On and On the Temporal Way

Clare Dixon and Michael Fisher

Department of Computer Science
University of Liverpool, Liverpool, United Kingdom {cldixon,mfisher}@liv.ac.uk

Abstract

"Up and Down the Temporal Way" was a paper published by Howard Barringer in the 1980s that used temporal logics to formally specify a lift system. Based on that temporal specification, we describe some advances and extensions to temporal specification and verification that the authors have been involved with since then.

1 Introduction

In 1985, Howard produced a temporal logic specification for a *lift* system [2] which was subsequently published in [3]. Our paper is (loosely) inspired by Howard's specification [3] and we outline some of the advances, extensions and refinements within temporal specification and verification that we have been involved with over the last 25 years. Throughout, we will endeavour to use Howard's lift example to provide examples.

The lift specification given by Howard in [2, 3] began with a series of informal properties of lifts and then moved on to the temporal logic specification first of an individual lift, and then multiple lifts. Whilst verification of lift properties was not carried out, an informal justification of whether the specification met the (English) requirements was provided. Also the paper cited [30] as a route for proving properties from this specification, utilizing tableau calculi for a number of propositional temporal logics.

In this paper, we provide two specifications for a single lift, one with a single call button for each floor and the other with two call buttons for each floor (up and down). We also specify multiple lifts. We follow [3] as far as possible but do introduce some additional propositions that make the formulation easier for the contemporary provers we use. Specifically, we describe several temporal resolution calculi and their related implementations and use these to prove some of the properties for the lift specification.

This paper is structured as follows. To begin with, in Section 2 we provide a quick review of *Propositional (Linear) Temporal Logic* [29], recalling temporal operators such as '$\bigcirc$', '$\Box$', '$\Diamond$', '$\mathcal{U}$' ("until"), and '$\mathcal{W}$' ("unless"). We also describe a temporal normal form, Separated Normal Form (SNF) [21], on which our subsequent proof and execution techniques will be based. In Section 3 we recall some of the informal lift requirements from Howard's paper, for example

> "when a lift has no outstanding requests to service, it remains at the current floor with doors closed."

We next define the lift-related propositions that we will use as a basis for the specification. These are similar to the original paper, for example, where $i \in \{0 \ldots n\}$,

at_i is true if lift is at floor i and false otherwise;

d_i is true if the door at floor i is open and false otherwise.

So, at_0 is true if the lift is at the *ground floor* and d_0 is true if the lift doors are open on the *ground floor*, while at_n is true if the lift is at the *top floor* and d_n is false if the lift doors are closed on the *top floor*. Given these basic propositions, we present some of the structural constraints described purely in propositional logic. For example, for all i and j in $\{0 \ldots n\}$ if a lift is at some floor it cannot also be at another floor.

$$at_i \Rightarrow \bigwedge_{j \neq i} \neg at_j .$$

We also develop a *temporal logic* specification of the dynamic changes between such configurations.

Since we are considering movement of lifts, we also introduce the concept of a lift direction, simply via '*up*' and '*down*' (actually for the latter we will use '$\neg$*up*'). Then we can specify properties such as "initially, the lift is at the ground floor and the direction is *up*" via:

$$\textbf{start} \Rightarrow (at_0 \wedge up)$$

where '**start**' is a (nullary) connective that holds only at the beginning of time. Now we can describe various rules concerning lift behaviour. For example, let a proposition c_j denote that the lift is called to floor j via the external call button. Also assume that the proposition ca_i denotes a request for the lift above floor i, defined as pressing the external call buttons or the internal send buttons for a floor above floor i. We might specify

$$\Box(at_i \wedge ca_i \wedge up) \Rightarrow \bigcirc at_{i+1}$$

to denote that the lift should continue travelling upwards to service a request above floor i. Next we show how to extend the specification, in an obvious way, to a multi-lift system.

Finally, we highlight some of the the general properties of the lift specification, which are either explicitly added to the specification, or that we would like to prove *from* the specification. These are properties, such as "all requests must eventually be serviced":

$$\Box(c_j \Rightarrow \Diamond d_i) .$$

This leads to a set of properties that we would like to prove of the specification.

In Section 4 we discuss how such a specification could be executed using METATEM, a programming framework based on executable temporal logic. This provides a way to animate the specification, attempting to build a model for it. Section 5 describes a resolution calculus for PTL and an implementation of this, allowing us to apply a propositional prover to the lift specification and various properties. In Section 6 we describe two temporal calculi that allow constraints on temporal formulae as part of the input. These provide a much more concise specification and constrained proof mechanism. In Section 7 we extend the logic to a first-order setting and describe a resolution calculus for this logic. In Section 8 we provide further extensions to the basic specifications and present some concluding remarks.

2 Propositional Temporal Logic

Propositional Linear Time Temporal Logic (PTL) can be thought of as classical propositional logic extended with operators to deal with time [41, 29]. The future-time temporal connectives we use include '$\bigcirc$' (in the next moment) and '$\mathscr{U}$' (until). Formally, PTL formulae are constructed from the following elements:

- a set, PROP, of propositional symbols

- propositional connectives, **true**, $\neg, \vee$; and
- temporal connectives, $\bigcirc$, and $\mathcal{U}$.

The set of well-formed formulae (WFF) of PTL, is defined as the smallest set satisfying the following:

- any elements of PROP and **true** are in WFF;
- if φ and ψ are in WFF, then so are $\neg\varphi, \varphi \vee \psi, \bigcirc\varphi, \varphi \mathcal{U} \psi$.

A *literal* is defined as either a proposition symbol or the negation of a proposition symbol. We (ambiguously) assume that the negation of $\neg p$ is p.

A model for PTL formulae can be characterised as a *sequence of states*, σ, of the form $\sigma = s_0, s_1, s_2, s_3, \ldots$, where each state s_i is a set of propositional symbols representing those propositions, which are satisfied at the i^{th} moment in time. The notation $(\sigma, i) \models \varphi$ denotes the truth of formula φ in the model σ at the state of index $i \in \mathbb{N}$ and is defined as follows.

$$
\begin{array}{lll}
(\sigma,i) \models & \textbf{true} & \\
(\sigma,i) \models & p & \text{iff } p \in s_i \text{ where } p \in \text{PROP} \\
(\sigma,i) \models & \neg\varphi & \text{iff it is not the case that } (\sigma,i) \models \varphi \\
(\sigma,i) \models & \varphi \vee \psi & \text{iff } (\sigma,i) \models \varphi \text{ or } (\sigma,i) \models \psi \\
(\sigma,i) \models & \bigcirc\varphi & \text{iff } (\sigma,i+1) \models \varphi \\
(\sigma,i) \models & \varphi \mathcal{U} \psi & \text{iff } \exists k \in \mathbb{N}.\ k \geq i \text{ and } (\sigma,k) \models \psi \text{ and} \\
 & & \quad \forall j \in \mathbb{N}, \text{if } i \leqslant j < k \text{ then } (\sigma, j) \models \varphi
\end{array}
$$

Note we can obtain **false** and the other Boolean operators via the usual equivalences and we define '$\Box$' (always in the future), '$\Diamond$' (sometime in the future) and '$\mathcal{W}$' (unless or weak until) operators as follows.

$$
\begin{array}{rcl}
\Diamond\varphi & \equiv & \textbf{true}\, \mathcal{U}\, \varphi \\
\Box\varphi & \equiv & \neg\Diamond\neg\varphi \\
\varphi \mathcal{W} \psi & \equiv & (\varphi \mathcal{U} \psi) \vee (\Box\varphi)
\end{array}
$$

For any formula φ, model σ, and state index $i \in \mathbb{N}$, either $(\sigma, i) \models \varphi$ holds or $(\sigma, i) \models \varphi$ does not hold, denoted by $(\sigma, i) \not\models \varphi$. If there is some σ such that $(\sigma, 0) \models \varphi$, then φ is said to be *satisfiable*. If $(\sigma, 0) \models \varphi$ for all models, σ, then φ is said to be *valid* and is written $\models \varphi$. A set $\mathcal{N}$ of formulae is *satisfiable* in the model σ at the state of index $i \in \mathbb{N}$ if, and only if, for all $\varphi \in \mathcal{N}, (\sigma, i) \models \varphi$. A formula of the form $\Diamond\varphi$ or $\psi \mathcal{U} \varphi$ is called an *eventuality*.

2.1 Normal Form

It is often convenient to operate on formulae represented in a normal form. Separated Normal Form (SNF) was first introduced for PTL in [20] and also discussed in [25]. To assist in the definition of the normal form we introduce a further (nullary) connective '**start**' that holds only at the beginning of time, i.e.,

$$(\sigma, i) \models \textbf{start} \quad \text{iff} \quad i = 0.$$

This allows the general form of the (temporal clauses of the) normal form to be implications. In the following, small Latin letters, k_i, l_j, m represent literals in the language PROP. A normal form for PTL is of the form

$$\Box \bigwedge_h X_h$$

where each X_h is an *initial*, *step*, or *sometime* clause (respectively) as follows:

$$\begin{array}{lll} \mathbf{start} \Rightarrow & \bigvee_i l_i & \textit{(initial)} \\ \bigwedge_i k_i \Rightarrow & \bigcirc \bigvee_j l_j & \textit{(step)} \\ \bigwedge_i k_i \Rightarrow & \Diamond m & \textit{(sometime)} \end{array}$$

We can translate any PTL formula φ into a formula φ' such that φ is satisfiable if and only if φ' is satisfiable.

Theorem 1. *[25] Any PTL formula can be transformed into an equi-satisfiable PTL formula in SNF with at most a linear increase in the size of the formula.*

The translation uses standard equivalences from propositional and temporal logic, renames complex subformulae using new propositions, linking the new propositions with the satisfaction of the renamed subformula everywhere in the model and unwinding temporal operators into formulae to be satisfied now, and in the next moment in time, using their fixpoint definitions.

