Temporal Verification of Fault-Tolerant Protocols

Michael Fisher, Boris Konev, and Alexei Lisitsa

Department of Computer Science, University of Liverpool, Liverpool, United Kingdom
{MFisher, Konev, A.Lisitsa}@liverpool.ac.uk

1 Introduction

The automated verification of concurrent and distributed systems is a vibrant and successful area within Computer Science. Over the last 30 years, *temporal logic* [10, 20] has been shown to provide a clear, concise and intuitive description of many such systems, and automata-theoretic techniques such as *model checking* [7, 14] have been shown to be very useful in practical verification. Recently, the verification of *infinite-state* systems, particularly parameterised systems comprising *arbitrary* numbers of identical processes, has become increasingly important [5]. Practical problems of an open, distributed nature often fit into this model, for example robot swarms of arbitrary sizes.

However, once we move beyond finite-state systems, which we do when we consider systems with arbitrary numbers of components, problems can occur. Although temporal logic still retains its ability to express such complex systems, verification techniques such as model checking must be modified. *Abstraction* techniques are typically used to reduce an infinite-state problem down to a finite-state variant suitable for application of standard model checking techniques. However, it is clear that such abstraction techniques are not always easy to apply and that more sophisticated verification approaches must be developed.

In assessing the reliability of such infinite-state systems, formal verification is clearly desirable and, consequently, several new approaches have been developed:

1. *model checking for parameterised and infinite state-systems* [1, 2];
2. *constraint based verification using counting abstractions* [9, 11];
3. *verification based on interactive theorem proving* [21, 22], *including that for temporal logic* [4, 23];
   and
4. *deductive verification in first-order* decidable *temporal logics* [12, 8].

The last of these approaches is particularly appealing, often being both complete (unlike (1)) and decidable (unlike (2)), able to verify both safety *and* liveness properties, and adaptable to more sophisticated systems involving asynchronous processes or communication delays. It is also (unlike (3)) fully mechanisable and does not require human interaction during the proof.

Now we come to the problem of verifying fault tolerance in protocols involving an arbitrary number of processes. What if some of the processes develop faults? Will the protocol still work? And how many processes must fail before the protocol fails?

Rather than specifying *exactly* how many processes will fail, which reduces the problem to a simpler version, we wish to say that there is *some* number of faulty processes, and that failure can occur at any time. Again we can capture this using temporal logics. If we allow there to be an infinite number of failures, then the specification and verification problem again becomes easier; however, such scenarios appear unrealistic. In many cases, correctness of the protocols depends heavily on the assumption of a known number of failures.

So, we are left with the core problem: *can we develop deductive temporal techniques for the verification of parameterised systems where a **finite**, but unknown, number of failures can occur?* This question is exactly what we address here.

We proceed as follows. Section 2 gives a brief review of *first-order temporal logic* (FOTL) and its properties. In Section 3, we propose two mechanisms for adapting deductive techniques for FOTL to the problem of finite numbers of failures in infinite-state systems, and in Section 4 we outline a case study. Finally, in Section 5, we provide concluding remarks.

2 Monodic First-Order Temporal Logics

First-order (linear time) temporal logic (FOTL) is a very powerful and expressive formalism in which the specification of many algorithms, protocols and computational systems can be given at a natural level of abstraction [20]. Unfortunately, this power also means that, over many natural time flows, this logic is highly undecidable (not even recursively enumerable). Even with incomplete proof systems, or with proof systems complete only for restricted fragments, FOTL is interesting for the case of parameterised verification: one proof may certify correctness of an algorithm for infinitely many possible inputs, or correctness of a system with infinitely many states.

FOTL is an extension of classical first-order logic by temporal operators for a discrete linear model of time (isomorphic to $\mathbb{N}$, being the most commonly used model of time). Formulae of this logic are interpreted over structures that associate with each element n of $\mathbb{N}$, representing a moment in time, a first-order structure $\mathfrak{M}_n = (D, I_n)$ with the same non-empty domain D.

The *truth* relation $\mathfrak{M}_n \models^{\mathfrak{a}} \phi$ in the structure $\mathfrak{M}$ and a variable assignment $\mathfrak{a}$ is defined inductively in the usual way under for the following (sample) temporal operators:

$\mathfrak{M}_n \models^{\mathfrak{a}} \bigcirc\phi$ 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 $n \leq i < m$ $\mathfrak{M}_i \models \phi$;
$\mathfrak{M}_n \models^{\mathfrak{a}} \bullet\phi$ iff $n > 0$ and $\mathfrak{M}_{n-1} \models^{\mathfrak{a}} \phi$;
$\mathfrak{M}_n \models^{\mathfrak{a}} \blacksquare\phi$ iff for all $m \leq n$,$\mathfrak{M}_m \models^{\mathfrak{a}} \phi$;
$\mathfrak{M}_n \models^{\mathfrak{a}} \blacklozenge\phi$ iff there exists $0 \leq m < n$ such that $\mathfrak{M}_m \models^{\mathfrak{a}} \phi$;
$\mathfrak{M}_n \models^{\mathfrak{a}} \phi \,\mathcal{S}\, \psi$ iff there exists $m \leq n$ such that $\mathfrak{M}_m \models^{\mathfrak{a}} \psi$ and for all $m < i \leq n$ $\mathfrak{M}_i \models \phi$.

