Chapter 1

The hardware verification workbench

Dirk W. Hoffmann, Lex Holt, Ewan Klein, Thomas Kropf, and Klaus Schneider

1.1 Overview

The hardware verification workbench, also referred to as the HVWB, is an example application intended to evaluate the practical applicability of the PROSPER infrastructure. It has been designed to serve as a research platform for developing and evaluating new methods and trends in formal hardware verification. The hardware verification workbench makes use of external verification back-ends and does not implement any proof procedure on its own. The communication between the hardware verification workbench and the involved proof-backends is maintained by PROSPER’s integration interface (PII) which is described in detail in Chapter ??.

The design of the workbench has mainly been influenced by three criteria:

- **Combining different proof paradigms:** The hardware verification workbench is not restricted to a single proof-paradigm and combines different proof methodologies in a single tool. The combination of different proof paradigms is a key-question which PROSPER is trying to solve and yet an unresolved problem in academia and industry. However, it is commonly believed that the combination of different proof methodologies is a prerequisite for making formal methods powerful enough to tackle real-life verification problems.

- **Achieving a high degree of automation:** In the past, it has turned out that a high degree of automation is a crucial property for verification tools to be
accepted in industrial environments (push-button principle). A verification tool requiring high amounts of user interaction also requires hardware engineers with profound logic expertise. As a consequence, highly interactive tools can only be driven by a few design engineers which hinders the integration into the standard design flow. Moreover, interactive proofs usually consume a high amount of verification time which imposes additional problems on the design flow. The hardware verification workbench therefore focuses on automatic proof procedures such as model checking, equivalence checking, and automated theorem proving.

- **Hiding formal details**: Formal verification tools which are intended to be an integrated part of the design flow must be usable by design engineers without deep knowledge in theoretical computer science. Hence, a prerequisite of easy-to-use verification tools is the existence of user friendly interfaces such as waveform viewers or graphical specification editors. Developing and evaluating user friendly interfaces is another key aspect of the PROSPER project. Using natural language for specifying system properties or using waveforms to display formula ambiguities, are two examples of user friendly interfaces that have been developed and examined within PROSPER. Both interfaces are integrated parts of the HVWB.

**Tool architecture**

The hardware verification workbench consists of the following components:

- the language converters
- the natural language front-end
- the core hardware verification workbench
- the internal object-database
- the internal goal-database
- the graphical user interface

All components interact as shown in Figure 1.1. The hardware verification workbench allows the user to read in different objects. Objects can be either circuit descriptions (restricted subsets of Verilog, VHDL, and IL), or logic formulas (LTL, CTL, or PL0).

The natural language interface (NL interface) developed at the University of Edinburgh has been integrated into the workbench. It provides the means of automatically generating LTL or CTL formulas out of natural language descriptions in plain English. The NL interface considerably decreases the logic expertise required to specify the correct behaviour of a design. The translation from LTL to $\omega$-automata (LTL_CONV) is part of HOL’s temporal logic library that has been developed by Klaus Schneider at the University of Karlsruhe. This library provides an embedding
of linear time temporal logic in HOL and a translation from this temporal logic to \( \omega \)-automata by HOL’s inference rules. It furthermore contains an interface to the SMV system, which is a symbolic model checker for CTL formulas that has been developed at the Carnegie Mellon University.

Internally, the HVWB maintains two major databases for storing data. The first database is referred to as the object database and contains all circuits or temporal formulas that have been read into the workbench. Two distinct objects can be unified to a proof-goal where one object serves as the specification and the other one as the implementation. Depending on the type of the specification/implementation pair, a different proof goal is set up. The type of the proof-goal is automatically determined by the workbench by analyzing the types of the specification and implementation objects. Note that not all object combinations lead to a well formed proof-goal. All meaningful combinations of specification/implementation pairs are listed in Figure 1.1. The hardware verification workbench is currently linked to proof-backends performing Boolean equivalence checking, propositional theorem proving, CTL model checking, LTL model checking, and LTL tautology checking.
The hardware verification workbench

<table>
<thead>
<tr>
<th>Specification type</th>
<th>Implementation type</th>
<th>Resulting proof-goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>IL</td>
<td>Sequential Equivalence Checking</td>
</tr>
<tr>
<td>Combinational IL</td>
<td>Combinational IL</td>
<td>Boolean Equivalence Checking</td>
</tr>
<tr>
<td>PL0</td>
<td>PL0</td>
<td>Propositional theorem proving</td>
</tr>
<tr>
<td>CTL</td>
<td>IL</td>
<td>CTL model checking</td>
</tr>
<tr>
<td>LTL</td>
<td>IL</td>
<td>LTL model checking</td>
</tr>
<tr>
<td>LTL</td>
<td>LTL</td>
<td>LTL tautology checking</td>
</tr>
</tbody>
</table>

Table 1.1: Different specification/implemention scenarios.

From the user’s point of view, the hardware verification workbench can be driven by two distinct interfaces:

- **Command line interface:** The hardware verification workbench is steared by usual text commands. This interface allows the user to write scripts and therefore to automate often repeated command sequences.

- **Graphical user interface:** The graphical user interface (GUI) provided the easiest and most convenient way to drive the workbench and provides the same functionality as the command line interface. Using the GUI, no command line options have to be learned. However, the GUI version also allows the user to directly send commands to the workbench kernel.

**Scope of this chapter**

In this chapter, we exemplarily describe a specific verification flow implemented in the hardware verification workbench: We present the translation of natural language specifications into finite state $\omega$-automata. This conversion is the basis for solving LTL/LTL proof-goals (LTL tautology checking). The translation is performed in two steps. The natural language description is first converted into a temporal formula in linear time temporal logic (LTL). In the second step, the corresponding LTL formulas are converted into an equivalent finite state $\omega$-automaton. The translation into $\omega$-automata enables the usage of all proof procedures, including those based on symbolic model checking, for the verification of specifications given in LTL. In this presentation, we focus on the technical aspects of the translation rather than on the practical steps that have to be performed within the hardware verification workbench by the designer. Hence, this chapter is not intended to be a description of all of the workbench’s features nor is it meant to replace the workbench’s user manual.