When specifying the behaviour of systems, it is sometime convenient to consider 'traditional' clauses of the form

$$\bigvee_i l_i \qquad \textit{(global)}$$

Every global clause can, if necessary, be represented as a combination of an initial and a step clause:

$$\mathbf{start} \Rightarrow \bigvee_i l_i \qquad \text{and} \qquad \mathbf{true} \Rightarrow \bigcirc \bigvee_i l_i$$

To save space we will sometimes use global clauses in Section 3 or formulae that can easily be translated into global clauses. We also assume clauses are kept in their simplest form by performing classical style simplification; for example that

$$p \Rightarrow \bigcirc(q \vee q \vee r)$$

would always be re-written as

$$p \Rightarrow \bigcirc(q \vee r).$$

3 Lift Specification

Next we examine the lift specification [3] and, based on this, provide a specification for several lift systems. The following are the rules for the lift system as stated in [3].

L1 Each lift has a set of buttons, one button for each floor. These illuminate when pressed and cause the lift to visit the corresponding floor. The illumination is cancelled when the corresponding floor is visited (i.e. stopped at) by the lift.

L2 Each floor has two buttons (except the ground floor and top floor), one an "up request" and one a "down request". These buttons illuminate when pressed. The buttons are cancelled when a lift visits the floor and is either travelling in the desired direction, or visiting the floor with no requests outstanding. In the latter case, if both floor-request buttons are illuminated only one should be cancelled. The algorithm to decide which to service should minimise the waiting time for both requests.

L3 When a lift has no request to service, it should remain at its final destination with its doors closed and await further requests.

L4 All requests for lifts from a floor must be serviced eventually, with all floors given equal priority.

L5 All requests for floors within lifts must be serviced eventually, with floors being serviced sequentially in the direction of travel.

L6 Each lift has an emergency button which, when pressed, causes a warning signal to be sent to the site manager. the lift is then deemed to be 'out of service'. Each lift has a mechanism to cancel its 'out of service' status.

To save space we do not further consider emergency behaviour, (L6), but the lift specifications we have developed can be extended if necessary.

3.1 Simple Single Lift

First we specify a single lift with a single external call button (denoting both a call up and a call down) and send buttons within the lift. This aims to specify (L1)–(L5) ignoring (L6) where (L2) is modified as there is only one call button on each floor. We assume that the ground floor is floor 0 and the top floor is floor n. We use the following propositions similar to those proposed in [3]. In each case, assume $0 \leqslant i \leqslant n$ unless it is stated otherwise.

at_i the lift is at floor i;

d_i the lift door is open (and $\neg d_i$ the lift door is closed);

c_i the external call button at floor i is pressed;

s_i the internal button to send the lift to floor i is pressed;

lc_i the external call light at floor i is lit;

ls_i the internal send light at floor i is lit;

ca_i the lift has been called (or sent) from a floor above i (for all floors $0 \leqslant i < n$);

cb_i the lift has been called (or sent) from a floor below i (for all floors $1 < i \leqslant n$);

up the lift movement is up (and $\neg up$ is the lift movement is down).

Note we add the propositions at_i and up which were not originally present in [3] but which make dealing with servicing requests easier and ca_i and cb_i, which are abbreviations for disjunctions of lc_j and ls_j where $j > i$ or $j < i$ respectively. In the subsequent text we relate formulae to those in [3]. To try avoid any confusion when referring to formulae from [3] we denote a formula (j) from [3] as (Bj).

Buttons and Lights

In [3] the following formulae labelled (B1).

$$\Box \bigwedge_{i=0}^{n} (c_i \Rightarrow (lc_i \mathcal{W} d_i))$$

and (B3)

$$\Box \bigwedge_{i=0}^{n} (\neg lc_i \Rightarrow (\neg lc_i \mathcal{W} c_i))$$

relate the calling of the lift with the lighting of the call buttons and the doors opening. This is partially to model L2. Note that we also assume that

$$\Box \bigwedge_{i=0}^{n} (lc_i \Rightarrow (lc_i \mathcal{W} d_i)).$$

From this we obtain the following formulae for all floors $0 \leqslant i \leqslant n$.

$$\Box(c_i \Rightarrow (lc_i \vee d_i)) \quad (1)$$
$$\Box(lc_i \Rightarrow \bigcirc(lc_i \vee d_i)) \quad (2)$$
$$\Box(\neg lc_i \Rightarrow \bigcirc(\neg lc_i \vee c_i)) \quad (3)$$

Similar formulae exist for the send button within the lift and its related light to model L1. In [3] the following formula labelled (B2)

$$\Box \bigwedge_{i=0}^{n} (s_i \Rightarrow (ls_i \mathcal{W} d_i))$$

and (B4)

$$\Box \bigwedge_{i=0}^{n} (\neg ls_i \Rightarrow (\neg ls_i \mathcal{W} s_i))$$

relates the sending of the lift via the internal buttons with the lighting of the send button within the lift to the lift doors opening. Note we also assume that

$$\Box \bigwedge_{i=0}^{n} (ls_i \Rightarrow (ls_i \mathcal{W} d_i)).$$

From this we obtain the following formulae for all floors $0 \leqslant i \leqslant n$.

$$\Box(s_i \Rightarrow (ls_i \vee d_i)) \quad (4)$$
$$\Box(ls_i \Rightarrow \bigcirc(ls_i \vee d_i)) \quad (5)$$
$$\Box(\neg ls_i \Rightarrow \bigcirc(\neg ls_i \vee s_i)) \quad (6)$$

From [3] formula (B5) we have

$$\Box \bigwedge_{i=0}^{n} ((lc_i \Rightarrow \neg d_i) \wedge (ls_i \Rightarrow \neg d_i))$$

i.e. if the call or send lights are on then the doors must be closed, and we obtain for all floors $0 \leqslant i \leqslant n$

$$\Box(lc_i \Rightarrow \neg d_i) \quad (7)$$
$$\Box(ls_i \Rightarrow \neg d_i) \quad (8)$$

Lift Door Behaviour

In [3] formulae (B7) are provided to ensure that at most one door is open at any moment.

$$\Box \left(\bigwedge_{i=0}^{n} \neg d_i \right) \vee \bigvee_{i=0}^{n} \left(d_i \wedge \bigwedge_{j \neq i} \neg d_j \right)$$

As a side effect of how we deal with lift servicing behaviour we introduce a proposition at_i to represent that the lift is at floor i and ensure that this holds for exactly one floor i (see (26) and (27)). In addition we relate d_i to at_i (see (22)) so we can try to prove the above holds rather than stating it.

Servicing behaviour

This is to deal with movement between floors, service requests, etc, as specified in L3, L4 and L5. In [3] the constraints required are as follows.

1. All requests must eventually be serviced, and no floor should be stopped at when there is not unserviced request for that floor.
2. The lift should not pass a floor for which there is an outstanding unserviced request.
3. The lift should not change direction if there is an outstanding request whose service would cause the lift to continue in the same direction.

The following formulae aim to keep the lift moving in the same direction as long as there is a service request still to be satisfied in that direction. The following are clauses for all floors $0 < i < n$.

$$\Box(at_i \wedge ca_i \wedge up \Rightarrow \bigcirc(at_{i+1} \wedge up)) \quad (9)$$
$$\Box(at_i \wedge cb_i \wedge \neg up \Rightarrow \bigcirc(at_{i-1} \wedge \neg up)) \quad (10)$$
$$\Box(at_i \wedge \neg ca_i \wedge cb_i \wedge up \Rightarrow \bigcirc(at_{i-1} \wedge \neg up)) \quad (11)$$
$$\Box(at_i \wedge ca_i \wedge \neg cb_i \wedge \neg up \Rightarrow \bigcirc(at_{i+1} \wedge up)) \quad (12)$$
$$\Box(at_i \wedge \neg ca_i \wedge \neg cb_i \Rightarrow \bigcirc at_i) \quad (13)$$

Note some of these clauses are modified for at_0 and at_n because there cannot be a call from below if the lift is at floor 0 and there cannot be a call from above if the lift is at floor n. We first provide the formulae for at_0.

$$\Box(at_0 \wedge ca_0 \Rightarrow \bigcirc(at_1 \wedge up)) \quad (14)$$
$$\Box(at_0 \wedge \neg ca_0 \Rightarrow \bigcirc at_0) \quad (15)$$

The following are the formulae for at_n.

$$\Box(at_n \wedge cb_n \Rightarrow \bigcirc(at_{n-1} \wedge \neg up)) \quad (16)$$
$$\Box(at_n \wedge \neg cb_n \Rightarrow \bigcirc at_n) \quad (17)$$

The following formulae define the propositions ca_i (call from above) for $i = n-1$.

$$\Box(ca_{n-1} \iff lc_n \vee ls_n) \quad (18)$$

and where for $0 \leqslant i < n-1$

$$\Box(ca_i \iff ca_{i+1} \vee lc_{i+1} \vee ls_{i+1}) \quad (19)$$

The following defines cb_i (call from below) for $i = 1$

$$\Box(ca_1 \iff lc_0 \vee ls_0) \tag{20}$$

and for for all floors $1 < i \leqslant n$

$$\Box(cb_i \iff cb_{i-1} \vee lc_{i-1} \vee ls_{i-1}) \tag{21}$$

Clearly $\Box\neg cb_0$ and $\Box\neg ca_n$. Now, the following states that for a door to be open the lift must be at that floor and it should only open if either the external call light or the internal send light for that floor was lit at the last moment or either the external call button has been pressed or the internal send button has been pressed.

$$\Box(d_i \Rightarrow at_i) \tag{22}$$

$$\Box(\neg lc_i \wedge \neg ls_i \Rightarrow \bigcirc(\neg d_i \vee c_i \vee s_i)) \tag{23}$$

The following ensures that if the lift has been called or sent to a floor (i.e. the call or send lights are on) then in the next moment if the lift is at that floor then the lift door is open, for all floors $0 \leqslant i \leqslant n$

$$\Box(lc_i \Rightarrow \bigcirc(d_i \vee \neg at_i)) \tag{24}$$

$$\Box(ls_i \Rightarrow \bigcirc(d_i \vee \neg at_i)) \tag{25}$$

The lift must be at exactly one floor at any moment

$$\Box(\bigvee_{i=0}^{n} at_i) \tag{26}$$

and for each $0 \leqslant i \leqslant n$, $0 \leqslant j \leqslant n$, $i \neq j$

$$\Box(\neg at_i \vee \neg at_j) \tag{27}$$

Actions

In [3] various actions are specified to describe the pushing of the call and send buttons and the opening and closing of doors. These are described as not being essential so we do not specify these as they seem to be forcing certain behaviour on the environment.

