A HOL Conversion for Translating Linear Time Temporal Logic to ω-Automata*

Klaus Schneider and Dirk W. Hoffmann

University of Karlsruhe, Department of Computer Science,
Institute for Computer Design and Fault Tolerance (Prof. D. Schmid),
P.O. Box 6980, 76128 Karlsruhe, Germany,
{Klaus.Schneider, Dirk.Hoffmann}@informatik.uni-karlsruhe.de,
http://goethe.ira.uka.de/

Abstract We present an embedding of linear time temporal logic LTL in HOL together with an elegant translation of LTL formulas into equivalent ω-automata. The translation is completely implemented by HOL rules and is therefore safe. Its implementation is mainly based on preproven theorems such that the conversion works very efficiently. In particular, it runs in linear time in terms of the given formula. The main application of this conversion is the sound integration of symbolic model checkers as (unsafe) decision procedures in the HOL theorem prover. On the other hand, the conversion also enables HOL users to directly verify temporal properties by means of HOL's induction rules.

1 Introduction

Specifications of reactive systems such as digital hardware circuits are conveniently given in temporal logics (see [1] for an survey). As the system that is to be checked can be directly viewed as a model of temporal logic formulas, the verification of these specification is usually done with so-called model checking procedures which have found a growing interest in the past decade. Tools such as SMV [2], SPIN [3], COSPAN [4], HSIS [5], and VIS [6] have already found bugs in real-world examples [7, 8, 9] with more than 10^{30} states.

While this is an enormous number for control-oriented systems, this number of states is quickly exceeded if data paths are involved. In these cases, the verification with model checking tools often suffers from the so-called state-explosion problem which roughly means that the number of states grows exponentially with the size of the implementation.

For this reason, we need interactive theorem provers such as HOL [10] for the verification of systems that are build up from control and data paths. However, if lemmata or specifications about the control flow are to be verified which do not affect the manipulation of data, then the use of a model checker is often more convenient, whereas the HOL proofs are time-consuming and tedious. This is even more true, if the verification fails and the model checker is able to present a counterexample. Therefore, it is a broadly

* This work has been financed by DFG project 'Verification of embedded systems' and the ESPRIT LTR Project 26241 (Prosper).

Y. Bertot et al. (Eds.): TPHOLs'99, LNCS 1690, pp. 255–272, 1999.

accepted claim that neither the exclusive use of model checkers nor the exclusive use of theorem provers is sufficient for an efficient verification of entire systems.

Instead, we need a combination of tools of both categories. For this reason, the integration of model checkers as (unsafe) tactics of the theorem prover are desirable. We use the notion 'unsafe' to indicate that the validity of the theorem has been checked by an external tool that is trusted, i.e., the validity is *not* checked by HOL's rules.

The HOL theorem prover has already some built-in decision procedures as e.g. `ARITH_CONV`, but a decision procedure for temporal logics, and even more an embedding of temporal logic at all is currently missing, although there is a long-term interest of the HOL users in temporal logic. In fact, one main application of the HOL system is the verification of concurrent systems (cf. the aim of the Prosper project http://www.dcs.gla.ac.uk/prosper). Other theorem provers for higher order logic such as PVS have already integrated model checkers [11].

For a sound integration of a model checker in the HOL theorem prover, we have to address the problem that we must first embed a temporal logic in the HOL logic. The effort of such an embedding depends crucially on the kind of temporal logic that is to be embedded. In general there are linear time and branching time temporal logics. Linear time temporal logics LTL state facts on computation paths of the systems, i.e., subformulas of this logic are viewed as HOL formulas of type $\mathbb{N} \to \mathbb{B}$ (where $\mathbb{N}$ represents the type of natural numbers and $\mathbb{B}$ the type of Boolean values). Branching time temporal logics, as, e.g., CTL* [12] can moreover quantify over computation paths, i.e., they can express facts as 'for all computation paths leaving this state some linear time property holds' or 'there is a computation path leaving this state that satisfies some linear time property'.

We will see in the following that the embedding of LTL is straightforwardly done in HOL since we only have to define the temporal operators as functions that are applied on arguments of type $\mathbb{N} \to \mathbb{B}$. Embeddings of branching time temporal logics would additionally require to formalize the computation paths of the system under consideration and therefore cause much more effort. This is the reason why we have decided to embed LTL in HOL instead of the more powerful logic CTL*.

Unfortunately the most efficient model checking tools, namely symbolic model checkers such as SMV, are not able to directly deal with LTL specifications. Instead, they are only able to handle a very restricted subset of CTL* called CTL [13] which is a subset of the alternation free μ-calculus. For this reason, model checking can be reduced to the computation of fixpoints.

Hence, the use of symbolic model checking procedures as decision procedures for LTL formulas in HOL requires the conversion of the LTL formulas into a format that the model checker can handle. In general, it is not possible to translate LTL to CTL, although there are some fragments of LTL that allow this [14]. However, it is well-known that LTL can be translated to finite-state ω-automata so that verifying an LTL formula is reduced to a language containment problem. The latter can be presented as a fixpoint problem that can be finally solved with a symbolic model checker.

Therefore, the contribution of this paper is twofold: first, we present an embedding of LTL together with a large theory of preproven theorems that can be used for various purposes. The verification of specifications given in this temporal logic can be veri-

fied with traditional means in HOL and is then not limited to propositional temporal logic. For example it also allows the verification of properties as the ones given in [15]. Second, we have implemented a transformation of LTL into a generalized form of non-deterministic Büchi automata in the form of a HOL conversion. It is to be noted that the implementation only makes use of primitive HOL rules and is therefore save (i.e., the `mk_thm` function is not used). Using this conversion, we can translate each LTL formula into an equivalent ω-automaton and have then the choice to verify the remaining property by means of traditional HOL tactics or by calling a model checker such as SMV. An implementation of the theory and the conversion is freely available under http://goethe.ira.uka.de/~schneider/.

The paper has the following outline: in the next section, we list some related work done in the HOL community and discuss then several approaches for the translation of LTL into finite state ω-automata that have been developed outside HOL. In Section 4, we present our embedding of LTL and the description of ω-automata in HOL. Section 5 describes the algorithm for transforming LTL formulas into generalized Büchi automata and Section 6 explains the application of the work within the Prosper project[1]. The paper then ends with some conclusions.

2 Embedding Temporal Logic and Automata in HOL

Pioneering work on embedding automata theory in HOL has been done by Loewenstein [16, 17]. Schneider, Kumar and Kropf [18] presented non-standard proof procedures for the verification of finite state machines in HOL. A HOL theory for finite state automata for the embedding of hardware designs has been presented by Eisenbiegler and Kumar [19]. Schneider and Kropf [20] presented a unified approach based on automaton-like formulas for combining different formalisms. In this paper, we have, however, no real need for an automata theory and have therefore taken a direct representation of automata by means of HOL formulas similar to [20] as given in section 4.2.

In the domain of temporal logics, Agerholm and Schjodt [21] were the first who made a model checker available for HOL. Von Wright [22] mechanized TLA (Temporal Logic of Actions) [23] in HOL. Andersen and Petersen [24] have implemented a package for defining minimal or maximal fixpoints of Boolean function transformers. As applications of their work, they embedded the temporal logic CTL [13] and Unity [25] in HOL which enables them to reason about Unity programs in HOL [26, 27] (which was their primary aim).

However, none of the mentioned authors considered the temporal logic LTL, although this logic is often preferred in specification since it is known to be a very readable temporal logic. Therefore, the work that comes closest to the one presented here is probably done by Schneider [28] (also [20]) who presented a subset of LTL that can be translated to deterministic ω-automata by means of closures. This paper presents however another approach to translate full LTL to ω-automata, but can be combined with the approach of [28] to reduce the number of states of the automaton [29].

[1] http://www.dcs.gla.ac.uk/prosper

3 Translating Linear Time Temporal Logic to ω-Automata

In this section, we discuss the state of the art of the translation of LTL formulas to ω-automata. Introductions to temporal logics and ω-automata are given in [1] and [30], respectively.

