Monitoring Modulo Theories

Normann Decker, Martin Leucker, and Daniel Thoma

Institute for Software Engineering and Programming Languages
Universität zu Lübeck, Germany

{decker, leucker, thoma}@isp.uni-luebeck.de

Abstract. This paper considers a generic approach to enhance traditional runtime verification techniques towards first-order theories in order to reason about data. This allows especially for the verification of multi-threaded, object-oriented systems. It presents a general framework lifting the monitor synthesis for propositional temporal logics to a temporal logic over structures within some first-order theory. To evaluate such temporal properties, SMT solving and classical monitoring of propositional temporal properties is combined. The monitoring procedure was implemented for linear-time temporal logic (LTL) based on the Z3 SMT solver and evaluated regarding runtime performance.

1 Introduction

In this paper we consider runtime verification of multi-threaded, object-oriented systems, representing a major class of today's practical software. As opposed to other verification techniques such as model checking or theorem proving, runtime verification (RV) does not aim at the analysis of the whole system but on evaluating a correctness property on a particular run, based on log-files or on-the-fly. To this end, typically a monitor is synthesized from some high-level specification that is monitoring the run at hand.

In recent years, a variety of synthesis algorithms has been developed, differing in the underlying expressiveness of the specification formalism and the resulting monitoring approach. Typically, a variant of linear-time temporal logic (LTL) is employed as specification language and monitoring is automata-based or rewriting-based.

Within the setting of multiple, in general arbitrarily many instances of program parts, for example in terms of threads or objects, a software engineer is naturally interested in verifying that the interaction of individual instances follows general rules. The ability of taking the dynamics of data structures and values into account is a desirable feature for specification and verification approaches. As such, the expressiveness of plain propositional temporal logics such as LTL does not suffice, as they do not allow for specifying complex properties on data.

In this paper, we enhance traditional runtime verification techniques for propositional temporal logics by first-order theories for reasoning about data, based on SMT solvers. In result, we obtain a powerful tool for verifying complex properties at runtime which exceeds the expressiveness of previous approaches. The implementation in our tool jUnit$^{\mathrm{RV}}$ [1] also shows that the framework is suitable for practical applications.

Today's SMT solvers are highly optimized tools that can check the satisfiability of formulae over a variety of first-order theories such as arithmetics, arrays, lists and uninterpreted functions. They allow for reasoning on a large class of data structures used in modern software systems. We hence aim at integrating their capabilities with the efficient monitoring approaches for temporal properties. We formulate example properties showing the specific strength of our framework in terms of expressiveness. Our benchmarks for monitoring Java programs show that such specifications can be monitored efficiently.

Combining monitoring and SMT. In the following we outline the idea of our approach my means of a running example. Consider a mutual exclusion property where a resource must not be accessed while it is locked, stated in LTL as $\mathrm{G}(\mathsf{lock} \rightarrow \neg\mathsf{access}\,\mathrm{U}\,\mathsf{unlock})$. If there are several resource objects available at runtime, this is too restrictive and one might specifically limit foreign access to locked resources. Using variables r and p, p', intended to represent resources and processes, respectively, and suitable predicates, the property can be stated as

$$\mathrm{G}(\mathsf{lock}(p, r) \rightarrow (\forall p' \neq p : \neg\mathsf{access}(p', r))\,\mathrm{U}\,\mathsf{unlock}(p, r)). \tag{1}$$

The free variables r and p are then implicitly universally quantified. Formally, all variables range over some universe that is fixed by the application. In our example, this could be the set of object identifiers in a certain Java program.

Data theories. To define a formal semantics for expressions as those above we note that we essentially use an LTL formula and exchanged propositions by first-order formulae. In LTL, propositions are evaluated at a position in some word, i.e. a letter. To evaluate first-order formulae, such a letter must now be a first-order structure describing a system state.

In the example we fixed the universe to object IDs and used a binary predicate "=" indicating equality. This is covered by the first-order theory of natural numbers with equality and can be handled by essentially all SMT solvers. It is possible to use more powerful theories, e.g., with linear order or arithmetics. Section 3.4 provides more examples.

What remains are the predicates that are not part of the theory. These specifically characterize the current system state or, more accurately, are interpreted by the current system state in terms of the current observation. We call them *observation predicates*, in the example we use binary predicates lock, unlock and access. Inspecting the program states, we obtain, at any time, an observation g that interprets all observation predicates in terms of relations on the universe.

The first-order formulae reason about the data structures under a specific observation. We therefore refer to this logic as *data logic.* Data logic formulae may have free variables, such as p and r in the example. Summing up, we can define the semantics of a data logic formula in terms of (1) an observation interpreting the observation predicates (and possibly observation functions), (2) a theory that fixes the interpretation of all other predicates and functions and (3) a valuation that assigns a value from the universe to each free variable.

Temporal data logic. In the example, the data logic replaces the propositional part that LTL is based on. To the logic expressing the temporal aspect, we generically refer to as the *temporal logic.* Our assumptions on the temporal logic must be that it is linear (defined on words) and that it only uses atomic propositions to "access" the word. For example, the semantics of some temporal operator must not depend on the current letter directly but only on the semantics of some proposition. We formally define that requirement in Section 3 but for now only remark that typical temporal logics like LTL, the linear μ-calculus or the temporal logic of calls and returns (CaRet) [2] fit into that schema.

Given a suitable temporal logic and data logic we can define the formalism we aim at. Taking the temporal logic and replacing the atomic propositions by data formulae, we obtain what we call a *temporal data logic*. The theory and universe is fixed by the data logic and the semantics of temporal data logic formulae can thus be defined over a *sequence of observations*. The free variables are bound universally so the formula is evaluated over the observation sequence for all possible valuations. The semantics of the formula is the conjunction (more generally the infimum) of these results.

Monitor construction. Assuming a monitor construction for the (propositional) temporal logic, we can evaluate a sequence of observations on-the-fly. The idea is to construct a *symbolic* monitor that deals atomically with data formulae. In the example formula (Equation 1) we treat $\mathsf{lock}(p, r)$, $\forall p' \neq p : \neg\mathsf{access}(p', r)$ and $\mathsf{unlock}(p, r)$ as three atomic propositions, say χ_1,χ_2 and χ_3. We obtain a temporal logic formula $\mathrm{G}(\chi_1 \rightarrow \chi_2 \,\mathrm{U}\, \chi_3)$ over a set of atomic propositions $AP = \{\chi_1, \chi_2, \chi_3\}$. A monitor can then be constructed that reads words over the finite symbolic alphabet $\Sigma := 2^{AP}$.

The free variables in the formula are p and r and range over the universe of natural numbers $\mathbb{N}$. Given a valuation $\theta : \{p, r\} \rightarrow \mathbb{N}$ for those, mapping, e.g., p to $\theta(p) = 1$ and r to $\theta(r) = 2$, we can map an observation g to the letter $a \in \Sigma$ that contains all formulae that are satisfied by g. For example, say g interprets the observation predicates as $\mathsf{lock}_g = \{(1, 2), (10, 7)\}$ and $\mathsf{access}_g = \mathsf{unlock}_g = \emptyset$ (because objects 1 and 10 happen to lock the resources 2 and 7, respectively, and otherwise nothing happened in the current execution step of the program). Then, under θ, g is mapped to $a = \{\chi_1, \chi_2\} \in \Sigma$ since χ_1 and χ_2 hold but χ_3 does not. The observation g might be mapped to some other symbolic letter for another valuation θ'. If, for example, $\theta'(p) = 2$ then χ_1 does not hold and g is projected to $a' = \{\chi_2\} \in \Sigma$.

