

Temporal Logic Theorem Proving and its Application to the Feature Interaction Problem ^{*}

Amy Felty

School of Information Technology and Engineering, University of Ottawa
150 Louis Pasteur, P.O. Box 450, Stn A
Ottawa, Ontario K1N 6N5, Canada
afelty@site.uottawa.ca

Abstract. We describe work in progress on a theorem prover for linear temporal logic (LTL) that will be used to automatically detect feature interactions in telecommunications systems. We build on previous work where we identified a class of LTL formulas used to specify the requirements of features, and developed a model checking tool to help find conflicts among feature requirements. The present work will generalize and improve our method in two ways. It will increase the class of conflicts we can find, and it will improve efficiency.

1 Introduction

The *feature interaction* problem is a well-known difficulty that arises in the design and implementation of telecommunications systems, and has received much attention from both research and industry. As more and more features are added to both residential and business telephone services, there is an increased potential for conflicting requirements among seemingly independent features. In a variety of recent approaches to the problem, temporal logic is used to specify feature requirements [4, 11], and model checking tools are used to help in the automatic detection of feature conflicts (for example [6, 10, 16]).

In previous work, as part of an industrial case study, we developed a method which found conflicts among features by considering two specification formulas, one each from two different features [10]. Each formula specifies a single requirement of a feature in temporal logic. A feature conflict results whenever the two formulas are logically inconsistent. We used a model checker to automate the process of finding such inconsistencies. We chose COSPAN since it was relatively easy to adapt to our task.

This paper describes work in progress on generalizing and improving our method by developing a theorem prover for *linear temporal logic* (LTL) that is specialized to our task. The main improvement we envision is to generalize the class of properties that can be handled by our technique. Using a model checker restricted us to consider only particular instantiations of specification formulas. Our proof search procedure will allow unrestricted use of quantification. We also

^{*} In *Proceedings of the IJCAR Workshop on Issues in the Design and Experimental Evaluation of Systems for Modal and Temporal Logics*, Technical Report DII 14/01, University of Siena, 2001.

hope to improve on performance. Although the model checker was quite efficient for our task, using general-purpose tools specialized to particular tasks almost always comes with some additional performance costs. Improving on performance will allow us to consider more difficult problems. Thus, our goal is to develop a specialized automated proof procedure that handles the class of linear temporal logic formulas that arise in the application of feature interaction detection. The addition of quantifiers will complicate matters, but we have begun to develop strategies for applying quantifier rules efficiently for our task.

There are many choices in automating a temporal logic. Here, we choose to translate LTL into first-order logic (FOL). We use a translation that can be viewed as a formalization of the semantics of LTL [14, 15] in FOL with integers. Because we are concentrating on a particular application, our proofs have a specific form and are such that they can be decomposed so that the temporal reasoning is simple and the challenging part is in the reasoning about formulas of FOL with integer arithmetic. In designing a proof search procedure, we hope to draw on current technology for automating this kind of reasoning.

In Sect. 2 we present LTL, and in Sect. 3 we describe our approach to feature conflict detection. We give an example of a conflict, and provide an informal proof in LTL which we use to help outline our proof search strategy. Our proposed strategy is given in Sect. 4. This section describes work in progress, and thus gives a rough outline of the strategy, and indicates where details have to be filled in. Implementing our search procedure is, of course, part of our planned future work. We discuss other future work, as well as related work and conclusions in Sect. 5.

2 Linear Temporal Logic

In the version of LTL we use, an atomic formula has the form $p(t_1, t_2, \dots, t_n)$ where p is a predicate and for $i = 1, \dots, n$, t_i is a term, just as in first-order logic. The syntax of formulas of LTL are defined as follows, where A represents an atomic formula.

$$L := A \mid \neg L_1 \mid L_1 \wedge L_2 \mid L_1 \vee L_2 \mid L_1 \supset L_2 \\ \mid \Box L_1 \mid \Diamond L_1 \mid L_1 \text{ U } L_2 \mid L_1 \text{ W } L_2 \mid \text{X}L_1.$$

