\omega,\mu\rangle} = [\![t]\!]_{\langle\omega^1,\mu\rangle}$.

Definition 11 (TL_D transition expressions)

Primitive relations $\langle\omega,\mu\rangle \models R(t_1,\ldots,t_n)$ *iff* $\mathcal{I}(R)([\![t_1]\!]_{\langle\omega,\mu\rangle}\ldots[\![t_n]\!]_{\langle\omega,\mu\rangle})$ *holds.*

Comparisons $\langle\omega,\mu\rangle \models t_1 = t_2$ *iff* $[\![t_1]\!]_{\langle\omega,\mu\rangle} = [\![t_2]\!]_{\langle\omega,\mu\rangle}$.

Collection membership $\langle\omega,\mu\rangle \models t_1 \in t_2$ *iff, for some i,* $1 \leq i \leq |[\![t_2]\!]_{\langle\omega,\mu\rangle}|$, $[\![t_1]\!]_{\langle\omega,\mu\rangle}$ *is the i-th element of* $[\![t_2]\!]_{\langle\omega,\mu\rangle}$.

Boolean Combinations
$\langle\omega,\mu\rangle \models \neg\varphi_1$ *iff* $\langle\omega,\mu\rangle \not\models \varphi_1$,
$\langle\omega,\mu\rangle \models \varphi_1 \wedge \varphi_2$ *iff* $\langle\omega,\mu\rangle \models \varphi_1$ *and* $\langle\omega,\mu\rangle \models \varphi_2$.

Quantifiers $\langle\omega,\mu\rangle \models \forall v \in t.\varphi$ *iff, for all* $q \in [\![t]\!]_{\langle\omega,\mu\rangle}$, $\langle\omega,\mu[q/v]\rangle \models \varphi$.

Definition 12 (TL_D regular expressions)

Transition expressions $\langle\omega,\mu\rangle \models^{i..j} \varphi$ *iff* $j = i+1$ *and* $\langle\omega^{i..},\mu\rangle \models \varphi$.

Empty word $\langle\omega,\mu\rangle \models^{i..j} \epsilon$ *iff* $i = j$.

Regular Operators
$\langle\omega,\mu\rangle \models^{i..j} r*$ *iff* $i = j$ *or there exists* k, $i < k \leq j$, $\langle\omega,\mu\rangle \models^{i..k} r$, $\langle\omega,\mu\rangle \models^{k..j} r*$;
$\langle\omega,\mu\rangle \models^{i..j} r_1; r_2$ *iff there exists* k, $i \leq k \leq j$, $\langle\omega,\mu\rangle \models^{i..k} r_1$, $\langle\omega,\mu\rangle \models^{k..j} r_2$;
$\langle\omega,\mu\rangle \models^{i..j} r_1 : r_2$ *iff there exists* k, $i < k \leq j$, $\langle\omega,\mu\rangle \models^{i..k} r_1$, $\langle\omega,\mu\rangle \models^{k-1..j} r_2$;
$\langle\omega,\mu\rangle \models^{i..j} r_1|r_2$ *iff* $\langle\omega,\mu\rangle \models^{i..j} r_1$ *or* $\langle\omega,\mu\rangle \models^{i..j} r_2$;
$\langle\omega,\mu\rangle \models^{i..j} r_1\&\&r_2$ *iff* $\langle\omega,\mu\rangle \models^{i..j} r_1$ *and* $\langle\omega,\mu\rangle \models^{i..j} r_2$.

Definition 13 (TL_D formulas)

Boolean Combinations
$\langle\omega,\mu\rangle \models \neg\varphi_1$ *iff* $\langle\omega,\mu\rangle \not\models \varphi_1$,
$\langle\omega,\mu\rangle \models \varphi_1 \wedge \varphi_2$ *iff* $\langle\omega,\mu\rangle \models \varphi_1$ *and* $\langle\omega,\mu\rangle \models \varphi_2$.

Temporal Operators
$\langle\omega,\mu\rangle \models X\varphi$ *iff* $\langle\omega^{1..},\mu\rangle \models \varphi$;
$\langle\omega,\mu\rangle \models \varphi_1 \mathbf{U} \varphi_2$ *iff there exists* $i \geq 0$ *such that* $\langle\omega^{i..},\mu\rangle \models \varphi_2$ *and for all* $0 \leq j < i$ $\langle\omega^{j..},\mu\rangle \models \varphi_1$.

Suffix Operators $\langle\omega,\mu\rangle \models \{r\}\varphi$ *iff there exists* $i \geq 0$ *such that* $\langle\omega^{i..},\mu\rangle \models \varphi$ *and* $\langle\omega,\mu\rangle \models^{0..i+1} r$.