In Section 4 we present a monitoring algorithm that maintains a copy of the symbolic monitor for each valuation. For a new observation, the algorithm simulates the individual transition for each copy by projecting the observation under the specific valuation. As the universe is in general infinite, the number of monitor instances is infinite as well but the algorithm uses a data structure to finitely represent the state of all monitor instances.

Related work. In runtime verification, handling data values to reason about the computation of a system more precisely has always been a concern. One of the first works extending LTL by parameters is by Stolz and Bodden [3]. Binding of parameters of propositions takes place in a Prolog-style fashion and the resulting approach is reasonable for the intended applications. However, no precise denotational semantics is given.

The works on Eagle and RuleR [4, 5] allow the formulation of first-order safety properties. The corresponding systems come with a rewriting-based semantics and are well-suited for specifying safety properties of especially finite, yet perhaps expanding traces. In [6] a runtime verification approach for the temporal evaluation of integer-modulo-constraints was presented. The underlying logic has a decidable satisfiability problem and the overall approach is anticipatory. However, only limited computations can be followed. To reason about the temporal evolution of data values along some computation, some form of bounded unrolling like in bounded model checking [7] can be used. For runtime verification, however, such an approach is not suitable, as the observed trace cannot be bounded.

Closely related to our work is that of Chen and Rosu [8]. It considers the setting of sequences of actions which are parameterized by identifiers (ID). The main idea is to divide the sequence of a program into sub-sequences, called slices, containing only a single ID, and monitor each slice independently. Hence, in contrast to our approach, no interdependencies between the different slices can be checked. Moreover, our monitoring approach is not limited to plain IDs but allows the user to reason more generally over data in terms of arbitrary (decidable) first-order theories. The work considers a dedicated temporal logic (LTL) together with the dedicated notion of parameters, whereas in our framework an arbitrary linear temporal logic is extended by a first-order theory.

Recently, Bauer et al. presented an approach combining LTL with a variant of first-order logic for runtime verification [9]. However, their approach restricts quantification to finite sets always determined in advance by the system observation. This allows for finitely instantiating quantifiers during monitor execution, but also profoundly limits the expressiveness of first-order logic. Basically, it is only possible to evaluate first-order formulae over finite system observations, and not to express properties in a declarative manner.

2 Preliminaries

First-order logic. A *signature* $S = (P, F, ar)$ consists of finite sets P, F of predicate and function symbols, respectively, each of some arity defined by $ar : P \cup F \to \mathbb{N}$. An *extension* of S is a signature $T = (P', F', ar')$ such that $P \subseteq P'$, $F \subseteq F'$ and $ar' \subseteq ar$.

The *syntax* of first-order formulae over the signature S is defined in the usual way using operators $\vee$ (or), $\wedge$ (and), $\neg$ (negation), variables $x_0, x_1, \ldots$, predicate and function symbols $p \in P$, $f \in F$, quantifiers $\forall$ (universal), $\exists$ (existential). *Free* variables are not in the scope of some quantifier and are assumed to come from some set $\mathcal{V}$. The set of all first-order formulae over a signature S is denoted $\mathrm{FO}[S]$. We consider constants as function symbols f with $ar(f) = 0$. A *sentence* is a formula without free variables.

An *S-structure* is a tuple $s = (\mathcal{U}, \mathfrak{s})$ comprising a non-empty *universe* $\mathcal{U}$ and a function $\mathfrak{s}$ mapping each predicate symbol $p \in P$ to a relation $p_s \subseteq \mathcal{U}^n$ of arity $n = ar(p)$ and each function symbol $f \in F$ to a function $f_s : \mathcal{U}^m \to \mathcal{U}$ of arity $m = ar(f)$. A T-structure $t = (\mathcal{T}, \mathfrak{t})$ is an *extension* of s if T is an extension of S, $\mathcal{T} = \mathcal{U}$ and $\mathfrak{s}(r) = \mathfrak{t}(r)$ for all symbols $r \in P \cup F$.

A *valuation* is a mapping $\theta : \mathcal{V} \to \mathcal{U}$ of free variables to values. The set of all such mappings may be denoted $\mathcal{U}^{\mathcal{V}}$. The semantics of first-order formulae is defined as usual. We write $(s, \theta) \models \chi$ if a formula χ is satisfied for some structure s and valuation θ. For sentences, we refer to a sole satisfying structure as a *model*, omitting a valuation. The *theory* $\mathfrak{T}$ of an S-structure s is the set of all sentences χ such that s is a model for χ.

Temporal specifications. We use AP to denote a finite set of *atomic propositions* and $\Sigma := 2^{AP}$ for the finite alphabet over AP. For arbitrary, possibly infinite alphabets we mostly use Γ. A *word* over some alphabet Γ is a sequence of letters from Γ and Γ^*, Γ^ω denote the sets of finite and infinite words over

Γ, respectively. The syntax of *linear-time temporal logic (LTL)* is defined in the usual way over atomic propositions AP using negation, boolean connectives and temporal operators X (next), U (until), G (globally) and F (eventually). We refer to the standard LTL semantics over infinite words $w \in \Sigma^\omega$ as LTL$_\omega$ given for an LTL formula φ by a mapping $[\![\varphi]\!]_\omega : \Sigma^\omega \to \mathbb{B}$ where $\mathbb{B} = \{\top, \bot\}$ denotes the two-valued boolean lattice. The finitary three-valued LTL semantics LTL$_3$ [10], is given for φ by a mapping $[\![\varphi]\!]_3 : \Sigma^* \to \mathbb{B}_3$ where $\mathbb{B}_3 = \{\top, ?, \bot\}$ denotes the three-valued boolean lattice ordered $\top > ? > \bot$. It is defined for $w \in \Sigma^*$ as $[\![\varphi]\!]_3(w) := \top$ if $\forall u \in \Sigma^\omega : [\![\varphi]\!]_\omega(wu) = \top$, $[\![\varphi]\!]_3(w) := \bot$ if $\forall u \in \Sigma^\omega : [\![\varphi]\!]_\omega(wu) = \bot$ and $[\![\varphi]\!]_3(w) := ?$ otherwise.

Monitor. A *monitor* $\mathcal{M} = (Q, \Gamma, \delta, q_0, \lambda, \Lambda)$ for a temporal property is a Moore machine where Q is a possibly infinite set of states, Γ is a possibly infinite input alphabet, $\delta : Q \times \Gamma \to Q$ is a deterministic transition function and $\lambda : Q \to \Lambda$ is a labeling function mapping states to labels from the set Λ.

3 Temporal Data Logic

The aim of the framework is to enable the user to specify and check complex properties of execution traces. As described above we consider two aspects, time and data. Note that we refer to *discrete* time, as opposed to continuous notions like in timed automata. In this section we therefore formalize how and under which assumptions two logics considering time (temporal logic) and data (data logic) can be combined to a specification formalism (temporal data logic) that can express the timely behaviour of a system with respect to the data it processes. The clear separation of the aspects will give rise to a monitoring procedure.