These formulas include the usual connectives of propositional logic, as well as the temporal operators “always” (\Box), “eventually” (\Diamond), “until” (U), “weak until” (W), and “next” (X). We do not include quantifiers here; we introduce them in Section 4 when we begin to discuss our new proof search procedure. The semantics of LTL describes how to evaluate an LTL formula on a *path* or set of paths in a *model* [14, 15]. A model is defined as a finite set of states labelled with atomic LTL formulas and a transition relation containing pairs of states. For a given state, the transition relation specifies which states it is possible to go to in one step. It is required that each state has at least one transition going out of it. One or more states are designated as *start* states. A path π is an infinite sequence of states $s_0 \rightarrow s_1 \rightarrow \dots$ such that s_0 is a start state, and every pair of consecutive states $s_i \rightarrow s_{i+1}$ ($i \geq 0$) is in the transition relation. We denote the suffix of a path π starting at s_i as π^i . We write $\pi \models L$ to denote that the path

π satisfies LTL formula L . This satisfaction relation is defined by induction on the structure of L as follows.

1. For atomic formula A , $\pi \models A$ iff A is one of the labels assigned to s_0 .
2. $\pi \models \neg L$ iff $\pi \models L$ is false.
3. $\pi \models (L_1 \wedge L_2)$ iff both $\pi \models L_1$ and $\pi \models L_2$.
4. $\pi \models (L_1 \vee L_2)$ iff $\pi \models L_1$ or $\pi \models L_2$.
5. $\pi \models (L_1 \supset L_2)$ iff whenever $\pi \models L_1$, it is also the case that $\pi \models L_2$.
6. $\pi \models \Box L$ iff for all $i \geq 0$, $\pi^i \models L$.
7. $\pi \models \Diamond L$ iff for some $i \geq 0$, $\pi^i \models L$.
8. $\pi \models (L_1 \text{ U } L_2)$ iff there exists some $i \geq 0$ such that $\pi^i \models L_2$ and for every $j = 0, \dots, i - 1$, we have $\pi^j \models L_1$.
9. $\pi \models (L_1 \text{ W } L_2)$ iff $\pi \models \Box L_1$ or $\pi \models (L_1 \text{ U } L_2)$.
10. $\pi \models \text{X}L$ iff $\pi^1 \models L$.

We say that an LTL formula L is *valid* if every path in every model satisfies L . All propositional tautologies, for example, are valid LTL formulas.

3 Feature Conflict Detection

Our approach to the detection of feature interactions begins by specifying each feature of a telecommunications system as a set of formulas in LTL. We have identified a general class of LTL formulas useful for feature specification [10]. Because we do not consider all of LTL, we will be able to specialize our new proof procedure to handle only this class of formulas. In this section, for illustration purposes, we consider a minor variation of one special case of this class of formulas and discuss other examples in the next section. For this special case, a specification formula has the form:

$$\Box[e \supset (p \text{ U } (r \vee d))]. \quad (1)$$

The symbols e , p , r , and d are formulas formed from atomic formulas and the propositional connectives. Informally, this formula states that if some enabling condition e holds, then p is a condition that persists, until eventually there is either a resolution r or a discharge condition d . If we were to consider exactly the class of formulas from our previous work, formula (1) should be $\Box[e \supset \text{X}(p \text{ U } (r \vee d))]$. We needed the next operator for cases when p was $\neg e$. It is not needed in the example we show here, so we leave it out to simplify the presentation, though adding it would only change small details of the proof and the search procedure.

We consider two features as examples: Anonymous Call Rejection (ACR) and Call Forwarding Busy Line (CFBL). Calls to a subscriber of the ACR feature will not go through when the caller prevents his or her number from being displayed on the subscriber's caller ID device. For CFBL, the subscriber gives a number to which all calls will be forwarded when the subscriber's line is busy. The property below states that if x subscribes to ACR and if there is a call request to x from y , and if furthermore the presentation of y 's number is restricted, this should

cause y to receive the ACR announcement denying the call, unless y gives up and goes back on hook first.

$$\begin{aligned} & \Box[(ACR(x) \wedge call_req(x, y) \wedge \neg DN_allowed(y)) \supset \\ & \quad (call_req(x, y) \text{ U } (ACR_annc(y, x) \vee onhook(y)))] \end{aligned} \quad (2)$$

