School of Informatics, University of Edinburgh

Centre for Intelligent Systems and their Applications

A modular, tactic-based approach to first-order temporal theorem proving

by

Claudio Castellini, Alan Smaill

Informatics Research Report EDI-INF-RR-0152

School of Informatics **October 2000**
http://www.informatics.ed.ac.uk/

A modular, tactic-based approach to first-order temporal theorem proving

Claudio Castellini, Alan Smaill

Informatics Research Report EDI-INF-RR-0152

SCHOOL *of* INFORMATICS
Centre for Intelligent Systems and their Applications

October 2000

appears in Proceedings of ICTL 2000, Dresden, Germany

Abstract :

In system specification and formal verification it is a central issue to deal with temporal logics. In particular, First-Order Temporal Logics (FOTLs) are needed whenever the modelled systems are infinite-state. Reasoning in FOTLs is hard and few approaches have so far proved effective. In this paper we propose a novel approach to FOTLs, in the style of labelled deduction in Quantified Modal Logics, which is modular in the structure of time and can therefore be adapted to a wide class of FOTLs. Moreover, we present a tactic-based temporal theorem prover enforcing this approach, obtained by applying Amy Felty's work on higher-order theorem proving in lambda-Prolog. Finally, we show some promising experimental results.

Keywords : first-order temporal logic, tactic theorem proving, lambda-Prolog, sequent calculus

Copyright © 2002 by The University of Edinburgh. All Rights Reserved

The authors and the University of Edinburgh retain the right to reproduce and publish this paper for non-commercial purposes.

Permission is granted for this report to be reproduced by others for non-commercial purposes as long as this copyright notice is reprinted in full in any reproduction. Applications to make other use of the material should be addressed in the first instance to Copyright Permissions, Division of Informatics, The University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, Scotland.

A modular, tactic-based approach to first-order temporal theorem proving

Claudio Castellini and **Alan Smaill** *

Abstract

In system specification and formal verification it is a central issue to deal with temporal logics. In particular, First-Order Temporal Logics (FOTLs) are needed whenever the modeled systems are infinite-state. Reasoning in FOTLs is hard and few approaches have so far proved effective. In this paper we propose a novel approach to FOTLs, in the style of labelled deduction in Quantified Modal Logics, which is modular in the structure of time and can therefore be adapted to a wide class of FOTLs. Moreover, we present a tactic-based temporal theorem prover enforcing this approach, obtained by applying Amy Felty's work on higher-order theorem proving in λProlog. Finally, we show some promising experimental results.

Keywords: first-order temporal logic, tactic theorem proving, lambda-Prolog, sequent calculus

1 Introduction

Temporal logics are extensions of classical logic, dealing with the concept of time and how properties of a dynamic system change through time ([Gabbay *et al.*, 1993]). They are used in specification and verification of multi-agent, concurrent and reactive systems, programs, circuits and protocols ([Manna and Pnueli, 1992]). Theorem proving in temporal logics can therefore formally ensure that a system respects its requirements, improving safety and effectiveness of a wide class of systems.

While several effective approaches have been developed for propositional temporal logics (for instance symbolic model checking, see [Burch *et al.*, 1992]), the situation is quite hard when we switch to First-Order Temporal Logics (FOTLs), which constitute a richer class of languages, mainly undecidable, and nevertheless necessary if we want to deal with infinite-state systems.

In this paper we propose a modular approach to first-order temporal theorem proving, in that:

1. much in the style of what is done in [Basin *et al.*, 1998] for Quantified Modal Logics, we give a labelled, modular presentation of FOTLs, setting up a sequent calculus, $\mathcal{TL}$, which allows different structures of time to be dealt with;
2. we describe a modular, tactic-based temporal theorem prover, FTL, which supports the approach described above, thanks to the logical module system of the language λProlog ([Nadathur and Miller, 1995]). The implementation closely follows Amy Felty's work on higher-order, tactic-based theorem proving in λProlog ([Felty, 1993]).

The choice of a modular approach to FOTLs in a labelled modal-logic style comes from the consideration that most FOTLs share the same syntax and only differ in the underlying structure of time (linear, branching, discrete, continuous etc.). Labelled deduction, in general, has recently become of great interest ([Gabbay, 1996]) and we think it is worthwhile to investigate its applicability to temporal logics. λProlog is a sensible choice for implementing tactic-based theorem provers for temporal logics ([Felty and Thery, 1996]).

The paper is structured as follows: Section 2 describes the sequent calculus $\mathcal{TL}$; Section 3 describes the theorem prover FTL; Section 4 shows some interesting experimental results we have obtained with FTL; finally, in Section 5, we briefly overview related work, we draw some conclusions and outline our future work.

2 A modular presentation of FOTLs

In this Section we describe the object logics we deal with and the associated sequent calculi.

In the style of [Basin *et al.*, 1998], where a family of quantified normal modal logics is presented which is modular with respect to the properties of the Kripke

*Division of Informatics, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, Scotland. e-mail: {claudio,smaill}@dai.ed.ac.uk

frame, we present FOTLs modularly with respect to the structure of time. Each FOTL consists of a first-order language augmented with a basic, easily extensible set of temporal operators; the formulae are labelled and the logic associated with labels (labelling algebra) enforces a Kripke-style semantics.

Accordingly, the associated sequent calculus $\mathcal{TL}$ consists of a core set of rules for the basic normal modal logic **K**, plus an additional set of rules for reasoning on time labels. Most of these rules are enforced by a single rule called *entailment*. $\mathcal{TL}$ can actually be seen as a modular family of sequent calculi.

The labelling algebra basically consists of a binary relation $\preceq$, whose properties determine the structure of time; we then augment it with functions whenever the structure of time allows their definition, for instance in the serial, convergent, or discrete case. The σ function is particularly interesting, since it plays the role of the "successor" instant in the discrete case, and has two associated sequent rules dealing with the $\bigcirc$ ("next time") operator.

Since the long-term aim of this work is to provide a framework for system verification, which usually involves discrete time, we include this function and rules into the labelling algebra since the beginning. In the following, when the structure of time is not intended as discrete, it suffices to ignore the $\bigcirc$ rules and the σ function.