3.1 Temporal Logic

The notion of a temporal logic (*TL*) that we consider for our monitoring framework is inspired by the intuition for LTL which is widely used for behavioural specifications, in particular in runtime verification. However, our monitoring approach does only rely on some specific properties that also come with other, also more expressive logics. In the following we identify the required features of a suitable temporal logic for our framework.

We require the desired temporal behaviour to be specified in a finitary, linear logic, that is, the semantics is defined on *finite words* over some alphabet Γ. The truth values of the semantics need to come from a complete semi-lattice $(\mathbb{S}, \sqcap)$ since we will handle multiple monitor instances and combine individual verdicts.

Second, there must be a *monitor construction* for the logic in question since our framework is intended to generically lift such a construction for handling data. We assume that such a construction turns a *TL* formula φ into a Moore machine $\mathcal{M}_\varphi$ with output $\mathcal{M}_\varphi(w) = [\![\varphi]\!](w)$ for $w \in \Gamma^*$. The restriction to Moore machines is not essential, our constructions are applicable to similar models, including Mealy machines and we do not rely on a finite state space.

As we aim at replacing atomic propositions, we require that the semantics of the temporal logic can only distinguish letters by means of the semantics of such propositions. This allows for lifting the semantics from a propositional to a complex alphabet where letters have more internal structure.

Proposition semantics. We formalize the distinction of positional and temporal aspects of a temporal logic formula using a *proposition semantics* $ps : AP \to 2^\Gamma$

mapping propositions $p \in AP$ to the set of letters $ps(p) \subseteq \Gamma$ that satisfy the proposition. Given, that the semantics of some propositional temporal logic can be defined by only referring to letters using a proposition semantics, it can be substituted without influencing the temporal aspect.

We refer to the canonical semantics for $\Gamma = \Sigma = 2^{AP}$ as $\mathsf{ps}_{AP} : AP \to 2^{\Sigma}$, with $\mathsf{ps}_{AP}(p) := \{a \subseteq AP \mid p \in a\}$. It is the "sharpest" in the sense that it distinguishes maximally many letters by means of combinations of propositions.

Symbolic abstraction. For an alphabet Γ, atomic propositions AP and a proposition semantics $ps : AP \to 2^{\Gamma}$, let $\pi_{ps} : \Gamma \to \Sigma$ be a projection with $\pi_{ps}(g) := \{p \in AP \mid g \in ps(p)\}$, mapping a letter $g \in \Gamma$ to the set of propositions that hold for it. For convenience, we lift the projection to words $g_1 \ldots g_n$ $(g_i \in \Gamma)$ by $\pi_{ps}(g_1 \ldots g_n) := \pi_{ps}(g_1) \ldots \pi_{ps}(g_n)$. Using π_{ps}, we consider the letters form Σ as symbolic abstractions of Γ wrt. AP and ps in the sense that π_{ps} maintains all the structure of Γ that is relevant for evaluating (boolean combinations of) propositions form AP.

As argued above, for the purpose of lifting a temporal logic over atomic propositions to propositions carrying data, i.e., structure, it is essential that the semantics of propositions can be encapsulated and exchanged without influencing the temporal aspect. We can formalize this requirement on a temporal logic TL using the symbolic abstraction. We assume the semantics of a TL formula φ to be a mapping that takes linear sequences from Γ^* and assigns a truth value from the complete semi-lattice $\mathbb{S}$. If the semantics satisfies our criterion we can make the proposition semantics $ps : AP \to 2^{\Gamma}$ an explicit parameter and assume the semantics of a formula φ is given by a mapping $[\![\varphi]\!](ps) : \Gamma^* \to \mathbb{S}$, or, generally, $[\![\varphi]\!] : (AP \to 2^{\Gamma}) \to (\Gamma^* \to \mathbb{S})$. Moreover, projecting the input word to a symbolic word and evaluating $[\![\varphi]\!](\mathsf{ps}_{AP})$ on it must not change the result.

Definition 1 (Propositional semantics). *Let AP be a set of atomic propositions and Γ an alphabet. A semantics $[\![\varphi]\!] : (AP \to 2^{\Gamma}) \to (\Gamma^* \to \mathbb{S})$ is* propositional *iff for all proposition semantics $ps : AP \to 2^{\Gamma}$ and all words $\gamma \in \Gamma^*$*

$$[\![\varphi]\!](ps)(\gamma) = [\![\varphi]\!](\mathsf{ps}_{AP})(\pi_{ps}(\gamma)).$$

Based on that notion of propositional semantics we can summarize the formal criteria for a temporal logic to be suitable for our monitoring framework.

Definition 2 (Temporal logic). *A* temporal logic *is a specification formalism TL over a set of atomic propositions AP that enjoys the following properties.*

1. *The semantics of formulae φ is given for finite words over an input alphabet Γ by a mapping $[\![\varphi]\!]_{TL} : (AP \to 2^{\Gamma}) \to (\Gamma^* \to \mathbb{S})$ where $(\mathbb{S}, \sqcap)$ is a complete semi-lattice.*
2. *The semantics is propositional.*
3. *A monitor construction is available that turns a formula φ into a Moore machine $\mathcal{M}_{\varphi}$ with output $\mathcal{M}_{\varphi}(w) = [\![\varphi]\!]_{TL}(\mathsf{ps}_{AP})(w)$ for $w \in \Sigma^*$.*

3.2 Data logic

To reason about data values our framework can use a so called *data logic DL* based on any first-order theory for which satisfiability is decidable. We assume the theory is represented by some structure which can be extended by additional predicate and function symbols that will represent observations from the system that shall be monitored.

Definition 3 (Data logic). *Let $T = (P, F, \mathsf{ar})$ be a signature, $t = (\mathcal{D}, \mathfrak{a})$ some T-structure and P', F' be additional predicate and function symbols with arity defined by $\mathsf{ar}' : P' \cup F' \to \mathbb{N}$, called* observation symbols*.*

A data logic *DL is a tuple $(t, G, \mathcal{V}, \mathcal{D})$ such that $G = (P \cup P', F \cup F', \mathsf{ar} \cup \mathsf{ar}')$ is an extension of T and $\mathcal{V}$ is a finite set of first-order variables.*

A DL formula *is a first-order formula over the signature G and possibly free variables from $\mathcal{V}$. A DL formula is called* observation-independent*, if it does not contain observation symbols. An* observation *is a G-structure $g = (\mathcal{D}, \mathfrak{g})$ that is an extension of t. The set of all observations is denoted Γ.*

The semantics *of a DL formula is defined over tuples $(g, \theta) \in \Gamma \times \mathcal{D}^{\mathcal{V}}$ consisting of an observation and a valuation $\theta : \mathcal{V} \to \mathcal{D}$ of free variables in the usual way.*

For an instance of the monitoring framework the structure t representing the theory is fixed. An observation-independent DL formula φ with free variables $x_1, \ldots, x_n \in \mathcal{V}$ can be evaluated just wrt. t, without considering an observation. A decision procedure for the theory of t can thus be applied directly. Further, φ can be interpreted as a constraint on the domain of variable valuations $\mathcal{D}^{\mathcal{V}}$ by considering the set $\Theta_\varphi := \{\theta \in \mathcal{D}^{\mathcal{V}} \mid (t, \theta) \models \varphi\}$.