Translations of LTL to ω-automata has been studied in a lot of papers. Among the first ones is the work of Lichtenstein and Pnueli [31, 32] and those of Wolper [33, 34]. These procedures explicitly construct an ω-automaton whose states are sets of subformulas of the given LTL formula. Hence, the number of states of the obtained automaton is of order $2^{O(n)}$ where n is the length of the considered LTL formula. Hence, also the translation procedure for converting a given LTL formula to an equivalent ω-automaton is of order $2^{O(n)}$.

Another approach for translating LTL into ω-automata has been given in [35]. There, a deterministic Muller automaton is generated by a bottom-up traversal through the syntax tree of the LTL formula. Boolean connectives are reduced to closures under complementation, union and intersection of the corresponding automata. The closure under temporal operators is done by first constructing a nondeterministic automaton that is made deterministic afterwards. Clearly, this approach suffers from the high complexity of the determinization algorithm which has to be applied for each temporal operator. Note that the determinization of ω-automata is much more complex than the determinization of finite-state automata on finite words: while for any nondeterministic automaton on finite words with n states there is an equivalent one with $2^{O(n)}$ states, it has been proved in [36] that the optimal bound for ω-automata is $2^{O(n \log(n))}$.

An elegant way for translating LTL into ω-automata by means of *alternating* ω-automata has been presented in [37]. Similar to the method presented in this paper, the translation with alternating ω-automata runs in linear runtime and space (in terms of the length of the given LTL formula), but the resulting alternating ω-automata cannot be easily handled with standard model checkers. Therefore, if we used this conversion, we would need another translation from alternating ω-automata to traditional ω-automata with an exponential blow-up.

The translation procedure we use neither needs to determinize the automata nor is it based on alternating ω-automata. Its runtime is nevertheless very efficient: we can translate any LTL formula of length n in time $O(n)$, i.e., in linear runtime in an ω-automaton with $2^{O(n)}$ states. The trick is not to construct the automaton explicitly, since this would clearly imply exponential runtime.

The translation goes back to [38] where it has been used for the implementation of a LTL front-end for the SMV model checker. In [39] it is shown that this translation can even be generalized to eliminate linear time operators in arbitrary temporal logic specifications so that even a model checking procedure for CTL* can be obtained.

4 Representing ω-Automata and Temporal Logic in HOL

The translation procedure that we use is a special case of the product model checking procedure given in [39] and has been presented in [38]. We first give the essential idea of the translation procedure and its implementation as a HOL conversion and give then the details of the LTL embedding and the formal presentation of the translation procedure.

The first step of the translation procedure is the computation of a 'definitional normal form' for a given LTL formula Φ, which looks in general as given below:

$$\exists \ell_1 \ldots \ell_n . \left(\bigwedge_{i=1}^{n} \ell_i = \varphi_i \right) \wedge \Psi$$

In the above formula, φ_i contains exactly one temporal operator which is the top-level operator of φ_i. Moreover, φ_i contains only the variables that occur in Φ plus the variables $\ell_1,\ldots,\ \ell_{i-1}$. Ψ is a propositional formula that contains the variables $\ell_1,\ldots,\ \ell_n$ and the variables occurring in Φ, so that Φ could be obtained from Ψ by replacing the variables $\ell_1,\ldots,\ \ell_n$ with $\varphi_1,\ldots,\varphi_n$, respectively, in this order.

The computation of this normal form is essentially done by the function tableau that is given in Figure 2. The equivalence between the above normal form and the original formula Φ can be proved by a very simple HOL tactic. One direction is proved by stripping away all quantifiers and the Boolean connectives so that a rewrite step with the assumptions proves the goal. The other direction is simply obtained by instantiating the witnesses φ_i, respectively. In HOL syntax, the tactic is written as follow:

```
EQ_TAC THENL
   [REPEAT STRIP_TAC,
    MAP_EVERY EXISTS_TAC [φ1, ..., φn]]
THEN ASM_REWRITE_TAC[]
```

Having computed this normal form and proven the equivalence with our original formula Φ, we then make use of preproven theorems that allow us to replace the 'definitions' $\ell_i = \varphi_i$ by equivalent nondeterministic ω-automata[2]. Repeating this substitution, we finally end up with a product ω-automaton, whose initial states are determined by Ψ, and whose transition relation and acceptance condition is formed by the equations $\ell_i = \varphi_i$.

4.1 Representing Linear Time Temporal Logic in HOL

We now present the formal definition of the linear time temporal logic that is considered in this paper. After defining syntax and semantics of the logic, we also present how we have embedded the logic in a very simple manner in the HOL logic. So, let us first consider the definition of the linear time temporal logic LTL.

Definition 1 (Syntax of LTL). The following mutually recursive definitions introduce the set of formulas of LTL over a given set of variables V_Σ:

- each variable in V_Σ is a formula, i.e., $V_\Sigma \subseteq \mathsf{LTL}$
- $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi \in \mathsf{LTL}$ if $\varphi, \psi \in \mathsf{LTL}$
- $\mathsf{X}\varphi$, $[\varphi \,\underline{\mathsf{U}}\, \psi] \in \mathsf{LTL}$ if $\varphi, \psi \in \mathsf{LTL}$

[2] See http://goethe.ira.uka.de/~schneider/my_tools/ltlprover/Temporal_Logic.html.

Informally, the semantics is given relative to a considered point of time t_0 as follows: $\mathsf{X}a$ holds at time t_0 iff a holds at time t_0+1; $[a \mathbin{\underline{\mathsf{U}}} b]$ holds at time t_0 iff there is a point of time $t+t_0$ in the future of t_0 where b becomes true and a holds until that point of time.

For a formal presentation of the semantics, we first have to define models of the temporal logic: Given a set of variables V_Σ, a model for LTL is a function π of type $\mathbb{N} \to \wp(V_\Sigma)$ where $\wp(M)$ denotes the powerset of a set M. The interpretation of formulas is then done according to the following definition.

Definition 2 (Semantics of LTL). Given a finite a set of variables V_Σ and a model $\pi : \mathbb{N} \to \wp(V_\Sigma)$. Then, the following rules define the semantics of LTL:

- $\pi \models x$ iff $x \in \pi^{(0)}$
- $\pi \models \neg\varphi$ iff $\pi \not\models \varphi$
- $\pi \models \varphi \wedge \psi$ iff $\pi \models \varphi$ and $\pi \models \psi$
- $\pi \models \varphi \vee \psi$ iff $\pi \models \varphi$ or $\pi \models \psi$
- $\pi \models \mathsf{X}\varphi$ iff $\lambda t.\pi^{(t+1)} \models \varphi$
- $\pi \models [\varphi \mathbin{\underline{\mathsf{U}}} \psi]$ iff there is a $\delta \in \mathbb{N}$ such that $\lambda t.\pi^{(t+\delta)} \models \psi$ and for all $d < \delta$ it holds that $\lambda t.\pi^{(t+d)} \models \varphi \wedge \neg\psi$

As a generalization to the above definition, temporal logics such as LTL are often interpreted over finite state Kripke structures. These are finite state transition systems where each state is labeled with a subset of variables of V_Σ. To interpret a LTL formula along a path π of such a Kripke structure, the above definition is used. As it is however easily possible to define a set of admissible paths in the form of a finite state transition system (similar to our representation of finite-state ω-automata as given below), we can 'mimic' Kripke structures easily in HOL without the burden of deep-embedding Kripke structures.