The next property states that if (a) x subscribes to CFBL, (b) x is not idle, (c) all previously forwarded calls from x to z have terminated, and (d) there is an incoming call from y , then the incoming call from y to x will be forwarded to z , unless y goes back on hook in the meantime.

$$\begin{aligned} & \Box[(CFBL(x) \wedge \neg idle(x) \wedge \neg \exists y forwarding(x, y, z) \wedge call_req(x, y)) \supset \\ & \quad (call_req(x, y) \text{ U } (forwarding(x, y, z) \vee onhook(y)))] \end{aligned} \quad (3)$$

The above two properties provide an example of feature interaction. Intuitively, there is a conflict because there should never be a time when a call is forwarded at the same time the caller hears an announcement saying that the call is denied. This is a conflict known to designers of telecommunications systems, and is resolved by giving priority to CFBL when $DN_allowed(y)$ holds, and to ACR otherwise.

Both of these properties have the same form as formula (1). In both cases the enabling condition e is a conjunct of several formulas, the persisting condition p is $call_req(x, y)$ (which holds from the time y starts to call x until the point where the call processing system makes a connection between x and y), the left disjunct is the resolution r , and the right disjunct $onhook(y)$ is the discharge condition d .

In the CFBL property, note the use of \exists , even though the definition of LTL so far doesn't include quantifiers. Quantifiers can be added directly to LTL [15], but it was important for our model checking approach to leave them out. The use of \exists here can be viewed as an abbreviation. When we are checking properties of a particular model of call processing, we will always include some fixed number of entities (parties with a phone). The formula with \exists stands for the disjunction of atomic formulas of the form $\neg forwarding(x, y, z)$ where in each disjunct, y is replaced by one of the entities, and there is a disjunct for every entity. When we add quantifiers to the logic in the next section, our proof search procedure will be designed to handle them directly, rather than as abbreviations.

A central idea to both our model checking and theorem proving approaches is that a feature interaction corresponds to logical inconsistency of specification formulas. Our definition of feature conflict [10], when applied to examples of the form of formula (1) above, states that two features interact when the following formula is valid:

$$\begin{aligned} & \neg(\Box[e_1 \supset (p_1 \text{ U } (r_1 \vee d_1))] \wedge \Box[e_2 \supset (p_2 \text{ U } (r_2 \vee d_2))] \\ & \quad \wedge \Box[SA] \wedge \Diamond[e_1 \wedge e_2 \wedge g]). \end{aligned} \quad (4)$$

This formula is valid precisely when there does not exist a model such that all four conjuncts inside the negation are satisfiable. The first two conjuncts are two feature specifications, each having the form of formula (1). In the third conjunct, SA represents *system axioms*, which describe properties that should

be true of *any* reasonable system implementation. The fourth conjunct restricts our attention to paths on which both enabled conditions can hold simultaneously and along which the call is not discharged prematurely, but is instead resolved successfully. The formula g , and the system axioms we use for this example are shown below.

$$\begin{aligned}
g &:= (p_1 \wedge p_2) \mathbf{U} (\neg p_1 \wedge \neg p_2 \wedge \neg d_1 \wedge \neg d_2) \\
SA &:= \neg(ACR_annc(y, x) \wedge forwarding(x, y, z)) \\
&\quad \wedge \neg(ACR_annc(y, x) \wedge call_req(x, y)) \\
&\quad \wedge \neg(forwarding(x, y, z) \wedge call_req(x, y)) \\
&\quad \wedge \neg(onhook(y) \wedge call_req(x, y)) \wedge \neg(onhook(y) \wedge forwarding(x, y, z))
\end{aligned}$$

See Felty & Namjoshi [10] for a fuller discussion of the role of these formulas.

To motivate our proof search strategy, we begin with an informal proof of formula (4) specialized as follows:

$$\begin{aligned}
e_1 &:= ACR(x) \wedge call_req(x, y) \wedge \neg DN_allowed(y) \\
r_1 &:= ACR_annc(y, x) \\
e_2 &:= CFBL(x) \wedge \neg idle(x) \wedge \neg \exists y forwarding(x, y, z) \wedge call_req(x, y) \\
r_2 &:= forwarding(x, y, z) \\
p_1, p_2 &:= call_req(x, y) \\
d_1, d_2 &:= onhook(y).
\end{aligned}$$

We denote this formula as ϕ . Formulas (2) and (3) are subformulas of ϕ representing the particular specification formulas from which we attempt to find a contradiction. We now present an informal proof of ϕ using the definition of the semantics of LTL given at the end of Sect. 2. We have formalized this proof in Nuprl [5], using an embedding of the semantics of temporal logic into Nuprl's type theory [9]. The proof we give here follows the Nuprl proof closely.

