Practical First-Order Temporal Reasoning

Clare Dixon, Michael Fisher, Boris Konev and Alexei Lisitsa
Department of Computer Science, University of Liverpool,
Liverpool, United Kingdom
{C.Dixon, M.Fisher, B.Konev, A.Lisitsa}@csc.liv.ac.uk

Abstract

In this paper we consider the specification and verification of infinite-state systems using temporal logic. In particular, we describe parameterised systems using a new variety of first-order temporal logic that is both powerful enough for this form of specification and tractable enough for practical deductive verification. Importantly, the power of the temporal language allows us to describe (and verify) asynchronous systems, communication delays and more complex liveness and fairness properties. These aspects appear difficult for many other approaches to infinite-state verification.

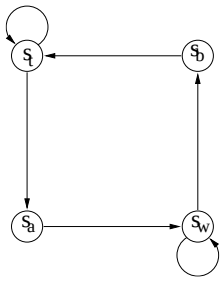
1. Introduction

The verification of infinite-state systems, particularly parameterised systems comprising of arbitrary numbers of identical processes, is increasingly important. Practical problems of an open, distributed nature often fit into this model. In assessing the reliability of these systems, formal verification is clearly desirable and so several approaches have been developed. Two of the most popular are *model checking for parameterised and infinite state-systems* [1, 2] and *constraint based verification using counting abstractions* [8, 9, 13], but both suffer problems. Within the model checking approach, formulae are translated into a Büchi transducer. Techniques from regular model checking are then used to search for models. This approach has been applied to several algorithms verifying safety properties and some liveness properties. However, the logic LTL(MSO), considered in [1], is *not* recursively enumerable; so regular model checking is incomplete. Constraint-based approaches [9] provide complete procedures for checking safety properties of broadcast protocols. However, these approaches have theoretically non-primitive recursive upper bounds for decision procedures (although they work well for small examples), are not suitable (or, have not been used) for asynchronous systems with delayed broadcast, and typically lead to undecidability problems if applied to liveness properties.

Thus, there is need for an approach that can tackle the verification of parameterised systems in a *complete* and *decidable* way, and also to tackle a wider class of system, incorporating asynchrony, communication delay, etc. In this paper we introduce a new *first-order temporal logic*, FOTLX, that allows us to achieve this. This logic combines previous work on the specification and verification of parameterised systems using *monodic* temporal logic [15] with more recent work on efficient temporal logics [11]. We show that the complexity of reasoning in FOTLX is lower than in unrestricted monodic first-order temporal logic. The resulting formalism is new, and now allows us to describe, and verify properties of, a wide and important class of parameterised systems.

The verification of concurrent systems often comes down to the analysis of families of finite-state automata, for example of the form given in Fig. 1(a). In describing such automata, both automata-theoretic (such as model checking) and logical approaches may be used. Recently, a propositional, linear-time temporal logic with improved deductive properties has been introduced [10, 11], providing the possibility of practical deductive verification in the future. The essence of this approach is to provide constraints between key propositions. These constraints state that exactly one proposition from an subset of propositions can be true at any moment in time. Thus, the automaton given in Fig. 1(a) can be described by the clauses given in Fig. 1(b), which are implicitly in the scope of a '$\Box$' ('always in the future') operator. Here '$\bigcirc$' is a temporal operator denoting 'at the next moment' and '**start**' is a temporal operator which holds only at the initial moment in time. The inherent assumption is that at any moment in time exactly one of s_a, s_b, s_t or s_w holds. With improved complexity results for such logics then the properties of any finite collection of such automata can be tractably verified using this *propositional* exactly-one temporal logic [10].

This naturally leads to the question of whether the exactly-one approach can be extended to *first-order tem-*



1. **start** $\Rightarrow s_t$
2. $s_t \Rightarrow \bigcirc(s_t \vee s_a)$
3. $s_b \Rightarrow \bigcirc s_t$
4. $s_a \Rightarrow \bigcirc s_w$
5. $s_w \Rightarrow \bigcirc(s_w \vee s_b)$

Figure 1. Finite-State Automaton (a) and its Propositional Temporal Specification (b)

poral logics and, if so, whether a form of tractability still applies. In such an approach, we can consider *infinite* numbers of finite-state automata (initially, all of the same structure). Previously, we have shown that FOTL (without explicit exactly-one constraints) can be used to elegantly (but with inherent complexity) specify such a system, simply by assuming the argument to each predicate represents a particular automaton [15]. Thus, in the following $s_a(x)$ is true if automaton x is in state s_a:

1. **start** $\Rightarrow \exists x.s_t(x)$
2. $\forall x.\,(s_t(x) \Rightarrow \bigcirc(s_t(x) \vee s_a(x)))$
3. $\forall x.\,(s_b(x) \Rightarrow \bigcirc s_t(x))$
4. $\forall x.\,(s_a(x) \Rightarrow \bigcirc s_w(x))$
5. $\forall x.\,(s_w(x) \Rightarrow \bigcirc(s_w(x) \vee s_b(x)))$

The contributions of this paper are the introduction of an exactly-one version of monodic first-order-temporal logic, the related complexity result and its application to a sophisticated process model. This allows us to not only specify broadcast protocols between synchronous components [13], but also to specify asynchronous systems with communication delays. We show that such a process model can be faithfully represented in FOTLX, making it possible to practically verify its properties.

Thus, in summary, we here tackle the problem of specifying and verifying parameterised systems by introducing a new temporal logic formalism. This logic not only allows us to describe, and prove properties about, a wider range of system, but also allows us to do so with improved complexity over previous temporal logic approaches.