3.3 Temporal data logic

Given a temporal and a data logic as described above, we can now define their combination, the temporal data logic TDL.

In TDL formulae we use brackets $\langle$ and $\rangle$ to clarify which parts come from the data logic.

Definition 4 (Temporal data logic). *Let TL be a temporal logic and $DL = (t, G, \mathcal{V}, \mathcal{D})$ a data logic. Let AP be a finite set $\{\langle\chi_1\rangle, \ldots, \langle\chi_n\rangle\}$ where $\chi_1, \ldots, \chi_n$ are DL formulae with free variables from $\mathcal{V}$.*

A TDL formula *is a TL formula over AP. A* structured word *is a finite sequence $\gamma \in \Gamma^*$ of DL observations. For a valuation $\theta \in \mathcal{D}^{\mathcal{V}}$, let the proposition semantics $\mathsf{ps}_\theta : AP \to 2^\Gamma$ be defined by $\mathsf{ps}_\theta(\langle\chi\rangle) := \{g \in \Gamma \mid (g, \theta) \models \chi\}$ for $\langle\chi\rangle \in AP$. The* semantics *of a TDL formula φ is a mapping $[\![\varphi]\!]_{TDL} : \Gamma^* \to \mathbb{S}$ defined for $\gamma \in \Gamma^*$ by*

$$[\![\varphi]\!]_{TDL}(\gamma) := \bigsqcap_{\theta \in \mathcal{D}^{\mathcal{V}}} [\![\varphi]\!]_{TL}(\mathsf{ps}_\theta)(\gamma).$$

Recall, for ps_θ we obtain a projection $\pi_{\mathsf{ps}_\theta} : \Gamma^* \to \Sigma^*$ from structured to symbolic words. In the following we abbreviate π_{ps_θ} by π_θ. From Definition 2 of the temporal logic it follows that we can evaluate the semantics of some TDL formula *symbolically*, which is an essential step in lifting a monitor construction for TL to the data setting.

Proposition 1. *Let φ be a TDL formula, $\mathcal{D}^{\mathcal{V}}$ the valuation space for free variables in φ, $\chi_1, \ldots, \chi_n$ the data logic formulae used in φ and $AP = \{\langle\chi_1\rangle, \ldots, \langle\chi_n\rangle\}$. For $\gamma \in \Gamma^*$ we have $[\![\varphi]\!]_{TDL}(\gamma) = \bigsqcap_{\theta \in \mathcal{D}^{\mathcal{V}}} [\![\varphi]\!]_{TL}(\mathsf{ps}_{AP})(\pi_\theta(\gamma))$.*

3.4 LTL and CaRet with Data

We now exemplify the instantiation of our framework by means of LTL. More precisely, we show that the the finitary, three-valued LTL$_3$ semantics $[\![\varphi]\!]_3 : \Sigma^* \to \mathbb{B}_3$ can be formulated to comply Definition 2. It is defined over $\Sigma = 2^{AP}$ based on the infinitary LTL$_\omega$ semantics. The inductive definition of

mutex	$\mathrm{G}(\langle \mathsf{lock}(f,t)\rangle \rightarrow \langle \forall t' \neq t : \neg\mathsf{access}(f,t')\rangle\, \mathrm{U} \langle \mathsf{unlock}(f,t)\rangle)$
access	$(\langle \mathsf{open}(x)\rangle\, \mathrm{R}\, \neg\langle \mathsf{access}(x)\rangle) \wedge \mathrm{G}(\langle \mathsf{close}(x)\rangle \rightarrow \mathrm{G}\,\neg\langle \mathsf{access}(x)\rangle)$
iterator	$\mathrm{G}((\langle \mathsf{iterator}(i)\rangle \vee \langle \mathsf{next}(i)\rangle) \rightarrow \mathrm{X}(\langle \mathsf{hasNext}(i, \mathrm{true})\rangle\, \mathrm{R}\, \neg\langle \mathsf{next}(i)\rangle))$
modified	$\mathrm{G}(\langle \mathsf{iterator}(c,i)\rangle \rightarrow \mathrm{G}(\langle \mathsf{add}(c)\rangle \rightarrow (\neg\langle \mathsf{next}(i)\rangle\, \mathrm{U} \langle \mathsf{finalize}(i)\rangle)))$
server	$\mathrm{G}(\langle \mathsf{request}(t,x)\rangle \rightarrow \mathrm{F}\langle \exists t' : \mathsf{response}(t',x,t)\rangle)$
response	$\mathrm{G}(\langle \mathsf{request}(t)\rangle \wedge \langle x = \mathsf{time}\rangle \rightarrow (\langle \mathsf{time} < x + 100\rangle\, \mathrm{U} \langle \mathsf{response}(t)\rangle))$
counter	$\mathrm{G}(\langle \mathsf{p}(x)\rangle \rightarrow \mathrm{X}\langle \mathsf{p}(x+1)\rangle)$
velocity	$\mathrm{G}(\langle \mathsf{s} = x \wedge \mathsf{t} = y\rangle \rightarrow \mathrm{X}\langle \mathsf{s} - x < \mathsf{vmax}\cdot(\mathsf{t} - y)\rangle)$
matching	$\mathrm{G}((\langle \mathsf{call}\rangle \wedge \langle \mathsf{printOpen}(x)\rangle) \rightarrow \mathrm{X}^{\mathrm{a}}\langle \mathsf{printClose}(x)\rangle)$
bound	$\mathrm{G}(\langle \mathsf{open}(x)\rangle \rightarrow \mathrm{X}(\neg\langle \mathsf{ret}\rangle \rightarrow \mathrm{G}^{\mathrm{a}}(\langle \mathsf{open}(y)\rangle \rightarrow \langle x > y\rangle)))$
depth	$\mathrm{G}(\langle \mathsf{open}(x)\rangle \rightarrow \mathrm{X}((\neg\langle \mathsf{ret}\rangle \wedge \mathrm{F}^{\mathrm{a}}\langle \mathsf{open}(\mathsf{x}-1)\rangle) \vee (\langle \mathsf{ret}\rangle \wedge \langle x = 0\rangle)))$

Table 1. Example properties using LTL and CaRet with data.

LTL_ω only refers to letters for atomic propositions. This can be easily reformulated in terms of an arbitrary proposition semantics $ps : AP \rightarrow 2^{\Gamma}$ over an arbitrary alphabet Γ. Instead of defining $[\![p]\!]_\omega(w) = \top$ iff $p \in w_0$, we let $[\![p]\!]_\omega(ps)(\gamma) := \top$ if $\gamma_0 \in ps(p)$ and $[\![p]\!]_\omega(ps)(\gamma) := \bot$ otherwise, for $\gamma \in \Gamma^\omega$. The rest of the definition remains untouched. The definition of the three-valued semantics $[\![\varphi]\!]_3$ does not at all refer to letters directly but only to LTL_ω. With these simple modifications LTL_3 fits to the notion of temporal logic in the sense of Definition 2. The corresponding monitor construction proposed in [10, 11] can be applied.