2.1 Syntax

Syntax can be defined separately for *static formulae* and *time labels*, since they have no common symbols.

Static formulae. Let $\mathcal{V}$, $\mathcal{F}$ and $\mathcal{P}$ be countable sets of variable symbols, function symbols and predicate symbols, and let $\zeta : \mathcal{F} \cup \mathcal{P} \to \mathbb{N}$ (arity); then static formulae are inductively defined as follows:

1. a *static term* is either *(i)* a variable symbol $v \in \mathcal{V}$, or *(ii)* a function symbol $f \in \mathcal{F}$ applied to an appropriate tuple of static terms, $f(s_1, \ldots, s_{\zeta(f)})$;
2. a *static atom* is a predicate symbol $p \in \mathcal{P}$, applied to an appropriate tuple of static terms, $p(s_1, \ldots, s_{\zeta(p)})$;
3. a *static formula* is a combination of static atoms and/or static formulae via standard Boolean connectives and quantifiers ($\vee$, $\neg$, $\forall$) and unary *temporal operators* $\Box$ ("always"), $\Diamond$ ("eventually"), $\bigcirc$ ("at next instant"). Other Boolean connectives can be defined from these as usual.

Examples of static formulae are: $\forall x.\Diamond\exists y.q(x,y)$, $\Box p(a)$ and $\bigcirc(p_1 \wedge p_2) \leftrightarrow (\bigcirc p_1 \wedge \bigcirc p_2)$, where we assume that $p(\cdot)$, $q(\cdot,\cdot)$, p_1 and p_2 are in $\mathcal{P}$ and a is in $\mathcal{F}$. Note that propositional symbols (Boolean variables) such as p_1 and p_2 can be regarded as nullary predicate symbols, while constants such as a can be regarded as nullary function symbols.

Time labels and **constraints.** Let $\mathcal{T}$ be a countable set of variable symbols such that $\mathcal{T} \cap \mathcal{V} = \emptyset$, and let 0 be a constant, σ a unary function and $\preceq$ a binary relation.

A *time label* is either *(i)* 0, or *(ii)* a variable symbol $t \in \mathcal{T}$, or *(iii)* the application of σ to a time label. Examples of time labels are 0, $\sigma(0)$ and $\sigma(\sigma(t_1))$.

A *constraint* is a formula of the form $\tau_1 \preceq \tau_2$, where τ_1, τ_2 are time labels.

Labelled formulae. The basic unit of information in our object logic is a *labelled formula* $\varphi@\tau$, informally meaning "at time τ, it is the case that φ" where φ is a static formula and τ is a time label.

Examples of labelled formulae are $p(a)@0$, $p_1 \wedge p_2@\sigma(\sigma(0))$ and $\forall x.p(x) \supset p(a)@\tau_1$. The @ operator is intended to bind less tightly than any other operator, therefore we will systematically omit parentheses around its arguments. The previous example $\forall x.p(x) \supset p(a)@\tau_1$ is read as $[\forall x.p(x) \supset p(a)]@\tau_1$.

Formulae. Labelled formulae and constraints are globally called *formulae*.

2.2 Semantics

The notion of truth for a formula is given by a *model* **M**. A model is a tuple

$$\mathbf{M} = \langle \mathcal{W}, \mathcal{D}, I_t, I_l, \alpha \rangle$$

where:

- $\mathcal{W}$ and $\mathcal{D}$ are disjoint nonempty sets (resp. the *set of instants* and the *domain*);
- the *time interpretation* I_t gives a meaning to $\preceq$, 0 and σ, and to all functions eventually introduced in the labelling algebra;
- the *logical interpretation* I_l gives a meaning to logical functions and predicates; that is, given a world w, it maps symbols in $\mathcal{F} \cup \mathcal{P}$ to a function $I_l(w, f)$ or a predicate $I_l(w, p)$ over $\mathcal{D}$;
- $\alpha : \mathcal{W} \times \mathcal{V} \to \mathcal{D}$ assigns appropriate values to variable symbols in $\mathcal{D}$, for each world.

The definitions above are largely based upon [Abadí and Manna, 1990]; as in that case, we distinguish between *rigid* and *flexible* symbols.

Note that we employ the symbol $\mathcal{W}$ ("worlds") for the set of time instants, suggesting that this semantics is basically a Kripke semantics. The $\preceq$ relation is an accessibility relation. Since we do not have any rules or assumptions about the existence of individuals at each instant, we are assuming the domain to be the same in all possible worlds.

We regard first-order temporal logics as quantified modal logics on particular Kripke frames, whose structure is determined by I_t. For instance, on a linear,

discrete time structure, time is isomorphic to $\mathbb{N}$ and $I_t(0) = 0 \in \mathbb{N}$, $I_t(\sigma)$ is the Peano successor function and $I_t(\preceq)$ is the $\leq$ relation over the naturals.

The truth of a formula is evaluated inductively as follows:

- if the static formula is atomic, i.e. the formula is of the form $p(s_1, \ldots, s_{\zeta(p)})@\tau$, it is evaluated inductively by means of I_l and α in the standard first-order way;
- if the static formula's top connective is not temporal, we follow the standard first-order semantic interpretation;
- for constraints and static formula whose top connective is temporal, we have

$$\begin{array}{lll} \mathbf{M} \models \tau_1 \preceq \tau_2 & \text{iff} & (I_t(\tau_1), I_t(\tau_2)) \in I_t(\preceq) \\ \mathbf{M} \models \bigcirc\varphi@\tau & \text{iff} & \mathbf{M} \models \varphi@\sigma(\tau) \\ \mathbf{M} \models \Box\varphi@\tau & \text{iff} & \text{for all } t', \mathbf{M} \models \tau \preceq t' \\ & & \quad \text{implies } \mathbf{M} \models \varphi@t' \\ \mathbf{M} \models \Diamond\varphi@\tau & \text{iff} & \text{there exists } t' \text{ such that} \\ & & \quad \mathbf{M} \models \tau \preceq t' \text{ and} \\ & & \quad \mathbf{M} \models \varphi@t' \end{array}$$