2. FOTLX

2.1. First-Order Temporal Logic

The language of First-Order (descrete, linear time) Temporal Logic, FOTL, is an extension of classical first-order logic by temporal operators for a discrete linear model of time (isomorphic to $\mathbb{N}$, that is, the most commonly used model of time). The signature of FOTL (without equality and function symbols) consists of a countably infinite set of *variables* x_0, x_1, ..., a countably infinite set of *constants* $c_0, c_1, \ldots$, a non-empty set of *predicate symbols* P, $P_0, \ldots$, each with a fixed arity ≥ 0, the *propositional operators* $\top$, $\neg$, $\vee$, the *quantifiers* $\exists x_i$ and $\forall x_i$, and the *temporal operators* $\Box$ ('always in the future'), $\Diamond$ ('eventually in the future'), $\bigcirc$ ('at the next moment'), and U ('until'). The set of formulae of FOTL is defined as follows: $\top$ is a FOTL formula; if P is an n-ary predicate symbol and t_1, ..., t_n are variables or constants, then $P(t_1, \ldots, t_n)$ is an *atomic* FOTL formula; if φ and ψ are FOTL formulae, then so are $\neg\varphi$, $\varphi \vee \psi$, $\exists x\varphi$, $\forall x\varphi$, $\Box\varphi$, $\Diamond\varphi$, $\bigcirc\varphi$, and $\varphi\,\mathsf{U}\,\psi$. We also use $\bot$, $\wedge$, and $\Rightarrow$ as additional operators, defined using $\top$, $\neg$, and $\vee$. Free and bound variables of a formula are defined in the standard way, as well as the notions of open and closed formulae. Given a formula φ, we write $\varphi(x_1, \ldots, x_n)$ to indicate that all the free variables of φ are among x_1, ..., x_n. As usual, a *literal* is either an atomic formula or its negation.

Intuitively, FOTL formulae are interpreted in *first-order temporal structures* which are sequences $\mathfrak{M}$ of *worlds*, $\mathfrak{M} = \mathfrak{M}_0, \mathfrak{M}_1, \ldots$ with truth values in different worlds being connected via temporal operators.

More formally, for every moment of time $n \geq 0$, there is a corresponding *first-order* structure, $\mathfrak{M}_n = \langle D, I_n \rangle$, where D is a non-empty set and I_n is an interpretation of predicate and constant symbols over D. We require that the interpretation of constants is *rigid*. Thus, for every constant c and all moments of time $i, j \geq 0$, we have $I_i(c) = I_j(c)$. A *(variable) assignment* $\mathfrak{a}$ is a function from the set of individual variables to D. We denote the set of all assignments by $\mathfrak{V}$.

The *truth* relation $\mathfrak{M}_n \models^{\mathfrak{a}} \phi$ in a structure $\mathfrak{M}$, is defined inductively on the construction of ϕ, see Fig. 2 for details (the semantics of Booleans is standard and is omitted from the figure).

$\mathfrak{M}$ is a *model* for a formula ϕ (or ϕ is *true* in $\mathfrak{M}$) if, and only if, there exists an assignment $\mathfrak{a}$ such that $\mathfrak{M}_0 \models^{\mathfrak{a}} \phi$. A formula is *satisfiable* if, and only if, it has a model. A formula is *valid* if, and only if, it is true in any temporal structure $\mathfrak{M}$ under any assignment $\mathfrak{a}$.

2.2. Monodicity and Monadicity

The set of valid formulae of FOTL is not recursively enumerable. Furthermore, it is known that even "small" fragments of FOTL, such as the *two-variable monadic* fragment (where all predicates are unary), are not recursively enumerable [24, 20]. However, the set of valid *monodic* formulae is known to be finitely axiomatisable [26].

Definition 1 *An FOTL-formula ϕ is called* monodic *if, and only if, any subformula of the form $\mathcal{T}\psi$, where $\mathcal{T}$ is one of*

$\mathfrak{M}_n \models^{\mathfrak{a}}$ **true**		$\mathfrak{M}_n \not\models^{\mathfrak{a}}$ **false**
$\mathfrak{M}_n \models^{\mathfrak{a}}$ **start**	iff	$n = 0$
$\mathfrak{M}_n \models^{\mathfrak{a}} P(t_1, \ldots, t_m)$	iff	$\langle I_n^{\mathfrak{a}}(t_1), \ldots I_n^{\mathfrak{a}}(t_m)\rangle \in I_n(P)$, where $I_n^{\mathfrak{a}}(t_i) = I_n(t_i)$, if t_i is a constant, and $I_n^{\mathfrak{a}}(t_i) = \mathfrak{a}(t_i)$, if t_i is a variable
$\mathfrak{M}_n \models^{\mathfrak{a}} \forall x \phi$	iff	$\mathfrak{M}_n \models^{\mathfrak{b}} \phi$ for every assignment $\mathfrak{b}$ that may differ from $\mathfrak{a}$ only in x
$\mathfrak{M}_n \models^{\mathfrak{a}} \exists x \phi$	iff	$\mathfrak{M}_n \models^{\mathfrak{b}} \phi$ for some assignment $\mathfrak{b}$ that may differ from $\mathfrak{a}$ only in x
$\mathfrak{M}_n \models^{\mathfrak{a}} \bigcirc \phi$	iff	$\mathfrak{M}_{n+1} \models^{\mathfrak{a}} \phi$
$\mathfrak{M}_n \models^{\mathfrak{a}} \Diamond \phi$	iff	there exists $m \geq n$ such that $\mathfrak{M}_m \models^{\mathfrak{a}} \phi$
$\mathfrak{M}_n \models^{\mathfrak{a}} \Box \phi$	iff	for all $m \geq n$, $\mathfrak{M}_m \models^{\mathfrak{a}} \phi$
$\mathfrak{M}_n \models^{\mathfrak{a}} (\phi \,\mathsf{U}\, \psi)$	iff	there exists $m \geq n$, such that $\mathfrak{M}_m \models^{\mathfrak{a}} \psi$ and, for all $i \in \mathbb{N}, n \leq i < m$ implies $\mathfrak{M}_i \models^{\mathfrak{a}} \phi$
$\mathfrak{M}_n \models^{\mathfrak{a}} (\phi \,\mathsf{W}\, \psi)$	iff	$\mathfrak{M}_n \models^{\mathfrak{a}} (\phi \,\mathsf{U}\, \psi)$ or $\mathfrak{M}_n \models^{\mathfrak{a}} \Box \phi$.