Initial conditions

Initially the lift doors are closed, there are no call or send requests and none of the lift buttons are lit.

$$\textbf{start} \Rightarrow \bigwedge_{i=0}^{n}(\neg d_i \wedge \neg c_i \wedge \neg s_i \wedge \neg lc_i \wedge \neg ls_i) \tag{28}$$

3.2 Single Lift with Up and Down Call Buttons

Following [3] we replace the single call button and lights by a "call up" button and light and a "call down" button and light. In particular c_i is replaced by cu_i and cd_i and lc_i is replaced by lcu_i and lcd_i. Note that there will be no call down button on floor 0 and no call up button on floor n.

cu_i the external call up button at floor i is pressed (for all floors $0 \leqslant i < n$);

cd_i the external call down button at floor i is pressed (for all floors $1 < i \leqslant n$);

lcu_i the external call up light at floor i is lit (for all floors $0 \leqslant i < n$);

lcd_i the external call down light at floor i is lit (for all floors $1 < i \leqslant n$);

The following formula define the propositions ca_i (call from above) for $i = n-1$

$$\Box(ca_{n-1} \iff (ls_n \vee lcd_n) \tag{29}$$

and where for $i = 0, \ldots n-2$

$$\Box(ca_i \iff (ca_{i+1} \vee lcu_{i+1} \vee ls_{i+1} \vee lcd_{i+1})) \tag{30}$$

The following defines cb_i (call from below) for $i = 1$

$$\Box(cb_1 \iff lcu_0 \vee ls_0) \tag{31}$$

and for $i = 2, \ldots n$

$$\Box(cb_i \iff cb_{i-1} \vee lcu_{i-1} \vee lcd_{i-1} \vee ls_{i-1}) \tag{32}$$

The following ensures that the lift door opens at floor i if the call up (respectively down) button was lit in the previous moment in time, the lift is at floor i and is travelling up (respectively down) or it was travelling down (respectively up) and there were no calls from below (respectively from above). For $0 < i < n$ we have the following

$$\Box(lcu_i \Rightarrow \bigcirc(d_i \vee \neg at_i \vee \neg up)) \tag{33}$$
$$\Box(lcu_i \Rightarrow \bigcirc(d_i \vee \neg at_i \vee up \vee cb_i)) \tag{34}$$
$$\Box(lcd_i \Rightarrow \bigcirc(d_i \vee \neg at_i \vee up)) \tag{35}$$
$$\Box(lcd_i \Rightarrow \bigcirc(d_i \vee \neg at_i \vee \neg up \vee ca_i)) \tag{36}$$

and additionally we have the following formulae for the top and bottom floors.

$$\Box(lcu_0 \Rightarrow \bigcirc(d_0 \vee \neg at_0)) \tag{37}$$
$$\Box(lcd_n \Rightarrow \bigcirc(d_n \vee \neg at_n) \tag{38}$$

In [3] the four unless formulae and formulae (7) and (8) in Section 3.1, Buttons and Lights, are replaced by the following formulae using the above and a notion of *up service at floor i*. The notion of *up service at floor i* is described as

- the lift must be at floor i;
- there must be a request to go up;
- if it is the case that the lift came from above (in the down direction) then
  - it must not be the case that the lift is already servicing a down request at floor i; and
  - it must not be the case that the lift is has a request to continue in the same direction.

The notion of *down service at floor i* is described analogously. We introduce some additional propositions uv_i and dv_i as abbreviations for other propositional formulae to capture these notions. We do not embed the need for a request to go up (respectively down) into the notion of an upservice (respectively downservice) to match with the previous specifications. Let

uv_i denote that there is an upservice at floor i (for all floors $0 \leqslant i < n$);

dv_i denote that there is an down service at floor i (for all floors $1 < i \leqslant n$);

We define *up service at i* as follows for $i = 0$

$$uv_i \iff d_i \tag{39}$$

and for $0 < i < n$

$$uv_i \iff (d_i \wedge (up \vee (\neg up \wedge \neg cb_i))) \tag{40}$$

Similarly we define *down service at i* as follows for $0 < i < n$.

$$dv_i \iff (d_i \wedge (\neg up \vee (up \wedge \neg ca_i))) \tag{41}$$

and also

$$dv_n \iff d_n \tag{42}$$

The unless formulae relating to calling the lift from Section 3.1, Buttons and Lights, now become the following for calls upwards.

$$\Box \bigwedge_{i=0}^{n-1} (cu_i \Rightarrow (lcu_i \mathcal{W} uv_i))$$
$$\Box \bigwedge_{i=0}^{n-1} (\neg lcu_i \Rightarrow (\neg lcu_i \mathcal{W} cu_i))$$
$$\Box \bigwedge_{i=0}^{n-1} (lcu_i \Rightarrow (lcu_i \mathcal{W} uv_i))$$

From this we obtain the following formulae for all floors $0 \leqslant i < n$.

$$\Box(cu_i \Rightarrow (lcu_i \vee uv_i)) \tag{43}$$
$$\Box(lcu_i \Rightarrow \bigcirc(lcu_i \vee uv_i)) \tag{44}$$
$$\Box(\neg lcu_i \Rightarrow \bigcirc(\neg lcu_i \vee cu_i)) \tag{45}$$

The unless formulae relating to calling the lift from Section 3.1, Buttons and Lights, now become the following for calls down.

$$\Box \bigwedge_{i=1}^{n} (cd_i \Rightarrow (lcd_i \mathcal{W} dv_i))$$
$$\Box \bigwedge_{i=1}^{n} (lcd_i \Rightarrow (lcd_i \mathcal{W} dv_i))$$
$$\Box \bigwedge_{i=1}^{n} (\neg lcd_i \Rightarrow (\neg lcd_i \mathcal{W} cd_i))$$

From this we obtain the following formulae for all floors $0 < i \leqslant n$.

$$\Box(cd_i \Rightarrow (lcd_i \vee dv_i)) \tag{46}$$
$$\Box(lcd_i \Rightarrow \bigcirc(lcd_i \vee dv_i)) \tag{47}$$
$$\Box(\neg lcd_i \Rightarrow \bigcirc(\neg lcd_i \vee cd_i)) \tag{48}$$

Together formulae (43)–(48) replace (1)–(3). Note that the unless formulae relating to the buttons within the lift remain the same as previously.

The formula (7) is now replaced by two formulae

$$\Box(lcu_i \Rightarrow \neg uv_i) \tag{49}$$
$$\Box(lcd_i \Rightarrow \neg dv_i) \tag{50}$$

The servicing behaviour is as before, namely (9)–(17).

Initially the lift doors are closed, there are no call or send requests and none of the lift buttons are lit.

$$\bigwedge_{i=0}^{n}(\neg d_i \wedge \neg s_i \wedge \neg ls_i) \wedge \bigwedge_{i=0}^{n-1}(\neg cu_i \wedge \neg lcu_i) \wedge \bigwedge_{i=1}^{n}(\neg cd_i \wedge \neg lcd_i) \tag{51}$$

3.3 Multiple Lifts

Following [3] we next consider multiple lifts. Consider m lifts and n floors. We extend the propositions at_i, d_i, s_i, ls_i, up, ca_i and cb_i to include an index relating to which lift is being referred to as follows.

at_{ji} the lift j is at floor i;

d_{ji} the lift j door is open at floor i (and $\neg d_{ji}$ the lift j door is closed at floor i);

s_{ji} the internal button to send the lift j to floor i is pressed;

ls_{ji} the internal send light in lift j at floor i is lit;

up_j lift j movement is up (and $\neg up_j$ is the lift j movement is down);

ca_{ji} denotes a request for lift j above floor i;

cb_{ji} denotes a request for lift j below floor i.

We add the following propositions relating to up and down services in relation to some lift and floor.

uv_{ji} denotes an up service by lift j at floor i;

dv_{ji} denotes a down service by lift j at floor i.

The following formula re-define the propositions ca_i (call from above) for $i = n-1$, $j = 1,\ldots m$

$$\Box(ca_{jn-1} \iff lcd_n \vee ls_{jn}) \tag{52}$$

and where for $i = 0,\ldots n-2$, $j = 1,\ldots m$

$$\Box(ca_{ji} \iff ca_{ji+1} \vee lcu_{i+1} \vee lcd_{i+1} \vee ls_{ji+1}) \tag{53}$$

The following defines cb_i (call from below) for $i = 1$, $j = 1,\ldots m$

$$\Box(cb_{j1} \iff lcu_0 \vee ls_{j0}) \tag{54}$$

and for $i = 2,\ldots n$, $j = 1,\ldots m$

$$\Box(cb_{ji} \iff cb_{ji-1} \vee lcu_{i-1} \vee lcd_{i-1} \vee ls_{ji-1}) \tag{55}$$

The formulae (33)–(38) relating to ensuring that the lift opens are extended to the multiple lift versions by adding an additional index to represent each lift. We define up service at floor $i = 0, \ldots n-1$ as follows

$$\Box(uv_i \iff (\bigvee_{j=1}^{m} uv_{ji})) \tag{56}$$

and down service at floor $i = 1, \ldots n$ as follows

$$\Box(dv_i \iff (\bigvee_{j=1}^{m} dv_{ji})) \tag{57}$$

The formulae for upservice (uv_{ji}) and down service for each lift (dv_{ji}) for $j = 1, \ldots m$ is the same as previously except with an additional lift index for each proposition, namely formulae (39)–(42). The formulae relating to the pressing and lighting of the the external call up and call down are as previously, i.e. (43)–(50). The send buttons within lifts, namely (4)–(6) are the same as previously but including an additional index for each lift. The servicing behaviour is as before, see (9)–(17), except each proposition is given an additional index for the lift under consideration. Each lift must be at exactly one floor at any moment for $j = 1, \ldots m$

$$\Box(\bigvee_{i=0}^{n} at_{ji}) \tag{58}$$

and for $j = 1, \ldots m$ for each $0 \leqslant i \leqslant n$, $0 \leqslant k \leqslant n$, $i \neq j$

$$\Box(\neg at_{ji} \vee \neg at_{jk}) \tag{59}$$

The formulae (22), (24), (25) are as previously but given an additional index for the lift in question. The initial conditions are as previously but with propositions extended with the lift index. Note that this means many clauses are repeated for each lift which makes the specification much longer.