The non-temporal aspects have semantics as follows:

$\mathfrak{M}_n \models^{\mathfrak{a}} \top$
$\mathfrak{M}_n \models^{\mathfrak{a}} P(t_1, \ldots, t_n)$ iff $(I_n(\mathfrak{a}(t_1)), \ldots, I_n(\mathfrak{a}(t_n))) \in I_n(P)$
$\mathfrak{M}_n \models^{\mathfrak{a}} \neg\varphi$ iff not $\mathfrak{M}_n \models^{\mathfrak{a}} \varphi$
$\mathfrak{M}_n \models^{\mathfrak{a}} \varphi \vee \psi$ iff $\mathfrak{M}_n \models^{\mathfrak{a}} \varphi$ or $\mathfrak{M}_n \models^{\mathfrak{a}} \psi$
$\mathfrak{M}_n \models^{\mathfrak{a}} \exists x\varphi$ iff $\mathfrak{M}_n \models^{\mathfrak{b}} \varphi$ for some assignment $\mathfrak{b}$ that may differ from $\mathfrak{a}$ only in x and such that $\mathfrak{b}(x) \in D$
$\mathfrak{M}_n \models^{\mathfrak{a}} \forall x\varphi$ iff $\mathfrak{M}_n \models^{\mathfrak{b}} \varphi$ for every assignment $\mathfrak{b}$ that may differ from $\mathfrak{a}$ only in x and such that $\mathfrak{b}(x) \in D$

$\mathfrak{M}$ is a *model* for a formula ϕ (or ϕ is *true* in $\mathfrak{M}$) if there exists an assignment $\mathfrak{a}$ such that $\mathfrak{M}_0 \models^{\mathfrak{a}} \phi$. A formula is *satisfiable* if it has a model. A formula is *valid* if it is satisfiable in any temporal structure under any assignment. The set of valid formulae of this logic is not recursively enumerable. Thus, there was a need for an approach that could tackle the temporal verification of parameterised systems in a *complete* and *decidable* way. This was achieved for a wide class of parameterised systems using *monodic temporal logic* [15].

Definition 1. *A* FOTL *formula is said to be* **monodic** *if, and only if, any subformula with its main connective being a temporal operator has at most one free variable.*

Thus, ϕ is called *monodic* if any subformula of ϕ of the form $\bigcirc\psi$, $\Box\psi$, $\Diamond\psi$, $\blacklozenge\psi$, etc., contains at most one free variable. For example, the formulae $\forall x.\ \Box\exists y.\ P(x, y)$ and $\forall x.\ \Box P(x, c)$ are monodic, while $\forall x, y.\ (P(x, y) \Rightarrow \Box P(x, y))$ is *not* monodic.

The monodic fragment of FOTL has appealing properties: it is axiomatisable [24] and many of its sub-fragments, such as the two-variable or monadic cases, are decidable. This fragment has a wide range of applications, for example in spatio-temporal logics [13] and temporal description logics [3]. A practical approach to proving monodic temporal formulae is to use *fine-grained temporal resolution* [17], which has been implemented in the theorem prover TeMP [16]. It was also used for deductive verification of parameterised systems [12]. One can see that in many cases temporal specifications fit into the even narrower, and decidable, monodic *monadic* fragment. (A formula is monadic if all its predicates are *unary*.)

3 Incorporating Finiteness

When modelling parameterised systems in temporal logic, informally, elements of the domain correspond to processes, and predicates to states of such processes [12]. For example $idle(x)$ means that a process x is in the idle state, $\Diamond\forall y.\ agreement(y)$ means that, eventually, all processes will be in agreement, while $\exists z.\ \Box inactive(z)$ means that there is at least one process that is always inactive. (See [12] for further details.)

For many protocols, especially when fault tolerance is concerned, it is essential that the number of processes is finite. The straightforward generalisation to infinite numbers of processes makes many protocols incorrect. Although decidability of monodic fragments holds also for the case of semantics where only temporal structures over *finite*

domains are allowed [15], the proof is model-theoretic and no practical procedure is known.

We here examine two approaches that allow us to handle the problem of finiteness within temporal specification:

- first, in 3.1 we consider proof principles which can be used to establish correctness of some parameterised protocols;
- then in 3.2 we prove that, for a wide class of protocols, decision procedures that do not assume the finiteness of a domain can still be used.