Proposition 2. *The MMT framework can be instantiated for LTL_3.*

The mutual exclusion property presented earlier is one example for a specification based on LTL and the theory of IDs. Other common examples of temporal properties are the correct use of iterators or global request/response properties. In the propositional versions of such properties the objects in question, iterators, resources or requests, are assumed to be unique. Adding data in terms of IDs, for example, allows for a much more realistic formulation. Table 1 lists formulations of these properties and also others that cannot be expressed without distinguishing at least identities. The property *modified* requires that an iterator must not be used after the collection it corresponds to has been changed.

Further, counting (*counter*) or arithmetic constraints (*response*, *velocity*), also on real numbers, are valuable features for a realistic specification.

RLTL and CaRet: Regular and nesting properties. Regular LTL [12] is an extension of LTL based on regular expressions. CaRet [2] is a temporal logic with calls and returns expressing non-regular properties. In addition to the LTL operators, CaRet allows for abstract temporal operators such as X^{a} and G^{a}, moving forward by jumping on a word from a calling position to matching return position, reflecting the intuition of procedure calls. For RLTL and CaRet monitor constructions have been proposed [6, 13]. Despite both are more complex the same arguments as for LTL apply. Example properties are listed in Table 1 and express matching call- and return values and nesting-depth bounds.

4 Monitoring

In this section we present our monitoring procedure for *TDL* formulae. It relies on the observation made in Proposition 1, namely that the *TDL* semantics for an input word $\gamma \in \Gamma^*$ is characterized by the *TL* semantics for projections of γ.

Any *TDL* formula can be interpreted as *TL* formula when considering all occurring data logic formulae as individual symbols. With this interpretation we can employ the monitor construction for *TL* to obtain a monitor over a finite alphabet constructed from these symbols.

Definition 5 (Symbolic monitor). *Let φ be a TDL formula and $\chi_1, \ldots, \chi_n$ the data logic formulae used in φ and $AP = \{\langle\chi_1\rangle, \ldots, \langle\chi_n\rangle\}$. The* symbolic alphabet *for φ is the finite set $\Sigma := 2^{AP}$. The* symbolic monitor *for φ is the monitor $\mathcal{M}_\Sigma$ constructed for φ interpreted as TL formula over AP.*

The symbolic monitor $\mathcal{M}_\Sigma$ for a *TDL* formula φ computes the semantics $[\![\varphi]\!](\mathsf{ps}_{AP}) : \Sigma^* \to \mathbb{S}$. Following Proposition 1, what remains is to maintain a monitor for each valuation $\theta \in \mathcal{D}^{\mathcal{V}}$ and to individually compute the corresponding projection π_θ on the input.

Within this section we present an algorithm for efficiently maintaining these, in general infinitely many, monitor instances. It uses a data structure, called constraint tree, that represents finitely many equivalence classes of symbolic monitors. The constraint tree also allows for easy computation of the infimum of the outputs of all monitor instances, which is the semantics of the property on the input trace read so far.

4.1 Representing and Evaluating Observations

While observations are formally defined as first-order structures, we want to use them algorithmically and must therefore choose a representation. An actual implementation of an SMT solver already fixes how to represent all objects essential for handling a certain theory, such as first-order formulae, predicates and function symbols. We have defined observations to be extensions of a structure representing the theory and want to handle them practically using an SMT solver. Consequently, we assume them to be extensions of the structure that the tool uses to represent and handle a theory. For the purpose of the implementation, it is a reasonable assumption that the semantics of observation symbols be expressible or, more precisely, expressed within the considered data theory.

Formally, for $DL = (t, G, \mathcal{V}, \mathcal{D})$ where t is a T-structure, we assume that any observation $g \in \Gamma$ induces a mapping $\hat{g} : FO[G] \to FO[T]$ s.t. for all *DL* formulae χ and all valuations $\theta \in \mathcal{D}^{\mathcal{V}}$ we have $(g, \theta) \models \chi$ iff $(t, \theta) \models \hat{g}(\chi)$. Note that this can be realized by substituting observation predicates by some observation-independent formula that characterizes its semantics wrt. g. Function symbols f can be replaced using existential substitution replacing expressions of the form $e(f(e'))$ by $\exists z : e(z) \wedge \xi_f(e', z)$ where an observation-free *DL* formula ξ_f characterizes the semantics of f wrt. g.

As noted earlier, we can also employ observation-free formulae ρ to describe sets of valuations $\Theta_\rho \subseteq \mathcal{D}^{\mathcal{V}}$. While this does not allow for representing any arbitrary set of valuations, the expressiveness of the data theory suffices to express any relevant set. If ρ represents an equivalence class wrt. some formula $\hat{g}(\chi)$, meaning $(t, \theta) \models \hat{g}(\chi)$ holds for all $\theta \in \Theta_\rho$ or none, we have that $(t, \theta) \models \hat{g}(\chi)$ iff there is any $\theta' \in \mathcal{D}^{\mathcal{V}}$ such that $(t, \theta') \models \hat{g}(\chi) \wedge \rho$.

Proposition 3. *Let χ be a DL formula and $g \in \Gamma$ an observation. Let ρ be an observation-free DL formula such that for all $\theta_1, \theta_2 \in \Theta_\rho$ we have $(t, \theta_1) \models \hat{g}(\chi)$ iff $(t, \theta_2) \models \hat{g}(\chi)$. Then, for all $\theta \in \Theta_\rho$, $(g, \theta) \models \chi$ iff $\hat{g}(\chi) \wedge \rho$ is satisfiable.*

Note that $\hat{g}(\chi) \wedge \rho$ is an observation-free *DL* formula and that checking it for satisfiability is exactly what we assume an SMT solver be able to do.

4.2 Constraint Trees

We next introduce constraint trees, a data structure storing the configurations of a set of instances of some symbolic monitor. It maintains sets of valuations $\Theta \subseteq \mathcal{D}^{\mathcal{V}}$ represented by constraints and stores for each such set a monitor state. The desired property regarding the use in our monitoring algorithm is that the sets of constraints induce a partition of the valuation space.

Definition 6 (Constraint tree). *Let $\mathcal{M}_{\Sigma}$ be a symbolic monitor with states Q and DL a data logic. A* constraint tree *is a tuple $T = (I, L, S_1, S_2, C, \lambda_I, \lambda_L)$ such that $(I \cup L, S_1, S_2)$ is a finite, non-empty binary tree with* internal nodes *I,* leaf nodes *L and* successor relations *$S_1, S_2 \subseteq I \times (I \cup L)$, C is a set of observation-independent DL formulae called* constraints, *$\lambda_I : I \to C$ labels internal nodes with constraints and $\lambda_L : L \to Q$ labels leaf nodes with monitor states.*

Let the DL formula $\rho(v_0 \ldots v_i)$ be the conjunction over all constraints along the path $v_0 \ldots v_{i-1}$, where all S_2-successors are negated and $\rho(v_0) = \text{true}$. A path constraint *in T is a DL formula $\rho(v_0 \ldots v_n)$ such that $v_0 \ldots v_n$ is a maximal path in T. A constraint tree T is* consistent *if the set of all path constraints in T induces a partition of $\mathcal{D}^{\mathcal{V}}$. The set of all constraint trees is denoted $\mathcal{T}$.*

In a constraint tree T, each inner node represents a constraint that is used to separate the valuation space $\mathcal{D}^{\mathcal{V}}$. S_1-branches represent the parts where the particular constraint holds while in the S_2-branches it does not.