4 Execution

Once we have our temporal specification, then we might like to *prototype* this to see if it behaves as we would expect. For this we can use METATEM [4].

The METATEM system essentially takes a temporal specification, again in SNF [21], and attempts to build a model for it. Given a set of SNF clauses, the basic METATEM execution process involving forward chaining from the initial clauses via the step clauses. During this the execution attempts to satisfy any '$\Diamond$' formulae generated from sometime clauses. In order to ensure that no satisfying path has been missed, the execution may backtrack to consider alternative choices [4, 5].

METATEM has been developed and refined over the years, but we here just use the basic model building capabilities. See [26, 24] for further details on aspects available in the current implementation, including: multiple processes/agents; communication between agents; limited first-order syntax; limited backtracking; preferences (and other meta-level predicates); and context.

In the case of our lift specification, we can take the SNF version and directly execute it. For the basic specification, this is likely to be uninteresting since no lift calls are *prescribed*. The lift could just sit at one floor forever and so happily satisfy its specification. So, in addition to the earlier specification, we need to have some description of the environment. Specifically, we need a description of the pattern

of call button presses. So long as we can describe this behaviour in temporal logic, then we just add this to our lift specification an execute using METATEM.

As an example, we took the specification for the 3 floor lift system and executed it using the METATEM implementation developed by Anthony Hepple and available at

`http://www.csc.liv.ac.uk/~michael/TLBook`

With some simple "print" statements added, the basic lift specification, when executed, just chose one floor to be at, in a fairly arbitrary way. However, once we added

$$\textbf{start} \Rightarrow \bigcirc(c_2 \wedge \bigcirc\bigcirc(c_0 \wedge \bigcirc\bigcirc c_1))$$

specifying a sequence of calls, the output became:

```
State 0: At floor 1
State 1: Called to floor 2; At floor 1
State 2: Called to floor 0; At floor 2
State 3: Called to floor 1; At floor 1
State 4: At floor 0
.............
```

Similarly, we can add formulae such as

$$\Box(\diamondsuit c_0 \wedge \diamondsuit c_1 \wedge \diamondsuit c_2)$$

and have an infinite number of calls.

This is a good way to explore straightforward specifications, but is not especially efficient. For example, once we get to examples with 4 or 5 floors, then METATEM becomes *very* slow. Note however, that the implementation used above essentially allows first-order temporal specifications and this, together with some programming directives which improve efficiency, can make such specifications execute much more quickly.

5 Proofs About Lifts

So, once we have our lift specification, and we are happy that it *appears* to do what we want, then we would like to *prove* some properties of it. Since we now have

Spec — a temporal formula specifying the lift system

Req — a temporal formula describing our requirements

then we would like to prove

$\vdash Spec \Rightarrow Req$

But, how shall we do this? We can, if necessary, use METATEM as seen in the last section, since METATEM is a complete model-building procedure for PTL (within one agent). However, letting METATEM backtrack over all possibilities is very inefficient. So, instead, we consider the *clausal temporal resolution* method [25].

5.1 Resolution for Propositional Temporal Logic

This is a refutation method, so we negate the formula we wish to prove. Then, since it is a *clausal* method, we translate the resulting formula into SNF again. Once a formula has been transformed into SNF, both step resolution and eventuality resolution rules can be applied. Step resolution effectively consists of the application of the standard classical resolution rule to formulae representing constraints at a particular moment in time, together with simplification rules, subsumption rules, and rules for transferring contradictions within states to constraints on previous states. Eventuality resolution resolves a sometime clause whose right-hand side is, for example, $\Diamond l$ with a set of clauses that together imply that l is always false. Pairs of initial or step clauses may be resolved using the following initial and step resolution rules (where A and B are disjunctions of literals, C and D are conjunctions of literals, and p is a proposition).

$$\frac{\begin{array}{lcl}\textbf{start} & \Rightarrow & A \vee p \\ \textbf{start} & \Rightarrow & B \vee \neg p\end{array}}{\begin{array}{lcl}\textbf{start} & \Rightarrow & A \vee B\end{array}} \qquad \frac{\begin{array}{rcl}C & \Rightarrow & \bigcirc(A \vee p) \\ D & \Rightarrow & \bigcirc(B \vee \neg p)\end{array}}{\begin{array}{rcl}(C \wedge D) & \Rightarrow & \bigcirc(A \vee B)\end{array}}$$

The following is used to remove clauses which imply **false** (where A is a conjunction of literals).

$$\{A \Rightarrow \bigcirc \textbf{false}\} \quad \longrightarrow \quad \left\{ \begin{array}{lcl}\textbf{start} & \Rightarrow & \neg A \\ \textbf{true} & \Rightarrow & \bigcirc \neg A\end{array} \right\}$$

Thus, if, by satisfying A, a contradiction is produced in the next moment, then A must never be satisfied.

The eventuality resolution rule effectively resolves together formulae containing the $\Box$ and $\Diamond$ connectives. However, as the translation to SNF restricts the clauses to be of a certain form, the application of such an operation will be between a sometime clause and a *set* of step clauses that together ensure a complementary literal will *always* hold.

The eventuality resolution rule applies between a sometime clause and a set of clauses that together imply $A \Rightarrow \bigcirc \Box \neg l$.

$$\frac{\begin{array}{lcl}A & \Rightarrow & \bigcirc \Box \neg l \\ C & \Rightarrow & \Diamond l\end{array}}{\begin{array}{lcl}C & \Rightarrow & \neg A \mathscr{W} l\end{array}}$$

The resolvent states that once C has been satisfied A must remain false unless l becomes true. The resolvent requires further translation into SNF. Note that the first premise does not occur in the normal form. The full temporal resolution rule, in detail, is as follows.

$$\frac{\begin{array}{lcl}A_0 & \Rightarrow & \bigcirc B_0 \\ \ldots & \Rightarrow & \ldots \\ A_r & \Rightarrow & \bigcirc B_r \\ C & \Rightarrow & \Diamond l\end{array}}{\begin{array}{lcl}C & \Rightarrow & \left[\bigwedge_{i=0}^{r} (\neg A_i)\right] \mathscr{W} l\end{array}}$$

where $A_i \Rightarrow \bigcirc B_i$ is a conjunction of one or more step clauses such that for all $i, 0 \leq i \leq r$ $B_i \Rightarrow \neg l$; and $B_i \Rightarrow \bigvee_{j=0}^{r} A_j$.

The side conditions ensure that

$$(\bigvee_i A_i) \Rightarrow \bigcirc \Box \neg l.$$

The set of clauses $A_i \Rightarrow \bigcirc B_i$ that satisfy these side conditions are together known as *a loop in* $\neg l$. Algorithms to find the loop within a set of SNF clauses are described in [9, 10]. The resolvent must again be translated into SNF before any further resolution steps. Whilst this introduces additional new propositions the number required is finite (one for each eventuality) so does not not affect the termination of the calculus (see [25] for more details). The step resolution process terminates when either no

new resolvents can be generated or a contradiction is derived by generating the following unsatisfiable formula

$$\textbf{start} \Rightarrow \textbf{false}.$$

Given any temporal formula, A, to be tested for unsatisfiability, the following steps are performed [24].

1. Translate A into SNF, giving A_s.
2. undertake step resolution (including simplification and subsumption) on A_s until either
   (a) **start** $\Rightarrow$ **false** is derived—terminate declaring that A is unsatisfiable; or
   (b) no further resolvents are generated—continue to step (3).
3. Choose an eventuality from the right-hand side of a sometime clause within A_s, for example $\Diamond l$. Search for loop-formulae for $\neg l$.
4. Construct loop resolvents for the loop detected and each sometime clause with $\Diamond l$ on the right-hand side. If any new formulae (i.e., that are not subsumed by clauses already present) have been generated, go to step (2).
5. If all eventualities have been resolved, i.e., no new formulae have been generated for any of the eventualities, terminate declaring A satisfiable; otherwise go to step (3).

The following theorem shows the correctness of the calculus.

Theorem 2. *[25] A clause set, S, is satisfiable if and only if temporal resolution procedure terminates declaring the set of clauses is "satisfiable" when applied to S.*

5.2 TRP++

TRP++ [33] is a theorem prover for PTL which is implemented in C++ and based on the above temporal resolution calculus. Input uses the SNF normal form but has a different syntax to that given above which is more amenable to mechanisation. To apply the initial and step resolution rules clauses are translated into first order logic using a natural arithmetic translation. For example an initial clause

$$\textbf{start} \Rightarrow p_1 \vee p_2 \vee p_3$$

is translated to

$$p_1(0) \vee p_2(0) \vee p_3(0)$$

where 0 is the natural number 0, and step clauses

$$p_1 \wedge p_2 \Rightarrow \bigcirc(p_3 \vee p_4)$$

are translated to

$$\forall x(\neg p_1(x) \vee \neg p_2(x) \vee p_3(s(x)) \vee p_4(s(x)))$$

where s represents the successor function over the natural numbers. Both initial and step resolution correspond to standard first-order ordered resolution. The eventuality resolution rule can also be implemented using an algorithm based on step resolution as described in [11]. Thus using this translation any first-order resolution prover could be used to perform initial and step resolution. However TRP++ implements its own 'near propositional' approach to do this. An earlier version of TRP++ has been compared with a implementations of a number of temporal tableau-based decision procedures and performs well [34].