3.1 Formalising Finiteness Principles

The language of FOTL is very powerful and one might ask if a form of finiteness can be defined inside the logic. We have found the following principles (which are valid over finite domains, though not in general) useful when analysing the proofs of correctness of various protocols and algorithms specified in FOTL (recall: $\blacklozenge\varphi$ means φ *was* true in the past):

Fin_1 (**deadline axiom**): $\Diamond(\forall x.\ (\Diamond P(x) \rightarrow \blacklozenge P(x)))$

Fin_2 (**finite clock axiom**): $[\forall x.\ \Box(P(x) \rightarrow \bigcirc \Box \neg P(x)] \Rightarrow [\Diamond \Box(\forall x.\ \neg P(x))]$

Fin_3 (**stabilisation axiom**):

$$[\Box(\forall x.\ (P(x) \rightarrow \bigcirc P(x))] \Rightarrow [\Diamond \Box(\forall x.\ (\bigcirc P(x) \rightarrow P(x))]$$

Actually the Fin_1 principle is a (more applicable) variant of the intuitively clearer principle $[\forall x.\ \Diamond P(x)] \Rightarrow [\Diamond \forall x.\ \blacklozenge P(x)]$ which is also valid over finite domains.

These principles have the following informal motivation. The deadline axiom principle, Fin_1, states that there is a moment after which "nothing new is possible"; that is, if, after the deadline, $P(x)$ becomes true for some domain element $a \in D$, there already was a moment in the past such that $P(x)$ was true on a at that moment. The final clock axiom, Fin_2, states that if the moments when the predicate $P(x)$ becomes true on some domain element are interpreted as clock ticks, the clock will eventually stop ticking. Finally, the stabilisation principle, Fin_3, states that if some domain area, where $P(x)$ is true, is growing then it will stop growing at some point. It can be easily seen that all these principles hold true in arbitrary finite domain structures.

Now, consider Fin_i for $i = 1, 2, 3$ as axiom schemes which can be added to a *reasonable* axiomatisation of FOTL (call this, Ax_{FOTL}) in order to capture, at least partially, "finite reasoning". By a 'reasonable' Ax_{FOTL}, we mean that we assume some Hilbert-style finitary axiomatic system for FOTL extending a standard, non-temporal, predicate logic axiomatisation by, at least, the axiom schemata presented in Fig. 1.

We show that all these three principles are actually equivalent modulo any reasonable Ax_{FOTL} (i.e. they can be mutually derived). The principle (an axiom scheme) F_1 is said to be derivable from F_2 if, for every instance α of F_1, we have $Ax_{\mathsf{FOTL}} + F_2 \vdash \alpha$. We will denote it simply $Ax_{\mathsf{FOTL}} + F_2 \vdash F_1$.

Theorem 1 *The principles Fin_1, Fin_2 and Fin_3 are mutually derivable.*

Future time axioms:

F0. $\vdash \Box\varphi \rightarrow \varphi$
F1. $\vdash \bigcirc\neg\varphi \leftrightarrow \neg\bigcirc\varphi$
F2. $\vdash \bigcirc(\varphi \rightarrow \psi) \rightarrow (\bigcirc\varphi \rightarrow \bigcirc\psi)$
F3. $\vdash \Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)$
F4. $\vdash \Box\varphi \rightarrow \Box\bigcirc\varphi$
F5. $\vdash (\varphi \rightarrow \bigcirc\varphi) \rightarrow (\varphi \rightarrow \Box\varphi)$
F6. $\vdash (\varphi \,\mathsf{U}\, \psi) \leftrightarrow \psi \vee (\varphi \wedge \bigcirc(\varphi \,\mathsf{U}\, \psi))$
F7. $\vdash (\varphi \,\mathsf{U}\, \psi) \rightarrow \Diamond\psi$

Past time axioms:

P1. $\vdash \neg\bullet\neg\varphi \rightarrow \bullet\varphi$
P2. $\vdash \bullet(\varphi \rightarrow \psi) \rightarrow (\bullet\varphi \rightarrow \bullet\psi)$
P3. $\vdash \varphi \,\mathcal{S}\, \psi \leftrightarrow \psi \vee (\varphi \wedge \neg\bullet\neg(\varphi \,\mathcal{S}\, \psi))$
P4. $\vdash \bullet$ **false**

Mixed axiom:

M8. $\vdash \varphi \rightarrow \bigcirc\bullet\varphi$

Interaction axioms:

I1. $\vdash \forall x.\, (\bigcirc\varphi(x)) \rightarrow \bigcirc(\forall x.\, \phi(x))$
I2. $\vdash \forall x.\, (\bullet\varphi(x)) \rightarrow \bullet(\forall x.\, \phi(x))$

Fig. 1. Ax_{FOTL}: Axioms of FOTL (all except the Interaction Axioms are taken from [18])

Proof

1. $Ax_{\mathsf{FOTL}} + Fin_1 \vdash Fin_2$.

   Assume $\forall x.\, \Box(P(x) \rightarrow \bigcirc \Box \neg P(x))$ $(*)$, which is the assumption of Fin_2.

   Consider then Fin_1 which is $\Diamond(\forall x.(P(x) \vee \Diamond P(x) \rightarrow \blacklozenge P(x)))$.

   In Fin_1 assume $\Diamond P(c)$ for an arbitrary c inside of $\Diamond(\ldots)$, then we have $\blacklozenge P(c)$ which together with $(*)$ gives $\neg P(c)$ and contradiction.

   That means, we have $\Diamond(\forall x.\neg(P(x) \vee \Diamond P(x)))$ which implies $\Diamond \Box(\forall x.\neg P(x))$.