Definition 14 (Satisfiability) *Given a formula* ϕ*, the satisfiability problem consists of finding a sequence of models* ω*, and an assignment* μ *to the free variables of* ϕ*, such that* $\langle\omega,\mu\rangle \models \phi$.

Definition 15 (Equi-Satisfiability) *Given two formulas* ϕ_1 *and* ϕ_2*, we say that* ϕ_1 *and* ϕ_2 *are equi-satisfiable when* ϕ_1 *is satisfiable iff* ϕ_2 *is satisfiable.*

Definition 16 (Entailment) *Given two formulas* ϕ *and* ψ*, the entailment problem consists of proving that, for all sequence of models* ω*, and assignments* μ *to the free variables of* ϕ *and* ψ*, if* $\langle\omega,\mu\rangle \models \phi$ *then* $\langle\omega,\mu\rangle \models \psi$.

It is possible to check whether ϕ entails ψ by checking the unsatisfiability of $\phi \wedge \neg\psi$. If ϕ entails ψ, then $\forall x.(\phi(x) \rightarrow \psi(x))$ is valid (and not the stronger $\forall x.(\phi(x)) \rightarrow \forall x.(\psi(x))$).

Remark 1 *When* Σ *contains only uninterpreted Boolean constant symbols (thus, with* $\mathcal{T}$ *empty),* TL_Σ *turns to be propositional Linear-time Temporal Logic (LTL) [26] extended with Regular Expressions (RELTL) [21].*

5.3 Running example

We can express the remaining requirements and the properties of the running examples as TL_D formulas on the class $Track$:

R#4 $\mathbf{G}\ \forall t \in trains.(t.current\_section = t.ma.end\_section \rightarrow \textsc{next}(t.position) \leq t.current\_section.final\_position)$

R#5 $\mathbf{G}\ \forall t \in trains.(t \notin \textsc{next}(trains) \rightarrow t.position = destination)$

R#6 $\mathbf{G}\ \forall t \in \textsc{next}(trains).(t \notin trains \rightarrow \textsc{next}(t.position) = 0)$

R#7 $\mathbf{G}\ ((\neg\mathbf{G}\ \forall t \in trains.(\textsc{next}(t.position) = position)) \rightarrow \mathbf{F}\ \exists t \in trains.(\textsc{next}(t.position) = t.current\_section.final\_position))$

P#1 $\mathbf{G}\ \forall t_1 \in trains.\forall t_2 \in trains.(t_1.position \neq t_2.position)$

P#2 $\mathbf{G}\ ((\neg\mathbf{G}\ \forall t \in trains.(\textsc{next}(t.position) = position)) \rightarrow \mathbf{F}\ \exists t \in trains.(\textsc{next}(t.position) = destination))$

6. Formal Analysis

6.1. Formal analysis and satisfiability

Given a set of requirements $\Phi = \{\phi_1, \ldots, \phi_n\}$, the requirement validation process consists of checking if the requirements are: *consistent*, i.e. if they do not contain some contradiction; *not too strict*, i.e. if they do allow some desired behavior ψ_d; *not too weak*, i.e. if they rule out some undesired behavior ψ_u. Each of these checks can be carried out by solving a satisfiability problem: consistency is checked by solving the satisfiability problem of $\bigwedge_{1\leq i\leq n} \phi_i$; the set of requirements is not too strict if $\bigwedge_{1\leq i\leq n} \phi_i \wedge \psi_d$ is satisfiable; it is not too weak if the $\bigwedge_{1\leq i\leq n} \phi_i \wedge \neg\psi_u$ is unsatisfiable.

We now outline our satisfiability procedure, based on a sequence of steps, each detailed in the rest of this section:

1. the formula is rewritten into an equi-satisfiable one free of quantifiers, by introducing a finite number of new function symbols;
2. the formula is rewritten into an equi-satisfiable one free of attributes symbols, by encoding the object variables into a finite domain;
3. the formula is encoded into a Fair Transition System (FTS) [22], which is a symbolic representation of an infinite-state system; the conversion is carried out by separating the temporal part from the first-order constraints;
4. finally, the FTS can be checked for language non-emptiness.