5.3 Properties to Prove

We now describe some temporal properties that we might like to prove of the above specifications. For simplicity, we assume that $n = 4$, i.e. the top floor is the 4th.

5.3.1 Simple Single Lift

P1a This property relates to the lift specification L1 about buttons and lights inside the lift. Pressing a button for some floor in the lift causes the light within the lift to illuminate (if the lift doors are not already open at that floor).

$$\Box \bigwedge_{i=0}^{4} ((s_i \wedge \neg d_i) \Rightarrow ls_i)$$

P1b This property relates to the lift specification L1 about buttons and lights inside the lift. If the send light is illuminated for some floor it is cancelled when the lift visits that floor.

$$\Box \bigwedge_{i=0}^{4} (d_i \Rightarrow \neg ls_i)$$

P2a Properties **P2a** and **P2b** relate to the lift specification L2 about buttons and lights outside the lift on each floor. Note that as in this specification we only have one button per floor we cannot deal with the properties that relate to the direction of the lift. (It is not simple to specify the property relating to minimising waiting times.) Pressing a button on some floor causes the light to illuminate (if the lift doors are not already open at that floor).

$$\Box \bigwedge_{i=0}^{4} ((c_i \wedge \neg d_i) \Rightarrow lc_i)$$

P2b If a call light is illuminated on some floor it is cancelled when the lift visits that floor.

$$\Box \bigwedge_{i=0}^{4} (d_i \Rightarrow \neg lc_i)$$

P3 This relates to lift specification L3. When a lift has no request to service it should remain at its final destination with its doors closed and await further requests.

$$\Box \left((\bigwedge_{i=0}^{4} (\neg lc_i \wedge \neg ls_i)) \Rightarrow \bigwedge_{i=0}^{4} (at_i \Rightarrow \bigcirc at_i) \right).$$

P4 This relates to lift specification L4. All requests for lifts from a floor must be serviced eventually. (It is not straightforward to model the part of L4 "with all floors given equal priority".)

$$\Box \bigwedge_{i=0}^{4} (c_i \Rightarrow \Diamond d_i)..$$

P5 This relates to L5. All requests for floors within lifts must be serviced eventually. We do not attempt to specify "with floors being serviced sequentially in the direction of travel".

$$\Box \bigwedge_{i=0}^{4} (s_i \Rightarrow \Diamond d_i)..$$

P6 We cannot specify this as we have not dealt with emergency behaviour.

P7 We show that the lift door is either closed on all floors or is open on at most one floor (see Section 3.1).

$$\bigwedge_{i,j=0,i\neq j}^{4} (\neg d_i \vee \neg d_j)$$

5.3.2 Single Lift with Up and Down Call Buttons

P1a and P1b As previously.

P2 Properties **P2a** and **P2b** relate to the lift specification L2 about buttons and lights outside the lift on each floor. (Again, it is not obvious how to specify the property relating to minimising waiting times.)

P2au' Pressing a call up button on some floor causes the light to illuminate (if there is not already an upservice at that floor).

$$\Box \bigwedge_{i=0}^{3} (cu_i \wedge \neg uv_i) \Rightarrow lcu_i$$

P2ad' Pressing a call down button on some floor causes the light to illuminate (if there is not already a down service at that floor).

$$\Box \bigwedge_{i=1}^{4} (cd_i \wedge \neg dv_i) \Rightarrow lcd_i$$

P2bu' If a call up light is illuminated on some floor it is cancelled when the lift visits that floor (i.e. there is an up service at that floor).

$$\Box \bigwedge_{i=0}^{3} (uv_i \Rightarrow \neg lcu_i)$$

P2bd' If a call down light is illuminated on some floor it is cancelled when the lift visits that floor (i.e. there is an down service at that floor).

$$\Box \bigwedge_{i=1}^{4} (dv_i \Rightarrow \neg lcd_i)$$

P3' This relates to lift specification L3. When a lift has no request to service it should remain at its final destination with its doors closed and await further requests. We now have to use the up and down versions of the light buttons

$$\Box \left(\bigwedge_{i=0}^{n-1} \neg lcu_i \wedge \bigwedge_{i=1}^{n} \neg lcd_i \wedge \bigwedge_{i=0}^{n} \neg ls_i \right) \Rightarrow \bigwedge_{i=0}^{4} (at_i \Rightarrow \bigcirc at_i) .$$

P4 This relates to lift specification L4. All requests for lifts from a floor must be serviced eventually. (Again, it is not straightforward to model the part of L4 "with all floors given equal priority".)

P4u' We have split this into two the first for call up buttons and the second for call down buttons. First the call up version.

$$\Box \bigwedge_{i=0}^{3} (cu_i \Rightarrow \Diamond uv_i).$$

P4d' Next the call down version.

$$\Box \bigwedge_{i=1}^{4} (cd_i \Rightarrow \Diamond dv_i).$$

P5 As previously

P5a' We also try proof this for a sample floor

$$\Box(s_2 \Rightarrow \Diamond d_2).$$

P6 As previously we cannot specify this as we have not dealt with emergency behaviour.

P7 As above.

5.3.3 Multiple Lifts

We can state properties in a similar manner for multiple lifts.

5.4 Proving Selected Properties

We use the above specifications as input to TRP++. Assume that $n = 4$, i.e. the top floor is floor 4.

5.4.1 Simple Single Lift

First we consider the specification of a simple single lift with just one external button per floor (see Section 3.1) and the properties in Section 5.3. Let *Spec* be the conjunction of the formulae (1)–(28) and *Req* be the property we wish to prove. The following table show the results from inputting $Spec \wedge \neg Req$ to TRP++. The first line shows that the specification on its own is satisfiable. The negation of the other properties returns unsatisfiable as expected.

Req	Satisfiable	Time (s)
-	Satisfiable	0.03
P1(a)	Unsatisfiable	0.36
P1(b)	Unsatisfiable	0.03
P2(a)	Unsatisfiable	0.36
P2(b)	Unsatisfiable	0.03
P3	Unsatisfiable	0.12
P4	Unsatisfiable	6.15
P5	Unsatisfiable	12.3
P7	Unsatisfiable	0.03

5.4.2 Single Lift with Up and Down Call Buttons

Next we consider the specification of a simple single lift with two external buttons per floor (see Section 3.1) and the properties in Section 5.3. Let $Spec1$ be the conjunction of the formulae (4)–(6), (8), (9)–(17), (22), (25)–(27), and (29)–(51) and Req be the property we wish to prove. The following table show the results from inputting $Spec1 \wedge \neg Req$ to TRP++. The first line shows that the specification on its own is satisfiable. The entry with the time denoted as > 3600 denotes that the prover had not finished within an hour. The negation of the other properties returns unsatisfiable as expected.

Req	Satisfiable	Time (s)
-	Satisfiable	1.62
P1(a)	Unsatisfiable	2.29
P1(b)	Unsatisfiable	1.78
P2(au)'	Unsatisfiable	2.78
P2(ad)'	Unsatisfiable	8.80
P2(bu)'	Unsatisfiable	2.28
P2(bd)'	Unsatisfiable	5.32
P3'	Unsatisfiable	6.76
P4u'	Unsatisfiable	24.89
P4d'	Unsatisfiable	55.71
P5	Unsatisfiable	> 3600
P5a'	Unsatisfiable	13.91
P7	Unsatisfiable	1.64

5.4.3 Multiple Lifts

Next we consider the specification of two lifts each with two external buttons per floor (see Section 3.3) and the properties in Section 5.3. Let $Spec2$ be the conjunction of the formulae (with propositions extended by the additional lift index where necessary) (4)–(6), (9)–(17), (22), (25), (33)–(51), and (52) -(59) and Req be the property we wish to prove. At this point we find the proof time becoming *very* large. This shows that some, more refined, representation is required instead of describing everything in PTL (and using a large number of propositions). In the next two sections, we describe two such improved representations.

6 Applying Constraints

When we write our specification in PTL, we of course get too many/big formulae. We might turn to first-order temporal logics to solve this (see later) but there is another approach. Consider the structural constraints we gave earlier, essentially in propositional (not temporal) logic. For example, with one lift and 5 floors, we need all of:

$$\begin{array}{lcl} at_0 & \Rightarrow & (\neg at_1 \wedge \neg at_2 \wedge \neg at_3 \wedge \neg at_4) \\ at_1 & \Rightarrow & (\neg at_0 \wedge \neg at_2 \wedge \neg at_3 \wedge \neg at_4) \\ at_2 & \Rightarrow & (\neg at_0 \wedge \neg at_1 \wedge \neg at_3 \wedge \neg at_4) \\ at_3 & \Rightarrow & (\neg at_0 \wedge \neg at_1 \wedge \neg at_2 \wedge \neg at_4) \\ at_4 & \Rightarrow & (\neg at_0 \wedge \neg at_1 \wedge \neg at_2 \wedge \neg at_3) \end{array}$$

to state that the lift is at most one floor at any moment (see (26) and (27)). Additionally we also need to add

$$at_0 \vee at_1 \vee at_2 \vee at_3 \vee at_4$$

to ensure that the lift is at some floor at any moment. This is even before we get to more interesting formulae that specify the dynamic aspects of the lift. If we go on to 10, or 100, or 1000 lifts then the above formulae are *huge*.

We here describe a class of temporal logics that have such structural constraints *built in*. These logics, called TLX [14, 16] and TLC [15], not only reduce the size of the temporal specifications we use but also significantly reduce the complexity of the decision problem for such specifications if the specifications are in or near SNF. Note that this approach involves reasoning *in the presence* of constraints rather than reasoning *about* them. We begin with the logic TLC.