2. $Ax_{\mathsf{FOTL}} + Fin_2 \vdash Fin_3$.

   Define $Q(x)$ to be $\neg P(x) \wedge \bigcirc P(x)$.

   Assume $\Box(\forall x.(P(x) \rightarrow \bigcirc P(x)))$ $(**)$.

   Then we have $\forall x.\, \Box(Q(x) \rightarrow \bigcirc \Box \neg Q(x))$ from the definition of $Q(x)$ and $(**)$.

   Applying Fin_2 we get $\Diamond \Box(\forall x.\, \neg Q(x))$ which is equivalent to $\Diamond \Box(\forall x.\, \neg \bigcirc P(x) \vee P(x))$ and to $\Diamond \Box(\forall x.\, (\bigcirc P(x) \rightarrow P(x)))$.

3. $Ax_{\mathsf{FOTL}} + Fin_3 \vdash Fin_1$.

   Applying to a valid formula, provable in Ax_{FOTL}, $\forall x(\blacklozenge P(x) \rightarrow \bigcirc \blacklozenge P(x))$ the principle Fin_3 we get

   $$\Diamond \Box(\forall x.(\bigcirc \blacklozenge P(x) \rightarrow \blacklozenge P(x)))\,.$$

   This implies [18] $\Diamond \Box(\forall x.\, \neg(\Diamond P(x)) \wedge \neg \blacklozenge P(x)))$.

   After propositionally equivalent transformations we get $\Diamond \Box(\forall x.\, (\Diamond P(x)) \rightarrow \blacklozenge P(x))$, which is Fin_1.

This theorem shows that all three principles are equivalent and so can be used interchangeably in the proofs. However, the differing syntactical forms may make some principles more suitable for *natural* proofs, yet may affect the efficiency of the automated proof search using these principles.

3.2 Eventually Stable Protocols

In Section 3.1 we highlighted some deduction principles capturing the finiteness of the domain. Alternatively, we can consider a family of protocols which terminate after a certain (but unknown) number of steps. For example, if every process sends only a finite number of messages, such protocol will eventually terminate. Consensus protocols [19], distributed commit protocols [6], and some other protocols fit into this class. Temporal models of specifications of such terminating protocols will eventually stabilise, that is, the interpretations I_n will be the same for sufficiently large n. We show that for these *eventually stable* specifications satisfiability over finite domains coincides with satisfiability over arbitrary domains.

Let $\mathcal{P}$ be a set of unary predicates. The *stabilisation principle w.r.t.* $\mathcal{P}$ is the formula:

$$\mathsf{Stab}_{\mathcal{P}} = \Box(\forall x \bigwedge_{P \in \mathcal{P}} [P(x) \equiv \bigcirc P(x)]).$$

Informally, if $\mathsf{Stab}_{\mathcal{P}}$ is true at some moment of time, from this moment the interpretation of predicates in $\mathcal{P}$ does not change. Let ϕ be a monodic temporal formula. Let $\mathcal{P}$ be the set of unary predicates occurring in ϕ. Then the formula

$$\phi_{\mathsf{Stab}} = \phi \wedge \Diamond\mathsf{Stab}$$

is called an *eventually stable formula*. We formulate the following proposition for monodic monadic formulae; it can be extended to other monodic classes obtained by *temporalisation by renaming* [8] of first-order classes with the finite model property.

Proposition 1. *Let ϕ be a monodic monadic formula. The eventually stable formula ϕ_{Stab} is satisfiable in a model with a finite domain if, and only if, ϕ_{Stab} is satisfiable in a model with an arbitrary domain.*

This proposition implies that if a protocol is such that it can be faithfully represented by an eventually stable formula, correctness of such protocol can be established by a procedure that does *not* assume the finiteness of the domain.

Proof For simplicity, we prove the proposition for formulae in Divided Separated Normal Form (DSNF) [8] only. The proof can be extended to the general case by the consideration of sub-formulae of ϕ.

A *monodic temporal problem P in divided separated normal form (DSNF)* is a quadruple $\langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$, where:

1. the universal part $\mathcal{U}$ and the initial part $\mathcal{I}$ are finite sets of first-order formulae;
2. the step part $\mathcal{S}$ is a finite set of clauses of the form $p \Rightarrow \bigcirc q$, where p and q are propositions, and $P(x) \Rightarrow \bigcirc Q(x)$, where P and Q are unary predicate symbols and x is a variable; and
3. the eventuality part $\mathcal{E}$ is a finite set of formulae of the form $\Diamond L(x)$ (a *non-ground* eventuality clause) and $\Diamond l$ (a *ground eventuality* clause), where l is a propositional literal and $L(x)$ is a unary non-ground literal with variable x as its only argument.

With each monodic temporal problem $\langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ we associate the FOTL formula $\mathcal{I} \wedge \Box\mathcal{U} \wedge \Box\forall x\mathcal{S} \wedge \Box\forall x\mathcal{E}$. When we talk about particular properties of a temporal problem (e.g., satisfiability, validity, logical consequences, etc) we refer to properties of this associated formula. Every monodic temporal formula can be transformed into divided separated normal form (DSNF) in a satisfiability equivalence preserving way with only linear growth in size [17].