The rest of this chapter is structured as follows: Section 1.2 outlines the translation of natural language descriptions into linear time temporal logic, and the algorithms for constructing finite state $\omega$-automata out the LTL formulas are presented in Section 1.3.
1.2 Translating natural language into TL

1.2.1 Motivation

Most formal verification approaches require the precise expression of a system’s intended properties (one might consider equivalence checkers, for example, to be exceptions). Whether the verification technique involves theorem proving or model checking, a prerequisite is to couch a specification of behaviour in the appropriate formal language—typically a logic. But many target users of verification technology are not logicians, although they may well have clear and precise intuitions about the properties they wish to verify. In the hardware realm, for example, timing diagrams are commonly used to make explicit and objective claims about the temporal behaviour of circuits or components. But timing diagrams are not logic (at least not as they stand; formalizations are possible, however [20]).

So there is strong motivation to build interfaces that permit hardware and software engineers who are not experts in logic to use verification tools. Since a key concern of the PROSPER project is making existing model checking and theorem proving technology easier to use, building a specification interface based on natural language seems a worthwhile goal.

1.2.2 System structure

The structure of the natural language converter is shown in Figure 1.2. There are four components:

1. *Coordination module*. This component mediates interaction between the main HVWB and the three other parts of the NL conversion system.

2. *Part-of-speech tagger*. Raw sentences are first ‘tagged’ by this module, in an attempt to statistically infer the most likely part of speech (grammatical category) for each word. This improves the performance of the subsequent parsing step, and also aids the recognition of hitherto unknown signal names. We employed the tagger from the LTG TTT text tokenisation system [25].

3. *ANLT parser*. We used the Alvey Natural Language Tools Grammar [23] to implement a parser for a restricted subset of English. This parser assigns one or more syntactic structures to each sentence and returns general-purpose semantic representations of their meaning.

4. *Alvey to TL translator*. The final component converts the semantic representations produced by the Alvey system into temporal logic formulas. This conversion process is described in more detail below.
1.2.3 Key issues

In the following sections, we highlight some of the more important issues in the design and implementation of the conversion system. We start by looking at the controlled language problem for this application: exactly what input language should the system accept? Next, we consider ambiguity in the NL input: a perennial issue for automatic natural language interpretation. Then, semantics. Most algorithms which convert between semantic or logical representations exploit strong compositionality in the formalisms involved. It turns out that this isn’t possible when translating representations of NL meaning into CTL and LTL. We examine a sample conversion in detail, and finally consider some issues for future development.

See [27] for comments on design decisions. Earlier work with similar goals is reported in [36].

1.2.4 Restricting the language

The definition of an appropriate subset of English for this task raised interesting issues of its own [28, 24]. The central problem is this: How to guarantee that English input has a valid temporal logic translation?

Restricting the input to a controlled language is the general approach. Started with a back-translation from CTL to English, we developed a hierarchy of subsets of English, according to

- the nature of semantic interpretation for sentences of that subset (compositional or not);
• the expressiveness of the ‘natural’ target formalism for that subset (for example: CTL, or CTL augmented with extended events, or first-order logic?).

We also developed a corpus of natural language circuit specifications, which currently contains around 400 sentences. Our main sources of data were:

• Written or spoken descriptions of timing diagrams by informants;
• Log files from our English-to-CTL web demo;
• The University of Texas VIS Benchmarks, in particular, CTL examples with English glosses.

Our current input language represents a middle point on the theoretical hierarchy, augmented by appropriate corpus data.

1.2.5 Ambiguity

Sentence (1.1), taken from our corpus of specification discourses, is ambiguous.

(1.1) After sig1 becomes active sig2 should not become active until sig3 becomes active.

The two LTL readings which our system assigns are shown in (1.2) and (1.2').

\[
\begin{align*}
&\text{(1.2)} \\
\sigma1 &\rightarrow X(\neg\sigma2) \cup \sigma3 \\
\sigma1 &\rightarrow G(\neg\sigma2 \cup \sigma3)
\end{align*}
\]

These correspond to the two readings obtained by the parser—a typical case of English syntactic ambiguity. The following paraphrases use additional punctuation to make the two readings more explicit:

(1.3) After sig1 becomes active, sig2 should not become active until sig3 becomes active.

(1.3') After sig1 becomes active sig2 should not become active—until sig3 becomes active.

Clearly, there needs to be a mechanism to help users of the HVWB’s NL interface resolve these kinds of ambiguity.
1.2.6 Semantic conversions

The process of converting the semantic representations produced by the Alvey system to temporal logic formulas has some interesting semantic aspects. For example, temporal information must be converted from first-order relations over Davidsonian event variables—the $e_1$, $e_2$ and $e_3$ of example (1.11) below—to modal operators.

The converter first treats the parser’s semantic representations as formulas of higher-order logic, and then generates a number of intermediate logical representations. The complete list is:

1. ‘Alvey representations’ (output of parser)
2. ‘Alvey logic’
3. ‘Event logic’
4. CTL and LTL

Note that Alvey representations are scoped, so that object quantification (e.g., every output) can be interpreted correctly.

1.2.7 Compositionality and expressibility

It transpires that a strictly compositional translation algorithm is inadequate. Consider the two example sentences:

(1.4) if sig1 is high then\[
sig1 \text{ is high until } \int \text{ sig2 is high for 3 cycles}
\]

Compositionally, we would like to translate \textit{sig2 is high for 3 cycles} identically in both cases. But there is no single LTL fragment that can be used; the two sentences translate to

\[
\begin{align*}
\text{sig1} & \rightarrow \text{sig2} \land X(\text{sig2} \land X \text{sig2}) \quad (1.5) \\
\text{sig1} & U (\text{sig2} \land \text{sig1} \land X (\text{sig2} \land \text{sig1} \land X \text{sig2})) \quad (1.5')
\end{align*}
\]

So the appropriate LTL fragment actually depends on the temporal context in which the corresponding English phrase occurs. In general, this happens when the English specification represents a relation over extended events—anything longer than a single cycle.