Essentially the logic TLC is PTL extended by the addition of *cardinality constraints* that restricts the numbers of literals that can be satisfied at any moment in time. To specify the constraints we allow statements stating that *up to k* literals, or *exactly k* literals from some subset of literals, are true at any moment in time. The above statement expressing that a lift may be at exactly one floor is written as

$$\mathscr{T}^{=1} = \{at_0, \ldots, at_n\}^{=1}$$

Other constraints that appear in this specification are the doors on some lift must be open on at most one floor.

$$\mathscr{D}^{\leqslant 1} = \{d_0, \ldots, d_n\}^{\leqslant 1}$$

Formally, a constraint $C_i^{\propto_i m_i}$ is a tuple $(C_i, \propto_i, m_i)$, where C_i is a set of literals with a cardinality restriction $\propto_i m_i$, such that $\propto_i \in \{=, \leq\}$ and $m_i \in \mathbb{N}$. The logic TLC is parametrised by a set of constraints, $\mathrm{TLC}(C_1^{\propto_1 m_1}, \cdots, C_n^{\propto_n m_n})$. The set of propositional symbols PROP is constructed as follows

$$\mathrm{PROP} = \{p \mid p \in C_i^{\propto_i m_i}\} \cup \{p \mid \neg p \in C_i^{\propto_i m_i}\} \cup \mathscr{A}$$

where $\mathscr{A}$ is a set of unconstrained propositions. The syntax of TLC formulae is the same as for PTL. The semantics of TLC uses the satisfiability of a constraint which we define next.

The notation $\mathscr{L} \models_{PL} \varphi$ denotes the truth of propositional logic formula φ with respect to a set of propositions $\mathscr{L}$. $\mathscr{L} \models_{PL} p$ iff $p \in \mathscr{L}$ where $p \in \mathrm{PROP}$ and the semantics of the operators $\neg$, $\vee$ is as usual. Let $\mathscr{L}$ be a set of propositions, $C^{\propto m}$ a constraint and

$$Eval(\mathscr{L}, C^{\propto m}) = \{p \mid p \in \mathscr{L} \text{ and } p \in C\} \cup \{\neg p \mid p \notin \mathscr{L} \text{ and } \neg p \in C\}$$

then

$$\begin{array}{lll} \mathscr{L} \models_{PL} C^{=m} & \text{iff} & |Eval(\mathscr{L}, C^{=m})| = m, \\ \mathscr{L} \models_{PL} C^{\leqslant m} & \text{iff} & |Eval(\mathscr{L}, C^{\leqslant m})| \leqslant m. \end{array}$$

Note that the operator $\models_{PL}$ is only defined for formulae from propositional logic (not from temporal logic). A set $\mathscr{C}$ of constraints is *satisfiable* ($\models_{PL} \mathscr{C}$) if, and only if, there is a set of propositions $\mathscr{L}$, such that, for each $C_i^{\propto_i m_i} \in \mathscr{C}$ ($i \in \mathbb{N}$), $\mathscr{L} \models_{PL} C_i^{\propto_i m_i}$.

A model for TLC($\mathscr{C}$) formulae can be characterised as a *sequence of states*, σ, of the form $\sigma = s_0, s_1, s_2, s_3, \ldots$, where each state s_i is a set of propositional symbols representing those propositions, which are satisfied at the i^{th} moment in time. Every s_i should satisfy the set of constraints, $\mathscr{C}$, i.e., for all s_i we have $s_i \models_{PL} \mathscr{C}$ (where s_i is a set of propositions).

Theorem 3. *[15] Satisfiability of a TLC($\mathscr{C}_1^{\propto_1 m_1}, \ldots \mathscr{C}_n^{\propto_n m_n}$) formula φ in Separated Normal Form can be decided in time*

$$O\left(|\varphi| \times \left(|\mathscr{C}_1^{\propto_1 m_1}|^{m_1} \times \cdots \times |\mathscr{C}_n^{\propto_n m_n}|^{m_n} \times 2^{|\mathscr{A}|}\right)^2\right)$$

where $|\varphi|$ is the length of φ, $|\mathscr{C}_i^{\propto_i m_i}|$ is the size of the set $\mathscr{C}_i^{\propto_i m_i}$ of constrained literals, and $|\mathscr{A}|$ is the size of the set $\mathscr{A}$ of unconstrained propositions.

If the TLC formulae are in, or close to, the normal form, in practice, this result means that TLC reasoning can be much more efficient than reasoning with the representation of constraints in PTL.

In [15] we provided a tableau-like algorithm to check satisfiability for TLC formulae. Essentially we construct a graph like structure where nodes are sets of literals that satisfy the constraints and edges represent temporal successors that satisfy step formulae. Deletions in the graph occur when a node has no edges from it (i.e. no node satisfies the right hand side of any step clauses) and terminal subgraphs where some eventuality cannot be fulfilled. In experiments the use of constraints appears not only to provide a succinct representation but also outperforms other tableau reasoners for temporal logic when a large number of propositions are constrained. However there are a number of disadvantages. First, we have to explicitly enumerate sets of propositions that satisfy the set of constraints. Second, although the algorithm only generates reachable states, we are often forced to construct *all* the states, e.g., when there are no initial clauses. This immediately results in the worst case complexity. Inherently, the incremental graph construction adopts a breadth-first style that requires us to construct, and keep in memory, a large number of nodes, resulting in an unavoidable inefficiency. We are currently developing a more standard tableau calculus along the lines of [44].

TLX is sublogic of TLC. The constraints in TLX are restricted to be "exactly one" sets (and as usual some unconstrained propositions are allowed), i.e. all the constraints are of the form $C_i^{=1}$. As a further restriction we require that the sets of propositions in each C_i are disjoint. Thus TLX($C_1, \ldots, C_n$) = TLC($C_1^{=1}, \ldots, C_n^{=1}$) with the additional restriction of disjointness. In [16] we devised a temporal resolution calculus for TLX, and established its completeness and complexity. Specifically, if a set of TLX clauses is unsatisfiable, then a contradiction will be deduced within time polynomial in $N_1 \times N_2 \times \cdots \times N_n \times 2^{\mathscr{A}}$ where N_1 is the size of C_1, N_2 is the size of C_2, etc, while $\mathscr{A}$ is the number of unconstrained propositions. TLX has a number of potential applications, and its relatively low complexity makes fast analysis feasible.

7 Many Lifts

As mentioned above, when we get many lifts/floors then our PTL specifications become large. We must duplicate many formulae for each lift, for example in relation to the movement, location and door status of each lift. Also we must make sure that formulae relating to interactions between lifts is correctly specified e.g., when to turn off a external call light which may have been serviced by any lift. However, since all lifts have the same temporal specification, then what we really want is a *first-order temporal logic* (FOTL) specification that we can instantiate for each particular lift. While appealing, FOTL has severe complexity problems. However, if we restrict ourselves to the *monodic* fragment [31] then the logic remains tractable.

7.1 First Order Temporal Logic

First-Order (discrete linear time) Temporal Logic, FOTL, is an extension of classical first-order logic with operators that deal with a linear and discrete model of time (isomorphic to $\mathbb{N}$ with the usual order relation, $<$, 'less than'). Formulae in FOTL are constructed in a standard way [21, 31] from:

- predicate symbols $P_0, P_1, \ldots$ each of which is of some fixed arity (null-ary predicate symbols are called *propositions*);
- individual variables $x_0, x_1, \ldots$;
- individual constants $c_0, c_1, \ldots$;

- Boolean operators **true**, $\neg$, and $\vee$;
- quantifiers $\exists$; and
- temporal operators '$\Diamond$' (*sometime in the future*), '$\bigcirc$' (*at the next moment in time*), and '$\mathcal{U}$' (*until*);

So,

- **true** is a FOTL formulae.
- If $t_1, ..., t_n$ are constants or variables and P is a n-ary predicate symbol, then $P(t_1, ..., t_n)$ is a FOTL formula.
- If ϕ and ψ are FOTL formulae and x is an individual variable then so are $\neg\phi$, $\phi \vee \psi$, $\exists x\phi$, $\Diamond\phi$, $\bigcirc\phi$, $\phi \mathcal{U} \psi$.

Note we can obtain **false**, $\forall$, and the other Boolean operators via the usual equivalences, and the temporal operators '$\Box$', '$\Diamond$' and '$\mathcal{W}$' operators using the equivalences in Section 2.

A FOTL formula ϕ is called monodic (see [31]) if any subformulae of the form $T\phi$, where T is one of $\Diamond$, $\Box$, $\bigcirc$ (or $\phi_1 T \phi_2$, where T is one of $\mathcal{U}$ or $\mathcal{W}$) contains at most one free variable.

Formulae in FOTL are interpreted in *first-order temporal structures* of the form $\mathfrak{M} = \langle D_i, I_i \rangle$, where D_i is a non-empty set and I_i is an interpretation of predicate and constant symbols over D_n. We make the expanding domains assumption, i.e., whenever $n < m$, $D_i \subseteq D_m$. A (variable) assignment, $\mathfrak{a}$, is a function from the set of individual variables to $\bigcup_{i \in \mathbb{N}} D_n$. We denote the set of all variable assignments by $\mathfrak{V}$. For every moment of time n, there is a corresponding *first-order* structure, $\mathfrak{M}_i = \langle D_i, I_i \rangle$; the corresponding set of variable assignments $\mathfrak{V}_n$ is a subset of the sets of all assignments,

$$\mathfrak{V}_i = \{\mathfrak{a} | \mathfrak{a}(x) \in D_i \text{ for every variable } x\}.$$

Intuitively, FOTL formulae are interpreted in sequences of such moments in time, $\mathfrak{M}_0, \mathfrak{M}_1, ...$ with truth values in different moments being connected via temporal operators.