Let $\mathsf{P} = \langle \mathcal{U}, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ be a monodic temporal problem in DSNF. We only have to show that if $\mathsf{P}_{\mathsf{Stab}}$ has a model, $\mathfrak{M} = \mathfrak{M}_0, \mathfrak{M}_1, \ldots$, with an infinite domain, it also has a model with a finite one. Let N be such that $\mathfrak{M}_N \models \mathsf{Stab}$. Consider now the temporal structure $\mathfrak{M}' = \mathfrak{M}_0, \mathfrak{M}_1, \ldots \mathfrak{M}_{N-1}, \mathfrak{M}_N, \mathfrak{M}_N, \mathfrak{M}_N, \ldots$ (i.e. from moment N the structure does not change). It can be seen that $\mathfrak{M}'$ is a model for P.

For every predicate, P, occurring in P, we introduce $N+1$ new predicates $P^0, P^1, \ldots, P^N$ of the same arity. Let ϕ be a first-order formula in the language of P. We denote by $[\phi]^i$, $0 \leq i \leq N$, the result of substitution of all occurrences of predicates in ϕ with their i-th counterparts; (e.g., $P(x_1, x_2)$ is replaced with $P^i(x_1, x_2)$).

– Let $\phi_{\mathcal{I}} = \bigwedge\{[\phi]^0 \mid \phi \text{ is in } \mathcal{I}\}$
– Let $\phi_{\mathcal{U}} = \bigwedge\{\bigwedge_{i=0}^{N} [\phi]^i \mid \phi \text{ is in } \mathcal{U}\}$.
– Let $\phi_{\mathcal{S}} = \bigwedge\{\bigwedge_{i=0}^{N-1} \big(\forall x(P^i(x) \Rightarrow Q^{i+1}(x)\big) \mid P(x) \Rightarrow \bigcirc Q(x) \text{ is in } \mathcal{S}\} \wedge \bigwedge\{\forall x(P^N(x) \Rightarrow Q^N(x) \mid P(x) \Rightarrow \bigcirc Q(x) \text{ is in } \mathcal{S}\}$
– Let $\phi_{\mathcal{E}} = \bigwedge\{[\forall x L(x)]^N \mid L(x) \text{ is in } \mathcal{E}\}$

Let $\phi_{FO} = \phi_{\mathcal{I}} \wedge \phi_{\mathcal{U}} \wedge \phi_{\mathcal{S}} \wedge \phi_{\mathcal{E}}$. Note that ϕ_{FO} does not contain any temporal operators. Consider now a first-order structure $\mathfrak{N}$ with the same domain D as $\mathfrak{M}$, interpreting constants in the same way, and such that $\mathfrak{N} \models P^i(a_1, \ldots, a_n)$, for some $a_1, \ldots, a_n \in D$, if, and only if, $\mathfrak{M}_i \models P(a_1, \ldots, a_n)$. It can be seen that that $\mathfrak{N} \models \phi_{FO}$. Since P is a monodic monadic problem, ϕ_{FO} is a monadic first-order formula, which has a model with a finite domain. Reversing the process, one can construct a model for P with a finite domain. ❑

4 Case Study: FloodSet Protocol

Next, we provide an example of how both methods described in Section 3 (explicit finiteness principles, and stabilisation principle for protocols with finite change) can be used for the proof of correctness of a protocol specified in monodic FOTL.

The setting is as follows. There are n processes, each having an *input bit* and an *output bit*. The processes work synchronously, run the same algorithm and use *broadcast* for communication. Any message sent by a non-faulty process is instantaneously delivered to all other processes. Some processes may fail and, from that point onward, such processes do not send any 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 processes. Crucially, there is a *finite* bound, f, on the number of processes that may fail.

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 non-faulty processes. It is required also that if all processes have the same input bit, that bit should be produced as an output bit.

This is a variant of *FloodSet algorithm with alternative decision rule* (in terms of [19], p.105) designed for solution of the Consensus problem in the presence of crash (or fail-stop) failures, and the basic elements of the protocol (adapted from [19][1]) are as follows.

– In the first round of computations, every process broadcasts its input bit.
– In every later round, a process broadcasts any value *the first time it sees it*.
– In every round the (tentative) output bit is set to the minimum value seen so far.

[1] In [19], every process *knows* the bound f in advance and stops the execution of the protocol after $f + 2$ rounds, producing the appropriate output bit. We consider the version where the processes do not know f in advance and produce a *tentative output bit* at every round.

The correctness criterion for this protocol is that, eventually (actually, no later than in $f + 2$ rounds) the output bits of all non-faulty processes will be the same.

Claim. The above FloodSet algorithm and its correctness conditions can be specified (naturally) within monodic monadic temporal logic without equality, and its correctness can be proved in monodic monadic temporal logic, using the above **finite clock axiom**.