Here is an even simpler example. No single interpretation of \textit{sig2 is high} will work for both of these sentences:
1.2 Translating natural language into TL

(1.6) \[
\begin{align*}
& \text{if } \text{sig1 is high} \\
& \text{if } \text{sig1 rises} \\
\end{align*}
\] 
then sig2 is high

The event \textit{sig1 rises} takes place over two cycles, and it is on the second of these two that the implication holds. The correct translations are therefore:

\[
\begin{align*}
& \text{sig1} \rightarrow \text{sig2} \\
& (\neg \text{sig1} \land X \text{sig1}) \rightarrow X \text{sig2}
\end{align*}
\]  

(1.7)

1.2.8 Conversion example

Let us consider this specification sentence:

(1.8) After sig1 is active, sig2 is active for three cycles.

The first stage in conversion is the part-of-speech tagger. This operates on the input sentence to produce

(1.9) After \textit{sig1\_NNP} is active\_JJ, \textit{sig3\_NNP} is active\_JJ for three cycles\_NNS

The tagger has used a variety of heuristics to guess the part-of-speech tag for each word. For example, it has correctly identified \textit{active} as an adjective and tagged it accordingly with \textit{JJ}.

The next stage is parsing. The full output of the ANLT parser is:

(1.10) (DECL
   (some (e1) e1
   (AFTER e1
      (some (e2) (PRES e2) (BE e2 (ACTIVE (name sig1) (degree unknown))))
      (3 (x1) (and (pl x1) (CYCLE x1))
      (some (e3) (PRES e3)
      (and
      (BE e3 (ACTIVE (name sig2) (degree unknown)))
      (FOR e3 x1))))
      (timespan unknown))))

While this ‘Alvey representation’ is not part of a well-defined logical language, it can fairly easily be viewed as a formula in a fragment of higher-order logic:

(1.11) DECL(some(e1, T,
      AFTER(e1,
      some(e2, PRES(e2), BE(e2, ACTIVE(sig1, DEGREE(unknown))))),
This is the starting point for a series of intermediate logical conversions. First, Davidsonian event variables are eliminated, along with aspects of the meaning representation not required for conversion to temporal logic:

(1.12) \(\text{AFTER} (\text{HIGH}(\text{sig}1), \text{FOR} (\text{HIGH}(\text{sig}2), 3), \text{TIMESPAN}(\text{unknown}))\)

Both 1.11 and 1.12 are formulas of ‘Alvey logic’. Next comes the conversion to ‘event logic’, which is rather like conventional temporal logic with the addition of extended events:

(1.13) \(\text{If} (\text{High}(\text{sig}1), \text{Next}(1, \text{For}(3, \text{High}(\text{sig}2)))\))

Finally, both CTL and LTL formulas can now be directly generated:

\[
\begin{align*}
\text{AG}(\text{sig}1 \rightarrow \text{AX}((\text{sig}2 \land \text{AX} \text{sig}2) \land \text{AX AX sig}2)) \\
\text{G}(\text{sig}1 \rightarrow \text{X}((\text{sig}2 \land \text{X sig}2) \land \text{XX sig}2))
\end{align*}
\]

(1.14) (1.15)

1.2.9 Prospects

Some outstanding issues:

- There is potential to glean additional information from the circuit model in order to correctly interpret higher-level natural language concepts. For example, signal roles and bit widths.

- Our handling of ambiguity could be improved. Two options to consider are displaying parse trees to the user, and generating disambiguated paraphrases.

- Context-sensitive phenomena. It might make sense to eliminate some or all of these via additional restrictions on the input language.

- Some non-compositional phenomena are not yet fully treated (e.g., persistent).

- Proving validity of conversion. We should be able to provide a formal justification of the conversion steps between intermediate logics.
1.3 Translating LTL to $\omega$-automata

1.3.1 Background

Specifications of reactive systems such as digital hardware circuits are conveniently given in temporal logics (see [17] for a comprehensive 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 [35], SPIN [29], COSPAN [26], and VIS [5] have already found bugs in real-world examples [6, 15, 13] with more than $10^{30}$ states.

While this is an enormous number for control-oriented systems, this number of states is quickly reached 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, interactive theorem provers such as HOL [22] are required for the verification of systems that consist of control and data paths. However, if lemmas 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 accepted claim that neither the exclusive use of model checkers nor the exclusive use of theorem provers is sufficient for a pragmatic verification. For this reason, the integration of model checkers as (unsafe) tactics of theorem provers like HOL is desired. The notion ‘unsafe’ is used here to indicate that the validity of the theorem has only been checked by an external tool, i.e., it is not checked by the theorem prover itself.

The integration of a model checker in a theorem prover requires first to embed the languages used by the model checker, i.e., finite automata and temporal logics, in the theorem prover’s logic. For this reason, we briefly list previous work on embedding automata and temporal logic in HOL. Pioneering work on embedding automata theory in HOL has been done by Loewenstein [33, 34]. Schneider, Kumar and Kropf [42] 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 [16]. Schneider and Kropf [41] presented a unified approach based on automaton-like formulas for combining different formalisms. In the domain of temporal logics, Agerholm and Schjodt [1] were the first who made a model checker available for HOL. Von Wright [46] mechanized TLA (Temporal Logic of Actions) [30] in HOL. Andersen and Petersen [3] 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 [11] and Unity [10] in HOL which enables them to reason about Unity programs in HOL [2, 4] (which was their primary aim). Initial work on the way to embed linear temporal logic (LTL) in HOL has been done by Schneider [38] (also contained in [41]) who presented a subset of LTL that can be translated to deterministic ω-automata by means of closures theorems that have been proved in HOL. This initial work has been recently completed by embedding the full linear time temporal logic with both future and past temporal logic operators in HOL [40]. Beneath the embedding of the temporal logic, there is also a library which contains conversions to translate LTL formulas into equivalent ω-automata, and an interface to the SMV model checker [35]. The library has been converted by K. Slind to HOL98, and is now distributed with the newer releases of HOL98.