The truth relation $\mathfrak{M}_i \models^{\mathfrak{a}} \phi$ in a structure $\mathfrak{M}$, only for those assignments $\mathfrak{a}$ that satisfy the condition $\mathfrak{a} \in \mathfrak{V}_n$, is defined inductively in the usual way under the following understanding of the temporal operators:

$$\begin{array}{lll}
\mathfrak{M}_i \models^{\mathfrak{a}} \bigcirc\phi & \text{iff} & \mathfrak{M}_{i+1} \models^{\mathfrak{a}} \phi \\
\mathfrak{M}_i \models^{\mathfrak{a}} \Box\phi & \text{iff} & \text{for all } m \in \mathbb{N}, \text{ if } (m \geq i) \text{ then } \mathfrak{M}_m \models^{\mathfrak{a}} \phi \\
\mathfrak{M}_i \models^{\mathfrak{a}} \Diamond\phi & \text{iff} & \text{there exists } m \in \mathbb{N} \text{ such that } (m \geq i) \text{ and } \mathfrak{M}_m \models^{\mathfrak{a}} \phi \\
\mathfrak{M}_i \models^{\mathfrak{a}} \phi \mathcal{U} \psi & \text{iff} & \text{there exists } m \leq i, \text{ and } \mathfrak{M}_m \models^{\mathfrak{a}} \psi, \text{ and for all } j \in \mathbb{N}, \\
 & & \text{if } (i \leq j < m) \text{ then } \mathfrak{M}_i \models^{\mathfrak{a}} \phi \\
\mathfrak{M}_i \models^{\mathfrak{a}} \phi \mathcal{W} \psi & \text{iff} & \mathfrak{M}_i \models^{\mathfrak{a}} \phi \mathcal{U} \psi \text{ or } \mathfrak{M}_i \models^{\mathfrak{a}} \Box\phi
\end{array}$$

$\mathfrak{M}$ 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 satisfied in any temporal structure under any assignment.

7.2 Divided Separated Normal Form (DSNF)

As for PTL, a normal form has been defined for FOTL. This was first defined in [8] and later used in [37]. A *temporal step clause* is a formula either of the form $p \Rightarrow \bigcirc l$, where p is a proposition and l is a propositional literal, or $(P(x) \Rightarrow \bigcirc M(x))$, where $P(x)$ is a unary atom and $M(x)$ is a unary literal. We call a clause of the first type an (original) *ground* step clause, and of the second type an (original) *non-ground* step clause. A *monodic temporal problem in Divided Separated Normal Form (DSNF)* is a quadruple $\langle \mathscr{U}, \mathscr{I}, \mathscr{S}, \mathscr{E} \rangle$, where

1. the universal part, $\mathscr{U}$, is given by a finite set of arbitrary closed first-order formulae;
2. the initial part, $\mathscr{I}$, is, again, given by a finite set of arbitrary closed first-order formulae;
3. the step part, $\mathscr{S}$, is given by a finite set of original (ground and non-ground) temporal step clauses, the left-hand sides of step clauses are pairwise distinct; and
4. the eventuality part, $\mathscr{E}$, is given by a finite set of clauses of the form $\diamondsuit L(x)$ (a *non-ground* eventuality clause) and $\diamondsuit l$ (a *ground eventuality* clause), where l is a propositional literal and $L(x)$ is a unary non-ground literal.

With each monodic temporal problem, we associate the formula

$$\mathscr{I} \wedge \Box \mathscr{U} \wedge \Box \forall x \mathscr{S} \wedge \Box \forall x \mathscr{E}.$$

where in the above we denote the conjunction $\bigwedge \mathscr{X}$ for a finite set of formulae $\mathscr{X}$ simply as $\mathscr{X}$.

Theorem 4 (see [8], Theorem 1)**.** *Any monodic first-order temporal formula ϕ can be transformed into a monodic temporal problem P in DSNF with at most linear increase in the size of the problem such that ϕ is satisfiable over expanding domains if, and only if, P is satisfiable over expanding domains.*

7.3 Lift Specification

We can use the specification for multiple lifts from Section 3.3 to develop a specification in first-order temporal logic. To remain monodic we must decide whether the variables relate to lifts or floors. Here we select the latter. For example the first-order version of the clause (9) is as follows.

$$\Box \forall x (at_i(x) \wedge ca_i(x) \wedge up(x) \Rightarrow \bigcirc (at_{i+1}(x) \wedge up(x)))$$

7.4 Temporal Resolution

Next we explain how clausal temporal resolution is extended to monodic FOTL [8, 37] Let P be a monodic temporal problem, and let

$$P_{i_1}(x) \implies \bigcirc M_{i_1}(x), \ldots, P_{i_k}(x) \implies \bigcirc M_{i_k}(x) \tag{60}$$

be a subset of the set of its step clauses. Then formulae of the form

$$P_{i_j}(c) \implies \bigcirc M_{i_j}(c), \tag{61}$$

$$\exists x \bigwedge_{j=1}^{k} P_{i_j}(x) \implies \bigcirc \exists x \bigwedge_{j=1}^{k} M_{i_j}(x), \tag{62}$$

are called *derived* step clauses where $c \in const(\mathsf{P})$ and $j = 1 \ldots k$.

Let $\{\Phi_1 \implies \bigcirc \Psi_1, \ldots, \Phi_n \implies \bigcirc \Psi_n\}$ be a set of derived step clauses or original *ground* step clauses. Then

$$\bigwedge_{i=1}^{n} \Phi_i \Rightarrow \bigcirc \bigwedge_{i=1}^{n} \Psi_i$$

is called a *merged derived step clause*.

Let $\mathscr{A} \implies \bigcirc\mathscr{B}$ be a merged derived step clause, let $P_1(x) \implies \bigcirc M_1(x), \ldots, P_k(x) \implies \bigcirc M_k(x)$ be a subset of the original step clauses, and let $\mathscr{A}(x) \stackrel{\text{def}}{=} \mathscr{A} \wedge \bigwedge_{i=1}^{k} P_i(x)$, $\mathscr{B}(x) \stackrel{\text{def}}{=} \mathscr{B} \wedge \bigwedge_{i=1}^{k} M_i(x)$. Then

$$\forall x(\mathscr{A}(x) \implies \bigcirc\mathscr{B}(x))$$

is called a *full merged step clause*.

Next we define the temporal resolution rules for the expanding domain case where $\mathscr{A} \Rightarrow \bigcirc\mathscr{B}$ and $\mathscr{A}_i \Rightarrow \bigcirc\mathscr{B}_i$ denote merged derived step clauses, $\forall x(\mathscr{A}(x) \Rightarrow \bigcirc\mathscr{B}(x))$ and $\forall x(\mathscr{A}_i(x) \Rightarrow \bigcirc\mathscr{B}_i(x))$ denote full merged step clauses, and $\mathscr{U}$ denotes the universal part of the problem.

Step resolution rule w.r.t. $\mathscr{U}$:

$$\frac{\mathscr{A} \Rightarrow \bigcirc\mathscr{B}}{\neg\mathscr{A}} \text{ where } \mathscr{U} \cup \{\mathscr{B}\} \models \textbf{false}.$$

Initial termination rule w.r.t. $\mathscr{U}$*:*

$$\frac{}{\textbf{false}} \text{ if } \mathscr{U} \cup \mathscr{I} \models \textbf{false}.$$

Eventuality resolution rule w.r.t. $\mathscr{U}$*:*

$$\frac{\forall x(\mathscr{A}_1(x) \Rightarrow \bigcirc\mathscr{B}_1(x)) \quad \ldots \quad \forall x(\mathscr{A}_n(x) \Rightarrow \bigcirc\mathscr{B}_n(x)) \qquad \Diamond L(x)}{\forall x \bigwedge_{i=1}^{n} \neg\mathscr{A}_i(x)}$$

where $i \in \{1,\ldots,n\}$, the side conditions $\forall x(\mathscr{U} \wedge \mathscr{B}_i(x) \implies \neg L(x))$ and $\forall x(\mathscr{U} \wedge \mathscr{B}_i(x) \implies \bigvee_{j=1}^{n}(\mathscr{A}_j(x)))$ are both valid

Ground eventuality resolution rule w.r.t. $\mathscr{U}$*:*

$$\frac{\mathscr{A}_1 \Rightarrow \bigcirc\mathscr{B}_1 \quad \ldots \quad \mathscr{A}_n \Rightarrow \bigcirc\mathscr{B}_n \qquad \Diamond l}{\bigwedge_{i=1}^{n} \neg\mathscr{A}_i}$$

where $\mathscr{U} \wedge \mathscr{B}_i \models \neg l$ and $\mathscr{U} \wedge \mathscr{B}_i \models \bigvee_{j=1}^{n} \mathscr{A}_j$ for all $i \in \{1,\ldots,n\}$ are both valid.

Note that all the inference rules have side conditions which are first-order problems. In general, these side conditions will therefore only be semi-decidable and in the case a side condition is false, it may happen that the test of this side condition does not terminate. So, to ensure fairness we must make sure that each such test cannot indefinitely block the investigation of alternative applications of inference rules in a derivation. Let $\mathsf{P} = \langle \mathscr{U}, \mathscr{I}, \mathscr{S}, \mathscr{E} \rangle$ be a monodic temporal problem, then the set of formulae

$$\mathsf{P}^c = \langle \mathscr{U}, \mathscr{I}, \mathscr{S}, \mathscr{E} \cup \{\Diamond L(c) \mid \Diamond L(x) \in \mathscr{E}, c \in const(\mathsf{P})\} \rangle$$

is termed the *constant flooded form* of P. P^c is satisfiability equivalent to P.

Theorem 5 (see [8, Theorem 10])**.** *The rules of the calculus preserve satisfiability over expanding domains. If a monodic temporal problem* P *is unsatisfiable over expanding domains, then any* fair *derivation from* P^c *successfully terminates.*

Note the papers [8, 37] also describes how to extend this to constant domains.

The paper [37] provides a more machine-oriented clausal resolution calculus based on that in [8]. Two implementations of this calculus have been carried out TeMP [32] and TSPASS [39]. TeMP [32] implements [8] using the kernel of the first order prover Vampire [42] to carry out step resolution in a way similar to that described for TRP++. TSPASS [39] is also based on [8] but ensures that derivations are fair and uses the first order prover SPASS [6] to carry out step resolution. TSPASS was developed by Michel Ludwig and Ullrich Hustadt and is available for use at

`http://www.csc.liv.ac.uk/~michel/software/tspass/`

8 Extensions and Conclusions

Finally we consider possible extensions to the specifications and provide concluding remarks.