We give a larger specification below, but first note the keys points concerning this:

1. Each process (s) must be categorised as one of the above types:
   $\Box(\forall x(Normal(x) \mid Failure(x) \mid Faulty(x)))$
   here the symbol $\mid$ means that exactly one of the predicates $Normal(x)$, $Failure(x)$, and $Faulty(x)$ is true.
2. If we see a '0' (the process has this already, or receives a message with this value) then we output '0':
   $\Box(\forall x(\neg Faulty(x) \wedge Seen(x, 0) \rightarrow \bigcirc Output(x) = 0))$
3. If we have not seen a '0' but *have* seen a '1', then we output '1':
   $\Box(\forall x.\, (\neg Faulty(x) \wedge \neg Seen(x, 0) \wedge Seen(x, 1) \rightarrow \bigcirc Output(x) = 1))$
4. The condition to be verified, namely that eventually all (non faulty) processes agree on the bit '0', or eventually all agree on the bit '1':

$$\Diamond((\forall x.\, \neg Faulty(x) \Rightarrow Output(x) = 0) \vee (\forall x.\, \neg Faulty(x) \Rightarrow Output(x) = 1))$$

We do not include the whole proof here, but will reproduce sample formulae to give the reader a flavour of the specification and proof.

4.1 Specification

A FOTL specification of the above $FloodSet$ algorithm φ is given as a conjunction of the following formulae, divided for convenience, into four groups as follows

1. **RULES:**
   $\Box(1stRound \rightarrow \bigcirc\neg 1stRound)$
   $\Box(\forall x.\, (Failure(x) \rightarrow \bigcirc Faulty(x)))$
   $\Box(\forall x.\, (1stRound \wedge Normal(x) \rightarrow \bigcirc(Send(x, Input(x)) \wedge Seen(x, Input(x)))))$
   $\Box(\forall x.\, (1stRound \wedge Failure(x) \rightarrow \bigcirc(Send\_Failure(x, Input(x)) \wedge Seen(x, Input(x)))))$
   $\Box(\forall x.\forall y.\, (\neg 1stRound \wedge Normal(x) \wedge Received(x, y) \wedge \neg Seen(x, y) \rightarrow \bigcirc Seen(x, y) \wedge Send(x, y)))$
   $\Box(\forall x.\forall y.\, (\neg 1stRound \wedge Failure(x) \wedge Received(x, y) \wedge \neg Seen(x, y) \rightarrow \bigcirc(Seen(x, y) \wedge Send\_Failure(x, y))))$
   $\Box(\forall x.\forall y.\, (Faulty(x) \rightarrow \bigcirc(\neg Send(x, y) \wedge \neg Send\_Failure(x, y))))$
   $\Box(\forall x.\forall y.\, (\neg 1stRound \wedge \neg Faulty(x) \wedge (\neg Received(x, y) \vee Seen(x, y)) \rightarrow \bigcirc(\neg Send(x, y) \wedge \neg Send\_Failure(x, y))))$
   $\Box(\forall x.\, (\neg Faulty(x) \wedge Seen(x, 0) \rightarrow \bigcirc Output(x) = 0))$
   $\Box(\forall x.\, (\neg Faulty(x) \wedge \neg Seen(x, 0) \wedge Seen(x, 1) \rightarrow \bigcirc Output(x) = 1))$

2. **FRAME CONDITIONS:**
   $\Box(\neg 1stRound \rightarrow \bigcirc\neg 1stRound)$

$\Box(\forall x.\,(Faulty(x) \rightarrow \bigcirc Faulty(x)))$
$\Box(\forall x.\,(\neg Faulty(x) \wedge \neg Failure(x) \rightarrow \bigcirc \neg Faulty(x)))$
$\Box(\forall x.\,\forall y.\,(Seen(x,y) \rightarrow \bigcirc Seen(x,y)))$

3. **CONSTRAINTS:**
$\Box(\forall x.\,\forall m.\,(Send(x,m) \rightarrow \forall y.\,Received(y,m)))$
$\Box(\forall x.\,\forall m.\,(Received(x,m) \rightarrow \exists y(Send(y,m) \vee Send\_Failure(y,m))))$
$\Box(\forall x.\,\forall m.\,\neg(Send(x,m) \wedge Send\_Failure(x,m)))$
$\Box(\forall x.\,(Normal(x) \mid Failure(x) \mid Faulty(x)))$

$\Box(\forall x.\,(Output(x) = 0 \vee Output(x) = 1))$
$\Box(\forall x.\,(Input(x) = 0 \vee Input(x) = 1))$
$\Box(\forall x.\,\forall y.\,(Send(x,y) \vee Received(x,y) \vee Seen(x,y)) \rightarrow (y = 0 \vee y = 1))$

4. **INITIAL CONDITIONS:**
$\Box($**start** $\Rightarrow 1stRound)$
$\Box($**start** $\Rightarrow \forall x.\ Normal(x))$
$\Box($**start** $\Rightarrow \forall x.\,\forall y.\,\neg Seen(x,y))$
$\Box($**start** $\Rightarrow \forall x.\ (Input(x) = 0 \vee Input(x) = 1))$

Note. One can get rid of all equalities in this example by using finiteness of the set of values, which are supposed to be second argument of $Seen(\_,\_)$, $Send(\_,\_)$ and $Send\_Failure(\_,\_)$.

