Proof Planning for Feature Interactions: a preliminary report

by

Claudio Castellini, Alan Smaill

Informatics Research Report EDI-INF-RR-0153

School of Informatics **October 2002**
http://www.informatics.ed.ac.uk/

Proof Planning for Feature Interactions: a preliminary report

Claudio Castellini, Alan Smaill

Informatics Research Report EDI-INF-RR-0153

SCHOOL *of* INFORMATICS
Centre for Intelligent Systems and their Applications
October 2002

appears in Proceedings of LPAR 2002, Tbilisi, Georgia

Abstract :
We report on an initial success obtained in investigating the Feature Interaction problem (FI) via proof planning. FIs arise as an unwanted/unexpected behaviour in large telephone networks and have recently attracted interest not only from the Computer Science community but also from the industrial world. So far, FIs have been solved mainly via approximation plus finite-state methods (model checking being the most popular); in our work we attack the problem via proof planning in First-Order Linear Temporal Logic (FOLTL), therefore making use of no finite-state approximation or restricting assumption about quantification. We have integrated the proof planner lambda-CLAM with an object-level FOLTL theorem prover called FTL, and have so far re-discovered a feature interaction in a basic (but far from trivial) example.

Keywords : proof planning, feature interactions, formal methods, temporal logics

Proof Planning for Feature Interactions: a preliminary report

Claudio Castellini and Alan Smaill

Division of Informatics
University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, UK

Abstract. We report on an initial success obtained in investigating the Feature Interaction problem (FI) via proof planning. FIs arise as an unwanted/unexpected behaviour in large telephone networks and have recently attracted interest not only from the Computer Science community but also from the industrial world. So far, FIs have been solved mainly via approximation plus finite-state methods (model checking being the most popular); in our work we attack the problem via proof planning in First-Order Linear Temporal Logic (**FOLTL**), therefore making use of no finite-state approximation or restricting assumption about quantification. We have integrated the proof planner λCLAM with an object-level **FOLTL** theorem prover called FTL, and have so far re-discovered a feature interaction in a basic (but far from trivial) example.

1 Introduction

In computer science and particularly in formal methods, propositional temporal logics, mainly in their linear-time version LTL, play a central role. They are used in specification, verification and synthesis of transition systems, programs, circuits, protocols and so on. Years of literature and practice have now shed light on the field, its complexity and applications. But the situation becomes more obscure and unexplored when we switch to first-order temporal logics. Because of their high complexity and the lack of tools, they are still quite a virgin territory to computer scientists; first-order linear temporal logic (**FOLTL**), for instance, is not recursively enumerable, and only recently (see, e.g., [13, 16]) some recursively enumerable and/or decidable fragments of it have been isolated and studied. Nevertheless, its expressive power makes it ideal to formalise the behaviour and requirements of infinite-state or parametrised discrete systems.

Encouraged by these results, we are trying to apply automated deduction via proof planning to **FOLTL**, and have so far obtained some interesting results, which are the subject to this paper. In particular, we have found a class of problems, the so-called Feature Interactions, which fits perfectly in this framework.

1.1 Case study: Feature Interactions

According to its most general definition, a *feature* is a service marketed to the customer of a company, usually in addition to a basic service. In the past decade at least, this problem, as experimented in large telephone networks, has received great attention (see, e.g., [12]), both from the academical and the industrial world. In this particular setting, the basic service is represented by the plain telephone switch network connecting users to one another; features are additional services such as call-waiting and call-forwarding. Features are specified and implemented without any knowledge of what other features may be concurrently required by other users in the network. This facilitates modular design but also introduces potential undesired / unwanted behaviours when more than one feature is activated.

A well-known example is the interaction arising between Anonymous Call Rejection (ACR) and Call Forwarding Busy Line (CFBL). Informally, ACR prescribes that anonymous calls (i.e., calls from a user hiding her number) should be rejected, while CFBL prescribes that all calls to the subscriber should be forwarded to a third party if the subscriber is busy. Assume user x subscribes to both features: what happens if anonymous user y calls x while he is engaged? Should y's call be rejected according to ACR or forwarded to z according to CFBL? The situation is usually repaired by establishing a priority relation among features, and in this case, ACR would have priority over CFBL. But this is not the focus of this paper.

Although **FOLTL** has already been used for FI, usually the problem is then approximated assuming a finite number of users and solved via finitary techniques such as model checking, as in [10] and [4], or satisfiability (see [3] for a quite exhaustive perspective); but in this case a positive answer (i.e., "there is no interaction and never will be") is not definitive: if, e.g., the approximation assumes 3 users, an interaction could arise when more of them join in.