Figure 2. FOTL semantics

$\bigcirc$, $\Box$, $\Diamond$ (or $\psi_1 \mathcal{T} \psi_2$, where $\mathcal{T}$ is one of U , W *), contains at most one free variable.*

We note that the addition of either equality or function symbols to the monodic fragment generally leads to the loss of recursive enumerability [26, 7, 18]. Further, even with its recursive enumerability, monodic FOTL is generally undecidable. To recover decidability, the easiest route is to restrict the first order part to some decidable fragment of first-order logic, such as the guarded, two-variable or monadic fragments. We here choose the latter, since monadic predicates fit well with our intended application to parameterised systems. Recall that monadicity requires that all predicates have arity of at most '1'. Thus, we use monadic, monodic FOTL [6].

The resolution theorem-prover TeMP [21] provides a a practical approach to proving monodic temporal formulae. In the past, TeMP has been successfully applied to problems from several domains [16], in particular, to examples specified in the temporal logics of knowledge (the fusion of propositional linear-time temporal logic with multi-modal S5). From this work it is clear that monodic first-order temporal logic is an important tool for specifying complex systems. However, it is also clear that the complexity, even of *monadic* monodic first-order temporal logic, makes this approach difficult to use for larger applications [16, 15].

2.3. Exactly-One Restrictions

An additional restriction we make to the above logic involves implicit exactly-one constraints over predicates. Such restrictions were introduced into propositional temporal logics in [10], where the correspondence with Büchi automata was described, and generalised in [11]. In both cases, the decision problem is of much better (polynomial even in the first case) complexity than that for the standard, unconstrained, logic. However, in these papers only *propositional* temporal logic was considered. We now add such an exactly-one constraint to monodic FOTL.

The set of predicate symbols $\Pi = \{P_0, P_1, \ldots\}$, is now partitioned into a set of exactly-one subsets, X_1, X_2, ..., X_n, with one *unconstrained* set N such that

1. all X_i are disjoint with each other,
2. N is disjoint with every X_i,
3. $\Pi = \bigcup_{j=1}^{n} X_j \cup N$, and
4. for each X_i, exactly *one* predicate within X_i is satisfied (for any element of the domain) at any moment in time.

Example 1 *Consider two exactly-one sets, $X_1 = \{P_1, P_2\}$ and $X_2 = \{P_4, P_7, P_8\}$. Then for any element of the domain, a, exactly one of $P_1(a)$ or $P_2(a)$ must be satisfied and exactly one of $P_4(a)$, $P_7(a)$ or $P_8(a)$ must be satisfied.*

2.4. Normal Form

Every monodic FOTLX formula can be translated to a *normal form* in a satisfiability preserving way using a renaming and unwinding technique which substitutes non-atomic subformulae and replaces temporal operators by their fixed point definitions as described, for example, in [14].

A *monodic temporal problem in Divided Separated Normal Form (DSNF)* [6] is a quadruple $\langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$, where:

1. the universal part, $\mathcal{U}$, is a finite set of arbitrary closed first-order formulae;
2. the initial part, $\mathcal{I}$, is, again, a finite set of arbitrary closed first-order formulae;
3. the step part, $\mathcal{S}$, is a finite set of step clauses of the form $P_1(x) \wedge \ldots P_k(X) \Rightarrow \bigcirc Q_1(x) \vee \ldots \vee Q_l(x)$, where P_i and $Q_j$are unary predicate symbols;

4. the eventuality part, $\mathcal{E}$, is a finite set of eventuality clauses of the form $\Diamond L(x)$, where $L(x)$ is a unary literal.

In what follows, we will not distinguish between a finite set of formulae $\mathcal{X}$ and the conjunction $\bigwedge \mathcal{X}$ of formulae within the set. With each monodic temporal problem, we associate the formula $\mathcal{I} \wedge \Box\mathcal{U} \wedge \Box\forall x\mathcal{S} \wedge \Box\forall x\mathcal{E}$. Now, when we talk about particular properties of a temporal problem (e.g., satisfiability, validity, logical consequences etc) we mean properties of the associated formula.

To translate a formula into the normal form, we recursively rename each innermost open subformula $\xi(x)$, whose main connective is a temporal operator, by $P_\xi(x)$, where $P_{\xi(x)}$ is a new unary predicate, and rename each innermost closed subformula ζ, whose main connective is a temporal operator, by p_ζ, where p_ζ is a new propositional variable.

The translation linearly increases the size of the formula—full details can be found e.g. in [14]. While renaming introduces new, unconstrained predicates and propositions, practical problems stemming from verification are nearly in the normal form, see Section 3.

2.5. Complexity

First-order temporal logics are notorious for being of a high complexity. Even decidable sub-fragments of monodic first-order temporal logic can be too complex for practical use. For example, satisfiability of monodic monadic FOTL logic is known to be EXPSPACE-complete [19]. However, imposing exactly-one restrictions we obtain better complexity bounds.