Theorem 1. *ϕ is a valid formula.*

Proof. This theorem states that it is not possible to define a model such that there exists a path where all four conjuncts inside the outermost negation in ϕ hold. Let π be an arbitrary path in an arbitrary model in π . We assume that the four conjuncts are satisfied in π , and we derive a contradiction. The fourth conjunct tells us that there is a $k \geq 0$ such that $(e_1 \wedge e_2 \wedge g)$ holds at s_k . From this, we derive the following three facts.

1. Since g holds at s_k , by the definition of \mathbf{U} , we know that there is a $j \geq k$ such that $(\neg p_1 \wedge \neg p_2 \wedge \neg d_1 \wedge \neg d_2)$ holds at s_j , and for $j' = k, \dots, j - 1$, we have that $(p_1 \wedge p_2)$ holds at $s_{j'}$.
2. Since e_1 holds at s_k , by the first conjunct, $(p_1 \mathbf{U} (r_1 \vee d_1))$ also holds at s_k . Thus there is an $i_1 \geq k$ such that $(r_1 \vee d_1)$ holds at s_{i_1} , and for $i' = k, \dots, i_1 - 1$, we have that p_1 holds at $s_{i'}$.
3. Similarly, since e_2 holds at s_k , we know $(p_2 \mathbf{U} (r_2 \vee d_2))$ also holds at s_k . Thus there is an $i_2 \geq k$ such that $(r_2 \vee d_2)$ holds at s_{i_2} , and for $i' = k, \dots, i_2 - 1$, we have that p_2 holds at $s_{i'}$.

From these facts, we know that $k \leq j, i_1, i_2$. The rest of the proof proceeds by cases on the relative values of j, i_1 , and i_2 .

- Case 1.** Assume $i_1 > j$. Here we have a contradiction because by fact 1 above, $\neg p_1$ holds at s_j , and by fact 2 above, p_1 holds at s_j since j is in the range $k, \dots, i_1 - 1$.
- Case 2.** Assume $i_2 > j$. Similarly, we have a contradiction because by fact 1, $\neg p_2$ holds at s_j , and by fact 3, p_2 holds at s_j since j is in the range $k, \dots, i_2 - 1$.
- Case 3.** Assume $i_1 < j$. By fact 2, $(r_1 \vee d_1)$ holds at s_{i_1} . By fact 1, p_1 holds at s_{i_1} because i_1 is in the range $k, \dots, j - 1$. Thus $(ACR_annc(y, x) \vee onhook(y))$ and $call_req(x, y)$ both hold at s_{i_1} . But since SA holds at every state, we get a contradiction because neither $ACR_annc(y, x)$ nor $onhook(y)$ can hold at the same time as $call_req(x, y)$.
- Case 4.** Assume $i_2 < j$. This case is similar to case 3, but this time both $call_req(x, y)$ and $(forwarding(x, y, z) \vee onhook(y))$ (which is $r_2 \vee d_2$) hold at state s_{i_2} , which again leads to a contradiction from SA .
- Case 5.** The only remaining case is when $j = i_1 = i_2$. At s_j , we have $(\neg p_1 \wedge \neg p_2 \wedge \neg d_1 \wedge \neg d_2)$ by fact 1, $(r_1 \vee d_1)$ by fact 2, and $(r_2 \vee d_2)$ by fact 3. By propositional reasoning, it follows that $r_1 \wedge r_2$ hold, which means that both $ACR_annc(y, x)$ and $forwarding(x, y, z)$ hold at s_j . We derive a contradiction from SA once more, since these atomic formulas cannot hold at the same time.