2.3 The sequent calculus $\mathcal{TL}$

The sequent calculus $\mathcal{TL}$ is directly modelled upon Gentzen's sequent calculus LK for classical first-order logic ([Gentzen, 1969]); rules for the temporal operators are introduced mimicking the operators' own semantic definitions:

$$\Box\varphi@\tau \text{ iff for all } \tau',\ \tau \preceq \tau' \text{ implies } \varphi@\tau'$$

$$\Diamond\varphi@\tau \text{ iff for some } \tau',\ \tau \preceq \tau' \text{ and } \varphi@\tau'$$

$$\bigcirc\varphi@\tau \text{ iff } \varphi@\sigma(\tau)$$

They are obtained by composing the appropriate classical rules, as in a first-order translation. For example, left-$\Box$ is the composition of left-$\forall$ and left-$\supset$, as is shown below:

$$\dfrac{\dfrac{\dfrac{\Gamma, \varphi@T \longrightarrow \Delta \quad \Gamma \longrightarrow \tau \preceq T, \Delta}{\Gamma, \tau \preceq T \supset \varphi@T \longrightarrow \Delta}\, l\supset}{\Gamma, \forall\tau'.\tau \preceq \tau' \supset \varphi@\tau' \longrightarrow \Delta}\, l\forall}{\Gamma, \Box\varphi@\tau \longrightarrow \Delta}\, \stackrel{\text{def}}{=}$$

$\mathcal{TL}$ has three sets of rules (see Table 1):

1. closing rules: axiom and entailment;
2. static rules, operating on propositional connectives and quantifiers (for conciseness, we only show the rules for $\vee$, $\neg$ and $\forall$;
3. temporal rules, operating on temporal operators $\Box$, $\Diamond$ and $\bigcirc$.

Temporal rules $r\Box$ and $l\Diamond$ involve first-order quantifiers of *universal force*, while temporal rules $r\Diamond$ and $l\Box$ involve first-order quantifiers of *existential force*. Therefore, $r\Box$ and $l\Diamond$ have provisos on their applications, exactly as $r\forall$ and $l\exists$ do.

Throughout the presentation of rules, we employ the symbol t for time eigenvariables (i.e., new time terms introduced by the application of universal force temporal rules) and the symbol T for time metavariables (i.e., new time terms introduced by the application of existential force temporal rules). This reflects the lazy approach to unification in sequent calculi which makes the proof search much more efficient (see e.g. [Shankar, 1992] for more details).

We choose 0 as the initial label, intending the starting instant. Static rules preserve time labels: labels in the conclusions are the same as in the premises and are propagated to subformulae. This is not the case for temporal rules, which enforce reasoning on time by acting on labels.

The entailment rule ENT allows a branch to be closed if the constraints among the antecedent formulae entail at least one constraint among the consequents. The entailment rule represents opaque time reasoning and enforces modularity: if we change the structure of time we just need to update the machinery associated with the entailment procedure. This rule can enforce properties such as seriality (for the modal logic **D**), reflexivity (for **T**), transitivity (for **S4**) and convergency (for **S4.2**).

This neat separation between temporal and logical reasoning is possible as long as the required properties are expressible as a Horn theory ([Basin *et al.*, 1997]), but if they involve disjunction, as is the case with linearity (for the modal logic **S4.3**), an additional rule must be explicitly inserted into the calculus. Still, this rule just operates on labels, therefore retaining a good degree of modularity. Such a rule is depicted in Table 2 and is equivalent to the application of the axiom $\forall\tau_0\tau'\tau''.(\tau_0 \preceq \tau' \wedge \tau_0 \preceq \tau'') \supset (\tau' \preceq \tau'' \vee \tau'' \preceq \tau')$.

3 A tactic theorem prover for $\mathcal{TL}$

$\mathcal{TL}$ is currently being implemented in λProlog ([Nadathur and Miller, 1995]); the result so far is a tactic-based, first-order temporal theorem prover we have called FTL. The choice of λProlog was motivated by some of its features, which are particularly well-suited for tactic-based theorem proving.

In this Section we first give an outline of the system and then we show how a simple automatic compound tactic is implemented.

3.1 Overview of FTL

FTL mainly consists of five λProlog *modules*, four of which are depicted in Figure 1.

$$\frac{}{\Gamma, \varphi@\tau \longrightarrow \varphi@\tau, \Delta}\ \mathsf{AX} \qquad \frac{\Gamma \vdash \mathcal{C}}{\Gamma \longrightarrow \mathcal{C}, \Delta}\ \mathsf{ENT}$$

$$\frac{\Gamma, \varphi@\tau \longrightarrow \Delta \quad \Gamma, \psi@\tau \longrightarrow \Delta}{\Gamma, \varphi \vee \psi@\tau \longrightarrow \Delta}\ l\vee \qquad \frac{\Gamma \longrightarrow \varphi@\tau, \psi@\tau, \Delta}{\Gamma \longrightarrow \varphi \vee \psi@\tau, \Delta}\ r\vee$$

$$\frac{\Gamma \longrightarrow \varphi@\tau, \Delta}{\Gamma, \neg\varphi@\tau \longrightarrow \Delta}\ l\neg \qquad \frac{\Gamma, \varphi@\tau \longrightarrow \Delta}{\Gamma \longrightarrow \neg\varphi@\tau, \Delta}\ r\neg$$

$$\frac{\Gamma, \varphi[C/x]@\tau \longrightarrow \Delta}{\Gamma, \forall x\varphi@\tau \longrightarrow \Delta}\ l\forall \qquad \frac{\Gamma \longrightarrow \varphi[a/x]@\tau, \Delta}{\Gamma \longrightarrow \forall x\varphi@\tau, \Delta}\ r\forall$$

$$\frac{\Gamma, \varphi@T \longrightarrow \Delta \quad \Gamma \longrightarrow \tau \preceq T, \Delta}{\Gamma, \Box\varphi@\tau \longrightarrow \Delta}\ l\Box \qquad \frac{\Gamma, \tau \preceq t \longrightarrow \varphi@t, \Delta}{\Gamma \longrightarrow \Box\varphi@\tau, \Delta}\ r\Box$$

$$\frac{\Gamma, \tau \preceq t, \varphi@t \longrightarrow \Delta}{\Gamma, \Diamond\varphi@\tau \longrightarrow \Delta}\ l\Diamond \qquad \frac{\Gamma \longrightarrow \varphi@T, \Delta \quad \Gamma \longrightarrow \tau \preceq T, \Delta}{\Gamma \longrightarrow \Diamond\varphi@\tau, \Delta}\ r\Diamond$$

$$\frac{\Gamma, \varphi@\sigma(\tau) \longrightarrow \Delta}{\Gamma, \bigcirc\varphi@\tau \longrightarrow \Delta}\ l\bigcirc \qquad \frac{\Gamma \longrightarrow \varphi@\sigma(\tau), \Delta}{\Gamma \longrightarrow \bigcirc\varphi@\tau, \Delta}\ r\bigcirc$$