Theorem 1 *Satisfiability of monodic monadic FOTLX formulae (in the normal form) can be decided in* $2^{O(N_1 \cdot N_2 \cdot \ldots \cdot N_n \cdot 2^{N_a})}$ *time, where* $N_1, \ldots, N_n$ *are cardinalities of the sets of exactly-one predicates, and* N_a *is the cardinality of the set of the unconstrained predicates.*

Corollary 2 *If the number,* n*, of sets of exactly-one predicates and the cardinality,* N_a*, of the set of unconstrained predicates is fixed, satisfiability of FOTLX formulae can be decided in deterministic exponential time (which is an improvement compared to the EXPSPACE complexity of the general case).*

Before we sketch the proof of this result, we show how the exactly-one restrictions influence the complexity of the satisfiability problem for monadic first-order (non-temporal) logic.

Lemma 3 *Satisfiability of monadic first-order formulae can be decided in* $\mathsf{NTime}(O(m \cdot N_1 \cdot N_2 \cdot \ldots \cdot N_n \cdot 2^{N_a}))$*, where* m *is the length of the formula, and* $N_1, \ldots, N_n$*,* N_a *are as in Theorem 1.*

Proof Similar to [3], Proposition 6.2.9. ❑

Proof [of Theorem 1, Sketch] For simplicity of presentation, we assume the formula contains no propositions. Satisfiability of a monodic FOTL formula is equivalent to a property of the *behaviour graph* for the formula, checkable in time polynomial in the product of the number of different predicate colours and the size of the graph, see [6], Theorem 5.15. For unrestricted FOTL formulae, the size of the behaviour graph is double exponential in the number of predicates. We estimate now the size of the behaviour graph and time needed for its construction for FOTLX formulae.

A predicate colour, γ, is a set of unary literals such that for every predicate $P(x)$ from the set of all predicates $X_1 \cup \ldots, \cup X_n \cup N$, either $P(x)$ or $\neg P(x)$ belongs to γ. Let Γ be a set of predicate colours and ρ be a map from the set of constants, $\mathrm{const}(\mathsf{P})$, to Γ. A pair $\langle \Gamma, \rho \rangle$ is called a *colour scheme*. Nodes of the behaviour graph are colour schemes. Clearly, there are no more than $2^{O(N_1 \cdot N_2 \cdot \ldots \cdot N_n \cdot 2^{N_a})}$ different colour schemes. However, not every colour scheme is a node of the behaviour graph: a colour scheme $\mathcal{C}$ is a node if, and only if, a monadic formula of first-order (non-temporal) logic, constructed from the given FOTLX formula and the colour scheme itself, is satisfiable (for details see [6]). A similar first-order monadic condition determines which nodes are connected with edges. So, the size of the formula is polynomial. By Lemma 3, satisfiability of monadic first-order formulae can be decided in deterministic $2^{O(N_1 \cdot N_2 \cdot \ldots \cdot N_n \cdot 2^{N_a})}$ time.

Overall, the behaviour graph, representing all possible models, for an FOTLX formula can be constructed in $2^{O(N_1 \cdot N_2 \cdot \ldots \cdot N_n \cdot 2^{N_a})}$ time. ❑

3. Parameterised Systems

Next we present a model suitable for the specification of both synchronous and asynchronous systems (protocols) with (possibly) delayed broadcast and give its faithful translation into FOTLX. This shows that the logic developed does indeed achieve what we intended. In addition, given the improved complexity results of the previous section, we believe this approach provides a route towards the *practical* verification of temporal properties of such infinite state systems.

A parameterised finite state machine based model, suitable for the specification and verification of protocols over arbitrary numbers of processes was defined in [13, 8]. Essentially, this uses a family of identical, and synchronously executing, finite state automata with a rudimentary form of communication: if one automaton makes a transition (an action) a, then it is required that *all* other automata simultaneously make a complementary transition (reaction) $\bar{a}$. In [15] we translated this automata model into monodic FOTL

and used automated theorem proving in that logic to verify parameterised cache coherence protocols [9]. The model assumed not only synchronous behaviour of the communicating automata, but instantaneous broadcast.

3.1. Process Model

We now describe both the asynchronous model, and the delayed broadcast approach.

Definition 2 (Protocol) *A protocol, P is a tuple $\langle Q, I, \Sigma, \tau \rangle$, where*

- *Q is a finite set of states;*
- *$I \subseteq Q$ is a set of initial states;*
- *$\Sigma = \Sigma_L \cup \Sigma_M \cup \bar{\Sigma}_M$, where*

  Σ_L is a finite set of local actions;

  Σ_M is a finite set of broadcast actions, i.e. "send a message";

  $\bar{\Sigma}_M = \{\bar{\sigma} \mid \sigma \in \Sigma_M\}$ is the set of broadcast reactions, i.e. "receive a message";
- *$\tau \subseteq Q \times \Sigma \times Q$ is a transition relation that satisfies the following property*
  $\forall \sigma \in \Sigma_M.\ \forall q \in Q.\ \exists q' \in Q.\ \langle q, \bar{\sigma}, q' \rangle \in \tau$ i.e., "readiness to receive a message in any state".

Further, we define a notion of global machine, which is a set of n finite automata, where n is a parameter, each following the protocol and able to communicate with others via (possibly delayed) broadcast. To model asynchrony, we introduce a special automaton action, $idle \notin \Sigma$, meaning the automaton is not active and so its state does not change. At any moment an arbitrary group of automata may be idle and all non-idle automata perform their actions in accordance with the transition function τ; different automata may perform different actions.