Constraint trees $T = (I, L, E, C, \lambda_I, \lambda_L)$ will be used to represent mappings $t : \mathcal{D}^{\mathcal{V}} \to Q$ assigning a monitor state $q \in Q$ to each valuation $\theta \in \mathcal{D}^{\mathcal{V}}$. If T is consistent, every valuation θ satisfies exactly one path constraint ρ in T which in turn corresponds to a unique path ending in some leaf node $v \in L$. The mapping is thereby defined as $t(\theta) = \lambda_L(v)$. Note that t would not be necessarily well-defined for constraint trees that are not consistent. Where convenient, we may identify a path constraint ρ with the set Θ_ρ of valuations satisfying it and write, e.g., $\theta \in \rho$ if some valuation $\theta \in \Theta_\rho$ satisfies ρ.

4.3 Symbolic Monitor Execution

In the following we present an algorithm incrementally processing a sequence of observations in order to compute the semantics of some *TDL* formula φ. It maintainings a consistent constraint tree as a finite representation of a mapping of valuations to states of the symbolic monitor $\mathcal{M}_{\Sigma} = (Q, \Sigma, \delta, q_0, \lambda, \mathbb{S})$ for φ.

The algorithm starts on the trivial constraint tree consisting only of one leaf node labeled by the initial state q_0. This means that the monitor instances for all valuations are in state q_0. Intuitively, for an input word $\gamma \in \Gamma$ the algorithm executes one monitor instance for each valuation $\theta \in \mathcal{D}^{\mathcal{V}}$ on the respective projection $\pi_\theta(\gamma)$. For the empty word $\gamma = \epsilon$, all projections are equal and all instances are in the same state. When reading a new observation $g \in \Gamma$ which is, for all valuations, projected to the same symbolic letter $a \in \Sigma$, all monitor instances read the same projection and their state changes equally to $\delta(q_0, a)$. Otherwise, if g is mapped to different symbolic letters for different valuations, the so far uniformly handled valuation space is *split*.

Consider two valuations $\theta, \theta' \in \mathcal{D}^{\mathcal{V}}$ and an input symbol $g \in \Gamma$ such that their projections $a = \pi_\theta(g) \neq b = \pi_{\theta'}(g)$ are different. Then there is some proposition

$\langle\chi\rangle \in AP$ that distinguishes a and b, e.g., let $\langle\chi\rangle \in a$ and $\langle\chi\rangle \notin b$. In general, the behaviour of all monitor instances reading a letter including $\langle\chi\rangle$ may diverge from those reading a letter not including $\langle\chi\rangle$. Therefore, the algorithm records this fact by splitting the valuation space in two parts, one for which χ holds under observation g and another for which it does not. A new node is added to the tree, labeled by the constraint $\hat{g}(\chi)$ precisely distinguishing the two parts.

A part may be split up further in the same way in case other propositions again distinguish valuations from it. Additional nodes are created in the constraint tree accordingly and so the path constraint ρ on the path to a leaf node $v \in L$ characterizes exactly the set of valuations Θ_ρ for which the projection of the observation g is equal and thus the state of all corresponding monitor instances. This process is continued when reading further observations. For each part Θ_ρ represented in the constraint tree, a new observation $h \in \Gamma$ is processed by checking for each proposition $\langle\chi\rangle \in AP$ if there are valuations in Θ_ρ that observe a projection including $\langle\chi\rangle$ by checking satisfiability of $\rho \wedge \hat{h}(\chi)$ and if there are others observing a projection not including $\langle\chi\rangle$ by checking the satisfiability of $\rho \wedge \neg\hat{h}(\chi)$. If one of the formulae is empty, meaning that one of the hypothetical new parts $\Theta_{\rho\wedge\hat{h}(\chi)} = \Theta_\rho \cap \Theta_{\hat{h}(\chi)}$ and $\Theta_{\rho\wedge\neg\hat{h}(\chi)} = \Theta_\rho \cap \overline{\Theta_{\hat{h}(\chi)}}$ is empty, the new observation h is projected equally wrt. $\langle\chi\rangle$ for all valuations in the part which is thus not split by $\hat{h}(\chi)$. Only if both new parts are non-empty, the part is split by adding a new node to the constraint tree labeled by $\hat{h}(\chi)$. Once all necessary splits are performed for an observation, all propositions are evaluated yielding the projections for each (possibly new) part. According to those, the leaf nodes are updated using the transition function of the symbolic monitor.

The procedure described above is listed explicitly as Algorithm 1. There, for the set of all constraint trees $\mathcal{T}$, we use constructors $\texttt{InnerCTree} : \mathrm{FO}[S] \times \mathcal{T} \times \mathcal{T} \to \mathcal{T}$ and $\texttt{LeafCTree} : Q \to \mathcal{T}$ for sub-trees and leafs, respectively, where $\mathrm{FO}[S]$ is the set of observation-independent *DL* formulae. For $T = \texttt{LeafCTree}(q)$ we assume that T consists of a single node $v \in L$ that is labeled by $\lambda_L(v) = q$ and for $T = \texttt{InnerCTree}(\varphi, T_1, T_2)$ we assume that T has at least three nodes v, v_1, v_2 such that v is the root of T labeled by $\lambda_I(v) = \varphi$, v_1, v_2 are the roots of T_1 and T_2, respectively, and $(v, v_1) \in S_1$ and $(v, v_2) \in S_2$.

Based on constraint trees as data structure and the algorithm for modifying constraint trees regarding a new observation we can now define the data monitor for a *TDL* formula, where, as before, the data logic *DL* is defined over observations Γ and the temporal logic *TL* uses truth values $\mathbb{S}$.

Definition 7 (Data monitor). *Let φ be a TDL formula, $\Sigma = 2^{AP}$ the symbolic alphabet and $\mathcal{M}_\Sigma = (Q, \Sigma, \delta, q_0, \lambda_Q, \mathbb{S})$ the symbolic monitor for φ.*

The data monitor *for φ is a Moore machine $\mathcal{M}_\Gamma = (\mathcal{T}, \Gamma, \texttt{step}, T_0, \lambda_\mathcal{T}, \mathbb{S})$ using constraint trees $\mathcal{T}$ as states.*

The transition function $\texttt{step} : \mathcal{T} \times \Gamma \to \mathcal{T}$ is given by Algorithm 1 and the initial tree T_0 consists of a single leaf node labeled with the initial state q_0 of $\mathcal{M}_\Sigma$. For a constraint tree $T \in \mathcal{T}$ where the leaf nodes L are labeled by λ_L, the monitor output is defined by $\lambda_\mathcal{T} : \mathcal{T} \to \mathbb{S}$ with $\lambda(T) = \bigsqcap_{v \in L} \lambda_\Sigma(\lambda_L(v))$.