6.2 Removing guarded quantifiers

Given a TL_D formula ϕ, we can obtain an equi-satisfiable quantifier-free formula ϕ_G. First, for every class $c \in C$, for every attribute $a \in c.A$ we introduce a new function symbol a_i for all $1 \leq i \leq \max(c.a.\mathit{Mult})$. Second, we perform the following top-down transformation

- $\chi(\forall v \in e.a.(\phi)) = \bigwedge_{1\leq i\leq m}(e.a.size \geq i \rightarrow \phi[e.a_i/v])$, with $m = \max(e.a.\mathit{Mult})$.
- $\chi(\neg\varphi_1) = \neg\chi(\varphi_1)$.
- $\chi(\varphi_1 \wedge \varphi_2) = \chi(\varphi_1) \wedge \chi(\varphi_2)$.
- $\chi(t \in e.a) = \bigvee_{1\leq i\leq m}(e.a.size \geq i \wedge t = e.a_i)$, with $m = \max(e.a.\mathit{Mult})$.
- $\chi(\phi) = \phi$ otherwise.

Theorem 1 *φ is satisfiable iff $\chi(\varphi)$ is satisfiable.*

Proof. If $\langle\omega, \mu\rangle \models \varphi$, we consider the extension ω' of ω that interprets the symbols introduced by χ as follows: for all $j \geq 0$, for all class $c \in C$, for all $a \in c.A$ with multiplicity different from $1..1$, for all $o \in U_c$, $\mathcal{I}^j(a_i)(o)$ is the i-th element of $\mathcal{I}^j(a)(o)$ if $|\mathcal{I}^j(a)(o)| \geq i$, otherwise $\mathcal{I}^j(a_i)(o)$ is equal to an arbitrary element of $U_{a.\mathit{Type}}$. Then $\langle\omega', \mu\rangle \models \chi(\varphi)$.

If $\langle\omega, \mu\rangle \models \chi(\varphi)$, we consider the extension ω' of ω that interprets the symbols not in χ as follows: for all $j \geq 0$, for all class $c \in C$, for all $a \in c.A$, for all $o \in U_c$, $\mathcal{I}^j(a)(o)$ is the sequence $\langle o_1, \ldots, o_h\rangle$ where $h = |\mathcal{I}^j(a)(o)|$ and $o_i = \mathcal{I}^j(a_i)(o)$. Then $\langle\omega', \mu\rangle \models \varphi$. $\diamond$

6.3. Automatic finite instantiation

Given a formula ϕ, we can translate the formula into an equi-satisfiable one where all object variables are encoded into a finite domain. More precisely, given a class type c, let n_c be the number of temporal terms of type c that occur in ϕ. For all variable x of type c that occurs in ϕ, let us introduce a new variable x_b over the range $[1..n_c]$. For all attributes a of the class c that occurs in ϕ, let introduce a new uninterpreted function a_b such that: if a is of class type d, then a_b is of type $[1..n_c] \rightarrow [1..n_d]$; if a is of primitive type τ, then a_b is of type $[1..n_c] \rightarrow U_\tau$. Let ϕ_b be the result of substituting in ϕ every variable v with v_b, and every attribute expression $e.a$ recursively with $a_b(e)$.

Theorem 2 *ϕ and ϕ_b are equi-satisfiable.*

Proof. Let $\langle\omega, \mu\rangle \models \phi$. For every $i \geq 0$, for every class c, let us consider the set $D_c^i \subseteq U_c$ such that $o \in D_c^i$ if and only if there exists a temporal term t in ϕ such that $[\![t]\!]_{\langle\omega^i,\mu\rangle} = o$. Let us consider an infinite sequence of injective functions $m_c^i : D_c^i \rightarrow [1..n_c]$, such that for all $i \geq 0$, for all terms t in ϕ of type c, if $[\![t]\!]_{\langle\omega^i,\mu\rangle} = [\![t]\!]_{\langle\omega^{i+1},\mu\rangle} = o \in D_c^i$, then $m_c^i(o) = m_c^{i+1}(o)$. Let us consider a model $\langle\omega_b, \mu_b\rangle$ such that for all $i \geq 0$, for all terms t_b in ϕ_b of type c, if $[\![t]\!]_{\langle\omega^i,\mu\rangle} = o$ then $[\![t_b]\!]_{\langle\omega_b^i,\mu_b\rangle} = m_c^i(o)$. Then $\langle\omega_b, \mu_b\rangle \models \phi_b$

Let $\langle\omega_b, \mu_b\rangle \models \phi_b$. For every $i \geq 0$, for every class c, let us consider an injective function $m_c : [1..n_c] \rightarrow U_c$. Let us consider a model $\langle\omega, \mu\rangle$ such that for all $i \geq 0$, for all terms t in ϕ of type c, if $[\![t_b]\!]_{\langle\omega_b^i,\mu_b\rangle} = j$ then $[\![t]\!]_{\langle\omega^i,\mu\rangle} = m_c(j)$. Then $\langle\omega, \mu\rangle \models \phi$ $\diamond$

Note in particular, that if ϕ contains only Boolean attributes, ϕ_b is propositional, and we can check its satisfiability by classic finite-state model checking.