Definition 3 (Asynchronous Global Machine) *Given a protocol, $\mathcal{P} = \langle Q, I, \Sigma, \tau \rangle$, the global machine $\mathcal{M}_G$ of dimension n is the tuple $\langle Q_{\mathcal{M}_G}, I_{\mathcal{M}_G} \tau_{\mathcal{M}_G}, \mathcal{E} \rangle$, where*

- *$Q_{\mathcal{M}_G} = Q^n$*
- *$I_{\mathcal{M}_G} = I^n$*
- *$\tau_{\mathcal{M}_G} \subseteq Q_{\mathcal{M}_G} \times (\Sigma \cup \{idle\})^n \times Q_{\mathcal{M}_G}$ is a transition relation that satisfies the following property*

$$\begin{array}{l} \langle \langle s_1, \ldots, s_n \rangle, \langle \sigma_1, \ldots \sigma_n \rangle, \langle s'_1, \ldots, s'_n \rangle \rangle \in \tau_{\mathcal{M}_G} \\ \quad \textit{iff} \quad \forall 1 \leq i \leq n. \\ \qquad ((\sigma_i \neq idle \Rightarrow \langle s_i, \sigma_i, s'_i \rangle \in \tau) \wedge \\ \qquad (\sigma_i = idle \Rightarrow s_i = s'_i)) \,. \end{array}$$

- *$\mathcal{E} = 2^{\Sigma_M}$ is a communication environment, that is a set of possible sets of messages in transit.*

An element $G \in Q_{\mathcal{M}_G} \times (\Sigma \cup \{idle\})^n \times \mathcal{E}$ is said to be a global configuration of the machine. A run of a global machine $\mathcal{M}_G$ is a possibly infinite sequence $\langle s^1, \sigma^1, E_1 \rangle \ldots\ldots \langle s^i, \sigma^i, E_i \rangle \ldots\ldots$ of global configurations of $\mathcal{M}_G$ satisfying the properties (1)–(6) listed below. In this formulation we assume $s^i = \langle s^i_1, \ldots, s^i_n \rangle$ and $\sigma^i = \langle \sigma^i_1, \ldots, \sigma^i_n \rangle$.

1. *$s^1 \in I^n$ ("initially all automata are in initial states");*
2. *$E_1 = \emptyset$ ("initially there are no messages transit");*
3. *$\forall i.\ \langle s^i, \sigma^i, s^{i+1} \rangle \in \tau_{\mathcal{M}_G}$ ("an arbitrary automaton can fire");*
4. *$\forall a \in \Sigma_M.\ \forall i.\ \forall j.\ ((\sigma^i_j = a) \Rightarrow \forall k.\ \exists l \geq i.\ (\sigma^l_k = \bar{a}))$ ("delivery to all participants is guaranteed");*
5. *$\forall a \in \Sigma_M.\ \forall i.\ \forall j.\ [(\sigma^i_j = \bar{a}) \Rightarrow (a \in E_i) \vee \exists k.\ \sigma^i_k = a)]$ ("one can receive only messages kept by the environment, or sent at the same moment of time ")*

In order to formulate further requirements we introduce the following notation:

$$Sent_i = \{a \in \Sigma_M \mid \exists j.\ \sigma^i_j = a\}$$

$$Delivered_k = \left\{ a \in \Sigma_M \;\middle|\; \begin{array}{l} \exists i \leq k.\ (a \in Sent_i) \wedge \\ (\forall l.\ (i < l < k) \rightarrow a \notin Sent_l) \wedge \\ (\forall j. \exists l.\ (i \leq l \leq k) \wedge (\sigma^l_j = \bar{a})) \end{array} \right\}$$

Then, the final requirement the run should satisfy is

6. *$\forall i.\ E_{i+1} = (E_i \cup Sent_i) - Delivered_i$*

This process model is quite expressive, capturing many interesting and useful systems. In particular, it is rich enough to allow us to describe, for example, such diverse systems as cache coherence protocols, multi-agent swarms, and distributed atomic commitment protocols including the two- and three-phase commit protocols [17, 25] and their modifications [5, 4]. For the sake of space, however, we will only consider a simple asynchronous distributed consensus example, called the *FloodSet protocol*, in Section 3.3

3.2. Temporal Translation

Given a protocol $\mathcal{P} = \langle Q, I, \Sigma, \tau \rangle$, we define its translation to FOTLX as follows. For each $q \in Q$, introduce a monadic predicate symbol P_q and for each $\sigma \in \Sigma \cup \{idle\}$ introduce a monadic predicate symbol A_σ. For each $\sigma \in \Sigma_M$ we introduce also a propositional symbol m_σ. Intuitively, elements of the domain in the temporal representation will represent exemplars of finite automata, and the formula $P_q(x)$ is intended to represent "automaton x is in state q". The formula $A_\sigma(x)$ is going to represent "automaton x performs action σ". Proposition m_σ will denote the fact

"message σ is in transit" (i.e. it has been sent but not all participants have received it). Because of the intended meaning we define two exactly-one sets: $X_1 = \{P_q \mid q \in Q\}$ and $X_2 = \{A_\sigma \mid \sigma \in \Sigma \cup \{idle\}\}$. All other predicates belong to the set of unconstrained predicates.
We define the temporal translation of $\mathcal{P}$, called $T_\mathcal{P}$, as a conjunction of the formulae in Fig. 3. Note that, in order to define the temporal translation of requirement (6) above, (on the dynamics of environment updates) we introduce the unary predicate symbol $Received_\sigma$ for every $\sigma \in \Sigma_m$. When an automaton a receives message σ, $Received_\sigma(a)$ becomes true. When $Received_\sigma$ becomes true for all x, the message is delivered and is not in transit any more, that is, m_σ becomes false.
We now consider the correctness of the temporal translation. This translation of protocol $\mathcal{P}$ is faithful in the following sense.