4 A Sequent Calculus Approach to Temporal Logic Theorem Proving

The translation we use from LTL to FOL is straightforward, and based directly on the semantic definition of LTL given at the end of Sect. 2. To each atomic LTL predicate, we add a new argument to obtain the corresponding predicate in FOL. This argument represents the index i of the state in the path in which it holds, i.e., given path π , π^i satisfies $p(t_1, t_2, \dots, t_n)$ in LTL if and only $p(t_1, t_2, \dots, t_n, i)$ holds in FOL. The translation function f takes two arguments, an LTL formula, and a path index. An LTL formula L is translated by computing $f(L, 0)$. The translation is defined inductively in Fig. 1.

$$\begin{aligned}
f(p(t_1, \dots, t_n), i) &:= p(t_1, \dots, t_n, i) \\
f(\neg L, i) &:= \neg f(L, i) \\
f(L_1 \wedge L_2, i) &:= f(L_1, i) \wedge f(L_2, i) \\
f(L_1 \vee L_2, i) &:= f(L_1, i) \vee f(L_2, i) \\
f(L_1 \supset L_2, i) &:= f(L_1, i) \supset f(L_2, i) \\
f(\Box L, i) &:= \forall j (j \geq i \supset f(L, j)) \\
f(\Diamond L, i) &:= \exists j (j \geq i \wedge f(L, j)) \\
f(L_1 \mathbf{U} L_2, i) &:= \exists j (j \geq i \wedge f(L_2, j) \wedge \forall k (i \leq k < j \supset f(L_1, k))) \\
f(L_1 \mathbf{W} L_2, i) &:= f(L_1 \mathbf{U} L_2, i) \vee f(\Box L_1, i) \\
f(\mathbf{X}L, i) &:= f(L, i + 1)
\end{aligned}$$

Fig. 1. Translation of LTL formulas to FOL formulas

We outline a strategy for building proofs of FOL formulas obtained by translating LTL formulas of the form (4). We illustrate with the particular formula ϕ and often refer to the proof of Theorem 1, but we make note of which parts of this proof do not depend on the particular instantiations of the subformulas $e_1, p_1, r_1, d_1, e_2, p_2, r_2, d_2$. As stated, formula (4) represents just one subclass of the class of formulas we wish to consider, but serves to illustrate the general ideas. Extending the proof procedure to handle the full class is ongoing work. It is expected to be straightforward to include the whole class, but challenging to make proof search efficient.

The first step is to determine if it is possible for both features to be enabled together. Two features cannot be enabled together if $\neg\Diamond(e_1 \wedge e_2)$ is valid. We try to find a proof of the translation of this formula. This check depends on the instances of e_1 and e_2 and generally involves trying to find a fairly simple proof in propositional logic. Second, if the features can be enabled together, we try to find a proof of formula (4). In the sequel, we focus on the automatic generation of proofs of formulas of this form.

For our first prototype, we have chosen to implement a Gentzen-style sequent calculus for first-order classical logic [12] with rules for reasoning about equality and inequality. The feature conflict examples we have worked with do not require induction, so we do not include it. The main rules of this inference system are given in Fig. 2. Rules for equality and inequality are not shown, though they

$$\begin{array}{c}
\frac{\Gamma \longrightarrow A, \Delta \quad \Gamma \longrightarrow B, \Delta}{\Gamma \longrightarrow A \wedge B, \Delta} \wedge\text{-R} \qquad \frac{A, B, \Gamma \longrightarrow \Delta}{A \wedge B, \Gamma \longrightarrow \Delta} \wedge\text{-L} \\
\\
\frac{\Gamma \longrightarrow A, B, \Delta}{\Gamma \longrightarrow A \vee B, \Delta} \vee\text{-R} \qquad \frac{A, \Gamma \longrightarrow \Delta \quad B, \Gamma \longrightarrow \Delta}{A \vee B, \Gamma \longrightarrow \Delta} \vee\text{-L} \\
\frac{A, \Gamma \longrightarrow B, \Delta}{\Gamma \longrightarrow A \supset B, \Delta} \supset\text{-R} \qquad \frac{\Gamma \longrightarrow A, \Delta \quad B, \Gamma \longrightarrow \Delta}{A \supset B, \Gamma \longrightarrow \Delta} \supset\text{-L} \\
\frac{A, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \neg A, \Delta} \neg\text{-R} \qquad \frac{\Gamma \longrightarrow A, \Delta}{\neg A, \Gamma \longrightarrow \Delta} \neg\text{-L} \\
\frac{\Gamma \longrightarrow [y/x]A, \Delta}{\Gamma \longrightarrow \forall xA, \Delta} \forall\text{-R} \qquad \frac{[t/x]A, \Gamma \longrightarrow \Delta}{\forall xA, \Gamma \longrightarrow \Delta} \forall\text{-L} \\
\frac{\Gamma \longrightarrow [t/x]A, \Delta}{\Gamma \longrightarrow \exists xA, \Delta} \exists\text{-R} \qquad \frac{[y/x]A, \Gamma \longrightarrow \Delta}{\exists xA, \Gamma \longrightarrow \Delta} \exists\text{-L}
\end{array}$$