It is also easily seen that we can consider the projections $\pi_x := \lambda t.\pi^{(t)} \cap \{x\}$ for all variables $x \in V_\Sigma$ instead of the path $\pi : \mathbb{N} \to \wp(V_\Sigma)$ itself. Using these projections, we can reestablish the path π as $\pi := \lambda t. \bigcup_{x \in V_\Sigma} \pi_x^{(t)}$. Hence, it is only a matter of taste whether we use the path π or its projections π_x as a model for the LTL logic. Using the projections directly leads to our HOL representation of LTL formulas: We represent variables of LTL in the HOL logic directly as HOL variables of type $\mathbb{N} \to \mathbb{B}$. So this simplification of the semantics allows a very easy treatment of LTL in HOL that even circumvents a deep-embedding of the LTL syntax at all. With this point of view, we have the following definitions of temporal operators, i.e., the embedding of LTL, in HOL:

Definition 3 (Defining Temporal Operators in HOL). The definition of temporal operators X and $\underline{\mathsf{U}}$ in HOL is as follows (for any $p, q : \mathbb{N} \to \mathbb{B}$):

- $\mathsf{X}p := \lambda t.p^{(t+1)}$
- $[p \mathbin{\underline{\mathsf{U}}} q] := \lambda t.\exists\delta.q^{(t+\delta)} \wedge \forall d.d < \delta \to p^{(t+\delta)} \wedge \neg q^{(t+\delta)}$

Note that X is of type $(\mathbb{N} \to \mathbb{B}) \to (\mathbb{N} \to \mathbb{B})$ and $\underline{\mathsf{U}}$ is of type $(\mathbb{N} \to \mathbb{B}) \to (\mathbb{N} \to \mathbb{B}) \to (\mathbb{N} \to \mathbb{B})$.

Clearly, it is possible and desirable to have more temporal operators that describe other temporal relationships as the ones given above. We found that the following set of temporal operators turned out to be adequate for practical use and have therefore added these operators to our LTL theory:

$$\begin{aligned}
&\mathsf{G}p = \lambda t.\forall d.p^{(t+d)}\\
&\mathsf{F}p = \lambda t.\exists d.p^{(t+d)}\\
&[p\ \mathsf{U}\ q] = \lambda t.\left[[\mathsf{F}p]^{(t)} \rightarrow [p\ \underline{\mathsf{U}}\ q]^{(t)}\right] \wedge \left[\neg[\mathsf{F}p]^{(t)} \rightarrow \mathsf{G}p^{(t)}\right]\\
&[p\ \underline{\mathsf{B}}\ q] = \lambda t.\exists\delta.p^{(t+d)} \wedge \left[\forall d.d \leq \delta \rightarrow \neg q^{(t+d)}\right]\\
&[p\ \mathsf{B}\ q] = \lambda t.\forall\delta.\left[\forall d.d < \delta \rightarrow \neg q^{(t+d)}\right] \wedge q^{(t+\delta)} \rightarrow \left[\exists d.d < \delta \wedge p^{(t+d)}\right]\\
&[p\ \underline{\mathsf{W}}\ q] = \lambda t.\exists\delta.p^{(t+d)} \wedge q^{(t+d)}\ \left[\forall d.d < \delta \rightarrow \neg q^{(t+d)}\right]\\
&[p\ \mathsf{W}\ q] = \lambda t.\forall\delta.\left[\forall d.d < \delta \rightarrow \neg q^{(t+d)}\right] \wedge q^{(t+d)} \rightarrow p^{(t+d)}
\end{aligned}$$

G, F, and [U] are the usual always, eventually, and until operators, respectively. There are some remarkable properties that can be found in our HOL theory on temporal operators. For example, we can compute a negation normal form $\mathsf{NNF}(\varphi)$ of a formula φ by the following equations:

$$\begin{aligned}
&\neg[\mathsf{G}p]^{(t)} = [\mathsf{F}[\lambda t.\neg p^{(t)}]]^{(t)} && \neg\,[p\ \mathsf{W}\ q]^{(t)} = \left[[\lambda t.\neg p^{(t)}]\ \underline{\mathsf{W}}\ p\right]^{(t)}\\
&\neg[\mathsf{F}p]^{(t)} = [\mathsf{G}[\lambda t.\neg p^{(t)}]]^{(t)} && \neg\,[p\ \underline{\mathsf{W}}\ q]^{(t)} = \left[[\lambda t.\neg p^{(t)}]\ \mathsf{W}\ p\right]^{(t)}\\
&\neg\,[p\ \underline{\mathsf{U}}\ q]^{(t)} = \left[[\lambda t.\neg p^{(t)}]\ \mathsf{B}\ p\right]^{(t)} && \neg\,[p\ \mathsf{B}\ q]^{(t)} = \left[[\lambda t.\neg p^{(t)}]\ \underline{\mathsf{U}}\ p\right]^{(t)}\\
&\neg\,[p\ \mathsf{U}\ q]^{(t)} = \left[[\lambda t.\neg p^{(t)}]\ \underline{\mathsf{B}}\ p\right]^{(t)} && \neg\,[p\ \underline{\mathsf{B}}\ q]^{(t)} = \left[[\lambda t.\neg p^{(t)}]\ \mathsf{U}\ p\right]^{(t)}
\end{aligned}$$

Also, it is shown that any of the binary temporal operators can be expressed by any other binary temporal operator. Hence, we could restrict our considerations, e.g., to the temporal operators X and $\underline{\mathsf{U}}$ and use the following reduction rules:

$$\begin{aligned}
&\mathsf{G}p = \lambda t.\neg\left[[\lambda t.\mathsf{T}]\ \underline{\mathsf{U}}\ [\lambda t.\neg p^{(t)}]\right]^{(t)}\\
&\mathsf{F}p = [p\ \underline{\mathsf{U}}\ [\lambda t.\mathsf{T}]]\\
&[p\ \mathsf{U}\ q] = \lambda t.\,[p\ \underline{\mathsf{U}}\ q]^{(t)} \vee [\mathsf{G}p]^{(t)}\\
&[p\ \mathsf{B}\ q] = \lambda t.\neg\left[[\lambda t.\neg p^{(t)}]\ \underline{\mathsf{U}}\ q\right]^{(t)}\\
&[p\ \mathsf{W}\ q] = \lambda t.\,\left[[\lambda t.\neg q^{(t)}]\ \underline{\mathsf{U}}\ [\lambda t.p^{(t)} \wedge q^{(t)}]\right]^{(t)} \vee [\mathsf{G}[\lambda t.\neg q^{(t)}]]^{(t)}\\
&[p\ \underline{\mathsf{W}}\ q] = \lambda t.\,\left[[\lambda t.\neg q^{(t)}]\ \underline{\mathsf{U}}\ [\lambda t.p^{(t)} \wedge q^{(t)}]\right]^{(t)}\\
&[p\ \underline{\mathsf{B}}\ q] = \lambda t.\neg\left[[\lambda t.\neg p^{(t)}]\ \underline{\mathsf{U}}\ q\right]^{(t)} \wedge [\mathsf{F}p]^{(t)}
\end{aligned}$$

Note, however, that the computation of the negation normal $\mathsf{NNF}(\varphi)$ of a formula φ form as given above reintroduces the B operator, so that we deal with X, $\underline{\mathsf{U}}$, and B in the following.

4.2 Representing ω-Automata in HOL

Let us now consider how we represent ω-automata in HOL. In general, an ω-automaton consists of a finite state transition system where transitions between two states are enabled if a certain input is read. A given sequence of inputs then induces one or more

sequences of states that are called *runs over the input word*. A word is accepted iff there is a run for that word satisfying the acceptance condition of the automaton. Different kinds of acceptance conditions have been investigated, consider e.g. [30] for an overview.

The representation of an ω-automaton as a formula in HOL is straightforward: We encode the states of the automaton by a subset of $\mathbb{B}^n$. Hence, a run is encoded by a finite number of state variables $q_0, \ldots, q_n$ which are all of type $\mathbb{N} \to \mathbb{B}$. Similarly, we encode the input alphabet by a subset of $\mathbb{B}^m$ (or isomorphic $\wp(\{x_0, \ldots, x_m\})$) and input sequences with variables $x_0, \ldots, x_m$ of type $\mathbb{N} \to \mathbb{B}$. Then, we represent an ω-automaton as a HOL formula of the following form:

$$\begin{array}{l}\exists q_0 \ldots q_n. \\ \quad \Phi_{\mathcal{I}}(q_0^{(0)}, \ldots, q_n^{(0)}) \wedge \\ \quad \Big[\forall t. \Phi_{\mathcal{R}}(q_0^{(t)}, \ldots, q_n^{(t)}, x_0^{(t)}, \ldots, x_m^{(t)}, q_0^{(t+1)}, \ldots, q_n^{(t+t)})\Big] \wedge \\ \quad \Phi_{\mathcal{F}}(q_0, \ldots, q_n)\end{array}$$

$\Phi_{\mathcal{I}}(q_0^{(0)}, \ldots, q_n^{(0)})$ is thereby a propositional formula where only the atomic formulas $q_0^{(0)}, \ldots, q_n^{(0)}$ may occur. $\Phi_{\mathcal{I}}$ represents the set of initial states of the automaton. Any valuation of the atomic formulas $q_0^{(0)}, \ldots, q_n^{(0)}$ that satisfies $\Phi_{\mathcal{I}}$ is an initial state of the automaton. Hence, the set of initial states is the set of Boolean tuples $(b_0, \ldots, b_n) \in \mathbb{B}^n$ such that $\Phi_{\mathcal{I}}(b_0, \ldots, b_n)$ is equivalent to T.