Proposition 1 *Given a protocol, $\mathcal{P}$, and a global machine, $\mathcal{M}_G$, of dimension n, then any temporal model $M_1, M_2, \ldots$ of $T_\mathcal{P}$ with the finite domain $c_1, \ldots c_n$ of size n represents some run $\langle s^1, \sigma^1, E_1\rangle \ldots \langle s^i, \sigma^i, E_i\rangle \ldots$ of $\mathcal{M}_G$ as follows:*

$\langle\langle s_1, \ldots, s_n\rangle, \langle \sigma_1, \ldots, \sigma_n\rangle, E\rangle$ is i-th configuration of the run iff $M_i \models P_{q_1}(c_1) \wedge \ldots P_{q_n}(c_n)$, $M_i \models A_{\sigma_1}(c_1) \wedge \ldots A_{\sigma_n}(c_n)$ and $E = \{\sigma \in \Sigma_m \mid M_i \models m_\sigma\}$

Dually, for any run of $\mathcal{M}_G$ there is a temporal model of $T_\mathcal{P}$ with a domain of size n representing this run.

Thus, given a parameterised system that fits into the above model, we can translate its specification (faithfully) into FOTLX.

3.3. Example

We here consider a variant of the *FloodSet algorithm with alternative decision rule* (in terms of [22], p.105) designed for solution of the Consensus problem.

The setting is as follows. There are n processes, each having an *input bit* and an *output bit*. The processes work asynchronously, run the same algorithm and use *broadcast* for communication. (The process is described graphically in Fig. 4.) The broadcast messages are guaranteed to be delivered, though possibly with arbitrary delays.

The goal of the algorithm is to eventually reach an agreement, i.e. to produce an output bit, which would be the same for all processes. It is required also that if all processes have the same input bit, that bit should be produced as an output bit.

The asynchronous FloodSet protocol we consider here is adapted from [22], the main differences being:

- the original protocol was synchronous, while our variant is asynchronous;

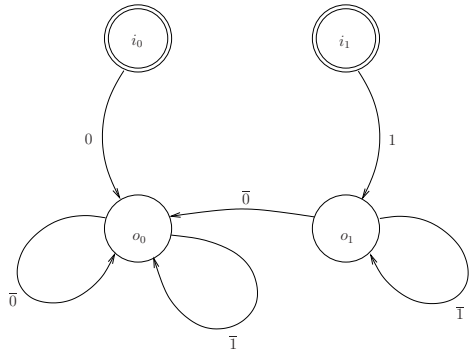


Figure 4. Asynchronous FloodSet Protocol Process.

- the original protocol assumed instantaneous message delivery, while we allow arbitrary delays in delivery; and
- although the original protocol was designed to work in the presence of crash (or fail-stop) failures, we assume, for simplicity, that there are no failures.

Because of the absence of failures the protocol is simplified and, unlike the original, does not require "retransmission" of any value. We will show later (in Section 3.5) how to include the case of crash failures in the specification (and verification). Thus, the asynchronous FloodSet protocol is defined, informally, as follows.

- At the first round of computations, every process broadcasts its input bit.
- At every round the (tentative) output bit is set to the minimum value ever seen so far.

The correctness criterion for this protocol is that, eventually, the output bits of all processes will be the same (either '0' or '1').
Now we can specify the asynchronous FloodSet as a protocol $\langle Q, I, \Sigma, \tau\rangle$, where:

- $Q = \{i_0, i_1, o_0, o_1\}$;
- $I = \{i_0, i_1\}$;
- $\Sigma = \Sigma_m \cup \bar{\Sigma}_m \cup \Sigma_L$ with $\Sigma_m = \{0, 1\}$, $\bar{\Sigma}_m = \{\bar{0}, \bar{1}\}$, $\Sigma_L = \emptyset$; and
- the transition relation $\tau = \{\langle i_0, 0, o_0\rangle, \langle o_0, \bar{0}, o_0\rangle, \langle o_0, \bar{1}, o_0\rangle, \langle i_1, 1, o_1\rangle, \langle o_1, \bar{0}, o_0\rangle, \langle o_1, \bar{1}, o_1\rangle\}$.

3.4. Verifying Properties of the Example

Now we have all the ingredients to perform the verification of parameterised protocols. Given a protocol $\mathcal{P}$, we

I. Each automaton either performs one of the actions available in its state, or is idle:
$\Box[\forall x.\ P_q(x) \rightarrow A_{\sigma_1}(x) \vee \ldots \vee A_{\sigma_k}(x) \vee A_{idle}(x)]$, where $\{\sigma_1, \ldots \sigma_k\} = \{\sigma \in \Sigma \mid \exists r \langle q, \sigma, r\rangle \in \tau\}$.

II. Action effects (non-deterministic actions):
$\Box[\forall x P_q(x) \wedge A_\sigma(x) \rightarrow \bigcirc \bigvee_{\langle q,\sigma,r\rangle \in \tau} P_r(x)]$ for all $q \in S$ and $\sigma \in \Sigma$.

III. Effect of being idle: $\Box[\forall x P_q(x) \wedge A_{idle}(x) \rightarrow \bigcirc P_q(x)]$, for all $q \in S$

IV. Initially there are no messages in transit and all automata are in an initial state: **start** $\rightarrow \neg m_\sigma$ for all $\sigma \in \Sigma_m$ and **start** $\rightarrow \forall x \bigvee_{q \in I} P_q(x)$.