6.4. Reduction to Fair Transition Systems Non-Emptiness

Fair Transition Systems (FTS) [22] are a symbolic representation of infinite-state systems. We generalize the definition in order to use formulas with symbols in Σ.

Definition 17 (FTS) *A* Fair Transition System (FTS) *is a tuple $S = \langle \Sigma, V, I, T, F \rangle$, where*

- *Σ is a first-order signature, that defines the state space; the states of the system are defined as the interpretations of the symbols in Σ; we consider only the interpretations that satisfy a given Σ-theory $\mathcal{T}$.*
- *V is a finite set of variables that represent parameters of the system.*
- *I is the initial condition expressed as a L_Σ formula.*
- *T is the transition condition expressed as a TL_Σ transition expression.*
- *F is the set of fairness conditions, each condition expressed as a L_Σ basic expression.*

Given an infinite sequence ω of Σ-interpretations and an assignment μ to the parameters in V, S accepts $\langle \omega, \mu \rangle$ iff $\langle \omega^{i\cdot\cdot}, \mu \rangle \models T$ for every $i \geq 0$, $\langle \omega^0, \mu \rangle \models I$, and, for all $\psi \in F$, there exist infinitely many i, such that $\langle \omega^i, \mu \rangle \models \psi$. The language $\mathcal{L}(S)$ is defined as the set of pairs $\langle \omega, \mu \rangle$ accepted by S.

In the case of RELTL, there exist efficient techniques to convert a temporal formula ϕ into an equivalent FTS S_ϕ (see, e.g., [10]). Formally,

Theorem 3 *Given a TL_Σ formula ϕ that contains only uninterpreted Boolean constants and Boolean variables V, there exists an FTS $S_\phi = \langle \Sigma, V, I, T, F \rangle$ such that $\langle \omega, \mu \rangle \models \phi$ iff $\langle \omega, \mu \rangle \in \mathcal{L}$.*

Thus, the check for the satisfiability of ϕ can be performed verifying that $\mathcal{L}(S_\phi) \neq \emptyset$.

As for the general case, we solve the satisfiability problem of a temporal formula ϕ as follows. First, for every transition expression ψ in ϕ, we introduce a new Boolean uninterpreted constant a_ψ. Second, we consider the Boolean abstraction ϕ^A of ϕ where every transition expression ψ has been substituted with the corresponding Boolean constant a_ψ.

Theorem 4 *ϕ is equi-satisfiable to $\phi^A \wedge \bigwedge_{\psi \in \phi} \mathbf{G}\ (a_\psi \leftrightarrow \psi)$.*

Proof. Suppose $\langle \omega, \mu \rangle \models \phi$, where $\mathcal{I}^i$ is the interpretation of ω^i for all $i \geq 0$. Let us consider a new sequence of models ω' such that, for all $i \geq 0$, the interpretation $\mathcal{I}'^i$ of ω'^i is defined as follows: $\mathcal{I}'^i(s) = \mathcal{I}^i(s)$ for all symbols occurring in ϕ, while $\mathcal{I}'^i(a_\psi) = \top$ iff $\langle \omega'^i, \mu \rangle \models \psi$. Then $\langle \omega', \mu \rangle \models \phi^A \wedge \bigwedge_{\psi \in \phi} \mathbf{G}\ (a_\psi \leftrightarrow \psi)$.

Suppose $\langle \omega, \mu \rangle \models \phi^A \wedge \bigwedge_{\psi \in \phi} \mathbf{G}\ (a_\psi \leftrightarrow \psi)$ Note that, for all $i \geq 0$, $\langle \omega^i, \mu \rangle \models a_\psi$ iff $\langle \omega^i, \mu \rangle \models \psi$. Thus $\langle \omega, \mu \rangle \models \phi$. $\diamond$

Theorem 5 *If ϕ^A is equivalent to the FTS S_{ϕ^A}, then $\phi^A \wedge \bigwedge_{\psi \in \phi} \mathbf{G}\ (a_\psi \leftrightarrow \psi)$ is equivalent to the FTS S_ϕ obtained by adding $\bigwedge_{\psi \in \phi} (a_\psi \leftrightarrow \psi)$ to the transition condition of S_{ϕ^A}.*

Proof. Suppose $\langle \omega, \mu \rangle \models \phi^A \wedge \bigwedge_{\psi \in \phi} \mathbf{G}\ (a_\psi \leftrightarrow \psi)$. In particular, $\langle \omega, \mu \rangle \models \phi^A$. Thus, $\langle \omega, \mu \rangle \in \mathcal{L}(S_{\phi^A})$. Moreover, $\langle \omega, \mu \rangle \models \bigwedge_{\psi \in \phi} \mathbf{G}\ (a_\psi \leftrightarrow \psi)$. Then, for all $i \geq 0$, $\langle \omega^{i\cdot\cdot}, \mu \rangle \models \bigwedge_{\psi \in \phi} (a_\psi \leftrightarrow \psi)$. Thus, $\langle \omega, \mu \rangle \in \mathcal{L}(S_\phi)$.