Similarly, $\Phi_{\mathcal{R}}(\ldots)$ is a propositional formula where only the atomic formulas $q_i^{(t)}$, $x_i^{(t)}$, and $q_i^{(t+1)}$ may occur. $\Phi_{\mathcal{R}}$ represents the transition relation of the ω-automaton as follows: there is a transition from state $(b_0, \ldots, b_n) \in \mathbb{B}^n$ to the state $(b'_0, \ldots, b'_n) \in \mathbb{B}^n$ for the input $(a_0, \ldots, a_m) \in \mathbb{B}^m$, iff $\Phi_{\mathcal{R}}(b_0, \ldots, b_n, a_0, \ldots, a_m, b'_0, \ldots, b'_n)$ is equivalent to T.

$\Phi_{\mathcal{F}}(q_0, \ldots, q_n)$ is the acceptance condition of the automaton. Note that $\Phi_{\mathcal{R}}$ may be partially defined, i.e., there may be input sequences x_i that have no run through the transition system, i.e., the formula may not be satisfied even without considering the acceptance condition. In general, the following types of acceptance conditions are distinguished, where all formulas Φ_k, Ψ_k are propositional formulas over $q_0^{(t_1+t_2)}, \ldots, q_n^{(t_1+t_2)}$:

Büchi:	$\forall t_1.\exists t_2.\Phi_0$
Generalized Büchi:	$\bigwedge_{k=1}^{a} [\forall t_1.\exists t_2.\Phi_k]$
Streett:	$\bigwedge_{k=1}^{a} [\forall t_1.\exists t_2.\Phi_k] \vee [\exists t_1.\forall t_2.\Psi_k]$
Rabin:	$\bigvee_{k=1}^{a} [\forall t_1.\exists t_2.\Phi_k] \wedge [\exists t_1.\forall t_2.\Psi_k]$

It can be shown that the nondeterministic versions of the above ω-automata have the same expressive power [30]. Therefore, we could use any of them for our translation. In the following, we focus on generalized Büchi automata. It will become clear after the next section, why this kind of ω-automaton is an appropriate means for a simple translation of temporal logic inside HOL.

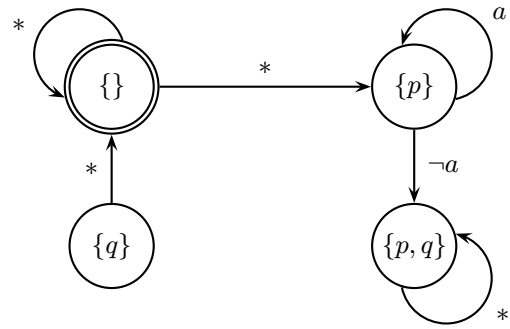

Fig.1. An example ω-automaton

As an example of the representation of an ω-automaton, consider Figure 1. This ω-automaton is represented by the following HOL formula:

$$\begin{array}{l}\exists p : \mathbb{N} \to \mathbb{B}.\ \exists q : \mathbb{N} \to \mathbb{B}.\\ \quad \neg p^{(0)} \wedge \neg q^{(0)}\\ \quad \wedge \forall t.\ \left(\begin{array}{l}(p^{(t)} \to p^{(t+1)}) \wedge (p^{(t+1)} \to p^{(t)} \vee \neg q^{(t)}) \wedge\\ (q^{(t+1)} = (p^{(t)} \wedge \neg q^{(t)} \wedge \neg a^{(t)}) \vee (p^{(t)} \wedge q^{(t)}))\end{array}\right)\\ \quad \wedge \forall t_1.\exists t_2.p^{(t_1+t_2)} \wedge \neg q^{(t_1+t_2)}\end{array}$$

This means that the only initial state is the one where neither p nor q does hold (drawn with double lines in figure 1). The transition relation is simple as given in figure 1. The acceptance condition requires that a run must visit infinitely often the state where p holds, but q does not hold. Hence, any accepting run starts in state $\{\}$ and must finally loop in state $\{p\}$ so that the automaton formula is equivalent to $\exists t_1.\forall t_2.a^{(t_1+t_2)}$.

5 The Tableau Procedure

Several tableau procedures have been designed for various kinds of logics, e.g., first order logic, μ-calculi, and, of course, temporal logics. While the standard tableau procedure for LTL has been presented in [33, 34], we follow the procedure given in [38]. This works roughly as follows: for each subformula φ that starts with a temporal operator and contains no further temporal operators, a new state variable $\ell_\varphi : \mathbb{N} \to \mathbb{B}$ is generated. On each path, the state transitions of the automaton should be such that the variable ℓ_φ exactly behaves like the subformula φ does, hence $\ell_\varphi = \varphi$ must hold along each path. By applying the substitution of subformulas recursively, we can reduce a given LTL formula Φ to a set of equations $E = \{\ell_i = \varphi_i \mid i \in I\}$ and some propositional formula Ψ such that $\bigwedge_{i \in I} \mathsf{G}[\ell_i = \varphi_i] \to (\Phi = \Psi)$ is valid. Clearly, if we resubstitute the variables ℓ_i by φ_i in Ψ, we obtain the original formula Ψ again (syntactically).

In the following, we simplify the syntax to obtain more readable formulas: we neglect applications on time and λ-abstraction of the time variable. For example, we write $\neg\,[[\neg p]\ \underline{\mathsf{U}}\ q] \wedge [\mathsf{F}p]$ instead of $\lambda t.\neg\left[[\lambda t.\neg p^{(t)}]\ \underline{\mathsf{U}}\ q\right]^{(t)} \wedge [\mathsf{F}p]^{(t)}$. It is clear from the context, where applications on time and λ-abstractions should be added to satisfy the type rules.

To illustrate the computation of the 'definitional normal form', consider the formula $(\mathsf{FG}a) \rightarrow (\mathsf{GF}a)$. The construction of E and Ψ results in $E = \{\ell_1 = \mathsf{G}a, \ell_2 = \mathsf{F}\ell_1, \ell_3 = \mathsf{F}a, \ell_4 = \mathsf{G}\ell_3\}$ and $\Psi = \ell_2 \rightarrow \ell_4$. The essential step is now to construct a transition relation out of E for an ω-automaton with the state variables ℓ_i such that each ℓ_i behaves as φ_i. This is done by the characterization of temporal operators as fixpoints as given in the next theorem.

Theorem 1 (Characterizing Temporal Operators as Fixpoints). *The following formulas are valid, i.e., they hold on each path:*

$$\begin{aligned}
&\mathsf{G}[y = a \wedge \mathsf{X}y] = \mathsf{G}[y = \mathsf{G}a] \vee \mathsf{G}[y = \mathsf{F}]\\
&\mathsf{G}[y = a \vee \mathsf{X}y] = \mathsf{G}[y = \mathsf{F}a] \vee \mathsf{G}[y = \mathsf{T}]\\
&\mathsf{G}[y = (b \Rightarrow a|\,\mathsf{X}y)] = \mathsf{G}[y = [a\ \mathsf{W}\ b]] \vee \mathsf{G}[y = [a\ \underline{\mathsf{W}}\ b]]\\
&\mathsf{G}[y = b \vee a \wedge \mathsf{X}y] = \mathsf{G}[y = [a\ \mathsf{U}\ b]] \vee \mathsf{G}[y = [a\ \underline{\mathsf{U}}\ b]]\\
&\mathsf{G}[y = \neg b \wedge (a \vee \mathsf{X}y)] = \mathsf{G}[y = [a\ \mathsf{B}\ b]] \vee \mathsf{G}[y = [a\ \underline{\mathsf{B}}\ b]]
\end{aligned}$$

Hence, each of the equations $y = a \wedge \mathsf{X}y$, $y = a \vee \mathsf{X}y$, $y = (b \Rightarrow a|\,\mathsf{X}y)$, $y = b \vee a \wedge \mathsf{X}y$, and $y = \neg b \wedge (a \vee \mathsf{X}y)$, has exactly two solutions for y.