Provisos: eigenvariables a and t cannot appear in the conclusion of $r\forall$, $l\Diamond$ and $r\Box$

Table 1: the calculus $\mathcal{TL}$.

$$\frac{\Gamma, \tau_0 \preceq \tau', \tau_0 \preceq \tau'', \tau' \preceq \tau'' \longrightarrow \Delta \quad \Gamma, \tau_0 \preceq \tau', \tau_0 \preceq \tau'', \tau'' \preceq \tau' \longrightarrow \Delta}{\Gamma, \tau_0 \preceq \tau', \tau_0 \preceq \tau'' \longrightarrow \Delta}\ \mathsf{LIN}$$

Table 2: the rule for linearity.

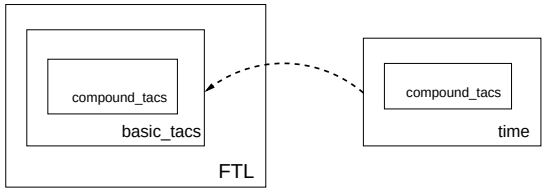

Figure 1: scheme of FTL.

A λProlog module is basically a piece of λProlog code on its own, and it can *accumulate* or *import* other modules. An accumulated module is textually included into a module, whereas an imported module dynamically makes its clauses available to a module. This mechanism provides a clean logical account of modular programming ([Nadathur and Miller, 1995]) and is particularly well suited in this case, where we want the machinery for reasoning on time to be opaque with respect to the prover.

In Figure 1 each box represents a module; inclusion of boxes into each other indicates accumulation, while the dashed arrow indicates importing. Module time, imported into module basic_tacs, enforces the $\vdash$ relation in the ENT rule.

We give now a brief description of each module, along with some remarks on their significance and characteristics. Module syntax, described below, is not depicted in Figure 1 for the sake of readability; it is accumulated by module compound_tacs.

Module syntax

Module syntax contains the `kind` and `type` declarations necessary to build the object language, plus some basic predicates for tactic theorem proving. In particular, kinds `i` and `time` are declared, which represent the two sorts of the elements of the domain and the instants of time.

```
kind i          type.
kind time       type.
```

All meta-level predicates are typed accordingly, so that there can be no confusion about the application of rules to one or to the other sort. This way, the need for a set of first-order axioms enforcing the disjointness of

$\mathcal{W}$ and $\mathcal{D}$ (as pointed out, e.g., in [Gabbay *et al.*, 1993]) is avoided.

Analogously, the kinds of static formulae and formulae are defined (the kind of time labels is `time` itself), and @ is declared as a constructor of formulae out of static formulae and time labels:

```
kind sformula   type.
kind formula    type.

type   @  sformula -> time -> formula.
```

Object-level operators are defined as static formulae constructors. For instance,

```
type neg          sformula -> sformula.
type and, or      sformula -> sformula -> sformula.
type forall       (i -> sformula) -> sformula.
type glob, evt    sformula -> sformula.
```

where `glob` and `evt` denote $\Box$ and $\Diamond$, respectively.

Finally, in the style of [Felty, 1993], we define sequents as pairs of lists of formulae, and we build *goals* out of a sequent and a proof:

```
kind  goal      type.
kind  proof     type.
kind  sequent   type.

type  -->       (list formula) -> (list formula) ->
                sequent.
type  proves    proof -> sequent -> goal.
```

The idea is that we want to establish a relation between a sequent and its proof (namely, that the proof proves the sequent). To do this, we try to reach the associated goal by means of steps enforced by tactics.

Modules basic_tacs **and** compound_tacs

Modules basic_tacs and compound_tacs enforce tactics. A *tactic* is a predicate linking two goals, stating one is reached from the other by means of some technique; in the case of *basic* tactics, it is the bare application of a sequent rule; in the case of *compound* tactics, it is the composition of more tactics.

Compound tactics. Compound tactics are in a sense more general than basic tactics, since they are independent from the object logic considered; in fact, module compound_tacs is accumulated both by basic_tacs and time. This reflects the need for compound tactics both in logical and in temporal reasoning.

Compound tactics enforce composition, conditional and exhaustive application of tactics. Since they are standard ([Felty, 1993]), we describe them very briefly:

- `then_tac Tac1 Tac2` enforces composition of two tactics, applying `Tac1` to an input goal and then `Tac2` to the obtained intermediate goal;
- `orelse_tac Tac1 Tac2` enforces conditional application of two tactics, applying `Tac1` and, upon failure, `Tac2`;
- `app_list_tac Tac_list` enforces application of a list of tactics by means of `orelse_tac`, so that the first successful tactic in the list is applied;
- `repeat_tac Tac` uses `then_tac` and `orelse_tac` to exhaustively apply tactic `Tac` to a goal, and fails if this is not possible;
- `complete_tac Tac` calls upon `repeat_tac` with `true_goal` as the output goal, therefore either the input goal is solved or we have failure.

Using these standard compound tactics, we are able to define exhaustive application of a list of tactics and iterative deepening proof search (tactics `exhaust_tac` and `idfs_tac`).