Suppose $\langle \omega, \mu \rangle \in \mathcal{L}(S_\phi)$. Thus, $\langle \omega, \mu \rangle \in \mathcal{L}(S_{\phi^A})$ and $\langle \omega, \mu \rangle \models \phi^A$. Moreover, for all $i \geq 0$, $\langle \omega^{i\cdot\cdot}, \mu \rangle \models \bigwedge_{\psi \in \phi} (a_\psi \leftrightarrow \psi)$. Then $\langle \omega, \mu \rangle \models \phi^A \wedge \bigwedge_{\psi \in \phi} \mathbf{G}\ (a_\psi \leftrightarrow \psi)$. $\diamond$

Corollary 1 *ϕ is satisfiable iff $\mathcal{L}(S_\phi) \neq \emptyset$.*

Therefore, we can check the satisfiability of ϕ, by checking the non-emptiness of the language accepted by S_ϕ.

6.5. Non-Emptiness Checking for FTS

Counterexample-Guided Abstraction Refinement We check the non-emptiness of S_ϕ by means of predicate abstraction. We adopt a counterexample-guided abstract refinement (CEGAR) loop [11], where the abstraction generation and refinement are completely automatized. The loop consists of four phases:

- *abstraction*, where the abstract system is built according to a given set of predicates;
- *verification*, where the non-emptiness of the language of the abstract system is checked; if the language is empty, it can be concluded that also the concrete system has an empty language; otherwise, an infinite trace is produced;
- *simulation*: if the verification produces a trace, the simulation checks whether it is realistic by simulating it on the concrete system; if the trace can be simulated in the concrete system, it is reported as a real witness of the satisfiability of the formula;
- *refinement*: if the simulation cannot find a concrete trace corresponding to the abstract one, the refinement discovers new predicates that, once added to the abstraction, are sufficient to rule out the unrealistic path.

The abstraction is performed by computing a Boolean formula equivalent to:

$$\exists V \exists V' (T(V, V') \wedge \bigwedge_{p \in P} (\hat{v}_p \leftrightarrow p(V) \wedge \hat{v}'_p \leftrightarrow p(V'))) \qquad (1)$$

The quantifiers are removed by passing the formula to a decision procedure, and enumerating the satisfying assignments to the abstract variables $\hat{v}_p, \hat{v}'_p$ [13].

As in [20] and [6], T is a first-order formula and the computation of all solutions is carried out by a SAT-modulo-Theories (SMT) solver.

We encode the simulation of the abstract trace on the concrete system into a Bounded Model Checking (BMC) problem [2], as proposed by [12]. Since the counterexample may be infinite, we consider lasso-shape paths. This way, if the counterexample is given by the abstract states $\hat{s}_0, \ldots, \hat{s}_k$ with $\hat{s}_l = \hat{s}_k$ for $l < k$, we then generate the SMT formula:

$$I(V_0) \wedge \bigwedge_{0 \leq i < k} T(V_i, V_{i+1}) \wedge \bigwedge_{0 \leq i \leq k} S_i(V_i) \wedge \bigwedge_{v \in V} v_l = v_k \qquad (2)$$

where the predicates S_i identify the abstract states $\hat{s}_i$. If the formula is not satisfiable, we refine the abstraction. As noted in [28], this concretization step is sound but not complete: we are fixing a specific lasso shape, so it may happen that the above formula is not satisfiable, but the counterexample is concretizable.

The initial predicates are arbitrarily chosen from the expressions that occur in ϕ. The new predicates discovered during the refinement are generated by analyzing the proof of the unsatisfiability of (2). To this purpose, for instance it is possible to use the interpolation-based techniques described in [8].

Bounded Model Checking via SMT Although incomplete, the CEGAR loop is often able to prove the inconsistency of a property ϕ by finding the right abstraction. On the opposite, if we aim at looking for a witness of the formula's satisfiability, a more effective procedure is to look for a lasso-shape trace of length up to a given bound.