The $\forall\text{-R}$ and $\exists\text{-L}$ rules have the proviso that the variable y cannot appear free in the lower sequent.

Fig. 2. A Sequent Proof System for First-Order Classical Logic

are of course important for our search strategy. A sequent is written $\Gamma \longrightarrow \Delta$ where Γ and Δ are *sets* of formulas. An initial sequent is a sequent $\Gamma \longrightarrow \Delta$ such that there is a formula A where $A \in \Gamma$ and $A \in \Delta$. The search procedure is goal-directed; it applies sequent rules and stores proofs and partial proofs as it goes along. The important part of proof search is determining the strategy for applying rules, i.e., which rules at which point in the proof and in what

order. One of the reasons we choose sequent proofs is that we believe that failed proofs can give us some insight in the debugging phases of writing specification formulas and system axioms. Our proof search procedure is similar to our model checking procedure [10] in that if we assume that the specification formulas and the system axioms are stated correctly and a proof is found, we can be sure there is a conflict. If no proof is found, there could be gaps in the system axioms or specifications.

For the generic parts of the proof that do not depend on instantiations of the subformulas $e_1, p_1, r_1, d_1, e_2, p_2, r_2, d_2$, we build proof skeletons. We simply fill them in with particular formulas, thus reducing the amount of search needed. For example, note that the proof of Theorem 1 up to the beginning of the case analysis is generic. The partial sequent proof representing this part of the informal proof has the sequent $\rightarrow \phi$ at the root, a series of applications of the rules in Fig. 2 each of which has exactly one premise, and a leaf node of the form $\Gamma \longrightarrow \Delta$ where Δ is empty and Γ is:

$$\begin{aligned} & \{k \geq 0, j \geq k, i_1 \geq k, i_2 \geq k, \\ & f(e_1, k), f(e_2, k), \neg f(p_1, j), \neg f(p_2, j), \neg f(d_1, j), \neg f(d_2, j), \\ & \forall j' (k \leq j' < j \supset (f(p_1, j') \wedge f(p_2, j'))), \\ & f(r_1, i_1) \vee f(d_1, i_1), \forall j' (k \leq j' < i_1 \supset f(p_1, j')), \\ & f(r_2, i_2) \vee f(d_2, i_2), \forall j' (k \leq j' < i_2 \supset f(p_2, j')), \\ & f(\Box[SA], 0)\} \end{aligned}$$

From this point, our approach to automating search is to do a case analysis on the relative values of j , i_1 , and i_2 . There are 13 possible cases: 6 for the cases where all 3 values are different, 6 for the cases when 2 are the same and the other is different, and 1 for the case when $j = i_1 = i_2$. We could just break the proof into these cases and apply some kind of general search to each. But in the informal proof of Theorem 1, there are only 5 cases. We only needed 5 because most of them cover more than one of the 13 subcases mentioned here. The first case in the informal proof, for example, covers the subcases: $i_1 > j > i_2$, $i_1 > i_2 > j$, $i_2 > i_1 > j$, $i_1 = i_2 > j$, $i_1 > i_2 = j$. In general, a clever decomposition into a small number of cases is hard to automate, but we are at least able to incorporate part of the structure of the case analysis of Theorem 1 into our strategy. In particular, we always use the first two cases because they are generic; they are independent of the particular predicates for p_1 and p_2 . As we have done so far, we provide a sequent proof skeleton which gets filled in with particular instances. These 2 cases in the informal proof cover 7 of the 13 subcases; thus only 6 remain.