The following formulas are also valid and show how one of the two solutions of the above fixpoint equations can be selected with different fairness constraints:

$$\begin{aligned}
&\mathsf{G}[y = \mathsf{G}a] = \mathsf{G}[y = a \wedge \mathsf{X}y] \wedge \mathsf{GF}[a \rightarrow y]\\
&\mathsf{G}[y = \mathsf{F}a] = \mathsf{G}[y = a \vee \mathsf{X}y] \wedge \mathsf{GF}[y \rightarrow a]\\
&\mathsf{G}[y = [a\ \mathsf{W}\ b]] = \mathsf{G}[y = (b \Rightarrow a|\,\mathsf{X}y)] \wedge \mathsf{GF}[y \vee b]\\
&\mathsf{G}[y = [a\ \underline{\mathsf{W}}\ b]] = \mathsf{G}[y = (b \Rightarrow a|\,\mathsf{X}y)] \wedge \mathsf{GF}[y \rightarrow b]\\
&\mathsf{G}[y = [a\ \mathsf{U}\ b]] = \mathsf{G}[y = b \vee a \wedge \mathsf{X}y] \wedge \mathsf{GF}[y \vee \neg a \vee b]\\
&\mathsf{G}[y = [a\ \underline{\mathsf{U}}\ b]] = \mathsf{G}[y = b \vee a \wedge \mathsf{X}y] \wedge \mathsf{GF}[\neg y \vee \neg a \vee b]\\
&\mathsf{G}[y = [a\ \mathsf{B}\ b]] = \mathsf{G}[y = \neg b \wedge (a \vee \mathsf{X}y)] \wedge \mathsf{GF}[y \vee a \vee b]\\
&\mathsf{G}[y = [a\ \underline{\mathsf{B}}\ b]] = \mathsf{G}[y = \neg b \wedge (a \vee \mathsf{X}y)] \wedge \mathsf{GF}[\neg y \vee a \vee b]
\end{aligned}$$

If we define an ordering relation on terms of type $\mathbb{N} \rightarrow \mathbb{B}$ by $\alpha \preceq \beta :\Leftrightarrow \forall t.\alpha^{(t)} \rightarrow \beta^{(t)}$, then we can also state that $\mathsf{G}a$ is the greatest fixpoint of $f_a(y) := a \wedge \mathsf{X}y$, and so on. Consequently, the above theorem characterizes each temporal operator as a least or greatest fixpoint of some function.

As can be seen, the strong and weak binary temporal operators satisfy the same fixpoint equations, i.e., they are both solutions of the same fixpoint equation. Furthermore, the equations of the first part show that there are exactly two solutions of the fixpoint equations. The strong version of a binary operator is the least fixpoint of the equations, while the weak version is the greatest fixpoint. Hence, replacing an equation as, e.g., $\ell = [a\ \mathsf{U}\ b]$ by adding $\ell = b \vee a \wedge \mathsf{X}\ell$ to the transition relation fixes ℓ such that it behaves either as $[a\ \mathsf{U}\ b]$ or $[a\ \underline{\mathsf{U}}\ b]$.

Moreover, the formulas of the second part of the above theorem show how we can assure that the newly generated variables ℓ_i can be fixed to be either the strong or the weak version of an operator by adding additional fairness constraints. These fairness constraints distinguish between the two solutions of the fixpoint equation and select safely one of both solutions.

function tableau(Φ)
  case Φ **of**
    is_prop(φ) : **return** $(\{\}, \varphi)$;
    $\neg\varphi$: $(E_1, \varphi_1) \equiv$ tableau(φ);
      return $(E_1, \neg\varphi_1)$;
    $\varphi \wedge \psi$: $(E_1, \varphi_1) \equiv$ tableau(φ); $(E_2, \psi_1) \equiv$ tableau(ψ);
      return $(E_1 \cup E_2, \varphi_1 \wedge \psi_1)$;
    $\varphi \vee \psi$: $(E_1, \varphi_1) \equiv$ tableau(φ); $(E_2, \psi_1) \equiv$ tableau(ψ);
      return $(E_1 \cup E_2, \varphi_1 \vee \psi_1)$;
    $\mathsf{X}\varphi$: $(E_1, \varphi_1) \equiv$ tableau(φ); $\ell =$ new_var;
      return $(E_1 \cup \{\ell = \mathsf{X}\varphi_1\}, \ell)$;
    $[\varphi\ \mathsf{B}\ \psi]$: $(E_1, \varphi_1) \equiv$ tableau(φ); $(E_2, \psi_1) \equiv$ tableau(ψ); $\ell =$ new_var;
      return $(E_1 \cup E_2 \cup \{\ell = [\varphi_1\ \mathsf{B}\ \psi_1]\}, \ell)$;
    $[\varphi\ \underline{\mathsf{U}}\ \psi]$: $(E_1, \varphi_1) \equiv$ tableau(φ); $(E_2, \psi_1) \equiv$ tableau(ψ); $\ell =$ new_var;
      return $(E_1 \cup E_2 \cup \{\ell = [\varphi_1\ \underline{\mathsf{U}}\ \psi_1]\}, \ell)$;

function trans(ℓ, Φ)
  case Φ **of**
    $\ell = \mathsf{X}\varphi$: **return** $\ell = \mathsf{X}\varphi$;
    $\ell = [\varphi\ \mathsf{B}\ \psi]$: **return** $\ell = \neg\psi \wedge (\varphi \vee \mathsf{X}\ell)$;
    $\ell = [\varphi\ \underline{\mathsf{U}}\ \psi]$: **return** $\ell = \psi \vee \varphi \wedge \mathsf{X}\ell$;

function fair(ℓ, Φ)
  case Φ **of**
    $\ell = \mathsf{X}\varphi$: **return** T;
    $\ell = [\varphi\ \mathsf{B}\ \psi]$: **return** $\mathsf{GF}(\ell \vee \varphi \vee \psi)$;
    $\ell = [\varphi\ \underline{\mathsf{U}}\ \psi]$: **return** $\mathsf{GF}(\ell \rightarrow \neg\varphi \vee \psi)$;

function Tableau(Φ)
  $(\{\ell_1 = \varphi_1, \ldots, \ell_n = \varphi_n\}, \Phi_{\mathcal{I}}) :=$ tableau(NNF(Φ));
  $\Phi_{\mathcal{R}} := \bigwedge_{i=1}^{n}$ trans(ℓ_i, φ_i);
  $\Phi_{\mathcal{F}} := \bigwedge_{i=1}^{n}$ fair(ℓ_i, φ_i);
  return $\mathcal{A}_{\exists}(\{\ell_1, \ldots, \ell_b\}, \Phi_{\mathcal{I}}, \Phi_{\mathcal{R}}, \Phi_{\mathcal{F}})$;

Fig.2. Algorithm for translating LTL to ω-automata

It is clear that the equations in the second part of the theorem tell us how to replace the definitions $\ell_i = \varphi_i$ by an equivalent ω-automaton. For example, the definition $\ell_1 = \mathsf{G}a$ is replaced with the formula:

$$[\forall t.\ell_1^{(t)} = a^{(t)} \wedge \ell_1^{(t+1)}] \wedge \forall t_1.\exists t_2.a^{(t_1+t_2)} \rightarrow \ell_1^{(t_1+t_2)}$$

As the introduced variables ℓ_i occur under an existential quantifier, this formula corresponds directly to a generalized Büchi automaton. Hence, the following theorem holds:

Theorem 2 (Translating LTL to ω-Automata). *For the algorithm given in Figure 2, the following holds for any* LTL *formula Φ and for $(E, \Psi) =$* tableau(Φ)*:*

- $|E| \in O(|\Phi|)$ *and* $|\Psi| + \sum_{\ell_i=\varphi_i \in E} |\varphi_i| \in O(|\Phi|)$*, which implies linear runtime in terms of* $|\Phi|$ *of* $\mathsf{tableau}(\Phi)$
- Ψ *is a propositional formula*
- *for each* $\ell_i = \varphi_i \in E$*,* φ_i *is a formula with exactly one temporal operator that occurs on top of* φ_i
- φ_i *contains at most the variables that occur in* Φ *plus the variables* $\ell_1,\ldots,\ell_{i-1}$
- $\Phi = \exists \ell_1 \ldots \ell_n . \Psi \wedge \bigwedge_{i\in I} \mathsf{G}[\ell_i = \varphi_i]$

The result of the function Tableau *results in an equivalent generalized Büchi automaton with the initial states* Ψ*, transition relation* $\bigwedge_{\ell_i=\varphi_i\in E} \mathsf{trans}(\ell_i,\varphi_i)$ *and acceptance condition* $\bigwedge_{\ell_i=\varphi_i\in E} \mathsf{fair}(\ell_i,\varphi_i)$.

The construction yields in an automaton with $2^{O(|\Phi|)}$ states and an acceptance condition of length $O(|\Phi|)$. Note that the constructed ω-automaton is in general nondeterministic. This can not be avoided, since deterministic Büchi automata are not as expressive as nondeterministic ones [30].

For our example $\mathsf{FG}a \rightarrow \mathsf{GF}a$, we derive the following ω-automaton:

$$
\begin{array}{l}
\exists \ell_1\, \ell_2\, \ell_3\, \ell_4. \\
\quad \left[\ell_2^{(0)} \rightarrow \ell_4^{(0)}\right] \wedge \\
\quad \forall t. \begin{bmatrix} \ell_1^{(t)} = a^{(t)} \wedge \ell_1^{(t+1)} \wedge \ell_2^{(t)} = \ell_1^{(t)} \vee \ell_2^{(t+1)} \wedge \\ \ell_3^{(t)} = a^{(t)} \vee \ell_3^{(t+1)} \wedge \ell_4^{(t)} = \ell_3^{(t)} \wedge \ell_4^{(t+1)} \end{bmatrix} \wedge \\
\quad \left(\begin{bmatrix} \forall t_1.\exists t_2. a^{(t_1+t_2)} \rightarrow \ell_1^{(t_1+t_2)} \end{bmatrix} \wedge \begin{bmatrix} \forall t_1.\exists t_2. \ell_2^{(t_1+t_2)} \rightarrow \ell_1^{(t_1+t_2)} \end{bmatrix} \wedge \right. \\
\quad \left. \begin{bmatrix} \forall t_1.\exists t_2. \ell_3^{(t_1+t_2)} \rightarrow a^{(t_1+t_2)} \end{bmatrix} \wedge \begin{bmatrix} \forall t_1.\exists t_2. \ell_3^{(t_1+t_2)} \rightarrow \ell_4^{(t_1+t_2)} \end{bmatrix} \wedge \right)
\end{array}
$$

As another example, consider the translation of a property that is to be verified for the single pulser circuit [40]:

$$\mathsf{G}[\neg i \wedge \mathsf{X}i \rightarrow \mathsf{X}([o\ \mathsf{B}\ (\neg i \wedge \mathsf{X}i)] \vee [o\ \mathsf{W}\ (\neg i \wedge \mathsf{X}i)])]$$

The formula specifies that after a rising edge of the input i, the output o must hold at least once before or at the time where the next rising edge of i occurs. The translation begins with the abbreviation of the subformulas starting with temporal operators. We obtain the definitions $\ell_0 := \mathsf{X}i$, $\ell_1 := [o\ \mathsf{B}\ (\neg i \wedge \ell_0)]$, $\ell_2 := [o\ \mathsf{W}\ (\neg i \wedge \ell_0)]$, $\ell_3 := \mathsf{X}(\ell_1 \vee \ell_2)$, and $\ell_4 := \mathsf{G}[\neg i \wedge \ell_0 \rightarrow \ell_3]$. Replacing these definition with the preproven theorems, we finally end up with the following generalized Büchi automaton:

$$
\begin{array}{l}
\exists \ell_0\, \ell_1\, \ell_2\, \ell_3\, \ell_4. \\
\quad \ell_4^{(0)} \wedge \\
\quad \forall t. \left(\begin{bmatrix} \ell_0^{(t)} = i^{(t+1)} \end{bmatrix} \wedge \begin{bmatrix} \ell_1^{(t)} = \neg(\neg i^{(t)} \wedge \ell_0^{(t)}) \wedge (o^{(t)} \vee \ell_1^{(t+1)}) \end{bmatrix} \wedge \right. \\
\quad \left. \begin{bmatrix} \ell_2^{(t)} = \left(\neg i^{(t)} \wedge \ell_0^{(t)} \Rightarrow o^{(t)} \middle| \ell_2^{(t+1)}\right) \end{bmatrix} \wedge \begin{bmatrix} \ell_3^{(t)} = \ell_1^{(t+1)} \vee \ell_2^{(t+1)} \end{bmatrix} \wedge \right. \\
\quad \left. \begin{bmatrix} \ell_4^{(t)} = (\neg i^{(t)} \wedge \ell_0^{(t)} \rightarrow \ell_3^{(t)}) \wedge \ell_4^{(t+1)} \end{bmatrix} \wedge \right) \wedge \\
\quad \left(\begin{array}{l} (\forall t_1.\exists t_2. \ell_1^{(t_1+t_2)} \vee o^{(t_1+t_2)} \vee \neg i^{(t_1+t_2)} \wedge \ell_0^{(t_1+t_2)}) \wedge \\ (\forall t_1.\exists t_2. \ell_2^{(t_1+t_2)} \vee \neg i^{(t_1+t_2)} \wedge \ell_0^{(t_1+t_2)}) \end{array} \right)
\end{array}
$$

We have implemented a HOL conversion that computes a corresponding generalized Büchi automaton as explained above and then proves the equivalence with our pre-proven theorems of Theorem 1. It is easily seen that the computation of the generalized Büchi automaton is done in linear runtime wrt. the length of the given formula.

6 Applications

The Prosper project focuses on reducing the gap between formal verification and industrial aims and needs. Although formal methods have proven their usefulness and importance in academia for years, they are only rarely applied in industrial environments. One reason for this is that existing proof tools require profound knowledge in logic. Since only a very few system engineers have this expertise, formal methods are often not applied at all. Another reason is the lack of automation. Many proof tools need considerable user interaction which complicates its usage and slows down the complete design cycle.

Within Prosper, examples of proof tools are being produced which provide user friendly access to formal methods. An open proof architecture allows the integration of different verification tools in a uniform higher order logic environment. Besides providing easy and consistent access to these tools, a high degree of automation can be achieved by integrating various decision procedures as plug-ins. Examples of already integrated decision procedures are a Boolean tautology checker based on BDDs, PROVER from Prover Technology, and the CTL model checker SMV.

As mentioned before, the translation of LTL formulas to ω-automata as described in the previous section can be used to model check LTL formulas with a CTL model checker. Using HOL's CTL model checker plug-in which has been developed as part of the Prosper project, our transformation procedure shows how LTL model checking can be performed directly within HOL.