On the other side, if we express the problem in **FOLTL**, we need to make no finitary approximation whatsoever; moreover, the presentation is compact and intuitive. But, due to the high complexity of this logic, it is highly unlikely that any general **FOLTL** theorem prover could be of use in a feasible amount of time/space. In order to overcome this problem, we use proof planning ([2]) as a high-level guidance for such a theorem prover.

It must be remarked that, in this paper, we will introduce only a relevant subset of **FOLTL** — namely, then one we are interested in for Feature Interactions. The most striking difference with usual treatments of this logic is the absence of the $\bigcirc$ ("next time") operator. Notice, though, that the approach works, in principle, for full **FOLTL**, and, as well, that we are likely to need the $\bigcirc$ operator when we enlarge the class of problems we tackle.

Outline of the paper: in Section 2 the concept of **FOLTL** is quickly revised and an appropriate sequent calculus is outlined; in Section 3 proof planning is introduced, and a sketch of our system is given; in Section 4 the experiment and the results are described, while Section 5 presents some conclusions and future work.

2 Formal background

2.1 First-Order Linear Temporal Logic

The language of **FOLTL** we consider includes nonempty sets of variable, function and predicate symbols $\mathcal{V}, \mathcal{F}$ and $\mathcal{P}$; it also has the classical operators $\neg$, $\supset$, $\forall$, the unary modal operators $\Box$ ("always") and $\Box^{\tau}$ ("bounded always"), and the binary modal operator $\mathcal{U}$ ("until"). Other operators are defined from these basic ones as usual (e.g., $\exists$ is $\neg\forall\neg$ and so on). Terms are defined in a standard way.

We employ Labelled Deduction (see, e.g., [11]) and have therefore a separate language for labels (denoting time instants) and constraints (predicates over labels). The language of labels and constraints consists of 0, $=$ and $\prec$ ($\preceq$ is defined as the disjunction of $=$ and $\prec$) and variable symbols from a non empty set $\mathcal{V}_t$. The notation $\varphi \mathbin{@} \tau$ informally means: at time τ, formula φ holds.

The semantics we use is also largely standard, see, e.g., [1]. A *structure* is a tuple

$$\mathbf{M} = \langle \mathbb{N}, <, =, \mathcal{D}, I \rangle$$

where

- $\mathbb{N}$, $<$ and $=$ denote the set of natural numbers and the usual less-than and equality relation;
- $\mathcal{D}$ is a nonempty set (the domain of quantification);
- I maps each function symbol $f \in \mathcal{F}$ to a $\mathcal{D}$-valued function $I(f)$ and each natural number i and predicate symbol $p \in \mathcal{P}$ to a predicate $I(i,p)$ over $\mathcal{D}$.

Due to this definition, the logic we consider has constant domains and rigid designators (see, e.g., [19]).

An *assignment* α is a function mapping variable symbols in $\mathcal{V}$ to values in $\mathcal{D}$. The assignment $\alpha^{[d/x]}$ assigns $d \in \mathcal{D}$ to x, leaving all the other symbols as in α. The *denotation* of a term s in the structure $\mathbf{M}$ w.r.t. α, written $s^{\mathbf{M},\alpha}$, is recursively defined as follows:

- if s is $v \in \mathcal{V}$, then $s^{\mathbf{M},\alpha} = \alpha(v)$;
- if s is $f(s_1, \ldots, s_n)$, then $s^{\mathbf{M},\alpha} = I(f)(s_1^{\mathbf{M},\alpha}, \ldots, s_n^{\mathbf{M},\alpha})$.

To give a semantics to labels and constraints, we also introduce an interpretation I_l mapping $\prec$ and $=$ to the usual relations $<$ and $=$ over the naturals, 0 to the natural number 0 and an assignment $\alpha_l : \mathcal{V}_t \mapsto \mathbb{N}$. The denotation of labels is analogous to that of logical terms. The letters $\tau_i, \tau_n, \tau_m, \ldots$ will be used for labels denoting the numbers $i, n, m, \ldots$

The notion of a formula φ being *true in a structure* $\mathbf{M}$ *under the assignment* α, written $\mathbf{M}, \alpha \models \varphi$, is defined recursively as follows:

$$
\begin{array}{ll}
\mathbf{M}, \alpha \models \tau_n = \tau_m & \text{iff } n = m \\
\mathbf{M}, \alpha \models \tau_n \prec \tau_m & \text{iff } n < m \\
\mathbf{M}, \alpha \models \tau_n \preceq \tau_m & \text{iff } \mathbf{M}, \alpha \models \tau_n = \tau_m \text{ or } \mathbf{M}, \alpha \models \tau_n \prec \tau_m \\
\mathbf{M}, \alpha \models p(s_1, \ldots, s_k) \mathbin{@} \tau_i & \text{iff } (s_1^{\mathbf{M},\alpha}, \ldots, s_k^{\mathbf{M},\alpha}) \in I(i, p) \\
\mathbf{M}, \alpha \models \neg\varphi \mathbin{@} \tau_i & \text{iff not } \mathbf{M}, \alpha \models \varphi \mathbin{@} \tau_i \\
\mathbf{M}, \alpha \models \varphi \supset \psi \mathbin{@} \tau_i & \text{iff not } \mathbf{M}, \alpha \models \varphi \mathbin{@} \tau_i \text{ or } \mathbf{M}, \alpha \models \psi \mathbin{@} \tau_i \\
\mathbf{M}, \alpha \models \forall x.\varphi \mathbin{@} \tau_i & \text{iff for all } d \in \mathcal{D}, \mathbf{M}, \alpha^{[d/x]} \models \varphi \mathbin{@} \tau_i \\
\mathbf{M}, \alpha \models \Box\varphi \mathbin{@} \tau_i & \text{iff for all } n \in \mathcal{N}, \\
 & \quad \text{not } \mathbf{M}, \alpha \models \tau_i \preceq \tau_n \text{ or } \mathbf{M}, \alpha \models \varphi \mathbin{@} \tau_n \\
\mathbf{M}, \alpha \models \Box^{\tau_n}\varphi \mathbin{@} \tau_i & \text{iff for all } m \in \mathcal{N}, \\
 & \quad \text{not } (\mathbf{M}, \alpha \models \tau_i \preceq \tau_m \text{ and } \mathbf{M}, \alpha \models \tau_m \prec \tau_n) \text{ or} \\
 & \quad \mathbf{M}, \alpha \models \varphi \mathbin{@} \tau_m \\
\mathbf{M}, \alpha \models \varphi\mathcal{U}\psi \mathbin{@} \tau_i & \text{iff there is } n \in \mathcal{N} \text{ such that} \\
 & \quad \mathbf{M}, \alpha \models \tau_i \preceq \tau_n \text{ and } \mathbf{M}, \alpha \models \psi \mathbin{@} \tau_n \text{ and} \\
 & \quad \mathbf{M}, \alpha \models \Box^{\tau_n}\varphi \mathbin{@} \tau_i
\end{array}
$$

If a formula φ is true in $\mathbf{M}$ under all possible assignments α, we say that the structure $\mathbf{M}$ is a *model* for φ, and that φ is *true in the structure (model)* $\mathbf{M}$, written $\mathbf{M} \models \varphi$. If a formula φ is true in all possible models, we say it is *valid* and write $\models \varphi$.

2.2 A labelled sequent calculus for FOLTL

The formal framework in which we do proof search is represented by a labelled sequent calculus modelled upon those presented in [6]. In that paper we have devised a methodology for building sound and complete sequent calculi for quantified modal logics whose frame properties can be axiomatised by a finite set of first-order sentences including equality.

This seems a good starting point, so we follow that approach and build a labelled sequent calculus for **FOLTL** that we call $\mathcal{C}_{\mathbf{FOLTL}}$, by adding rules for reflexivity, transitivity and strong connectedness to the basic calculus $\mathcal{C}_{\mathbf{QK}}$ in [6]. Moreover, we add to $\mathcal{C}_{\mathbf{FOLTL}}$ the rules shown in Figure 1.

The key point in $\mathcal{C}_{\mathbf{FOLTL}}$ is the way the $\mathcal{U}$ operator is taken care of. Consider rules $l\mathcal{U}$ and $l\Box^*$: basically, rule $l\mathcal{U}$ expands the "existential" part of $\mathcal{U}$, that is, it introduces a fresh time variable $\tau_a \in \mathcal{V}_t$ and a constraint stating that ψ will hold at a time τ_a in the future; moreover, it introduces the requirement that, "in the meantime", φ must hold ($\Box^{\tau_a}\varphi \mathbin{@} \tau$).

Rule $l\Box^*$ then, takes care of this latter requirement (the "universal" part of $\mathcal{U}$), and expands it into three assertions: *(i)* its argument φ holds at a time τ_d, and *(ii, iii)* this time must lie between the current time τ and the time τ_a associated with the $\Box^{\tau_a}$ operator. An analogous explanation can be made for rules $r\mathcal{U}$ and $r\Box^*$.