In Theorem 1, the last 3 cases depend on particular facts about ACR and CFBL. Since we cannot depend on always having the same structure as in this example, our strategy is to decompose the proof into 6 cases at this point. Proving these cases is where good automation is required, and perhaps we will need sophisticated heuristics. Our initial prototype implements a simple depth-first search procedure [8]. This procedure combines direct sequent proof with *amplification* similar to that used in Andrew's mating search procedure [1], which is complete for FOL without integers. An important challenge will be automating reasoning about integer inequalities. We may be able to incorporate decision procedures to handle these parts of the proof. For example, the *Sup-Inf* method [3,

19] may be useful here. We are not attempting to implement a complete procedure, but we believe it will not be difficult to handle those formulas already handled in our model checking approach and hopefully many more.

One drawback of our model checking approach was that we had to use all possible instantiations of the system axioms SA using the constants that represented the different parties in a call. In a call with 3 parties a , b , and c , there are 3 possible instances for x , 3 for y , and 3 for z . Here, we can universally quantify over x , y , and z in the formula SA , and our procedure uses unification to determine the instances we need. Amplification mentioned above can handle cases where more than one instance is needed in a particular proof. We are unlikely to ever need all 3^3 instances.

Our initial prototype can find proofs of some formulas of the form (4), but the automated reasoning for the 6 subcases needs improvement. Also, there are certainly other forms that specification formulas can take besides that of formula (1). In our previous work, we defined a feature specification language [10] which provides a general template. Specifications using these templates are easily translated to LTL formulas of a regular form. For example, two other common forms resulting from the template are below.

$$\Box \neg (e_1 \wedge X((p_1 \wedge \neg e_2) \cup (e_2 \wedge \neg(p_2 \cup (r \vee d))))) \quad (5)$$

$$\Box [e \supset X(\neg p \text{ W } r)] \quad (6)$$

Formula (5) expresses a property that must hold after two enabling events e_1 and e_2 happen in sequence. In particular, the precondition on the property is that e_1 is followed by e_2 with p_1 persisting between them. The postcondition is $(p_2 \cup (r \vee d))$, which is similar to the postcondition in formula (1) and states that after this series of events, a second persisting condition p_2 holds until there is either a resolution r or a discharge d . The formula uses double negation to express this form of requirement: the property is false if and only if there is a point where the precondition holds but the postcondition fails to hold. Formula (6) is often used to express a safety property: if there is some enabling event e , then some property p should not hold between e and the resolution r . For example, if the called party is idle, the caller should not hear a busy tone between the time the call is started and the time it ends. Our approach must certainly be extended to handle these and all formulas that result from translation of the specification language. Also, formula (5) illustrates that the enabling conditions can be more complex, so we will need more theorem proving power to determine if two formulas can be enabled together. But even handling the full specification language only requires a small subset of full LTL. We believe that we can integrate the general form of our template language into the proof search strategy outlined so far without encountering significant difficulties.

5 Conclusion

We have described work in progress on a proof procedure for automating feature conflict detection. Our next step will be to fill in the rest of the strategy for this procedure, implement it, and apply what we learn from examples to improve the strategy and increase the class of examples that we can handle.

When we began work on automating conflict detection, in addition to model checkers, we examined several existing modal logic theorem provers such as Mod-LeanTAP[2] and the Logics Workbench (LWB) [13]. These systems, as well as several other existing systems, did not directly support the logic we were planning to use, though perhaps it would not be difficult to adapt them to do so. Since that time, a module for propositional linear temporal logic has been added to LWB [18]. We plan to try this system on some examples without quantification, e.g., any of the examples used in our model checking approach, and compare the results to our new approach.

An alternative theorem proving approach would be to use a theorem prover that implements a higher-order logic, and use an embedding of LTL into the higher-order logic used by the system. As mentioned, we used such an embedding in Nuprl [9] to formalize Theorem 1. LTL has also been formalized in the HOL theorem prover [17]. Although there are some differences between the Nuprl and HOL formalizations, the application of either one to the kinds of proofs we encounter in feature conflict detection should be similar. Both systems provides primitives which we can use to implement an automated procedure for the class of LTL formulas we have identified and thus we could implement our conflict detection fully within a higher-order system. Initial experiments with Nuprl showed that the overhead of using one logic embedded in another for this application has too great a cost. We expect the outcome to be similar using HOL, and we plan to do similar experiments to find out. Our translation to FOL can be viewed as a compromise. The expected advantage is a gain in efficiency. The disadvantage is the use of a translation to build a notion of time into FOL. In the higher-order embeddings, time is more elegantly defined in terms of primitives already existing in the logic. The FOL approach requires a correctness theorem to show that a CTL formula is valid if and only if its translation is provable in FOL, but the proof of such a theorem should be straightforward.