Algorithm 1 Split constraints and simulate monitor steps

```
function split =
  //recursively process subtrees, accumulate constraints
  case ($P$, $\rho$, $a$, InnerCTree($\varphi$, $t_0$, $t_1$), $g$) then
     InnerCTree($\varphi$,split($P$, $\rho \wedge \neg\varphi$, $a$, $t_0$, $g$),split($P$, $\rho \wedge \varphi$, $a$, $t_1$, $g$))

  //evaluate propositions, split partition if necessary
  case ($\{\langle\chi\rangle\} \cup P$, $\rho$, $a$, LeafCTree($s$), $g$) then
     $T_0$ = if SAT($\rho \wedge \neg\hat{g}(\chi)$) then
              split($P$, $\rho \wedge \neg\hat{g}(\chi)$, $a$, LeafCTree($s$), $g$)
           else Empty
     $T_1$ = if SAT($\rho \wedge \hat{g}(\chi)$) then
              split($P$, $\rho \wedge \hat{g}(\chi)$, $a \cup \{\langle\chi\rangle\}$, LeafCTree($s$), $g$)
           else Empty
     if ($t_0$ = Empty) then $t_1$
     else if ($t_1$ = Empty) then $t_0$
     else InnerCTree($\hat{g}(\chi)$, $t_0$, $t_1$)

  //store new state
  case ($\emptyset$, $\rho$, $a$, LeafCTree($s$), $g$) then
     LeafCTree($\delta(s, a)$)

function step($t$: CTree, $g \in \Gamma$): CTree =
  split($AP$, true, $\emptyset$, $t$, $g$)
```

4.4 Correctness

Proposition 4 (Termination). *On a constraint tree T, the function* `step` *in Algorithm 1 terminates and has a running time in $\mathcal{O}(|T| \cdot |\Sigma|)$ where $|T|$ is the number of nodes in T and $|\Sigma| = 2^{|AP|}$ is the number of abstract symbols.*

The monitoring procedure presented above is correct in that the data monitor $\mathcal{M}_\Gamma$ for a *TDL* formula φ computes the correct semantics for all input words.

Theorem 1 (Correctness). *Let φ be a TDL formula and $\mathcal{M}_\Gamma$ the data monitor for φ. Then, for all $\gamma \in \Gamma^*$, $M_\Gamma(\gamma) = [\![\varphi]\!]_{TDL}(\gamma)$.*

In order to prove correctness, we first settle some observations. Recall that the semantics $[\![\varphi]\!]_{TDL}$ can be represented as the conjunction $\bigsqcap_{\theta \in \mathcal{D}^\mathcal{V}} [\![\varphi]\!]_{TL}(\mathsf{ps}_{AP})(\pi_{\mathsf{ps}_\theta}(\gamma))$ over projections $\pi_{\mathsf{ps}_\theta}(\gamma)$ (Proposition 1). We fix the data logic *DL* for this section and write π_θ for π_{ps_θ} in the following.

Despite the conjunction above is infinite, given a finite word $\gamma \in \Gamma^*$, the valuation space $\mathcal{D}^\mathcal{V}$ can be partitioned into finitely many equivalence classes $\Theta_1, \ldots, \Theta_n$ such that the projection of γ is unique for each class Θ_i, i.e., $\forall_{\theta,\theta' \in \Theta_i} : \pi_\theta(\gamma) = \pi_{\theta'}(\gamma)$. It therefore suffices to maintain this set of equivalence classes which can in turn be finitely represented by constraints ρ_i. Let $w_i = \pi_\theta(\gamma) \in \Sigma^*$ for $\theta \in \Theta_i$ be the projection of γ for the class Θ_i ($i \in \{1, \ldots, n\}$). The semantics can then be computed as the finite conjunction $[\![\varphi]\!]_{TDL}(\gamma) = \bigsqcap_{i=1}^{n} [\![\varphi]\!]_{TL}(\mathsf{ps}_{AP})(w_i)$.

It remains to reason that this partition exists which we do by showing that it is in fact computed by the monitoring algorithm. More precisely, we show that the constraint tree T that is the configuration of the monitor $\mathcal{M}_\Gamma$ after reading a word γ is consistent. That is, the path constraints ρ represented by T cover the whole valuation space and are disjoint. Moreover, for all valuations $\theta \in \rho$ of such an equivalence class ρ, the symbolic monitor $\mathcal{M}_\Sigma$ behaves the same on all corresponding projections $\pi_\theta(\gamma)$.

Lemma 1. *Let $\mathcal{M}_\Gamma = (\mathcal{T}, \Gamma, \delta_\Gamma, t_0, \lambda_\Gamma)$ and $\mathcal{M}_\Sigma = (Q, \Sigma, \delta_\Sigma, q_0, \lambda_\Sigma)$ be the data monitor and the symbolic monitor, respectively, for some TDL formula*

φ. Let for $\gamma \in \Gamma^*$ be $T = \delta_\Gamma(T_0, \gamma)$ and R_T the set of path constraints in T. If T is consistent, $T(\rho)$ denote for $\rho \in R_T$ the unique label of the leaf in T corresponding to ρ. Then, (i) $\{\Theta_\rho \mid \rho \in R_T\}$ is a partition of $\mathcal{D}^\mathcal{V}$ and (ii) $\forall_{\rho\in R_T}\forall_{\theta\in\Theta_\rho} : T(\rho) = \delta_\Sigma(q_0, \pi_\theta(\gamma))$.

We can now proof that the data monitor computes the correct semantics.

Proof (Theorem 1).
Let $T = \delta_\Gamma(T_0, \gamma)$. We have, using Lemma 1 (*i*) and (*ii*),

$$\mathcal{M}_\Gamma(\gamma) = \lambda_\Gamma(T) = \bigsqcap_{v\in L} \lambda_\Sigma(\lambda_L(v)) \overset{(i)}{=} \bigsqcap_{\rho\in R_T} \lambda_\Sigma(T(\rho)) \overset{(ii)}{=} \bigsqcap_{\rho\in R_T} \bigsqcap_{\theta\in\rho} \delta_\Sigma(q_0, \pi_\theta(\gamma))$$

$$\overset{(i)}{=} \bigsqcap_{\theta\in\mathcal{D}^\mathcal{V}} \delta_\Sigma(q_0, \pi_\theta(\gamma)) = \bigsqcap_{\theta\in\mathcal{D}^\mathcal{V}} [\![\varphi]\!]_{TL}(\mathsf{ps}_{AP})(\pi_\theta(\gamma)) = [\![\varphi]\!]_{TDL}(\gamma)$$

□

4.5 Remarks and Optimizations

Impartiality and anticipation. An impartial semantics distinguishes between preliminary and final verdicts. A final verdict for some word indicates that it will not change for any continuation. Impartiality is desirable as monitoring can be stopped as a soon as a final verdict is encountered (c.f. [14, 6]). In the context of our framework this gains even more importance. When the underlying monitor is impartial, a branch already yielding a final verdict can be pruned. This immensely improves runtime performance. If the symbolic monitor is impartial, the data monitor (partially) inherits this property in the typical cases. Another desired property is anticipation, i.e., evaluating to a final verdict as early as possible. While in general not transfered from the symbolic to the data monitor, this may still lead to better performance.

Dedicated theories as first-class citizens. The monitoring framework is also flexible in the sense that one can trade efficiency for generality. When the properties intended to monitor are simple enough it is reasonable to extend the algorithm to directly evaluate constraints. As we show in the experiments this works well, in particular for properties concerning only object IDs.