Model checking or tautology checking of LTL formulas is usually performed by reducing the problems to equivalent automata problems. For this reason, translations of LTL to ω-automata have been extensively studied in the past: Among the many approaches are those of Lichtenstein and Pnueli [31, 32], Vardi and Wolper [47, 45, 48], Jong [14], Burch et. al. [9, 12, 40], Vardi [44], and Gerth et. al. [21]. In general, there are procedures that explicitly construct an ω-automaton [31, 32, 47, 45, 48, 21] and others that construct an implicit description of the automaton like [9, 12, 40]. Explicit representations of automata enumerate the states and the transitions of an automaton, and therefore suffer from large state spaces. Implicit representations, in contrast, use propositional formulas to encode these sets. For this reason, the members of the sets must first be encoded by some set of Boolean variables. After that, one can use propositional formulas to represent sets: Each propositional interpretation that satisfies the considered formula corresponds with a member of the set. As formulas of length n may have $2^n$ satisfying interpretations, the implicit representation may be very compact (in particular if efficient data structures like BDDs [7] are used).

In general, the number of states of an automaton that is equivalent to an LTL formula of length n may be of order $2^{|\text{length } n|}$. As a consequence, all translation procedures that construct explicitly represented automata have an exponential runtime wrt. the length of the formulas. Instead, procedures that are based on implicit representations like [9, 12, 40] have a linear runtime wrt. the length of the formulas. The temporal logic library of Schneider also makes use of implicit representations and therefore allows a very efficient translation of temporal logic formulas into automata that is done by HOL’s inference rules. Hence, the translation from temporal logic into equivalent automata is safe, i.e. done inside HOL, but checking the automata via SMV is unsafe in that SMV’s result is transferred to HOL via the mk.thm function.

### 1.3.2 Representing ω-automata and temporal logic in HOL

For a sound integration of a model checker in the HOL theorem prover, one has to address the problem to first embed a temporal logic in the HOL logic. The effort
of such an embedding crucially depends on the kind of temporal logic that is to be embedded. In general, there are linear time and branching time temporal logics (see [17] for a survey). Linear time temporal logics express facts on computation paths of the systems, i.e., subformulas of this logic are viewed as HOL formulas of type $\mathbb{N} \rightarrow \mathbb{B}$ (where $\mathbb{N}$ represents the type of natural numbers and $\mathbb{B}$ represents the type of Boolean values). Hence, the embedding of LTL as presented in [40] and reviewed below is straightforwardly done in HOL, since the temporal operators can be defined as functions that are applied on arguments of type $\mathbb{N} \rightarrow \mathbb{B}$.

Branching time temporal logics like CTL* [18], on the other hand, 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’. Therefore, embeddings of branching time temporal logics would additionally require to formalize the set of computation paths of the system under consideration and therefore cause much more effort.

### Representing Linear Time Temporal Logic in HOL

We now present the formal definition of the linear time temporal logic that has been used in [40]. After defining syntax and semantics of the logic, it is shown below how the temporal logic has been embedded in the HOL logic.

**Definition 1 (Syntax of LTL)** The following mutually recursive definitions introduce the set of LTL formulas over a given set of variables $\mathcal{V}$:

- each variable of $\mathcal{V}$ is a formula, i.e., $\mathcal{V} \subseteq \text{LTL}$
- $\neg \varphi$, $\varphi \land \psi$, $\varphi \lor \psi \in \text{LTL}$ if $\varphi, \psi \in \text{LTL}$
- $\begin{aligned} X \varphi \in \text{LTL} & \text{ if } \varphi \in \text{LTL} \\
\left[\begin{array}{c} \varphi \end{array}\right] \psi \in \text{LTL} & \text{ if } \varphi, \psi \in \text{LTL} \\
\begin{array}{c} \overline{X} \varphi \end{array} \in \text{LTL} & \text{ if } \varphi, \\psi \in \text{LTL} \\
\end{aligned}$

Informally, the semantics is given relative to a considered point of time $t_0$ as follows: $X \varphi$ holds at time $t_0$ iff $\varphi$ holds at time $t_0 + 1$; $[\varphi \left[ \begin{array}{c} \psi \end{array}\right] \psi]$ holds at time $t_0$ iff there is a point of time $t_0 + \delta$ in the future of $t_0$ where $\psi$ becomes true and $\varphi$ holds until that point of time. The operators $\overline{X}$ and $\left[ \begin{array}{c} \overline{U} \end{array}\right.]$ are the past time variants of $X$ and $\left[ \begin{array}{c} \overline{U} \end{array}\right.]$, respectively: $\overline{X} \varphi$ holds at time $t_0$ if either $t_0 = 0$ holds or $\varphi$ holds at time $t_0 - 1$. Finally, $[\varphi \left[ \begin{array}{c} \overline{U} \end{array}\right. \psi]$ holds at time $t_0$ iff there has been point of time $t_0 - \delta$ in the past of $t_0$ where $\psi$ was true and $\varphi$ held since that point of time.

For a formal presentation of the semantics, the models of the temporal logic must be defined in advance: Given a set of variables $\mathcal{V}$, a model for LTL is a function $\pi$ of type $\mathbb{N} \rightarrow \varphi(\mathcal{V})$ where $\varphi(M)$ 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 set of variables $\mathcal{V}$, and a function $\pi : \mathbb{N} \rightarrow \varphi(\mathcal{V})$. Then, the following rules define the semantics of LTL:

- $\pi, t_0 \models x$ iff $x \in \pi^{(t_0)}$
- $\pi, t_0 \models \neg \varphi$ iff $\pi, t_0 \not\models \varphi$
- $\pi, t_0 \models \varphi \land \psi$ iff $\pi, t_0 \models \varphi$ and $\pi, t_0 \models \psi$
- $\pi, t_0 \models \varphi \lor \psi$ iff $\pi, t_0 \models \varphi$ or $\pi, t_0 \models \psi$
- $\pi, t_0 \models X \varphi$ iff $\pi, t_0 + 1 \models \varphi$
- $\pi, t_0 \models [\varphi \mathbf{U} \psi]$ iff there is a $\delta \in \mathbb{N}$ such that $\pi, t_0 + \delta \models \psi$ and for all $d < \delta$ it holds that $\pi, t_0 + d \models \varphi$
- $\pi, t_0 \models \overline{X} \varphi$ iff $t_0 = 0$ holds or if $\pi, t_0 - 1 \models \varphi$ holds
- $\pi, t_0 \models [\varphi \mathbf{U} \psi]$ iff there is a $\delta \in \mathbb{N}$ such that $\pi, t_0 - \delta \models \psi$ and for all $d < \delta$ it holds that $\pi, t_0 - d \models \varphi$

As a generalization of the above definition, temporal logics are often interpreted over finite state Kripke structures. These are finite state transition systems where each state is labeled with a subset of $\mathcal{V}$ (the set of variables). LTL formulas are then interpreted on the paths of these Kripke structures. The above definition is then used to interpret a LTL formula at position $t_0$ on a path $\pi$ of such a Kripke structure. As it is however easily possible to define the set of admissible paths of a finite state transition system (similar to our representation of finite-state $\omega$-automata as given below), Kripke structures can be easily simulated in HOL without the burden of deep-embedding Kripke structures.

It is also easily seen that the projections $\pi_x := \lambda t. \pi^{(t)} \cap \{x\}$ for all variables $x \in \mathcal{V}$ can be used instead of $\pi : \mathbb{N} \rightarrow \varphi(\mathcal{V})$ itself. Using these projections, one could reestablish $\pi$ as $\pi := \lambda t. \bigcup_{x \in \mathcal{V}} \pi_x^{(t)}$, which shows that no information is lost. Hence, it is only a matter of taste whether $\pi$ or its projections $\pi_x$ is used as a model for the LTL logic. Using these projections directly leads to the HOL representation of LTL formulas as used in [40]: Variables of LTL are represented in the HOL logic directly as HOL variables of type $\mathbb{N} \rightarrow \mathbb{B}$. 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. Therefore, the temporal logic library is based on the following definitions of temporal operators:

Definition 3 (Defining Temporal Operators in HOL) The definition of temporal operators $X$, $[\varphi \mathbf{U} \psi]$, $\overline{X}$ and $[\varphi \mathbf{U} \psi]$ in HOL is as follows (for any $\varphi, \psi : \mathbb{N} \rightarrow \mathbb{B}$):

- $X \varphi := \lambda t_0. \varphi^{(t_0 + 1)}$
- $\overline{X} \varphi := \lambda t_0. (t_0 = 0) \lor \varphi^{(t_0 - 1)}$
- $[\varphi \mathbf{U} \psi] := \lambda t_0. \exists t_1, t_0 \leq t_1 \land \psi^{(t_1)} \land (\forall t, t_0 \leq t < t_1 \rightarrow \varphi^{(t)})$
- $[\varphi \mathbf{U} \psi] := \lambda t_0. \exists t_1, t_0 \leq t_1 \land \psi^{(t_1)} \land (\forall t, t_1 < t < t_0 \rightarrow \varphi^{(t)})$
1.3 Translating LTL to $\omega$-automata

Note that $X$ and $\overline{X}$ are of type $(\mathbb{N} \rightarrow \mathbb{B}) \rightarrow (\mathbb{N} \rightarrow \mathbb{B})$ and $[\cdot \cup \cdot]$ and $[\cdot \overline{\cup} \cdot]$ are of type $(\mathbb{N} \rightarrow \mathbb{B}) \rightarrow (\mathbb{N} \rightarrow \mathbb{B}) \rightarrow (\mathbb{N} \rightarrow \mathbb{B})$.

Clearly, it is possible and desirable to have more temporal operators that describe other temporal relationships as the ones given above. For this reason, the temporal logic library defines also the following further operators:

\[
\begin{align*}
\overline{X} \varphi &= \lambda t_0.(t_0 \neq 0) \land \varphi^{(t_0-1)} \\
G \varphi &= \lambda t_0. \forall t. t_0 \leq t \rightarrow \varphi(t) \\
\overline{G} \varphi &= \lambda t_0. \exists t. t_0 \leq t \rightarrow \varphi(t) \\
F \varphi &= \lambda t_0. \exists t. t_0 \leq t \land \varphi(t) \\
\overline{F} \varphi &= \lambda t_0. \exists t. t_0 \leq t \land \varphi(t) \\
[\varphi \mathbb{B} \psi] &= \lambda t_0. \exists t_1.t_0 \leq t_1 \land \varphi(t_1) \land (\forall t. t_0 \leq t \land t \leq t_1 \rightarrow -\psi(t)) \\
[\varphi \mathbb{W} \psi] &= \lambda t_0. \exists t_1.t_0 \leq t_1 \land \varphi(t_1) \land (\forall t. t_0 \leq t \land t < t_1 \rightarrow -\psi(t)) \\
[\varphi \mathbb{U} \psi] &= \lambda t_0.[\varphi \mathbb{U} \psi]^{(t_0)} \lor \overline{G}\varphi^{(t_0)} \\
[\varphi \overline{\mathbb{U}} \psi] &= \lambda t_0.[\varphi \overline{\mathbb{U}} \psi]^{(t_0)} \lor \overline{G}\varphi^{(t_0)} \\
[\varphi \mathbb{B} \psi] &= \lambda t_0.[\varphi \mathbb{B} \psi]^{(t_0)} \lor [G \neg \psi]^{(t_0)} \\
[\varphi \overline{\mathbb{B}} \psi] &= \lambda t_0.[\varphi \overline{\mathbb{B}} \psi]^{(t_0)} \lor [G \neg \psi]^{(t_0)} \\
[\varphi \mathbb{W} \psi] &= \lambda t_0.[\varphi \mathbb{W} \psi]^{(t_0)} \lor [G \neg \psi]^{(t_0)} \\
[\varphi \overline{\mathbb{W}} \psi] &= \lambda t_0.[\varphi \overline{\mathbb{W}} \psi]^{(t_0)} \lor [G \neg \psi]^{(t_0)}
\end{align*}
\]