We have identified a class of formulas from LTL that correspond to those that result from translation from our template language. It would be interesting to give a more precise characterization of this fragment of LTL and compare it to various other fragments that have been identified in the literature, such as decidable fragments.

Another orthogonal direction for our work is to investigate the use of the Mona model checker [7] for our application. Mona implements branching time satisfiability, which generalizes linear time. In addition to detecting conflicts, we plan to use it to investigate whether we can use the feature specifications to synthesize efficient programs that meet these specifications.

References

1. P. B. Andrews. Theorem proving via general matings. *Journal of the Association for Computing Machinery*, 28:193–214, 1981.
2. B. Beckert and R. Goré. System description: leanK 2.0. In *Fifteenth International Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 51–55. Springer-Verlag, 1998.
3. W. W. Bledsoe. A new method for proving certain Presburger formulas. In *Proceedings of the Fourth Joint Conference on Artificial Intelligence*, pages 15–21, 1975.

4. J. Blom, R. Bol, and L. Kempe. Automatic detection of feature interactions in temporal logic. In K. E. Cheng and T. Ohta, editors, *Feature Interactions in Telecommunications Systems III*, pages 1–19. IOS Press, 1995.
5. R. L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, 1986.
6. L. du Bousquet. Feature interaction detection using testing and model-checking, experience report. In *World Congress on Formal Methods*. Springer Verlag, 1999.
7. J. Elgaard, N. Klarlund, and A. Møller. Mona 1.x: New techniques for WS1S and WS2S. In *Tenth International Conference on Computer Aided Verification*, volume 1427. Springer-Verlag Lecture Notes in Computer Science, 1998.
8. A. Felty. *Specifying and Implementing Theorem Provers in a Higher-Order Logic Programming Language*. PhD thesis, University of Pennsylvania, Technical Report MS-CIS-89-53, August 1989.
9. A. P. Felty, D. J. Howe, and F. A. Stomp. Protocol verification in Nuprl. In *Tenth International Conference on Computer Aided Verification*, pages 428–439. Springer-Verlag Lecture Notes in Computer Science, June 1998.
10. A. P. Felty and K. S. Namjoshi. Feature specification and automatic conflict detection. In M. Calder and E. Magill, editors, *Feature Interactions in Telecommunications and Software Systems VI*, pages 179–192. IOS Press, 2000.
11. A. Gammelgaard and J. E. Kristensen. Interaction detection, a logical approach. In W. Bouma and H. Velthuisen, editors, *Feature Interactions in Telecommunications Systems*, pages 178–196. IOS Press, 1994.
12. G. 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.
13. A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfried. Propositional logics on the computer. In *Theorem Proving with Analytic Tableaux and Related Methods*, volume 918 of *Lecture Notes in Artificial Intelligence*, pages 310–323. Springer-Verlag, 1995.
14. M. R. A. Huth and M. D. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2000.
15. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, 1991.
16. M. Plath and M. Ryan. Plug-and-play features. In K. Kimbler and L. G. Bouma, editors, *Feature Interactions in Telecommunications and Software Systems V*, pages 150–164. IOS Press, 1998.
17. K. Schneider and D. W. Hoffmann. A HOL conversion for translating linear time temporal logic to ω -automata. In *Twelfth International Conference on Theorem Proving in Higher Order Logics*, volume 1690, pages 255–272. Springer-Verlag Lecture Notes in Computer Science, 1999.
18. S. Schwendimann. A new one-pass tableau calculus for PLTL. In *Theorem Proving with Analytic Tableaux and Related Methods*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 277–291. Springer-Verlag, 1998.
19. R. Shostak. On the SUP-INF method for proving Presburger formulas. *JACM*, 24(4):529–543, 1977.