5 Experimental Results

We implemented our framework based on jUnit$^{\text{RV}}$ [1], a tool for monitoring temporal properties for applications running on the Java Virtual Machine. The previous version of jUnit$^{\text{RV}}$ supported classical LTL specifications referring to, e.g., the invocation of a method of some class. With the approach proposed here it is now possible, for example, to specify properties that relate to individual objects and their evolution in time. The implementation is based on a generic interface to an SMT solver. We present benchmarks using the SMT solver Z3 [15]. For comparison, we additionally implemented a dedicated solver for the theory of IDs (i.e., conjunctions of equality constraints on natural numbers). For the benchmarks, we have chosen representative properties from Table 1. The property *mutex* is a typical example for interaction patterns in object-oriented systems. It was evaluated on a program with resource objects and user objects randomly accessing them. The *iterator* example was evaluated on a simple program using

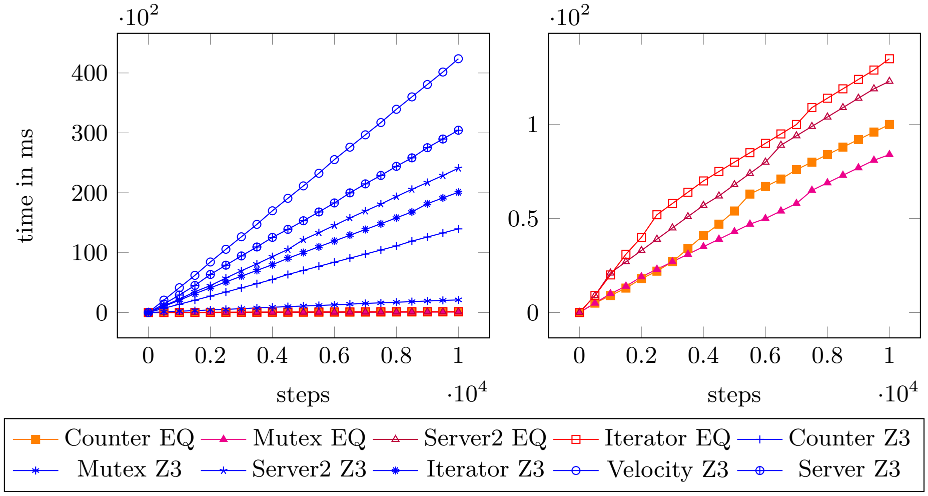

Fig. 1. Experimental results

randomly one of two iterator objects for traversing a list. Third, we evaluated a typical client-server response pattern (*server*) on a program simulating a number of server threads that receive requests and responses. For handling existential quantification, we rely on Z3. For comparison, we also evaluate the property $\mathrm{G}(\langle \mathsf{request}(t, x)\rangle \to F\langle \mathsf{response}(x, t)\rangle)$ (*server2*) as a variation that can be handled by our simple solver. The *counter* property covers the counting of natural numbers which is a very elementary aspect in computer programs and uses an unbounded number of different data values. A property involving a rather complex theory is *velocity*. The free variables refer to real numbers as data values and the constraints that have to be checked are multi-dimensional.

In our experiments we measured the execution time of a program with an integrated monitor over the number of monitoring steps. The measurements were taken up to 10^4 steps. Very simple programs were used, since the measured runtime is thereby essentially the runtime of the monitoring algorithm. The linear graphs obtained for every example show that the execution time for a monitoring step is constant. The most complex properties, *velocity* and *server* induce the most overhead due to a higher computational cost by the SMT solver. However, even the performance for *velocity* of 4.2 ms/step is acceptable for many applications. Thus, employing an SMT solver is viable whenever performance is not a main concern, for instance in case a monitoring step is not expected to happen frequently wrt. to the overall computation steps. Our dedicated implementation is much faster (by factor 100) and hence can only be distinguished in the right-hand diagram. These results demonstrate, that performance can be improved for specific settings and the approach can still be employed when performance is more critical. As mentioned before, the number of calls to the SMT solver is linear in the size of the constraint tree. Hence, the overhead may increase up to linearly in the number of runtime objects that need to be tracked. In our example the maximal size of the constraint tree was six. All experiments were carried out on an Intel i5 (750) CPU.

6 Conclusion

With the combination of propositional temporal logics and first-order theories, the framework we propose in this paper allows for a precise, yet high-level and universal formulation of behavioural properties. This helps the user to avoid modeling errors by formulating specifications describing a system on a higher level of abstraction than required for an actual implementation.

The clear separation of the aspects of time and data allows for efficient runtime verification as the different aspects are handled separately in terms of a symbolic monitor construction and solving satisfiability for first-order theories. The independent application of techniques from monitoring and SMT solving benefits from improvements in both fields.

Our implementation and the experimental evaluation show that the approach is applicable in the setting of object-oriented systems and that the runtime overhead is reasonably small. Note that this is despite the properties expressible in our framework are hard to analyze. The satisfiability problem, for example, is already undecidable for the combination of LTL and the very basic theory of identities.

References

1. Decker, N., Leucker, M., Thoma, D.: jUnit$^{\mathrm{RV}}$-Adding Runtime Verification to jUnit. In: NASA Formal Methods. LNCS, Springer (2013)
2. Alur, R., Etessami, K., Madhusudan, P.: A temporal logic of nested calls and returns. In: TACAS. LNCS, Springer (2004)
3. Stolz, V., Bodden, E.: Temporal assertions using aspectj. Electr. Notes Theor. Comput. Sci. (2006)
4. Goldberg, A., Havelund, K.: Automated runtime verification with eagle. In: MSVVEIS, INSTICC Press (2005)
5. Barringer, H., Rydeheard, D.E., Havelund, K.: Rule systems for run-time monitoring: From eagleto ruler. In: RV. LNCS, Springer (2007)
6. Dong, W., Leucker, M., Schallhart, C.: Impartial anticipation in runtime-verification. In: ATVA. LNCS, Springer (2008)
7. Biere, A., Clarke, E.M., Raimi, R., Zhu, Y.: Verifiying safety properties of a power pc microprocessor using symbolic model checking without bdds. In: CAV. LNCS, Springer (1999)
8. Chen, F., Rosu, G.: Parametric trace slicing and monitoring. In: TACAS. LNCS, Springer (2009)
9. Bauer, A., Küster, J.C., Vegliach, G.: From propositional to first-order monitoring. In: RV. LNCS, Springer (2013)
10. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: FSTTCS. LNCS, Springer (2006)
11. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for ltl and tltl. ACM Trans. Softw. Eng. Methodol. (2011)
12. Leucker, M., Sánchez, C.: Regular linear temporal logic. In: ICTAC. LNCS, Springer (2007)
13. Decker, N., Leucker, M., Thoma, D.: Impartiality and anticipation for monitoring of visibly context-free properties. In: RV. LNCS, Springer (2013)
14. Bauer, A., Leucker, M., Schallhart, C.: The good, the bad, and the ugly, but how ugly is ugly? In: RV. LNCS, Springer (2007)
15. de Moura, L.M., Bjørner, N.: Z3: An efficient smt solver. In: TACAS. LNCS, Springer (2008)