Basic tactics. Basic tactics enforce sequent rules. A *rule* in this setting is a proof constructor, that is, a mapping from a proof to another proof. As an example, this is the definition of the basic tactic `r_imp_tac`, which encapsulates rule $r\supset$, called `r_imp` (rows are numbered for the comfort of the reader):

```
type r_imp          proof -> proof.
type r_imp_tac      goal -> goal -> o.

(0) r_imp_tac
(1)     ((r_imp P) proves (Gamma --> Delta))
(2)     (P proves ( ((Phi @ Tau)::Gamma) -->
(3)                 ((Psi @ Tau)::Delta'))) :-
(4)     member (Phi imp Psi @ Tau) Delta,
(5)     delete (Phi imp Psi @ Tau) Delta Delta'.
```

- if the conclusion set Δ contains $\varphi\supset\psi@\tau$ then we remove it and call the new conclusion set Δ' (rows 4 and 5);
- then, proof `(r_imp P)`, obtained by application of rule $r\supset$ to proof P, proves sequent $\Gamma\longrightarrow\Delta$ (row 1),
- provided that proof P proves sequent $\Gamma,\varphi@\tau\longrightarrow\psi@\tau,\Delta'$ (rows 2 and 3).

Since this tactic enforces a static rule, the time label τ is preserved from the premises to the conclusions, while this is not the case with temporal basic tactics, such as the one for rule $l\Box$:

```
type l_glob     time -> proof -> proof -> proof.
type l_glob_tac goal -> goal -> o.

l_glob_tac
  ((l_glob Tau P1 P2) proves (Gamma --> Delta))
  (and_goal
    (P1 proves (((Phi @ T)::Gamma') --> Delta))
    (P2 proves (Gamma' -->
        ((constraint (Tau wbefore T))::Delta)))) :-
  member (glob Phi @ Tau) Gamma,
  delete (glob Phi @ Tau) Gamma Gamma'.
```

in which the elimination of a quantifier of existential force (the $\forall$ on the left embedded into rule $l\Box$) causes the introduction of the time metavariable T, and $\varphi@T$

replaces $\Box\varphi@\tau$[1]. Two branches are opened by means of the `and_goal` goal constructor, which succeeds if and only if both subgoals succeed, and constraint $\tau \preceq T$ is put in the other branch. This rule is basically the composition of a $l\forall$ rule and a $l\supset$ rule.

Notice that in this case we include the time label τ, at which we get rid of the $\Box$ operator, into the proof itself. Tactics are, in fact, flexible enough to allow incorporation of useful information into proofs.

Module time

Module time contains the machinery for reasoning on constraints and unification between time labels.

Types for 0 (`z`) and $\preceq$ (`wbefore`) are straightforward from the definitions. Again, the λProlog typing system is used to ensure that these objects only operate on time instants. Predicate `entails` is the entailment procedure:

```
type z          time.
type wbefore    time -> time -> o.

type entails    proof -> (list formula)
                      -> (list formula) -> o.
```

The entailment relies on the tactic mechanism to enforce a first-order proof search for the desired sequent. The list of supplied tactics enforces the desired properties of the Kripke frame:

```
entails Proof Gamma Delta :-
  exhaust_tac (refl_tac::trans_tac::nil)
              (Proof proves (Gamma --> Delta))
              true_goal.
```

Correspondence between properties of the Kripke frame and modal axioms ([van Benthem, 1984]) is reflected in the ability of `entails` to detect the a property. In the previous example, the use of `refl_tac` and `trans_tac` force the frame to be reflexive and transitive. As we expect, given this list of tactics, axioms **T** (reflexivity), **D** (seriality) and **4** (transitivity) are provable.

Module time can be augmented this way up to modal logic **S4.2**. Unfortunately, when it comes to linearity (**S4.3**) time tactics interact with logical reasoning and therefore they must be placed in the `basic_tacs` module. Moreover, discreteness is implemented using tactics which relate the accessibility relation to the σ function.

Module time can be tuned *ad libitum* to fit the desired properties of the time structure. Prospectively, it can be also replaced by an external procedure. In [McGuire *et al.*, 1994], for instance, it is suggested that a Presburger Arithmetic decision procedure can be needed for reasoning in a linear, discrete time temporal logic.

[1] Conversely, quantifiers of universal force are implemented via λProlog's meta-level universal quantification ([Nadathur and Miller, 1995; Felty, 1993]).

Module FTL

Module FTL provides the prover's top commands, which enforce both an *interactive* and an *automatic* mode. Proof search is goal-directed: given an input static formula φ to be checked for validity, FTL sets up the goal

```
P proves (nil --> Phi @ 0)
```

where `P` is an uninstantiated proof object. Iteratively, depending on the shape of the sequent in the goal, a tactic is chosen and applied, getting a new goal whose sequent reflects the application of the specified rule(s). Soundness of this approach is assured by the soundness of basic tactics, each of which enforces a sequent rule.

We go on like this in a goal-directed fashion. If every branch is eventually closed, the final result is a fully instantiated proof object — the *proof* of the original formula.

3.2 A simple automatic compound tactic

In the interactive mode, it is the user who feeds the tactics to FTL at each proof step, exactly as in most interactive provers; in the automatic mode, a simple compound tactic exhaustively applies basic tactics from an ordered sequence.

Following [Wallen, 1989], we group rules into four sets[2]:

- α propositional and temporal rules which do not cause the proof tree to branch: $r\supset$, $r\vee$, $l\wedge$, $l\leftrightarrow$, $l\neg$ $r\neg$, $l\bigcirc$ and $r\bigcirc$;
- β propositional rules which cause the proof tree to branch: $l\supset$, $r\leftrightarrow$, $r\wedge$ and $l\vee$;
- γ rules for quantifiers of existential force and temporal rules which involve quantifiers of existential force: $l\forall$, $r\exists$, $l\Box$ and $r\Diamond$;
- δ rules for quantifiers of universal force and temporal rules which involve quantifiers of universal force: $r\forall$, $l\exists$, $r\Box$ and $l\Diamond$.

The idea is *(i)* to *check immediately* whether a branch can be closed (closing rules AX and ENT); *(ii)* to *apply as soon as possible* rules which do not branch, nor introduce new terms (α); *(iii)* to *delay as much as possible* the application of branching rules (β). Rules which introduce new terms (first δ rules and then γ rules, so as to maximise the scope of eigenvariables) can be applied midway. So we choose the order: closing, α, δ, γ and β.