This complication is required by a completeness argument, which we explain informally: there are formulae, representing problems in our case study, which cannot be proved if we do not allow multiple instances of the universal part of $\mathcal{U}$. Therefore, it is necessary to handle the two parts separately. This is also why

Logical rules

$$\frac{\Gamma, \tau \preceq \tau_a, \psi \mathbin{@} \tau_a, \Box^{\tau_a}\varphi \mathbin{@} \tau \longrightarrow \Delta}{\Gamma, \varphi \mathcal{U} \psi \mathbin{@} \tau \longrightarrow \Delta}\ l\mathcal{U}$$

$$\frac{\Gamma \longrightarrow \tau \preceq \tau_b, \Delta \quad \Gamma \longrightarrow \psi \mathbin{@} \tau_b, \Delta \quad \Gamma \longrightarrow \Box^{\tau_b}\varphi \mathbin{@} \tau, \Delta}{\Gamma \longrightarrow \varphi \mathcal{U} \psi \mathbin{@} \tau, \Delta}\ r\mathcal{U}$$

$$\frac{\Gamma, \varphi \mathbin{@} \tau_c \longrightarrow \Delta \quad \Gamma \longrightarrow \tau \preceq \tau_c, \Delta \quad \Gamma \longrightarrow \tau_c \prec \tau_n, \Delta}{\Gamma, \Box^{\tau_n}\varphi \mathbin{@} \tau \longrightarrow \Delta}\ l\Box^*$$

$$\frac{\Gamma, \tau \preceq \tau_d, \tau_d \prec \tau_n \longrightarrow \varphi \mathbin{@} \tau_d, \Delta}{\Gamma \longrightarrow \Box^{\tau_n}\varphi \mathbin{@} \tau, \Delta}\ r\Box^*$$

Table 1. sequent rules for $\mathcal{U}$ and $\Box^\tau$. $\tau_a, \tau_d \in \mathcal{V}_t$ cannot appear free in the conclusion of $l\mathcal{U}$ and $r\Box^*$.

we have introduced the $\Box^\tau$ operator in our logic, which is not very common in **FOLTL**.

3 Proof planning and λCLAM

Proof planning was initially proposed by Bundy ([2]) and has then been developed mainly in Edinburgh, Birmingham and Saarbrücken; the underlying idea is to build an abstract representation of a proof (a *proof plan*) rather than a proof. Analogously to a proof tree, where each node is labelled by a pair (inference rule, sequent), a proof plan is a tree whose nodes are labelled by pairs (method, sequent). A *method* is a macro-step of reasoning resembling the activity of a human reasoner, i.e., a well chosen induction scheme or a considerate case-split rule. The advantages are mainly that *(i)* the search space at the abstract level is typically orders of magnitude smaller than that at the object level, and *(ii)* very little backtracking is likely to happen in proof planning.

Once a proof plan for a formula has been found, it is translated into the object-level specification of a proof and then validated by an object-level theorem prover acting as a proof checker. That is, each node of the proof plan is expanded into a sequence (or possibly a tree) of inference rules, specified in an operational way by *tactics*. The result is a *tactic tree* which is then executed by the prover until the proof of the original formula is obtained. This way, a task which would have been impossible for the object-level prover alone becomes feasible. A proof plan acts as a guidance for proof search at the object level.

The proof planner we use, λCLAM, was first envisioned in [18], in which the idea of higher-order terms for proof search is introduced. λCLAM employs the higher-order logic programming language λProlog (see [15]) and takes advantage of a number of features of this language, *in primis* its modularity. A typical method declaration in λCLAM looks like this:

```
atomic <theory-name> <method-name>
  <input-goal>
  <preconditions>
  <postconditions>
  <output-goal>
  <associated tactic>.
```

The method can be applied to the *input goal* if the *preconditions* hold (representing constraints on the shape of the main formula of the sequent or denoting the state of the planner in which the method is applicable); after application of the method, the *output goal* is generated and the *postconditions* must hold. The *associated tactic* replaces the method in the object-level specification of the proof.

3.1 FTL: a tactic-based theorem prover for FOLTL

FTL is written in λProlog as well and employs tactics, in the style of [7]. At the simplest level a tactic is a wrapper for a rule of inference, also carrying some information such as which formula the rule is applied to, which term has been guessed for an existential force quantifier, etc. It basically enforces a sound step of inference.

In general however, a tactic just links two sequents and can therefore represent conditional, iterative and exhaustive application of tactics, or even a tactic tree, which is the full specification of a proof tree.

FTL exploits a number of well-known characteristics of λProlog which are well suited for classical, intuitionistic and temporal logics (see, e.g., [9]). The main one is, once again, its modularity: in FTL, a separate λProlog module called time takes care of reasoning on time. Figure 1 sketches the system: each box represents a module, inclusion of boxes indicates textual inclusion of modules (*accumulation*), while an arrow denotes dynamic addition of a module's clauses to another module (*importing*). In this case, module time is imported into the main tactics module, basic_tacs.