In this case, we encode the simulation of such trace with the following SMT formula:

$$I(V_0) \wedge \bigwedge_{0 \leq i < k} T(V_i, V_{i+1}) \wedge \bigvee_{0 \leq l < k} \bigwedge_{v \in V} v_l = v_k \wedge \bigwedge_{\psi \in F} \bigvee_{l \leq h < k} \psi(V_h) \qquad (3)$$

6.6. Running Example

We first consider the consistency problem of the requirements of the running example. After removing the quantifiers, the formulas will be in the form:

R#1 $\mathbf{G}\ ((\bigvee_{1 \leq i \leq MAX}(sections.size \geq i \wedge (sections_i.initial\_pos = 0))) \wedge (\bigwedge_{1 \leq i \leq MAX}(sections.size \geq i \rightarrow (sections_i.final\_pos = destination \vee \bigvee_{1 \leq j \leq MAX}(sections.size \geq j \wedge sections_j.initial\_pos \leq sections_i.final\_pos \wedge section_j.final\_pos > section_i.final\_pos)))))$

R#2 $\mathbf{G}\ (\bigwedge_{1 \leq i \leq MAX}(trains.size \geq i \rightarrow (trains_i.position \geq trains_i.current\_section.initial\_pos \wedge trains_i.position \leq trains_i.current\_section.final\_pos)))$

R#3 $\mathbf{G}\ (\bigwedge_{1 \leq i \leq MAX}(trains.size \geq i \rightarrow \bigwedge_{1 \leq j \leq MAX}(trains.size \geq j \rightarrow ((trains_i \neq trains_j) \rightarrow (trains_i.ma.first\_section.initial\_pos > trains_j.ma.end\_section.final\_pos \vee trains_j.ma.first\_section.initial\_pos > trains_i.ma.end\_section.final\_pos)))))$

R#4 $\mathbf{G}\ (\bigwedge_{1 \leq i \leq MAX}(trains.size \geq i \rightarrow (trains_i.current\_section = trains_i.ma.end\_section \rightarrow \text{NEXT}(trains_i.position) \leq trains_i.current\_section.final\_pos))$

R#5 $\mathbf{G}\ (\bigwedge_{1 \leq i \leq MAX}(trains.size \geq i \rightarrow ((\bigwedge_{1 \leq j \leq MAX}(\text{NEXT}(trains.size) \geq j \rightarrow \text{NEXT}(trains_j) \neq trains_i)) \rightarrow trains_i.position = destination)))$

R#6 $\mathbf{G}\ (\bigwedge_{1 \leq i \leq MAX}(\text{NEXT}(trains.size) \geq i \rightarrow ((\bigwedge_{1 \leq j \leq MAX}(trains.size \geq j \rightarrow trains_j \neq \text{NEXT}(trains_i))) \rightarrow \text{NEXT}(trains_i.position) = 0)))$

R#7 $\mathbf{G}\ ((\neg \mathbf{G}\ \bigwedge_{1 \leq i \leq MAX}(trains.size \geq i \rightarrow (\text{NEXT}(trains_i.position) = position))) \rightarrow \mathbf{F}\ \bigvee_{1 \leq i \leq MAX}(trains.size \geq i \wedge (\text{NEXT}(trains_i.position) = trains_i.current\_section.final\_pos)))$

The temporal terms of type $Train$ that occur in the formula are $trains_i$, for $1 \leq i \leq MAX$. Thus, we can encode the objects of type $Train$ with $[1..MAX]$. The temporal terms of type $Section$ that occur in the formula are

- $sections_i$ for $1 \leq i \leq MAX$;
- $trains_i.current\_section$ for $1 \leq i \leq MAX$;
- $trains_i.ma.first\_section$ for $1 \leq i \leq MAX$;
- $trains_i.ma.end\_section$ for $1 \leq i \leq MAX$;

Thus, we can encode the objects of type $Section$ with $[1..4*MAX]$. Similarly MAs are encoded into $[1..MAX]$ and $Tracks$ into $[1..1]$.

We use NuSMV [7] to convert the proposition temporal part of the formula into an automaton. The resulting FTS is obtained by combining the automaton with the first-order temporal expressions. We check the language non-emptiness of the automaton by using an extended version of NuSMV that provides CEGAR functionalities [6] and has been interfaced with MathSAT [4]. We were able to prove the consistency of the running example in the case $MAX = 10$ in about 30 secs on an Intel 2.3GHz Laptop equipped with 1Gb of memory running Linux using a memory limit of 512Mb. The two properties were proved not to hold respectively in about 35 secs and 75 secs. The reason of the failure in both cases is due to missing assumptions. In order to modify the specification as to prove the two properties hold several iterations of refinement of the specification aiming to add assumptions and verification of the new specification have been performed. At the end we were able to prove the two properties hold for the final version of the specification.