The automatic mode is enforced by tactic `auto_tac`, which reads as follows:

```
type auto_tac      goal -> goal -> o.

auto_tac InGoal OutGoal :-
```

[2] Rules $l\leftrightarrow$ and $r\leftrightarrow$ just split the $\leftrightarrow$ connective into the conjunction of two $\supset$ connectives.

```
exhaust_tac ( <... closing rules ...> ::
              <... alpha rules ...> ::
              <... delta rules ...> ::
              <... gamma rules ...> ::
              <... beta rules ...> )
InGoal OutGoal.
```

`exhaust_tac` is given an ordered sequence of basic tactics (closing, α, δ, γ, β). Since the sequence comprises all basic tactics, including closing rules, when no tactic from the sequence can be applied the branch cannot be closed, neither any basic tactic can be further applied. `true_goal` cannot be reached and tactic `auto_tac` fails.

This order clearly achieves great efficiency: branching is postponed, making the search space smaller. But unfortunately it also introduces ordering problems ([Wallen, 1989]). Formula $\forall x.p(x) \supset (p(a) \wedge p(b))$, for instance, cannot be proved if rule $l\forall$ is employed before rule $r\wedge$, as is the case.

To overcome this problem we allow backtracking on the choice of the rule to apply. This way `auto_tac` tries all possible orderings of basic rules. When the above formula is given to FTL, first the "wrong" order is tried and a failure is reached, then backtracking happens and the "right" order is applied, leading to success.

4 Experimental results

We have first tested FTL on a set of easy valid formulae taken from [Manna and Pnueli, 1981] and [Abadí and Manna, 1990]. Then we have been able to prove the correctness of the specification of a simple Boolean circuit.

Rather than giving timings for the formulae we have been able to prove, which would be of little significance to the scope of this paper, we prefer to present some of the most interesting examples, along with sample proofs, and to give explanations about FTL's behaviour.

All proofs are obtained *automatically* enforcing reflexivity, transitivity and linearity plus axioms for σ.

Hereafter we assume that p, p_1, p_2 are nullary predicates and q, r are unary predicates, and that a is a constant of the domain. The initial label @0 is omitted to make the notation lighter and 1 is written in place of $\sigma(0)$.

4.1 Valid formulae

FTL can prove automatically most of the formulae from 1 to 49 in [Manna and Pnueli, 1981]. The ones we could not prove contain the "until" operator or require a naive form of induction, which we won't discuss here for lack of space. Here is a list of the most interesting successful ones.

$\boxed{\Box\neg p \leftrightarrow \neg\Diamond p}$. Duality between $\Box$ and $\Diamond$ is proved in a natural way. Here goes the proof of the forward direction: FTL uses α-rules to get to

$$\boxed{\dfrac{\dfrac{\dfrac{\Box\neg p, p@t, 0 \preceq t \longrightarrow}{\Box\neg p, \Diamond p \longrightarrow}\, l\Diamond}{\Box\neg p \longrightarrow \neg\Diamond p}\, r\neg}{\longrightarrow \Box\neg p \supset \neg\Diamond p}\, r\supset}$$

where eigenvariable t has been introduced as the witness time for p, once the $\Diamond$ operator has been stripped away. Continuing the proof β-rule $l\Box$ is used to branch; metavariable T is introduced.

$$\boxed{\dfrac{\dfrac{\overline{p@t, 0 \preceq t \longrightarrow p@T}}{p@t, 0 \preceq t, \neg p@T \longrightarrow}\, l\neg \quad \overline{p@t, 0 \preceq t \longrightarrow 0 \preceq T}}{p@t, 0 \preceq t, \Box\neg p \longrightarrow}\, l\Box}$$

On the left branch, closing by axiom is detected with label unification $\{T \leftarrow t\}$; this unification is carried over to the right branch, where the trivial entailment $0 \preceq t \longrightarrow 0 \preceq t$ is solved.

$\boxed{\Box p \supset \bigcirc\Box p}$. This formula is proved by unfolding $\Box$ on both sides and label unification. The entailment problem $1 \preceq t \longrightarrow 0 \preceq t$ is correctly solved. Note that the converse formula cannot be proved because the entailment problem $0 \preceq t \longrightarrow 1 \preceq t$ is queried and rejected.

$\boxed{\Box(p_1 \wedge p_2) \leftrightarrow (\Box p_1) \wedge (\Box p_2)}$. An interesting case where backtracking in proof search is required. FTL has the initial choice between applying $l\Box$ or $r\wedge$; given the ordering in `auto_tac`, rule $l\Box$ is employed *before* rule $r\wedge$, leading to a blocked proof. FTL then backtracks to the point where $l\Box$ was used and tries to apply the next applicable rule, that is, $r\wedge$. This way, the $\Box$ on the left is preserved in both branches and the proof succeeds.

$\boxed{[\bigcirc(r(a) \vee q(a)) \wedge \Box(\forall x.\neg r(x))] \supset \bigcirc q(a)}$. This is Example 4.5 from [Abadí and Manna, 1990]. The full proof is shown in Figure 2. FTL uses rules $r\supset, l\wedge, l\bigcirc, r\bigcirc, l\Box$ to get two branches. After $l\forall, l\neg, l\vee$ are applied to the left branch, two further branches are opened, which can be immediately closed by axiom using the unifier $\{C \leftarrow a, T \leftarrow 1\}$. Extending this unification to the right branch yields constraint $0 \preceq 1$, which is trivial. Note that the temporal resolution-style proof given in [Abadí and Manna, 1990] is quite clumsy if compared to this one.

4.2 Verification of a set-reset flip-flop

Using tactic `auto_tac` FTL can prove the correctness of the specification of a set-reset flip-flop with respect to a requirement on the outputs (it must be noted that the proof requires basically no temporal reasoning). Here we describe the problem and the proof.