Notice that the temporal specification uses, among others, the predicates $Normal(\_)$ to denote normal operating processes, $Failure(\_)$ to denote processes experiencing failure (at some point of time), $Faulty(\_)$ for the processes already failed. There are also predicates such as $Seen(\_,\_)$ specifying the effect of communications. Having these, it is straightforward to write down the temporal formulae describing the above protocol and correctness condition (i.e. (4) above). In the proof of correctness below, the **finite clock axiom** has to be instantiated to the $Failure(x)$ predicate (i.e. replace P by $Failure$ in Fin_2).

4.2 Refutation

In this section we will consider the actual proof concerning the correctness of the above specification with respect to the conditions we have presented. We will not present the full proof, but will provide an outline indicating how the major steps occur.

First of all, the clausal temporal resolution approach is a refutation procedure and so we add the negation of the required condition (i.e. $\neg\psi$) and attempt to derive a contradiction. We note that $\neg\psi$ is

$$\Box((\exists x \neg Faulty(x) \wedge Output(x) \neq 0)$$
$$\wedge$$
$$(\exists x \neg Faulty(x) \wedge Output(x) \neq 1))$$

We translate formulae such as '$Output(x) \neq 0$' to '$\neg\neg Output(x)$' since the only values allowed are '0' and '1'. Consequently the two temporal formulae derived from $\neg\psi$ are:

$C1$: $\Box(\exists x \neg Faulty(x) \wedge Output(x))$
$C2$: $\Box(\exists x \neg Faulty(x) \wedge \neg Output(x))$

Frame conditions and the finite clock axiom applied for the *Failure* predicate give

$C3$: $\Diamond\Box(\forall x. \neg Failure(x))$

From $C1$ and $C3$ we have

$C4$: $\Diamond\Box(\forall x. \neg Failure(x) \wedge \exists x(\neg Faulty(x) \wedge Output(x)))$

From $C4$ and constraints we now have

$C5$: $\Diamond\Box(\forall x. \neg Failure(x) \wedge \exists x(Normal(x) \wedge Output(x)))$

By rules concerning *Output* and $C5$ we get

$C6$: $\Diamond\Box(\exists x Normal(x) \wedge \bullet Seen(x, 0))$

Next, let us note a useful variant of the induction axiom called the *minimal element principle*:

$$\forall \bar{x}([\Diamond\varphi(\bar{x})] \rightarrow [\Diamond(\varphi(\bar{x}) \wedge \bullet\blacksquare\neg\varphi(\bar{x}))])$$

By the minimum element principle

$C7$: $\Diamond\Box(\exists x Normal(x) \wedge \blacklozenge(Seen(x, 0) \wedge \bullet\blacksquare\neg Seen(x, 0)))$

By rules from $C7$

$C8$: $\Diamond\Box(\exists x Normal(x) \wedge \blacklozenge(\bullet Received(x, 0)))$

By rules from $C8$

$C9$: $\Diamond\Box(\exists x Normal(x) \wedge \blacklozenge(\bullet(Received(x, 0) \wedge \neg Seen(x, 0))))$

By rules from $C9$

$C10$: $\Diamond\Box(\exists x Normal(x) \wedge \blacklozenge(Normal(x) \wedge Received(x, 0) \wedge \neg Seen(x, 0))))$

By rules from $C10$

$C11$: $\Diamond\Box(\exists x Normal(x) \wedge \blacklozenge(Send(x, 0)))$

By rules from $C11$

$C12$: $\Diamond\Box(\blacklozenge\forall x. Seen(x, 0)))$

From $C12$

$C13$: $\Diamond\Box(\forall x. Seen(x, 0))$

From $C2$ and rules

$C14$: $\Box(\exists y \neg Seen(y, 0))$

Finally, from $C13$ and $C14$ we get a contradiction.
□

4.3 Eventual Stabilisation of FloodSet Protocol

One may also verify the *FloodSet* protocol using the eventual stabilisation principle from Section 3.2. To establish the applicability of the principle one may use the following arguments: every process can broadcast at most twice, and taking into account finiteness of both the numbers of processes and of failures, one may conclude that eventually the protocol stabilises. Note that such an analysis only allows us to conclude that the protocol stabilises, but its properties still need to be proved. Let ϕ be a temporal specification of the protocol. Taking into account the stabilisation property, the protocol is correct iff $(\phi \wedge \neg\psi)_{\mathsf{Stab}}$ is not satisfiable over finite domains. By Proposition 1, there is no difference in satisfiability over finite and general domains for such formulae and so one may use theorem proving methods developed for monadic monodic temporal logics over general models to establish this fact. In this case, the proof follow(s) exactly the form of proof presented in the previous section, with the exception that statement $C3 : \Diamond\Box(\forall x.\, \neg Failure(x))$ is obtained in a different way. One of the conjuncts of the stabilisation principle with respect to $\phi \wedge \neg\psi$ is

$$\Diamond\Box(\forall x Failure(x) \equiv \bigcirc Failure(x)\,.$$

Together with the rule

$$\Box(\forall x.\, (Failure(x) \rightarrow \bigcirc Faulty(x)))$$

and the constraint

$$\Box(\forall x.\, (Normal(x) \mid Failure(x) \mid Faulty(x)))$$

this implies $C3$, as required.