$G$ and $F$ are the usual always (always in the future) operators, and $\overline{G}$ and $F$ are their past time duals. $[\varphi \mathbb{B} \psi]$ means that $\varphi$ holds before $\psi$ holds, and $\varphi$ must actually occur at some point in the future. $[\varphi \mathbb{W} \psi]$ means that $\varphi$ holds at the first point of time where $\psi$ holds, and there is such a point of time. $[\varphi \mathbb{B} \psi]$ and $[\varphi \mathbb{W} \psi]$ are the past time variants of $[\varphi \mathbb{B} \psi]$ and $[\varphi \mathbb{W} \psi]$, respectively. Moreover, $[\varphi \mathbb{U} \psi]$, $[\varphi \mathbb{B} \psi]$ and $[\varphi \mathbb{W} \psi]$ are the weak variants of $[\varphi \mathbb{U} \psi]$, $[\varphi \mathbb{B} \psi]$ and $[\varphi \mathbb{W} \psi]$, respectively. One can also distinguish between a weak and a strong version of the previous operators: the weak version $\overline{X} \varphi$ holds for any $\varphi$ at time $t = 0$, while the strong version $\overline{X} \varphi$ is false at time $t = 0$. Hence, $\overline{X} (\varphi \land \neg \varphi)$ is equivalent to $t = 0$, which shows that the temporal logic with past time operators is slightly more expressive than the version without past time modalities.

In the following, the syntax is simplified to obtain more readable formulas: all applications on time are neglected as well as $\lambda$-abstraction of the time variable. For
example, $\neg[(\neg \varphi) \cup \psi] \wedge [F \varphi]$ is written instead of $\lambda t.\neg[(\lambda t.\neg \varphi(t)) \cup \psi](t) \wedge [F \varphi](t)$. It is clear from the context, where applications on time and $\lambda$-abstractions should be added to satisfy the type rules.

There are some remarkable properties that can be found in the HOL theory on temporal operators. For example, the following equations allow to compute a negation normal form $\text{NNF}(\varphi)$ of any formula:

$$
\begin{align*}
\neg \Box \varphi &= \Box (\neg \varphi) \\
\neg \Diamond \varphi &= \Diamond (\neg \varphi) \\
\neg G \varphi &= F (\neg \varphi) \\
\neg F \varphi &= G (\neg \varphi) \\
\neg [\varphi \cup \psi] &= [(\neg \varphi) \Box \psi] \\
\neg [\varphi \cup \psi] &= [(\neg \varphi) \Box \psi] \\
\neg [\varphi \cup \psi] &= [(\neg \varphi) \Box \psi] \\
\neg [\varphi \cup \psi] &= [(\neg \varphi) \Box \psi] \\
\neg [\varphi \cup \psi] &= [(\neg \varphi) \Box \psi]
\end{align*}
$$

Also, it is shown that any of the binary temporal future/past operators can be expressed by any other binary temporal future/past operator. For example, all considerations can be restricted to the temporal operators $X$, $\lceil \cup \rceil$, $\bar{X}$ and $\lceil \bar{U} \rceil$, which is seen by the following reduction rules:

$$
\begin{align*}
\bar{X} \varphi &= \neg \bar{X} \neg \varphi \\
G \varphi &= \neg \lceil T \cup (\neg \varphi) \rceil \\
F \varphi &= \lceil T \cup \varphi \rceil \\
[\varphi \Box \psi] &= [\neg (\neg \varphi) \Box (\varphi \land \neg \psi)] \\
[\varphi \Box \psi] &= [\neg (\neg \varphi) \Box (\varphi \land \psi)] \\
[\varphi \Box \psi] &= [\neg (\neg \varphi) \Box (\varphi \land \psi)] \\
[\varphi \Box \psi] &= [\neg (\neg \varphi) \Box (\varphi \land \psi)] \\
[\varphi \Box \psi] &= [\neg (\neg \varphi) \Box (\varphi \land \psi)]
\end{align*}
$$

**Representing $\omega$-Automata in HOL**

This section briefly shows how $\omega$-automata are represented in HOL. In general, an $\omega$-automaton consists of a finite state transition system where a transition between two states is 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. [43] for an overview.
The representation of an $\omega$-automaton as a formula in HOL is straightforward: We use a finite number of state variables $q_0, \ldots, q_n$ to encode the states of the automaton. Similarly, the input alphabet are encoded by variables $x_0, \ldots, x_m$. In HOL, the state variables and the input variables are all of type $\mathbb{N} \rightarrow \mathbb{B}$. Then, an $\omega$-automaton is represented by a HOL formula of the following form:

$$\exists q_0 \ldots q_n. \Phi_I(q_0^{(0)}, \ldots, q_n^{(0)}) \land \left[ \forall t. \Phi_R(q_0^{(t)}, \ldots, q_n^{(t)}, x_0^{(t)}, \ldots, x_m^{(t)}, q_0^{(t+1)}, \ldots, q_n^{(t+1)}) \right] \land \Phi_F(q_0, \ldots, q_n)$$

$\Phi_I(q_0^{(0)}, \ldots, q_n^{(0)})$ is thereby a ‘propositional’ formula where only the atomic formulas $q_0^{(0)}, \ldots, q_n^{(0)}$ may occur. $\Phi_I$ implicitly represents the set of initial states of the automaton as follows: Any valuation of the atomic formulas $q_0^{(0)}, \ldots, q_n^{(0)}$ that satisfies $\Phi_I$ 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 $\Phi_I(b_0, \ldots, b_n)$ evaluates to $T$.

Similarly, $\Phi_R(\ldots)$ is a ‘propositional’ formula where only the atomic formulas $q_i^{(t)}$, $x_i^{(t)}$, and $q_i^{(t+1)}$ may occur. $\Phi_R$ represents the transition relation of the $\omega$-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 $\Phi_R(b_0, \ldots, b_n, a_0, \ldots, a_m, b'_0, \ldots, b'_n)$ evaluates to $T$.