The scheme of a set-reset flip-flop is visible in Figure 3. The intended behaviour of this small circuit is summarised as follows: if S is true and R is false, P must be true and Q must be false; vice-versa if S is false and

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\boxed{\text{OK}}}{r(a)@1 \longrightarrow r(C)@T, q(a)@1}\,\mathsf{AX} \quad \dfrac{\boxed{\text{OK}}}{q(a)@1 \longrightarrow r(C)@T, q(a)@1}\,\mathsf{AX}}{(r(a) \vee q(a))@1, \neg r(C)@T \longrightarrow q(a)@1}\,l\neg, l\vee}{(r(a) \vee q(a))@1, \forall x.\neg r(x)@T \longrightarrow q(a)@1}\,l\forall \quad \dfrac{\boxed{\text{OK}}}{[r(a) \vee q(a)]@1 \longrightarrow 0 \preceq T, q(a)@1}\,\mathsf{ENT}}{(r(a) \vee q(a))@1, \Box\forall x.\neg r(x) \longrightarrow q(a)@1}\,l\Box}{\bigcirc(r(a) \vee q(a)), \Box\forall x.\neg r(x) \longrightarrow \bigcirc q(a)}\,l\bigcirc, r\bigcirc}{\bigcirc(r(a) \vee q(a)) \wedge \Box\forall x.\neg r(x) \longrightarrow \bigcirc q(a)}\,l\wedge}{\longrightarrow [\bigcirc(r(a) \vee q(a)) \wedge \Box\forall x.\neg r(x)] \supset \bigcirc q(a)}\,r\supset$$

Figure 2: proof of a simple first-order example.

R is true; finally, if both S and R are true, the circuit maintains the previous state. The inputs are never false at the same time.

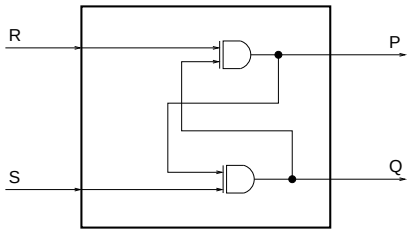

Figure 3: a set-reset flip-flop.

The circuit can be specified in our logic as a formula $\varphi \stackrel{\text{def}}{=} \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4$ in which

$$\begin{array}{lcl} \varphi_1 & \stackrel{\text{def}}{=} & Q \wedge \neg P \\ \varphi_2 & \stackrel{\text{def}}{=} & \Box(P \leftrightarrow \neg(Q \wedge R)) \\ \varphi_3 & \stackrel{\text{def}}{=} & \Box(Q \leftrightarrow \neg(P \wedge S)) \\ \varphi_4 & \stackrel{\text{def}}{=} & \Box(S \vee R) \end{array}$$

(P, Q, R and S are intended to be nullary predicates). Here φ_1 enforces the initial state, φ_2 and φ_3 enforce the equations for P and Q, and φ_4 represents a safety requirement on the inputs. φ_2, φ_3 and φ_4 are invariant through time, and therefore they have a $\Box$ operator as their top connective.

We want to check whether this specification respects requirement $\psi \stackrel{\text{def}}{=} \Box(P \leftrightarrow \neg Q)$, that is, if we can safely take Q as always being the negation of P. So, we want to check whether

$$\vdash (\varphi \supset \psi)@0$$

First, FTL eagerly applies α rules $r\supset$, $l\wedge$ four times, $l\neg$ and $r\Box$ to get

$$\boxed{\dfrac{0 \preceq t, \varphi_2, \varphi_3, \varphi_4, Q \longrightarrow (P \leftrightarrow \neg Q)@t, P}{\longrightarrow \varphi \supset \psi}}$$

Then it uses $l\Box, l\leftrightarrow, l\Box, l\leftrightarrow, l\Box$, to unfold the $\Box$s around φ_2, φ_3 and φ_4; the definitions of the $\leftrightarrow$ connectives are expanded into conjunctions of two implications and placed into Γ as single antecedents, labelled with different times.

Due to lack of space, we will not follow the proof any further. No backtracking is involved in the search; the proof tree has 58 nodes (each one representing the application of a basic tactic) and 18 branches. FTL gets to close by axiom 15 branches, in two cases using a non-atomic formula as the justification for the axiom rule. The remaining three branches are closed by entailment; the constraints are in all cases trivial. t is used here as the witness time at which the invariant formulae are true.

It is remarkable that this proof looks intuitive — that is, FTL proceeds in a natural style, first isolating the premises, then unfolding quantifiers and temporal operators and then trying to match the antecedents with the consequents.

5 Conclusions, related and future work

5.1 Related work

The ideas behind our modular presentation of FOTLs are largely inspired by [Basin *et al.*, 1998], where a modular approach to Quantified Modal Logics is presented; Alessandra Russo has developed a similar system in [Russo, 1996]. There are several analogies between our framework and theirs: for instance, rule ENT in $\mathcal{TL}$ bears a strong resemblance with rule $\mathcal{I}_{R-A}$ in Russo's propositional modal labelled deductive system. All these attempts can be seen as a gradual shift towards Dov Gabbay's Labelled Deductive Systems framework ([Gabbay, 1996]) in modal and temporal logic reasoning.

The use of λProlog for theorem proving has been pioneered by Amy Felty ([Felty, 1993]) who showed how the characteristics of this language made it perfectly suitable for implementing proof search through tactics and tacticals (respectively, what we call basic and com-

pound tactics). λProlog ([Nadathur and Miller, 1995]) was developed as a logic programming language supporting higher-order programming, polymorphic typing, modular programming, abstract data types and lambda-abstraction and application.

Given their usefulness, First-Order Temporal Logics have been widely studied so far (see for instance [Gabbay *et al.*, 1993], [Manna and Pnueli, 1992] and [Emerson, 1990]), but due to their complexity, very few attempts towards their mechanisation and execution have been successful. The most remarkable case is STeP ([Manna *et al.*, 1995]), which has been successfully employed in verification of hybrid and real-time systems ([Manna and Sipma, 1998; Bjoerner *et al.*, 1997]). STeP integrates deductive methods and model checking, plus other modules which can be called upon whenever the computation requires it. Modularity, which allows smart combination of different techniques, seems the leading path.

Temporal resolution ([Dixon *et al.*, 1998]) is another interesting approach which is being developed for the first-order case at Manchester Metropolitan University. It relies on Fisher's Normal Form for temporal logic formulae ([Fisher, 1997]).

5.2 Conclusions and future work

We have presented a temporal sequent calculus, called $\mathcal{TL}$, which is modular in the structure of time and can therefore be employed on several FOTLs. In $\mathcal{TL}$ labelled formulae are used, where labels represent instants of time, and reasoning on labels is devoted to a special rule called ENT, which is opaque to the calculus.