6.1 The Prosper Plug-In Interface

The basis for a uniform integration of different proof tools in HOL is Prosper's plug-in Interface [41, 42]. It provides an easy to use and formally specified communication mechanism between HOL and its various proof backends. Communication is either based on Internet sockets or on local Unix pipes, depending on the machine where the proof backend is running. The plug-in interface can roughly be divided into two parts. One part that manages communication with HOL (proof engine side) and another part which is responsible for the plugged in proof backend (plug-in side). If Internet sockets are used for communication, both sides of the plug-in might run on different machines. The proof engine side is directly linked to HOL whereas the plug-in side runs on the same machine as the proof backends. If more than one plug-in is running, each back end is linked by a separate plug-in interface. This allows the integration of arbitrary tools running either locally on the same machine or widespread over the world connected via Internet sockets. Overall, this leads to the sketch in Fig. 3

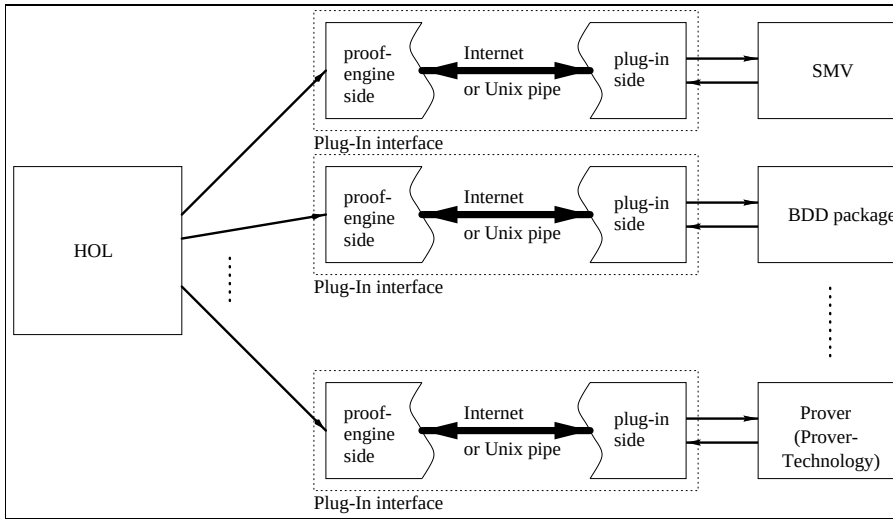

Fig.3. Sketch of the Prosper plug-in Interface

6.2 The CTL Model Checker Plug-In

As part of the Prosper project, we have integrated the SMV model checker [2] into HOL via Prosper's plug-in interface. SMV has been chosen since it is one of the widest spread and most used model checkers. Moreover, SMV is freely available and can therefore be distributed with HOL.

To be able to reason about SMV programs, the SMV language defined in [2] has been deeply embedded in HOL[3]. According to the grammar definition given in [2], a new HOL type is defined for each non-terminal, i.e., the following HOL types are defined:

```
smv_constant      smv_id            smv_expr
smv_ctl           smv_type_enum     smv_type
smv_var_decl      smv_alhs          smv_assign_decl
smv_define_decl   smv_declaration   smv_moduleports
smv
```

Having the SMV grammar in mind, these data types are defined in a straightforward manner, e.g., `smv_constant` is defined as

```
val smv_constant = define_type
    {name = "smv_constant",
     type_spec = ‘smv_constant = ATOM_CONSTANT of string
                               | NUMBER_CONSTANT of num
                               | FALSE_CONSTANT
                               | TRUE_CONSTANT‘,
     fixities = [Prefix,Prefix,Prefix,Prefix]};
```

[3] http://goethe.ira.uka.de/~hoff/smv.sml

All other data types are defined in a similar way. At the moment, the language is only embedded syntactically. The semantics which is also given in [2] will be formalized in HOL soon.

A parser exists which converts SMV programs to their corresponding representation in HOL. Once the datatype has been created, it can be manipulated within HOL, or SMV can be invoked with the `MC` command. Calling SMV via the plug-in interface, the HOL datatype is converted into SMV readable format and passed to the model checker. If the specification is true, the HOL term `T` is returned to HOL, otherwise `F` is returned. Once the semantics of SMV programs has been formalized (e.g., by defining a predicate $\mathsf{valid}(S)$ which is true if and only if S satisfies its specification), the model checker plug-in can be adapted easily to return a theorem $\models \mathsf{valid}(S)$ instead of the HOL term `T`.

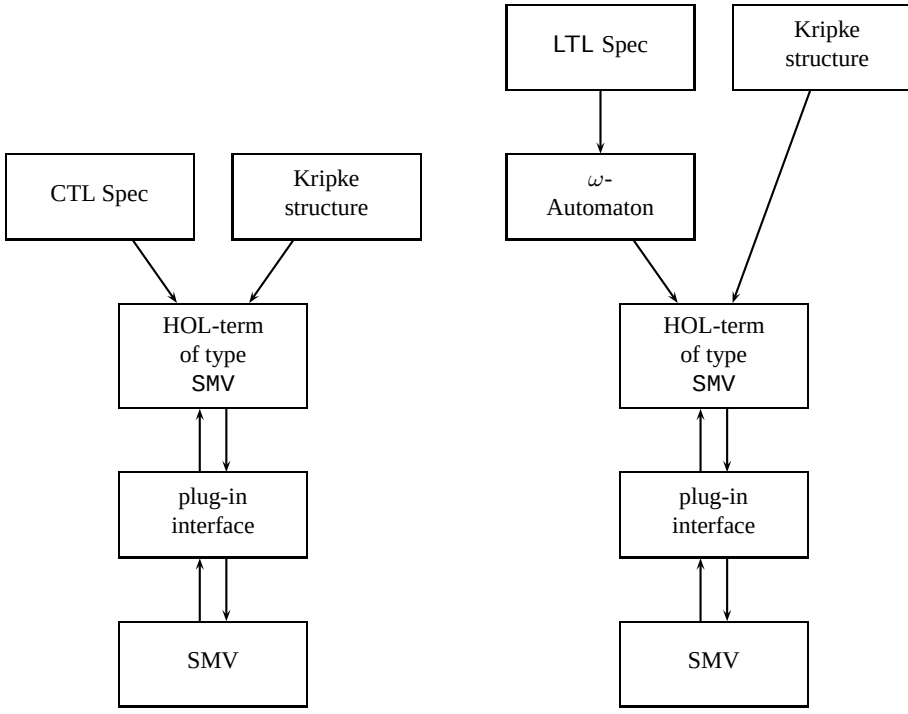

Fig.4. Invoking the SMV model checker via the Prosper plug-in interface. The left picture shows the usual call to SMV with a CTL specification. The right picture demonstrates how our transformation of `LTL` formulas to ω-automata can be used to model check `LTL` formulas with the same plug-in.

Internally, the SMV plug-in interface interacts with SMV by using standard I/O communication. This design decision has been made because it avoids changes in the SMV source code. Treating SMV as a black box tool, it can be easily upgraded when new versions of SMV are released.

Fig. 4 (left) shows how SMV is invoked with a CTL specification. Fig. 4 (right) shows how our procedure for transforming LTL formulas to ω-automata can be used to model check LTL formulas with the same plug-in.

7 Conclusions and Future Work

We have described a translation procedure for converting LTL formulas to equivalent ω-automata and its implementation in the HOL theorem prover. Together with the SMV plug-in this allows the usage of SMV as a decision procedure that can be conveniently called as a HOL tactic. As a result, temporal logic formulas given in LTL can now be used to specify and to verify the concurrent behavior conveniently by the model checker SMV, although this model checker is not directly able to handle LTL. The translation of LTL into ω-automata by our HOL conversion runs in linear time and is also in practice very efficient since it is mainly based on preproven theorems of the LTL theory that we also provided.

References

[1] E.A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 996–1072, Amsterdam, 1990. Elsevier Science Publishers.

[2] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell Massachusetts, 1993.

[3] G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[4] R. H. Hardin, Z. Har'El, and R. P. Kurshan. COSPAN. In Rajeev Alur and Thomas A. Henzinger, editors, *Conference on Computer Aided Verification (CAV)*, volume 1102 of *Lecture Notes in Computer Science*, pages 423–427, New Brunswick, NJ, USA, July/August 1996. Springer Verlag.

[5] A. Aziz, F. Balarin, S.-T. Cheng, R. Hojati, T. Kam, S.C. Krishnan, R.K. Ranjan, T.R. Shiple, V. Singhal, S. Tasiran, H.-Y. Wang, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. HSIS: A BDD-Based Environment for Formal Verification. In *ACM/IEEE Design Automation Conference (DAC)*, San Diego, CA, June 1994. San Diego Convention Center.