V. All messages are eventually received (Guarantee of Delivery): $\Box[\exists y A_\sigma(y) \rightarrow \forall x \Diamond A_{\bar{\sigma}}(x)]$, for all $\sigma \in \Sigma_m$.

VI. Only messages kept in the environment (in transit), or sent at the same moment of time can be received: $\Box[\forall x A_{\bar{\sigma}}(x) \rightarrow m_\sigma \vee \exists y A_\sigma(y)]$ for all $\sigma \in \Sigma_m$.

VII. Finally, for all $\sigma \in \Sigma_m$, we have the conjunction of the following formulae specifying the communication model:

1. **start** $\rightarrow \forall x.\ \neg Received_\sigma(x)$
2. $\Box[\forall x.\ (A_{\bar{\sigma}}(x) \wedge \neg\forall y.\ Received_\sigma(y)) \rightarrow \bigcirc Received_\sigma(x)]$
3. $\Box[\forall x.\ (Received_\sigma(x) \wedge \neg\forall y.\ Received_\sigma(y) \rightarrow \bigcirc Received_\sigma(x)]$
4. $\Box[\forall x.\ (\neg(A_{\bar{\sigma}}(x) \vee Received_\sigma(x)) \wedge \neg\forall y.\ Received_\sigma(y)) \rightarrow \bigcirc \neg Received_\sigma(x)]$
5. $\Box[\forall x.\ Received_\sigma(x) \rightarrow \bigcirc \neg m_\sigma]$
6. $\Box[\exists x.\ A_\sigma(x) \wedge \neg\forall y.\ Received_\sigma(y) \rightarrow \bigcirc m_\sigma]$
7. $\Box[\neg\exists x.\ A_\sigma(x) \wedge \neg\forall y.\ Received_\sigma(y) \rightarrow (m_\sigma \leftrightarrow \bigcirc m_\sigma)]$

Figure 3. Temporal Specification of Abstract Protocol Structure.

can translate it into a temporal formula $T_{\mathcal{P}}$. For the temporal representation, χ, of a required correctness condition, we then check whether $T_{\mathcal{P}} \rightarrow \chi$ is a valid temporal formula. If it is valid, then the protocol is correct for all possible values of the parameter (sizes).

Correctness conditions can, of course, be described using any legal FOTLX formula. For example, for the above FloodSet protocol(s) we have a liveness condition to verify that eventually all processes will agree on '0' or will agree on '1':

$$\Diamond(\forall x.\ o_0(x) \vee \forall x.\ o_1(x)).$$

3.5. Variations of the model

The above model allows us to introduce various extensions and the corresponding version of Proposition 1 still holds.

Determinism. The basic model allows non-deterministic actions. To specify the case of deterministic actions only, one should replace the "Action Effects" axiom in Fig. 3 by the following variant (for all $\langle q, \sigma, r\rangle \in \tau$):

$$\Box[\forall x.\ P_q(x) \wedge A_\sigma(x) \rightarrow \bigcirc P_r(x)]$$

Explicit bounds on delivery. In the basic mode, no explicit bounds on delivery time are given. To introduce bounds one has to replace the "Guarantee of Delivery" axiom by:

$$\Box[\exists y.\ A_\sigma(y) \rightarrow \forall x.\ A_{\bar{\sigma}}(x) \vee \bigcirc A_{\bar{\sigma}}(x) \vee \ldots \vee \bigcirc^n A_{\bar{\sigma}}(x)]$$

for all $\sigma \in \Sigma_m$ and some n (representing the maximal delay).

In [15], we considered a deterministic model with *instantaneous delivery* (that is, the explicit bounds case with $n = 0$).

Finite bounds on delivery. One may replace the "Guarantee of Delivery" axiom with (for all $\sigma \in \Sigma_m$) the following:

$$\Box[\exists y.\ A_\sigma(y) \rightarrow \Diamond\forall x.\ Received_{\bar{\sigma}}(x)]$$

Guarded actions. One can also extend the model with guarded actions, where actions can be performed depending on global conditions in global configurations.

Crashes. One may replace the "Guarantee of Delivery" axiom by an axiom stating that only the messages sent by normal (non-crashed) participants will be delivered to all

participants. (See [15] for examples of such specifications in a FOTL context.)

Returning to the FloodSet protocol, for example, one may consider a variation of the asynchronous protocol suitable for resolving the Consensus problem in the presence of *crash failures*. We can modify the above setting as follows. Now, processes may fail and, from that point onward, such processes send no further messages. Note, however, that the messages sent by a process *in the moment of failure* may be delivered to *an arbitrary subset* of the non-faulty processes. Then the FloodSet protocol considered above is modified by adding the following rule:

- At every round (after the first), a process broadcasts any value *the first time it sees it*.

The goal of the algorithm also has to be modified, so only *non-faulty* processes are required to eventually reach an agreement:

$$\Diamond \left[\begin{array}{l} (\forall x.\ \mathit{Non\text{-}faulty}(x) \rightarrow o_0(x)) \vee \\ (\forall x.\ \mathit{Non\text{-}faulty}(x) \rightarrow o_1(x)) \end{array} \right].$$

The above rules can be easily encoded in the model.

4. Concluding Remarks

In this paper, we have developed an exactly-one version of FOTL, providing: its syntax and semantics; conditions for decidability; and detailed complexity of the decision procedure. As well as being an extension and combination of the work reported in both [6] and [11], this work forms the basis for tractable temporal reasoning over infinite state problems. In particular, we have shown how the logic can capture quite a strong model of parameterised systems, incorporating more complex aspects of *asynchrony* and *communication*, and is also able to verify more sophisticated *liveness* and *fairness* properties. Thus, in contrast to many other approaches [23, 9, 2], not only safety, but also liveness and fairness properties, can be verified through (complete) automatic deductive verification.