Moreover, we have presented a tactic theorem prover written in λProlog, called FTL, which exemplifies $\mathcal{TL}$'s modularity and can reason about several FOTLs. $\mathcal{TL}$'s ENT rule is enforced in FTL via the Time module, which is imported in λProlog style into the prover's top shell, acting therefore as an opaque reasoner on labels.

Finally, we have discussed some interesting experimental results, which have been obtained in a totally automatic way thanks to a simple compound tactic which exhaustively applies rules to the target formula.

Immediate future work includes implementing the "until" operator and a lazy constraint management (i.e., delaying their solution to the very end of the proof tree).

Future work includes primarily the application of the proof planning paradigm to FTL via integration with the λCLAM system ([Richardson *et al.*, 1998]); secondarily we plan to embed inductive reasoning on time into FTL. Finally, since the long-term aim of this project is automated system verification, we plan to investigate the use of tactics for finding loop invariants and strengthening the requirements. These strategies have already proved very effective, mainly within STeP, which is our main reference point.

Acknowledgements This work has been carried out at the University of Edinburgh during 1999-2000 and is supported by the EPSRC Grant GR/M46624, "Mechanising first-order temporal logics". The authors wish to thank all other people involved in the project: Alan Bundy, Anatoli Degtyarev, Paul Jackson, Michael Fisher and Peter Quigley, as well as the two anonymous referees for their helpful comments.

References

[Abadí and Manna, 1990] Martin Abadí and Zohar Manna. Nonclausal deduction in first-order temporal logic. *Journal of the ACM*, 37(2):279–317, April 1990.

[Basin *et al.*, 1997] David Basin, Seán Matthews, and Luca Viganò. Labelled propositional modal logics: Theory and practice. *Journal of Logic and Computation*, 7(6):685–717, December 1997.

[Basin *et al.*, 1998] David Basin, Seán Matthews, and Luca Viganò. Labelled modal logics: Quantifiers. *Journal of Logic, Language, and Information*, 7(3):237–263, 1998.

[Bjoerner *et al.*, 1997] N. S. Bjoerner, Z. Manna, H. B. Sipma, and T. E. Uribe. Deductive verification of real-time systems using STeP. *Lecture Notes in Computer Science*, 1231:22–??, 1997.

[Burch *et al.*, 1992] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[Dixon *et al.*, 1998] Clare Dixon, Michael Fisher, and Michael Wooldridge. Resolution for temporal logics of knowledge. *Journal of Logic and Computation*, 8(3):345–372, June 1998.

[Emerson, 1990] E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 995–1072. Elsevier Science Publishers, Amsterdam, The Netherlands, 1990.

[Felty and Thery, 1996] Amy Felty and Laurent Thery. Interactive theorem proving with temporal logic. Technical Report RR-2804, Inria, Institut National de Recherche en Informatique et en Automatique, 1996.

[Felty, 1993] Amy Felty. Implementing tactics and tacticals in a higher-order logic programming language. *Journal of Automated Reasoning*, 11(1):43–81, August 1993.

[Fisher, 1997] Michael Fisher. A normal form for temporal logics and its applications in theorem-proving and execution. *Journal of Logic and Computation*, 7(4):429–456, August 1997.

[Gabbay *et al.*, 1993] Dov M. Gabbay, Ian Hodkinson, and Mark Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects. Volume 1: Mathematical Foundations*. Oxford University Press, *forthcoming*, 1993. Preprinted as Tech Report MPI-I-92-213, Max-Planck-Institut für Informatik, Saarbrücken, Germany.

[Gabbay, 1996] Dov M. Gabbay. *Labelled Deductive Systems, Volume 1*. Oxford University Press, Oxford, 1996.

[Gentzen, 1969] Gerhard Gentzen. Investigations into logical deductions, 1935. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland Publishing Co., Amsterdam, 1969.

[Manna and Pnueli, 1981] Zohar Manna and Amir Pnueli. Temporal verification of concurrent programs: the temporal framework for concurrent programs. In R.S̃. Boyer and J Strother Moore, editors, *The Correctness Problem in Computer Science*, chapter 5, pages 215–273. Academic Press, London, 1981.

[Manna and Pnueli, 1992] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, New York, 1992.

[Manna and Sipma, 1998] Z. Manna and H. B. Sipma. Deductive verification of hybrid systems using STeP. *Lecture Notes in Computer Science*, 1386:305–??, 1998.

[Manna *et al.*, 1995] Z. Manna, N. Bjoerner, A. Browne, and E. Chang. STeP: The Stanford Temporal Prover. *Lecture Notes in Computer Science*, 915:793–??, 1995.

[McGuire *et al.*, 1994] H. McGuire, Z. Manna, and B. Waldinger. Annotation-based deduction in temporal logic. In Dov M. Gabbay and Hans Jürgen Ohlbach, editors, *Proceedings of the 1st International Conference on Temporal Logic*, volume 827 of *LNAI*, pages 430–444, Berlin, July 1994. Springer.

[Nadathur and Miller, 1995] Gopalan Nadathur and Dale Miller. *Higher-order logic programming*, volume 5, chapter ? Oxford University Press, 1995.

[Richardson *et al.*, 1998] Julian Richardson, Alan Smaill, and Ian Green. Proof planning in higher-order logic with lambda-clam. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction (CADE-98)*, volume 1421 of *LNAI*, pages 129–133, Berlin, July 5–10 1998. Springer.

[Russo, 1996] Alessandra Russo. Generalising propositional modal logic using labelled deductive systems. In F. Baader and K. U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 57–74. Kluwer Academic Publishers, March 1996.

[Shankar, 1992] Natarajan Shankar. Proof search in the intuitionistic sequent calculus. In D. Kapur, editor, *Proceedings 11th Intl. Conf. on Automated Deduction, CADE'92, Saratoga Springs, CA, USA, 15–18 June 1992*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 522–536. Springer-Verlag, Berlin, 1992.

[van Benthem, 1984] Johan van Benthem. Correspondence theory. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Volume II: Extensions of Classical Logic*, volume 165 of *Synthese Library*, chapter II.4, pages 167–247. D. Reidel Publishing Co., Dordrecht, 1984.

[Wallen, 1989] L. A. Wallen. *Automated Deduction in Non-Classical Logics*. MIT Press, 1989.