$\Phi_F(q_0, \ldots, q_n)$ is the acceptance condition of the automaton. Note that $\Phi_R$ 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, an input sequence is accepted, if it has at least one run $(q_0, \ldots, q_n)$ through the automaton that satisfies the acceptance condition $\Phi_F(q_0, \ldots, q_n)$. In the literature, the following types of acceptance conditions are distinguished, where all formulas $\Phi_k, \Psi_k$ 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:** $\bigwedge_{k=1}^n [\forall t_1. \exists t_2. \Phi_k]$
- **Streett:** $\bigwedge_{k=1}^n [\forall t_1. \exists t_2. \Phi_k] \lor [\exists t_1. \forall t_2. \Psi_k]$
- **Rabin:** $\bigvee_{k=1}^n [\forall t_1. \exists t_2. \Phi_k] \land [\exists t_1. \forall t_2. \Psi_k]$

It can be shown that the nondeterministic versions of the above $\omega$-automata have the same expressive power [43]. Therefore, any of these classes can be used for the following translation. In the following, generalized Büchi automata are used. It will become clear in the next section, why this kind of $\omega$-automaton is an appropriate means for a simple translation of temporal logic inside HOL.
As an example of the representation of an \( \omega \)-automaton, consider Figure 1.3. This \( \omega \)-automaton is represented by the following HOL formula:

\[
\exists p : \mathbb{N} \to \mathbb{B}. \exists q : \mathbb{N} \to \mathbb{B}.
\neg p^{(0)} \land \neg q^{(0)}
\land \forall t. \left( (p^{(t)} \to p^{(t+1)}) \land (p^{(t+1)} \to p^{(t)} \lor \neg q^{(t)}) \land (q^{(t+1)} = (p^{(t)} \land \neg q^{(t)} \land \neg a^{(t)}) \lor (p^{(t)} \land q^{(t)})) \right)
\land \forall t_1. \exists t_2. p^{(t_1+t_2)} \land \neg q^{(t_1+t_2)}
\]

This means that the only initial state is the one where neither \( p \) nor \( q \) does hold (drawn with double lines in Figure 1.3). 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 \( \exists t_1. \forall t_2. a^{(t_1+t_2)} \).

### 1.3.3 The translation procedure

The translation procedure given in [40] is a variant of the approach in [9, 12], which are in turn special cases of the product model checking procedure given in [39]. The first step of the translation procedure is the computation of a ‘definition normal form’ for a given LTL formula \( \Phi \) that is obtained as follows: for each elementary subformula \( \varphi \), i.e., each subformula that starts with a temporal operator, a new variable \( \ell_\varphi : \mathbb{N} \to \mathbb{B} \) is generated. The state transitions of the automaton should be such that the variable \( \ell_\varphi \) exactly behaves like the subformula \( \varphi \) does, hence \( \ell_\varphi = \varphi \) should hold. By applying the substitution of subformulas recursively in a bottom up manner, any LTL formula \( \Phi \) is reduced to a set of equations \( E = \{ \ell_i = \varphi_i \mid i \in I \} \) and some propositional formula \( \Psi \) such that \( \bigwedge_{i \in I} [\ell_i = \varphi_i] \to (\Phi = \Psi) \) is valid. Therefore, the ‘definition normal form’ looks in general as given below:

\[
\exists \ell_1 \ldots \ell_n. \left( \bigwedge_{i=1}^n \ell_i = \varphi_i \right) \land \Psi
\]
1.3 Translating LTL to ω-automata

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

The above normal form is closely related to the Fischer-Ladner closure [19]: Subformulas of a temporal logic formula that start with a temporal operator are often called elementary subformulas. The Fischer-Ladner closure of a temporal logic formula is defined to be the set of its elementary subformulas. It is easily seen that in the above normal form, each variable $\ell_i$ is used to abbreviate an elementary subformula, which explains the relationship between the Fischer-Ladner closure and the above normal form.

The computation of the above normal form is essentially done by the function \texttt{ElemForms} that is given in Figure 1.4. The equivalence between the above normal form and the original formula $\Phi$ 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 will then prove the goal. The other direction is simply obtained by instantiating the witnesses $\varphi_1, \ldots, \varphi_n$ for $\ell_1, \ldots, \ell_n$, respectively. Hence, the HOL tactic is as follows:

\begin{verbatim}
EQ_TAC THENL
 [REPEAT STRIP_TAC,
  MAP_EVERY EXISTS_TAC [\varphi_1, \ldots, \varphi_n]]
 THEN ASM_REWRITE_TAC[]
\end{verbatim}

Having computed this normal form and proven the equivalence with the original formula $\Phi$, preproven theorems are used to replace the ‘definitions’ $\ell_i = \varphi_i$ by initialization parts, parts for the transition relation, and parts for the acceptance condition, so that finally an equivalent $\omega$-automaton (cf. theorem 1) is obtained.

As a first step, the next lemma shows how the part for the transition relation is determined. This part stems from the recursion laws that the temporal operator obey. Clearly, as any location variable $\ell_\varphi$ should behave as the elementary formula $\varphi$ that it abbreviates, it must obey the same recursion laws. These recursion laws determine the transition relation of the $\omega$-automaton. A complete list of the recursion laws is given in the following lemma:
Lemma 1 (Recursion Laws of Temporal Operators) The following formulas are generally valid, i.e., they hold at any point $t$ of any model $\pi$:

\[
\begin{align*}
G\varphi &= \varphi \land XG\varphi & \overline{G}\varphi &= \varphi \land \overline{X}\overline{G}\varphi \\
F\varphi &= \varphi \lor XF\varphi & \overline{F}\varphi &= \varphi \lor \overline{X}\overline{F}\varphi \\
[\varphi \ U \ \psi] &= \psi \lor \varphi \land X[\varphi \ U \ \psi] & [\varphi \ U \ \psi] &= \psi \lor \varphi \land \overline{X}[\varphi \ U \ \psi] \\
[\varphi \ B \ \psi] &= \neg \psi \land (\varphi \lor X[\varphi \ B \ \psi]) & [\varphi \ B \ \psi] &= \neg \psi \land (\varphi \lor \overline{X}[\varphi \ B \ \psi]) \\
[\varphi \ W \ \psi] &= \psi \land \varphi \lor \neg \psi \land X[\varphi \ W \ \psi] & [\varphi \ W \ \psi] &= \psi \land \varphi \lor \neg \psi \land \overline{X}[\varphi \ W \ \psi] \\
[\varphi \ W \ \psi] &= \psi \land \varphi \lor \neg \psi \land X[\varphi \ W \ \psi] & [\varphi \ W \ \psi] &= \psi \land \varphi \lor \neg \psi \land \overline{X}[\varphi \ W \ \psi]
\end{align*}
\]