8.1 Extensions

Fault Tolerant Lifts. Imagine we have a (monodic) FOTL specification of our lift system. What if one or more lifts fail/break? Will we still be able to service the calls/requests? This would partially address (L6), the part of the specification relating to lift emergencies. What if there are an unbounded number of lifts? What if we do not know how many lifts actually fail, but know that there is only a finite number of failures? And how can we verify all these? We could try apply techniques for fault tolerant and infinite state verification such as [27, 28].

Uncertain Lifts. What if we are not certain exactly where a lift will move? We might specify this using additional *probabilistic* operators, for example:

$$(at_2 \wedge up) \Rightarrow \bigcirc(P_{\frac{17}{20}}\, at_3 \wedge P_{\frac{3}{20}}\, at_1)$$

following work such as [7].

Epistemic Lifts. What if our lifts *know* something about other floors/lifts? For example, maybe the lift does not visit a floor it *knows* another lift is visiting (or is planning to visit)? How might we specify this? We can extend our temporal specification with the $S5_n$ modalities typically used for logics of knowledge [19] and then carry out some proof on such specifications, see for example [18, 12, 40, 17].

Autonomous Lifts. What if lifts are autonomous? What if they have goals and beliefs of their own and can decide whether or not to move or open their doors[1]? Pressing a button to call a lift just sends a request to the lift, which can then choose to add a new goal of getting to the requested floor. What if there are multiple, conflicting goals? How can we specify such systems? How can we prototype them? How can we verify their properties? Here can apply the work in, for example, [13, 22, 23, 26].

8.2 Concluding Remarks

Here we have presented several areas relating to temporal specification, verification and execution we have been involved with. We have used the paper [3] to provide motivating examples or on which we have based the specifications and properties to apply our techniques. There has been much other work in this area we have not mentioned, for example, tableau calculi for PTL [30, 44, 43, 36] and implementations such as [35, 1], and tableau for FOTL [38]. All of this shows, however, that the field of temporal specification and verification remains vibrant, even after all these years since Howard Barringer's original lift specification.

References

[1] P. Abate and R. Gore. The tableaux work bench. In *TABLEAUX03: Automated Reasoning with Analytic Tableaux and Related Methods*, volume 2796 of *LNAI*, pages 230–236. Springer, 2003.

[2] H. Barringer. Up and Down the Temporal Way. Technical Report UMCS-85-9-3, Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, U.K., September 1985.

[1] c.f. "Hitchhikers Guide to the Galaxy".

[3] H. Barringer. Up and Down the Temporal Way. *The Computer Journal*, 30(2):134–148, 1987.

[4] H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens. METATEM: An Introduction. *Formal Aspects of Computing*, 7(5):533–549, 1995.

[5] H. Barringer, M. Fisher, D. Gabbay, R. Owens, and M. Reynolds, editors. *The Imperative Future: Principles of Executable Temporal Logic*. Research Studies Press, May 1996.

[6] C.Weidenbach, R.A.Schmidt, T.Hillenbrand, R.Rusev, and D.Topic. System description: Spass version 3.0. In *Proceedings of the 21st international conference on Automated Deduction: Automated Deduction*, CADE-21, pages 514–520. Springer, 2007.

[7] N. de Carvalho Ferreira, M. Fisher, and W. van der Hoek. Specifying and reasoning about uncertain agents. *International Journal of Approximate Reasoning*, 49(1):35–51, 2008.

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

[9] C. Dixon. Search Strategies for Resolution in Temporal Logics. In M. A. McRobbie and J. K. Slaney, editors, *Proceedings of the Thirteenth International Conference on Automated Deduction (CADE)*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 672–687, New Brunswick, New Jersey, July/August 1996. Springer.

[10] C. Dixon. Temporal Resolution using a Breadth-First Search Algorithm. *Annals of Mathematics and Artificial Intelligence*, 22:87–115, 1998.

[11] C. Dixon. Using Otter for Temporal Resolution. In *Advances in Temporal Logic*, volume 16 of *Applied Logic Series*, pages 149–166. Kluwer, 2000. Proceedings the Second International Conference on Temporal Logic (ICTL). ISBN 0-7923-6149-0.

[12] C. Dixon. Using Temporal Logics of Knowledge for Specification and Verification–a Case Study. *Journal of Applied Logic*, 4(1):50–78, 2006.

[13] C. Dixon, M. Fisher, and A. Bolotov. Clausal Resolution in a Logic of Rational Agency. *Artificial Intelligence*, 139(1):47–89, July 2002.

[14] C. Dixon, M. Fisher, and B. Konev. Is There a Future for Deductive Temporal Verification? In *Proceedings of TIME 2006 the Thirteenth International Symposium on Temporal Representation and Reasoning*, Budapest, Hungary, June 2006. IEEE Computer Society Press.

[15] C. Dixon, M. Fisher, and B. Konev. Temporal Logic with Capacity Constraints. In *Proc. of the 6th International Symposium on Frontiers of Combining Systems (FroCoS)*, pages 163–177. LNAI, 2007.

[16] C. Dixon, M. Fisher, and B. Konev. Tractable Temporal Reasoning. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 318–23. AAAI Press, 2007.

[17] C. Dixon, M. Fisher, and B. Konev. Taming the Complexity of Temporal Epistemic Reasoning. In *Proc. of the 7th International Symposium on Frontiers of Combining Systems (FroCoS)*, volume 5749 of *LNAI*, pages 198–213. Springer, 2009.

[18] C. Dixon, M. Fisher, and M. Wooldridge. Resolution for Temporal Logics of Knowledge. *Journal of Logic and Computation*, 8(3):345–372, 1998.

[19] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[20] M. Fisher. A Resolution Method for Temporal Logic. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 99–104, Sydney, Australia, August 1991. Morgan Kaufman.

[21] M. Fisher. A Normal Form for Temporal Logic and its Application in Theorem-Proving and Execution. *Journal of Logic and Computation*, 7(4):429–456, August 1997.

[22] M. Fisher. Temporal Development Methods for Agent-Based Systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 10(1):41–66, January 2005.

[23] M. Fisher. Agent Deliberation in an Executable Temporal Framework. *Journal of Applied Logic*, 9(4):223–238, 2011.

[24] M. Fisher. *An Introduction to Practical Formal Methods Using Temporal Logic*. Wiley, 2011.

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

[26] M. Fisher and A. Hepple. Executing Logical Agent Specifications. In *Multi-Agent Programming: Languages, Tools and Applications*, pages 1–27. Springer, 2009.

[27] M. Fisher, B. Konev, and A. Lisitsa. Practical Infinite-State Verification with Temporal Reasoning. In *Proceedings of Verification of Infinite State Systems and Security*, volume 1 of *NATO Security through Science Series: Information and Communication*, pages 91–100. IOS Press, 2006.

[28] M. Fisher, B. Konev, and A. Lisitsa. Temporal Verification of Fault-Tolerant Protocols. In *Methods Models and Tools for Fault Tolerance*, volume 5454 of *Lecture Notes in Computer Science*, pages 44–56. Springer, 2009.

[29] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. The Temporal Analysis of Fairness. In *Proceedings of the Seventh ACM Symposium on the Principles of Programming Languages*, pages 163–173, Las Vegas, Nevada, January 1980.

[30] G. D. Gough. Decision Procedures for Temporal Logic. Master's thesis, Department of Computer Science, University of Manchester, October 1984. Also University of Manchester, Department of Computer Science, Technical Report UMCS-89-10-1.

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

[32] U. Hustadt, B. Konev, A. Riazanov, and A. Voronkov. **TeMP**: A temporal monodic prover. In David A. Basin and Michaël Rusinowitch, editors, *Proceedings of the Second International Joint Conference on Automated Reasoning (IJCAR)*, volume 3097 of *LNAI*, pages 326–330. Springer, 2004.

[33] U. Hustadt and Boris Konev. TRP++ 2.0: A temporal resolution prover. In *Automated Deduction—CADE-19*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 274–278. Springer, 2003.

[34] U. Hustadt and R. A. Schmidt. Scientific benchmarking with temporal logic decision procedures. In D. Fensel, F. Giunchiglia, D. McGuinness, and M.-A. Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Eighth International Conference (KR'2002)*, pages 533–544. Morgan Kaufmann, 2002.

[35] G. Jaeger, P. Balsiger, A. Heuerding, S. Schwendimann, M. Bianchi, K. Guggisberg, G. Janssen, W. Heinle, F. Achermann, A. D. Boroumand, P. Brambilla, I. Bucher, and H. Zimmermann. LWB–The Logics Workbench 1.1. http://www.lwb.unibe.ch/, 2002. University of Berne, Switzerland.

[36] G.L.J.M. Janssen. *Logics for Digital Circuit Verification: Theory, Algorithms, and Applications*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 1999.

[37] 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.

[38] R. Kontchakov, C. Lutz, F. Wolter, and M. Zakharyaschev. Temporalizing tableaux. *Studia Logica*, 76:91–134, 2004.

[39] M. Ludwig and U. Hustadt. Implementing a Fair Monodic Temporal Prover. *AI Communications*, 23(2-3):68–96, 2010.

[40] C. Nalon and C. Dixon. Clausal Resolution for Normal Modal Logics. *Journal of Algorithms*, 62(3-4):117–134, 2007.

[41] Amir Pnueli. The Temporal Logic of Programs. In *Proc, 18th Symposium on the Foundations of Computer Science (FOCS)*, pages 46–57. IEEE Computer Society Press, 1977.

[42] A. Riazanov and A.Voronkov. The design and implementation of vampire. *AI Communications*, 15(2,3):91–110, August 2002.

[43] S. Schwendimann. A New One-Pass Tableau Calculus for PLTL. In Harrie de Swart, editor, *Proceedings of Tableaux 98*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 277–291. Springer-Verlag, 1998.

[44] P. Wolper. The Tableau Method for Temporal Logic: An Overview. *Logique et Analyse*, 110–111:119–136, June-Sept 1985.