7. Related Work

Several attempts to the formal specification and validation of requirements have been proposed. The works that aim at solving problems similar to the ones tackled in this paper are Alloy [19], Formal Tropos [16] and OCL [25].

Alloy [19] is a language for describing structural properties of a system relying on a small kernel formal language based on the subset of Z [29] that allows for object modeling. An Alloy specification consists of basic structures representing classes together with constraints and operations describing how the structures change dynamically. Alloy only allows to specify attributes belonging to finite domains (no Reals or Integers). Thus, it would have been impossible to model the Train position of the running example presented in this paper in Alloy. Although Alloy supports the "next" operator ("prime" operator), that allow to specify the temporal evolution of a given object, it does not allow to express properties using LTL and regular expressions (at the basis of the logic presented in this paper). Thus, it is limited to state or transition invariants, and it does not allow to specify fairness conditions that are crucial in our context. Similarly to the approach proposed in this paper, Alloy supports two kinds of analysis: simulation and checking. In simulation the consistency of an invariant or operation is demonstrated by generating witness (a state or a transition). In checking a consequence of the specification is tested by attempting to generate a counterexample of a user specified length. Alloy, in order to perform the verification, requires the user to specify the bounds on the maximum number of class instances and a limit for the counterexample/witness length. Thus, it is only able to prove/disprove a property for the given bounds and for the given length. The approach presented in this paper overcomes these problems with the automatic finite instantiation that preserves satisfiability and with a reduction to known verification techniques that allow to prove/disprove a property regardless of any bound on the witness/counterexample length.

Tropos [3, 30] is a goal-oriented software development methodology which provides a visual modelling language that can be used to define an informal specification, allowing to model intentional and social concepts, such as those of actor, goal, and social relationships between actors. Formal Tropos (FT) [16] extends a Tropos specification with annotations that characterize the valid behaviors of the model. A FT specification consists of a sequence of class declarations such as actors, goals, and dependencies. Each declaration associates a set of attributes to the class. The temporal behavior of the instances is specified by means of temporal constraints expressed in a typed first-order LTL. FT is similar in spirit to the approach proposed in this paper. The difference between the proposed approach and FT are in the expressiveness of the formalization language. FT is limited to LTL temporal operators, while this approach allows to express constraints and properties with a logic that mixes LTL with regular expression (which is more expressive than LTL). FT, like Alloy, requires for the formal analysis the specification of bounds one the maximum number of class instances, and can only deal with finite domain class attributes and constraints over such finite domain values. The novel approach does not require the specification of the bounds on the number of class instances, and allows for the use of class attributes of infinite range like e.g. Integers and Reals.

OCL [25] is a formal language developed as a business modelling language to express additional constraints about the objects in an UML model. It is a pure specification language, so an OCL expression is guaranteed to be without side effects on the model. This means that the state of the system will never change because of the evaluation of an OCL expression, even though an OCL expression can be used to specify a state change. In fact, in a post-condition, and only in this context, the expression can refer to values for each property of an object (via the operator $@pre$) at two different time instants: the value of a property at the start of the operation or method and the value of a property upon completion of the operation or method. This allows to express and predicate on the temporal evolution of a property. Similarly to Alloy, OCL cannot express some temporal properties, such as fairness. In OCL no commitment is done on the possibility to execute analysis, such as model checking, moreover, as a general observation, the language is not decidable.

As far as the combination of temporal with first-order state formulas logic is concerned, we remark that it was proposed by Manna and Pnueli in [22, 23] as specification language for reactive systems. Many works studied the decidability of particular fragments of the logic (cf., e.g., [18, 14, 17]). We also considered a fragment of the language, by restricting temporal quantifications to the guarded quantifiers over bounded sets of objects and to implicit freezing quantifiers that relate next and current values in the transition expressions. Rather than focusing on which are the decidable theories we can use in the state formulas, our work focus more on object models, their specification and validation.

8. Conclusions

In this paper, we have proposed a formalism for the representation and the analysis of requirements. The formalism builds on class diagrams, and combines fragments of first order logic (to describe rich data and relationships between attributes and entities) with temporal operators (to describe the evolution of the scenarios). We also proposed a procedure to automatically check the satisfiability based on the finite domain encoding and on the reduction to language non-emptiness for FTS, and the application of verification techniques based on SMT and abstraction-refinement.