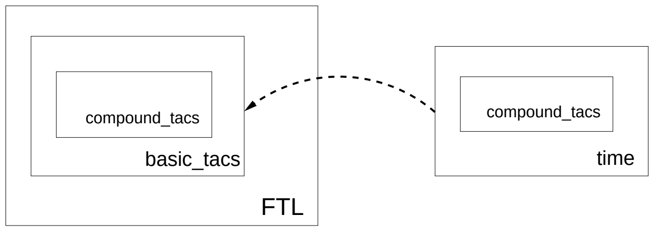

Fig. 1. a scheme of FTL.

FTL also employs well-known techniques borrowed from the automated reasoning community, such as the use of metavariables to handle non-generative rules (universal quantification in the hypotheses or existential quantification in the conclusions). Together with the standard λProlog unification algorithm, this technique enforces a "lazy" instantiation of existential force quantifiers. This technique, which achieves great efficiency by delaying the choice of terms to the very end of the proof, is thoroughly explained, e.g., in [17]. See [5] for a more detailed system description of `FTL`.

4 Experiments

In the following, we will omit labels attached to a formula whenever the label is 0. The way we tackle FIs closely follows the methodology outlined in [10]:

- the global behaviour of the telephone system is expressed as a set of invariant (i.e., wrapped by a $\Box$ operator) universally quantified first-order sentences;
- a feature, denoted by the subscript i, is specified via a formula like this:

$$\Box\forall\overline{x}\ e_i(\overline{x}) \supset\ [p_i(\overline{x})\ \mathcal{U}\ (r_i(\overline{x}) \vee d_i(\overline{x}))] \tag{1}$$

  informally meaning "after the feature is **e**nabled, a **p**ersisting condition holds until the feature is **r**esolved or **d**ischarged".
- a feature interaction (that is, an undesired behaviour) is found between two features 1 and 2 if the previous formulae together imply $\Box\forall\overline{x}\neg G(\overline{x})$, where $G(\overline{x})$ is:

$$e_1(\overline{x}) \wedge e_2(\overline{x}) \wedge [(p_1(\overline{x}) \wedge p_2(\overline{x}))\ \mathcal{U}\ (\neg p_1(\overline{x}) \wedge \neg p_2(\overline{x}) \wedge \neg d_1(\overline{x}) \wedge \neg d_2(\overline{x}))] \tag{2}$$

  informally meaning "enable the features at the same time, let them persist, then force them to resolve". The idea is that $G(\overline{x})$ represents the required behaviour, and $\Box\forall\overline{x}\neg G(\overline{x})$ is the *denial* of it. If the above implication is valid, then the required behaviour will never be possible, for any combination of users.

In order to test the feasibility of proof planning for FIs, we have analysed the hand-made proof of the ACR/CFBL example (see Subsection 1.1) as presented in [8], and tried to mimic it in a proof plan. Here is how we formalised the problem:

(i) ACR is defined by instantiating the schema 1 as follows:

$$\begin{aligned} e_1(x,y) &= \mathit{has\_acr}(x) \wedge \neg \mathit{display}(y) \wedge \mathit{call\_req}(y,x) \\ p_1(x,y) &= \mathit{call\_req}(y,x) \\ r_1(x,y) &= \mathit{acr\_announce}(x,y) \\ d_1(y) &= \mathit{onhook}(y) \end{aligned}$$

informally meaning: if user x has activated ACR and user y is anonymous, y will be trying to call x until either x sends a rejection message or y hangs up.

(ii) CFBL is defined by instantiating the same schema as follows:

$$\begin{aligned} e_2(x,y,z) &= has\_cfbl(x) \wedge \neg idle(x) \wedge \\ &\qquad \neg\exists t.forwarding(t,x,z) \wedge call\_req(y,x) \\ p_2(x,y) &= call\_req(y,x) \\ r_2(x,y,z) &= forwarding(y,x,z) \\ d_2(y) &= onhook(y) \end{aligned}$$

informally meaning: if user x has activated CFBL, is not idle and there are no calls to him being currently forwarded, y will be trying to call x until either the call is forwarded to z or y hangs up.

(iii) System axioms enforce simple properties of the predicates involved in the definition of the features, e.g.,

$$\Box \forall xy\ \neg(onhook(x) \wedge call\_req(x,y))$$

informally meaning: it is impossible to hang up and be trying to call someone at the same time.

(iv) Finally, the requirement is obtained by instantiating scheme 2 with the above definitions of p_i, e_i, d_i and r_i, $i = 1, 2$ (from now on, we omit the explicit reference to the variables $\overline{x}$ for conciseness).

In the end we are trying to prove validity of the formula:

$$\begin{aligned} \Box\forall\overline{x}\ [e_1 \supset\ p_1\, \mathcal{U}\ (r_1 \vee d_1)] &\quad \wedge \\ \Box\forall\overline{x}\ [e_2 \supset\ p_2\, \mathcal{U}\ (r_2 \vee d_2)] &\quad \wedge \\ \Box\forall\overline{x}[SA] &\quad \supset \\ \Box\forall\overline{x}\neg G & \end{aligned} \tag{3}$$

where SA denotes system axioms. Note that 3 involves a quite free mix of unary, binary and ternary predicates, temporal operators and first-order quantifiers, in such a way that it does not fall into any known well-behaved fragment of **FOLTL**, as far as we know.