[6] R. K. Brayton, A. L. Sangiovanni-Vincentelli, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, S. Qadeer, R. K. Ranjan, T. R. Shiple, G. Swamy, T. Villa, G. D. Hachtel, F. Somenzi, A. Pardo, and S. Sarwary. VIS: A system for verification synthesis. In *Computer-Aided Verification*, New Brunswick, NJ, July-August 1996.

[7] M.C. Browne, E.M. Clarke, D.L. Dill, and B. Mishra. Automatic Verification of Sequential Circuits Using Temporal Logic. *IEEE Transactions on Computers*, C-35(12):1034–1044, December 1986.

[8] D.L. Dill and E.M. Clarke. Automatic verification of asynchronous circuits using temporal logic. *IEE Proceedings*, 133 Part E(5):276–282, September 1986.

[9] E.M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D.E. Long, K.L. McMillan, and L.A. Ness. Verification of the Futurebus+ Cache Coherence Protocol. In D. Agnew, L. Claesen, and R. Camposano, editors, *IFIP Conference on Computer Hardware Description Languages and their Applications (CHDL)*, pages 5–20, Ottawa, Canada, April 1993. IFIP WG10.2, CHDL'93, IEEE COMPSOC, Elsevier Science Publishers B.V., Amsterdam, Netherland.

[10] M.J.C. Gordon and T.F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[11] N. Shankar. PVS: Combining specification, proof checking, and model checking. In M. Srivas and A. Camilleri, editors, *International Conference on Formal Methods in Computer Aided Design (FMCAD)*, volume 1166 of *Lecture Notes in Computer Science*, pages 257–264, Palo Alto, CA, USA, November 1996. Springer Verlag.

[12] E.A. Emerson and J.Y. Halpern. "sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, January 1986.

[13] E.M. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In D. Kozen, editor, *Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, Yorktown Heights, New York, May 1981. Springer-Verlag.

[14] K. Schneider. CTL and equivalent sublanguages of CTL*. In C. Delgado Kloos, editor, *IFIP Conference on Computer Hardware Description Languages and their Applications (CHDL)*, pages 40–59, Toledo,Spain, April 1997. IFIP, Chapman and Hall.

[15] K. Schneider, T. Kropf, and R. Kumar. Why Hardware Verification Needs more than Model Checking. In *Higher Order Logic Theorem Proving and its Applications*, Malta, 1994.

[16] P. Loewenstein. Formal verification of state-machines using higher-order logic. In *IEEE/ACM International Conference on Computer Design (ICCD)*, pages 204–207, 1989.

[17] P. Loewenstein. A formal theory of simulations between infinite automata. In L.J.M. Claesen and M.J.C. Gordon, editors, *Higher Order Logic Theorem Proving and its Applications*, pages 227–246, Leuven, Belgium, September 1992. IFIP TC10/WG10.2, North-Holland. IFIP Transactions.

[18] K. Schneider, R. Kumar, and T. Kropf. Alternative Proof Procedures for Finite-State Machines in Higher-Order Logic. In J.J. Joyce and C.-J.H. Seger, editors, *Higher Order Logic Theorem Proving and its Applications*, volume 780 of *Lecture Notes in Computer Science*, pages 213–227, Vancouver, Canada, August 1993. University of British Columbia, Springer-Verlag, published 1994.

[19] D. Eisenbiegler and R. Kumar. An Automata Theory Dedicated Towards Formal Circuit Synthesis. In E.T. Schubert, P.J. Windley, and J. Alves-Foss, editors, *Higher Order Logic Theorem Proving and its Applications*, volume 971 of *Lecture Notes in Computer Science*, pages 154–169, Aspen Grove, Utah, USA, September 1995. Springer-Verlag.

[20] K. Schneider and T. Kropf. A unified approach for combining different formalisms for hardware verification. In M. Srivas and A. Camilleri, editors, *International Conference on Formal Methods in Computer Aided Design (FMCAD)*, volume 1166 of *Lecture Notes in Computer Science*, pages 202–217, Palo Alto, USA, November 1996. Springer Verlag.

[21] S. Agerholm and H. Schjodt. Automating a model checker for recursive modal assertions in HOL. Technical Report DAIMI IR-92, DAIMI, January 1990.

[22] J. von Wright. Mechanizing the temporal logic of actions in HOL. In M. Archer, J.J. Joyce, K.N. Levitt, and P.J. Windley, editors, *Higher Order Logic Theorem Proving and its Applications*, Davis, California, August 1991. IEEE Computer Society, ACM SIGDA, IEEE Computer Society Press.

[23] L. Lamport. The temporal logic of actions. Technical Report 79, Digital Equipment Cooperation, 1991.

[24] F. Andersen and K.D. Petersen. Recursive Boolean Functions in HOL. In M. Archer, J.J. Joyce, K.N. Levitt, and P.J. Windley, editors, *Higher Order Logic Theorem Proving and its Applications*, Davis, California, August 1991. IEEE Computer Society, ACM SIGDA, IEEE Computer Society Press.

[25] K.M. Chandy and J. Misra. *Parallel Program Design*. Addison-Wesley, Austin, Texas, May 1989.

[26] F. Andersen. *A Theorem Prover for UNITY in Higher Order Logic*. PhD thesis, Horsholm, Denmark, March 1992.

[27] F. Andersen, K.D. Petersen, and J.S. Petterson. Program Verification using HOL-UNITY. In J.J. Joyce and C.-J.H. Seger, editors, *Higher Order Logic Theorem Proving and its Applications*, volume 780 of *Lecture Notes in Computer Science*, pages 1–16, Vancouver, Canada, August 1993. University of British Columbia, Springer-Verlag, published 1994.

[28] K. Schneider. Translating linear temporal logic to deterministic ω-automata. In M.Pfaff and R. Hagelauer, editors, *GI/ITG/GMM Workshop Methoden des Entwurfs und der Verifikation digitaler Systeme*, pages 149–158, 1997.

[29] K. Schneider. Yet another look at LTL model checking. In *IFIP WG10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, Lecture Notes in Computer Science, Bad Herrenalb, Germany, September 1999. Springer Verlag.

[30] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191, Amsterdam, 1990. Elsevier Science Publishers.

[31] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 97–107, New York, January 1985. ACM.

[32] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Conference on Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218, New York, 1985. Springer-Verlag.

[33] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.

[34] P. Wolper. On the relation of programs and computations to models of temporal logic. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, pages 75–123, Altrincham, UK, 1987. Springer-Verlag.

[35] G.G de Jong. An automata theoretic approach to temporal logic. In K.G. Larsen and A. Skou, editors, *Workshop on Computer Aided Verification (CAV)*, volume 575 of *Lecture Notes in Computer Science*, pages 477–487, Aalborg, July 1991. Springer-Verlag.

[36] S. Safra. On the complexity of ω automata. In *IEEE Symp.on Foundations of Computer Science*, pages 319–327, 1988.

[37] M. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff'94*, 1994.

[38] E.M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In David L. Dill, editor, *Conference on Computer Aided Verification (CAV)*, volume 818 of *Lecture Notes in Computer Science*, pages 415–427, Standford, California, USA, June 1994. Springer-Verlag.

[39] K. Schneider. Model checking on product structures. In G.C. Gopalakrishnan and P.J. Windley, editors, *Formal Methods in Computer-Aided Design*, volume 1522 of *Lecture Notes in Computer Science*, pages 483–500, Palo Alto, CA, November 1998. Springer Verlag.

[40] S.D. Johnson, P.S. Miner, and A. Camilleri. Studies of the single pulser in various reasoning systems. In T. Kropf and R. Kumar, editors, *International Conference on Theorem Provers in Circuit Design (TPCD)*, volume 901 of *Lecture Notes in Computer Science*, pages 126–145, Bad Herrenalb, Germany, September 1994. Springer-Verlag. published 1995.

[41] M. Norrish, L. Dennis, and R. Boulton. Prosper plug-in interface design. Prosper project report D3.2a, October 1998.

[42] M. Norrish, G. Collins, L. Dennis, and R. Boulton. Prosper plug-in interface user documentation. Prosper Project Report D3.2b, November 1998.