In the future, we plan to extend the expressiveness of the formalism to encompass richer data. We will also investigate enhancements in the verification engine, along the lines outlined in [9], and to more aggressive (problem-specific) abstraction techniques. Finally, we will investigate issues related to contract-based specifications, where only a part of the scenario is considered to be controllable, and to address problems beyond satisfiability, such as realizability and synthesis.

References

[1] UML Version 2.1.2. `http://www.omg.org/spec/UML/2.1.2/`.

[2] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *TACAS*, pages 193–207, 1999.

[3] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[4] R. Bruttomesso, A. Cimatti, A. Franzn, A. Griggio, and R. Sebastiani. The MathSAT 4 SMT Solver. In *CAV*, pages 299–303, 2008.

[5] R. E. Bryant, S. M. German, and M. N. Velev. Exploiting Positive Equality in a Logic of Equality with Uninterpreted Functions. In *CAV*, pages 470–482, 1999.

[6] R. Cavada, A. Cimatti, A. Franzén, K. Kalyanasundaram, M. Roveri, and R. K. Shyamasundar. Computing predicate abstractions by integrating BDDs and SMT solvers. In *FM-CAD*, pages 69–76. IEEE, 2007.

[7] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model checker. *STTT*, 2(4):410–425, 2000.

[8] A. Cimatti, A. Griggio, and R. Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Theories. In *TACAS*, pages 397–412, 2008.

[9] A. Cimatti, M. Roveri, V. Schuppan, and S. Tonetta. Boolean abstraction for temporal logic satisfiability. In *CAV*, volume 4590 of *LNCS*, pages 532–546. Springer, 2007.

[10] A. Cimatti, M. Roveri, and S. Tonetta. PSL Symbolic Compilation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2008. To appear.

[11] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *CAV*, pages 154–169, 2000.

[12] E. Clarke, A. Gupta, J. Kukula, and O. Strichman. SAT Based Abstraction-Refinement Using ILP and Machine Learning Techniques. In *CAV*, pages 265–279, 2002.

[13] E. Clarke, D. Kroening, N. Sharygina, and K. Yorav. Predicate Abstraction of ANSI-C Programs Using SAT. *Formal Methods in System Design*, 25(2-3):105–127, 2004.

[14] S. Demri, R. Lazic, and D. Nowak. On the freeze quantifier in Constraint LTL: Decidability and complexity. *Inf. Comput.*, 205(1):2–24, 2007.

[15] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 995–1072. 1990.

[16] A. Fuxman, L. Liu, J. Mylopoulos, M. Roveri, and P. Traverso. Specifying and analyzing early requirements in Tropos. *Requirements Engineering*, 9(2):132–150, 2004.

[17] S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Combination Methods for Satisfiability and Model-Checking of Infinite-State Systems. In *CADE*, pages 362–378, 2007.

[18] I. M. Hodkinson, F. Wolter, and M. Zakharyaschev. Decidable fragment of first-order temporal logics. *Ann. Pure Appl. Logic*, 106(1-3):85–134, 2000.

[19] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, 2002.

[20] S. K. Lahiri, R. Nieuwenhuis, and A. Oliveras. SMT techniques for fast predicate abstraction. In *CAV*, LNCS, pages 424–437. Springer, 2006.

[21] M. Lange. Linear Time Logics Around PSL: Complexity, Expressiveness, and a Little Bit of Succinctness. In *CONCUR*, pages 90–104, 2007.

[22] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Specification*. Springer, 1992.

[23] Z. Manna and A. Pnueli. *Temporal verification of reactive systems: safety*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[24] G. Nelson and D. C. Oppen. Fast Decision Procedures Based on Congruence Closure. *J. ACM*, 27(2):356–364, 1980.

[25] OMG. *Object Constraint Language: OMG available specification Version 2.0*, 2006.

[26] A. Pnueli. The temporal logic of programs. In *Proceedings of 18th IEEE Symp. on Foundation of Computer Science*, pages 46–57, 1977.

[27] A. Pnueli, Y. Rodeh, O. Strichman, and M. Siegel. The small model property: How small can it be? *Inf. Comput.*, 178(1):279–293, 2002.

[28] R. Sebastiani, S. Tonetta, and M. Vardi. Property-Driven Partitioning for Abstraction Refinement. In *TACAS*, pages 389–404, 2007.

[29] J. M. Spivey. *The Z Notation: a reference manual, 2nd edition*. Prentice Hall, 1992.

[30] A. Susi, A. Perini, P. Giorgini, and J. Mylopoulos. The Tropos Metamodel and its Use. *Informatica*, 29(4):401–408, 2005.