According to the hand-made proof, let us suppose G holds and try to derive a contradiction. By the definitions of $\Box$ and $\mathcal{U}$,

1. if G holds, there is a time $t_0 \geq 0$ at which both e_1 and e_2 hold; also, there is a time $t_G \geq t_0$ such that p_1 and p_2 hold until $\neg p_1 \wedge \neg p_2 \wedge \neg d_1 \wedge \neg d_2$ holds at t_G;
2. since both ACR and CFBL hold, they must be enabled at t_0; also, there are times $t_{ACR}, t_{CFBL} \geq t_0$ such that p_1 and p_2 hold until the features are either resolved or discharged respectively at t_{ACR} and t_{CFBL}.

The key to the proof is the relative positions of t_G, t_{ACR} and t_{CFBL}; Figure 2 is an example case in which $t_{ACR} < t_G$ and $t_{CFBL} > t_G$.

There are three subcases to be considered for t_G and t_{ACR} (i.e., $t_G < t_{ACR}$, $t_G > t_{ACR}$ and $t_G = t_{ACR}$) and three for t_G and t_{CFBL}, but it turns out that the situation is simpler:

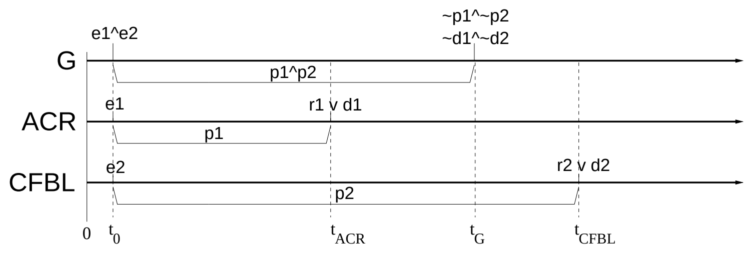

Fig. 2. a graphical representation of the interaction between ACR and CFBL. In this case, $t_{ACR} < t_G$ and $t_{CFBL} > t_G$.

– consider G and ACR: if $t_G < t_{ACR}$ then both $\neg p_1$ and p_1 must hold at t_G, which leads to a contradiction. Analogously, consider G and CFBL: if $t_G < t_{CFBL}$ then both $\neg p_2$ and p_2 must hold at t_G;
– consider G and ACR again: if $t_{ACR} < t_G$ then both p_1 and $r_1 \vee d_1$ must hold at t_{ACR}, which leads to a contradiction if the system axioms are taken into account. Analogously for G and CFBL, in which case a contradiction is derived from p_2 and $r_2 \vee d_2$ at t_{CFBL} w.r.t. the system axioms;
– lastly, consider the remaining case in which $t_G = t_{ACR} = t_{CFBL}$: by propositional reasoning, r_1 and r_2 must hold together with the system axioms, which once again leads to a contradiction.

As already noted in [8], system axioms are not involved in the first two cases, ruled out by simple propositional considerations. The remaning three cases are solved by first-order reasoning because no temporal operators are involved in the system axioms. In order to mimic this neat, intuitive and rigorous (although not formal) way of reasoning, we set up a λCLAM method called *fi_case_split* which simply splits the goal of proving formula 3 into three first-order subgoals (see Figure 3).

λCLAM finds the proof plan in about one minute on an Ultra 10 Sun machine without any backtracking, as we expect. The proof plan is then translated into a (big) tactic which is fed to FTL, which applies it to the formula and generates the actual proof of the formula itself.

It is interesting to have a closer look at the process of translation of the proof plan into an FTL tactic. In particular, the first-order reasoning which happens in λCLAM during the exploration of the three subgoals opened by *fi_case_split* only involves atomic methods, which embed inference rules of $\mathcal{C}_{\mathbf{FOLTL}}$ and are translated directly into basic tactics.