5 Concluding Remarks

In this paper we have introduced two approaches for handling the finiteness of the domain in temporal reasoning.

The first approach uses explicit finiteness principles as axioms (or proof rules), and has potentially wider applicability, not being restricted to protocols with the stabilisation property. On the other hand, the automation of temporal proof search with finiteness principles appears to be more difficult and it is still largely an open problem.

In the approach based on the stabilisation principle, all "finiteness reasoning" is carried out at the meta-level and essentially this is used to reduce the problem formulated for finite domains to the general (not necessarily finite) case. When applicable, this method is more straightforward for implementation and potentially more efficient. Applicability, however, is restricted to the protocols which have stabilisation property (and this property should be demonstrated in advance as a pre-condition).

Finally, we briefly mention some future work. Automated proof techniques for monadic monodic FOTL have been developed [8, 17] and implemented in the TeMP system [16], yet currently proof search involving the finiteness principles requires improvement. Once this has been completed, larger case studies will be tackled. The techniques themselves would also benefit from extension involving probabilistic, real-time and equational reasoning.

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. 17th International Conference on Concurrency Theory (CONCUR)*, volume 4137 of *LNCS*, pages 95–109. Springer, 2006.
3. A. Artale, E. Franconi, F. Wolter, and M. Zakharyaschev. A Temporal Description Logic for Reasoning over Conceptual Schemas and Queries. In *Proc. European Conference on Logics in Artificial Intelligence (JELIA)*, volume 2424 of *LNCS*, pages 98–110. Springer, 2002.
4. N. Bjorner, A. Browne, E. Chang, M. Colon, A. Kapur, Z. Manna, H. B. Sipma, and T. E. Uribe. STeP: Deductive-Algorithmic Verification of Reactive and Real-time Systems. In *Proc. 8th International Conference on Computer-Aided Verification (CAV)*, vol. 1102 of *LNCS*, pages 415–418, Springer-Verlag, 1996.
5. M. Calder, and A. Miller. An Automatic Abstraction Technique for Verifying Featured, Parameterised Systems. *Theoretical Computer Science*, to appear.
6. 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.
7. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, Dec. 1999.
8. A. Degtyarev, M. Fisher, and B. Konev. Monodic Temporal Resolution. *ACM Transactions on Computational Logic*, 7(1):108–150, January 2006.
9. G. Delzanno. Constraint-based Verification of Parametrized Cache Coherence Protocols. *Formal Methods in System Design*, 23(3):257–301, 2003.
10. E. A. Emerson. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier, 1990.
11. J. Esparza, A. Finkel, and R. Mayr. On the Verification of Broadcast Protocols. In *Proc. 14th IEEE Symp. Logic in Computer Science (LICS)*, pages 352–359. IEEE CS Press, 1999.
12. M. Fisher, B. Konev, and A. Lisitsa. Practical Infinite-state Verification with Temporal Reasoning. In *Verification of Infinite State Systems and Security*, volume 1 of *NATO Security through Science Series: Information and Communication*. IOS Press, January 2006.
13. D. Gabelaia, R. Kontchakov, A. Kurucz, F. Wolter, and M. Zakharyaschev. On the Computational Complexity of Spatio-Temporal Logics. In *Proc. 16th International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 460–464. AAAI Press, 2003.
14. G. J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.
15. I. Hodkinson, F. Wolter, and M. Zakharyaschev. Decidable Fragments of First-order Temporal Logics. *Annals of Pure and Applied Logic*, 106:85–134, 2000.
16. U. Hustadt, B. Konev, A. Riazanov, and A. Voronkov. TeMP: A Temporal Monodic Prover. In *Proc. 2nd Second International Joint Conference on Automated Reasoning (IJCAR)*, volume 3097 of *LNAI*, pages 326–330. Springer, 2004.
17. B. Konev, A. Degtyarev, C. Dixon, M. Fisher, and U. Hustadt. Mechanising First-order Temporal Resolution. *Information and Computation*, 199(1-2):55–86, 2005.
18. O. Lichtenstein and A. Pnueli, Propositional Temporal Logics: Decidability and Completeness, International Journal of the IGPL, Vol. 8 No , 55–85.
19. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
20. Z. Manna and A. Pnueli. *Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.

21. S. Owre, J. Rushby, N. Shankar, F. von Henke. *Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS.* IEEE Transactions on Software Engineering, Volume 21 No 2, 107-122
22. C. Röckl. *Proving write invalidate cache coherence with bisimulations in Isabelle/HOL*. In Proc. of FBT 2000, 69-78, Shaker, 2000.
23. A. Pnueli and T. Arons. *TLPVS: A PVS-based LTL verification system.* In *Verification: Theory and Practice*, volume 2772 of *LNCS*, pages 598–625. Springer, 2003.
24. 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.