Finally, our future work involves exploring further the framework described in this paper, in particular the development of an implementation to prove properties of protocols in practice. Further, we would like to see if we can extend the range of systems we can tackle beyond the monodic fragment. We also note that some of the variations we might desire to include in Section 3.5 can lead to undecidable fragments. However, for some of these variations, we have correct although (inevitably) incomplete methods, see [15]. We aim to explore these boundaries further.

References

[1] P. A. Abdulla, B. Jonsson, M. Nilsson, J. d'Orso, and M. Saksena. Regular Model Checking for LTL(MSO). In *Proc. 16th International Conference on Computer Aided Verification (CAV)*, volume 3114 of *LNCS*, pages 348–360. Springer, 2004.

[2] P. A. Abdulla, B. Jonsson, A. Rezine, and M. Saksena. Proving Liveness by Backwards Reachability. In *Proc. CONCUR*, volume 4137 of *LNCS*, pages 95–109. Springer, 2006.

[3] E. Börger, E Grädel, and Yu. Gurevich. *The Classical Decision Problem*. Springer, 1997.

[4] D. Chkliaev, J. Hooman, and P. van der Stock. Mechanical Verification of Transaction Processing Systems. In *Proc. 3th International Conference on Formal Engineering Methods (ICFEM 2000)*, pages 89–97, IEEE, 2000.

[5] D. Chkliaev, P. van der Stock, and J. Hooman. Mechanical verification of a Non-Blocking Atomic Commitment Protocol. In *Proc. ICDCS Workshop on Distributed System Validation and Verification*, pages 96–103, IEEE, 2000.

[6] A. Degtyarev, M. Fisher, and B. Konev. Monodic Temporal Resolution. *ACM Transactions on Computational Logic*, 7(1):108–150, January 2006.

[7] A. Degtyarev, M. Fisher, and A. Lisitsa. Equality and Monodic First-Order Temporal Logic. *Studia Logica*, 72(2):147–156, Nov. 2002.

[8] G. Delzanno. Automatic Verification of Parameterized Cache Coherence Protocols. In *Proc. 12th International Conference on Computer Aided Verification (CAV)*, volume 1855 of *LNCS*, pages 53–68, 2000.

[9] G. Delzanno. Constraint-Based Verification of Parametrized Cache Coherence Protocols. *Formal Methods in System Design*, 23(3):257–301, 2003.

[10] C. Dixon, M. Fisher, and B. Konev. Is There a Future for Deductive Temporal Verification? In *Proc. International Symposium on Temporal Representation and Reasoning (TIME)*, pages 11–18, 2006. IEEE CS Press.

[11] C. Dixon, M. Fisher, and B. Konev. Tractable Temporal Reasoning. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)* pages 318-323. AAAI Press, 2007.

[12] E. A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier, 1990.

[13] J. Esparza, A. Finkel, and R. Mayr. On the Verification of Broadcast Protocols. In *Proc. 14th IEEE Symposium on Logic in Computer Science (LICS)*, pages 352–359. IEEE CS Press, 1999.

[14] M. Fisher, C. Dixon, and M. Peim. Clausal Temporal Resolution. *ACM Transactions on Computational Logic*, 2(1):12–56, Jan. 2001.

[15] M. Fisher, B. Konev, and A. Lisitsa. Practical Infinite-state Verification with Temporal Reasoning. In *Verification of Infinite State Systems and Security*. IOS Press, January 2006.

[16] M.-C. F. Gago, U. Hustadt, C. Dixon, M. Fisher, and B. Konev. First-Order Temporal Verification in Practice. *Journal of Automated Reasoning*, 34(3):295–321, 2005.

[17] J. Gray. Notes on Database Operating Systems. *Operating Systems: An Advanced Course*, volume 60 of *LNCS*, pages 393–481. Springer, 1978.

[18] I. Hodkinson. Monodic Packed Fragment with Equality is Decidable. *Studia Logica*, 72(2):185–197, 2002.

[19] I. Hodkinson, R. Kontchakov, A. Kurucz, F. Wolter, and M. Zakharyaschev. On the Computational Complexity of Decidable Fragments of First-Order Linear Temporal Logics. In *Proc. International Symposium on Temporal Representation and Reasoning (TIME)*, pages 91–98. IEEE CS Press, 2003.

[20] I. Hodkinson, F. Wolter, and M. Zakharyashev. Decidable Fragments of First-Order Temporal Logics. *Annals of Pure and Applied Logic*, 106(1-3): 85-134, 2000.

[21] U. Hustadt, B. Konev, A. Riazanov, and A. Voronkov. TeMP: A Temporal Monodic Prover. In *Proc. 2nd International Joint Conference on Automated Reasoning (IJCAR)*, volume 3097 of *LNAI*, pages 326–330. Springer, 2004.

[22] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1996.

[23] M. Maidl. A Unifying Model Checking Approach for Safety Properties of Parameterized Systems. In volume 2102 of *LNCS*, pages 311–323. Springer, 2001.

[24] S. Merz. Decidability and Incompleteness Results for First-Order Temporal Logic of Linear Time. *Journal of Applied Non-Classical Logics*, 2:139–156, 1992.

[25] D. Skeen. Nonblocking Commit Protocols. In *Proc. SIGMOD International Conference on Management of Data*, pages 133–142, ACM, 1981.

[26] F. Wolter and M. Zakharyaschev. Axiomatizing the Monodic Fragment of First-Order Temporal Logic. *Annals of Pure and Applied Logic*, 118(1-2):133–145, 2002.