The case is quite different with the translation of the *fi_case_split* method itself, corresponding to a quite complicated sub-prooftree, visible in Figure 4. In particular, rules $l\mathcal{U}$ are employed once each for G and ACR, introducing time points t_G and t_{ACR}; then, strong connectedness (rule sconn) generates three subcases in which $t_G < t_{ACR}$, $t_G > t_{ACR}$ and $t_G = t_{ACR}$. The first two subcases are ruled out respectively by immediate contradiction and using the first sub-case

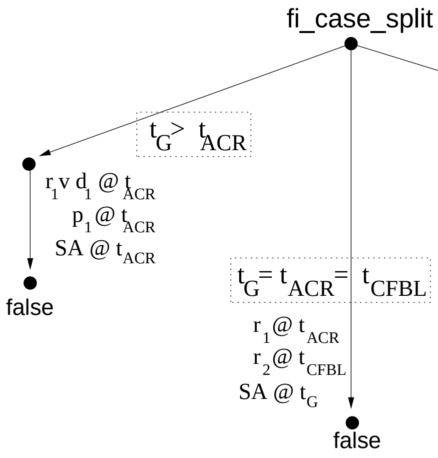

Fig. 3. method *fi_case_split* applied to the interaction between ACR and CFBL. The three generated subgoals are closed by first-order reasoning.

of the *fi_case_split* method; the third is brought forward, with the assumption that $t_G = t_{ACR}$.

Then, in perfect analogy, rule $l\mathcal{U}$ introduces time t_{CFBL} for CFBL and strong connectedness opens three more subcases, the first two of which are respectively closed by immediate contradiction, and by the second sub-case of *fi_case_split*. We are left with the assumptions $t_G = t_{ACR}$ and $t_G = t_{CFBL}$, and this branch is closed by the third sub-case of *fi_case_split*.

The subtree remarkably reflects the structure of the hand-made proof, also formally justifying it; moreover, the fact that its execution as a tactic proves the original formula in FTL ensures soundness of the proof plan. But the more remarkable property is that it clearly shows a sort of "pattern" in the way the $\mathcal{U}$-formulae are exploited and searched for contradiction: first use the $\mathcal{U}$ in G "against" that in ACR; once one branch only is left, use the $\mathcal{U}$ in G once again "against" that in $CFBL$; if in the end one branch only remains, try to close it by FO reasoning.

5 Conclusions

As a preliminary result, the experiment described in the previous Section seems encouraging. The spirit behind proof planning, a discipline on the border of AI and Cognitive Science, is that of capturing the common structure in proofs dealing with a particular problem, by means of proof plans — exactly as it happens in this example. We believe this class of problems is so hard that not only no brute-force approach is feasible, but also that effective heuristics will be hard to find for a long time; at the same time, several of the proofs devised for these problems actually seem to *share* the common structure seen at the end

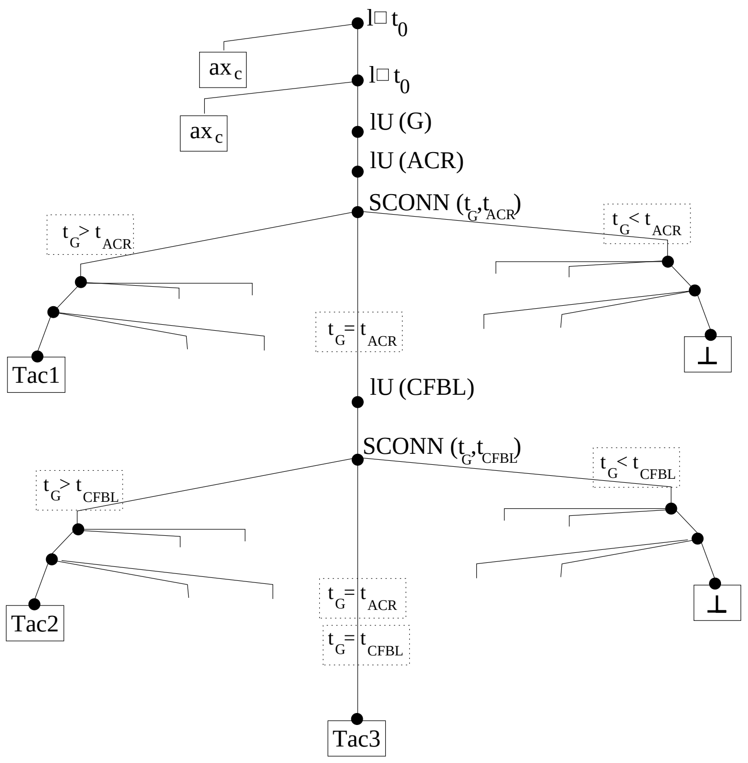

Fig. 4. the tactic tree obtained translating the *fi_case_split* method. $Tac1$, $Tac2$ and $Tac3$ are the tactics corresponding to the three subcases of the method. Branches which look open in the Figure are closed by rules $l\Box^*$, not shown.

of the previous Section: if this is confirmed by further experimenting, Feature Interactions are definitely a good benchmark for proof planning.

In fact, the idea is that of mimicking human proofs and then generalising the common structure found in them to more examples and problems; naïve as it may appear, the framework presented in this preliminary report will work for any two features whose shapes resemble the schema 1 and system axioms not containing temporal operators. Although this represents a small fragment of **FOLTL**, it appears that a relevant part of the Feature Interaction community is adopting a similar formalism; see, e.g., [14, 10, 4].

Of course the class of problems needs to be enlarged and the approach generalised. In particular, it seems not too far-fetched to tackle interactions among more than two features, and — more interesting — among a collection of users, each one having a different set of features enabled.

If proved successful, this approach could shed more light on the field of automated reasoning in **FOLTL**, bringing benefits to the whole community. As well, if feasible on a large scale, it could significantly improve the situation in infinite-state formal methods, thanks also to the possible integration of λCLAM with external decision procedures and model checkers.

Acknowledgements

This work is being carried out at the University of Edinburgh and is supported by the EPSRC Grant GR/M46624, "Mechanising First-Order Temporal Logics". The authors wish to thank all people involved in the project: Alan Bundy, Anatoli Degtyarev, Paul Jackson, Michael Fisher and Peter Quigley.

References

1. Martin Abadí and Zohar Manna. Nonclausal deduction in first-order temporal logic. *Journal of the ACM*, 37(2):279–317, April 1990.
2. A. Bundy. The use of explicit plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *9th International Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988. Longer version available from Edinburgh as DAI Research Paper No. 349.
3. Muffy Calder, Mario Kolberg, Evan H. Magill, and Stephan Reiff-Marganiec. Feature interaction: A critical review and considered forecast. *Computer Networks*, 2002.
4. Muffy Calder and Alice Miller. Using SPIN for feature interaction analysis — A case study. In M.B. Dwyer, editor, *Model checking software: 8th International SPIN Workshop, Toronto, Canada, May 19-20, 2001: proceedings*, pages 143–162. Springer, 2001. Lecture Notes in Computer Science No. 2057.
5. C. Castellini and A. Smaill. Tactic-based theorem proving in first-order modal and temporal logics. In E. Giunchiglia and F. Massacci, editors, *Issues in the Design and Experimental Evaluation of Systems for Modal and Temporal Logics*, TR DII 14/01. University of Siena, June 2001.
6. C. Castellini and A. Smaill. Modular, uniform and normalising sequent calculi for quantified modal logics. Technical report, Division of Informatics, 2002.
7. A. Felty. Implementing tactics and tacticals in a higher-order logic programming language. *Journal of Automated Reasoning*, 11(1):43–81, 1993.
8. Amy Felty. Temporal logic theorem proving and its application to the feature interaction problem. In E. Giunchiglia and F. Massacci, editors, *Issues in the Design and Experimental Evaluation of Systems for Modal and Temporal Logics*, TR DII 14/01. University of Siena, June 2001.
9. Amy Felty and Laurent Thery. Interactive theorem proving with temporal logic. *Journal of Symbolic Computation*, 23(4):367–397, 1997.
10. Amy P. Felty and Kedar S. Namjoshi. Feature specification and automated conflict detection. In *Feature Interactions Workshop*. IOS Press, 2000.
11. Dov M. Gabbay. *Labelled Deductive Systems, Volume 1.* Oxford University Press, Oxford, 1996.
12. Nancy Griffeth, Ralph Blumenthal, Jean-Charles Gregoire, and Tadashi Ohta. A feature interaction benchmark for the first feature interaction detection contest. *Computer Networks (Amsterdam, Netherlands: 1999)*, 32(4):389–418, April 2000.
13. I. Hodkinson, F. Wolter, and M. Zakharyaschev. Decidable fragments of first order temporal logics. *Annals of Pure and Applied Logic*, 106:85–134, 2000.

14. Bengt Jonsson, Tiziana Margaria, Gustaf Naeser, Jan Nyström, and Bernhard Steffen. Incremental requirement specification for evolving systems. *Nordic Journal of Computing*, 8(1):65–87, Spring 2001.
15. Gopalan Nadathur and Dale Miller. Higher-order logic programming. In D. Gabbay, C. Hogger, and A. Robinson, editors, *Handbook of Logic in AI and Logic Programming, Volume 5: Logic Programming*. Springer Verlag, Oxford, 1986.
16. Regimantas Pliuškevičius. On an ω-decidable deductive procedure for non-Horn sequents of a restricted FTL. *Lecture Notes in Computer Science*, 1861:523–537, 2000.
17. 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, pages 522–536. Springer-Verlag, Berlin, 1992.
18. Alan Smaill and Ian Green. Higher-order annotated terms for proof search. In Joakim von Wright, Jim Grundy, and John Harrison, editors, *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, volume 1275 of *Lecture Notes in Computer Science*, pages 399–414, Turku, Finland, 1996. Springer-Verlag. Also available as DAI Research Paper 799.
19. Luca Viganò. *Labelled Non-Classical Logics*. Kluwer Academic Publishers, Dordrecht, 2000.