As can be seen, the recursion laws do not uniquely determine a temporal expression. In particular, the strong and weak temporal operators satisfy the same fixpoint equations. It is therefore not sufficient to replace the equations $\ell_\varphi = \varphi$ by their corresponding transition relation parts.

For this reason, the solutions of the above fixpoint equations must be investigated. The next theorem shows clearly that any of the above fixpoint equations has exactly two solutions that correspond with the strong and weak variant of the temporal operators. It moreover shows how one of these solutions can be selected by adding either a part for the acceptance condition or a suitable initialization condition.

Theorem 1 (Characterizing Temporal Operators as Fixpoints) The following formulas are generally valid, i.e., they hold at any point $t$ of any model $\pi$:

\[
\begin{align*}
G[q = \varphi \land Xq] &\iff G[q = \varphi] \lor G[q = F] \\
G[q = \varphi \land Xq] &\iff G[q = F] \lor G[q = T] \\
G[q = \psi \lor \varphi \land Xq] &\iff G[q = [\varphi \ U \ \psi]] \lor G[q = [\varphi \ U \ \psi]] \\
G[q = \neg \psi \land (\varphi \lor Xq)] &\iff G[q = [\varphi \ B \ \psi]] \lor G[q = [\varphi \ B \ \psi]] \\
G[q = \psi \land \varphi \lor \neg \psi \land Xq] &\iff G[q = [\varphi \ W \ \psi]] \lor G[q = [\varphi \ W \ \psi]] \\
G[Xq = \varphi \land q] &\iff G[q = \overline{X}G\varphi] \lor G[q = F] \\
G[Xq = \varphi \land q] &\iff G[q = \overline{X}F\varphi] \lor G[q = T] \\
G[Xq = \psi \lor \varphi \land q] &\iff G[q = \overline{X}[\varphi \ U \ \psi]] \lor G[q = \overline{X}[\varphi \ U \ \psi]] \\
G[Xq = \neg \psi \land (\varphi \lor q)] &\iff G[q = \overline{X}[\varphi \ B \ \psi]] \lor G[q = \overline{X}[\varphi \ B \ \psi]] \\
G[Xq = \psi \land \varphi \lor \neg \psi \land q] &\iff G[q = \overline{X}[\varphi \ W \ \psi]] \lor G[q = \overline{X}[\varphi \ W \ \psi]]
\end{align*}
\]

Hence, each of the above equations has exactly two solutions for $q$. The following equations that are valid at initial time ($t = 0$) of every model $\pi$, show how one of
the two solutions of the above fixpoint equations can be selected with either a fairness constraint or a suitable initial condition:

\[
\begin{align*}
G[q = G\varphi] & \iff G[q = \varphi \land Xq] \land GF[q \to q] \\
G[q = F\varphi] & \iff G[q = \varphi \lor Xq] \land GF[q \to \varphi] \\
G[q = [\varphi U \psi]] & \iff G[q = \psi \lor \varphi \land Xq] \land GF[q \lor \neg \varphi] \\
G[q = [\varphi U B \psi]] & \iff G[q = \neg \psi \land (\varphi \lor Xq)] \land GF[q \lor \psi] \\
G[q = [\varphi B \psi]] & \iff G[q = \neg \psi \land (\varphi \lor Xq)] \land GF[q \to \varphi] \\
G[q = [\varphi W \psi]] & \iff G[q = \psi \land \varphi \lor \neg \psi \land Xq] \land GF[q \lor \psi] \\
G[q = [\varphi W B \psi]] & \iff G[q = \psi \land \varphi \lor \neg \psi \land Xq] \land GF[q \to \psi] \\
G[q = \neg G\varphi] & \iff (q = T) \land G[Xq = \varphi \land q] \\
G[q = \neg X \neg F \varphi] & \iff (q = F) \land G[Xq = \varphi \lor q] \\
G[q = \neg X [\neg \varphi U \psi]] & \iff (q = T) \land G[Xq = \psi \lor \varphi \land q] \\
G[q = \neg X [\neg \varphi U B \psi]] & \iff (q = F) \land G[Xq = \psi \lor \varphi \land q] \\
G[q = \neg X [\neg \varphi B \psi]] & \iff (q = T) \land G[Xq = \neg \psi \land (\varphi \lor q)] \\
G[q = \neg X [\neg \varphi W \psi]] & \iff (q = T) \land G[Xq = \psi \lor \varphi \lor \neg \psi \land q] \\
G[q = \neg X [\neg \varphi W B \psi]] & \iff (q = F) \land G[Xq = \psi \lor \varphi \lor \neg \psi \land q]
\end{align*}
\]

The relation \(\alpha \preceq \beta :\iff \forall t.\alpha^{(t)} \rightarrow \beta^{(t)}\) defines an ordering relation on terms of type \(\mathbb{N} \to \mathbb{B}\). Using this ordering relation, one can see that \(G\alpha\) is the greatest fixpoint of \(f_\alpha(y) := a \land Xq\), 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 future 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., \(\ell = [a U b]\) by adding \(\ell = b \lor a \land X\ell\) to the transition relation fixes \(\ell\) such that it behaves either as \([a U b]\) or \([a U b]\).

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

The situation is similar for the past time temporal operators. Again, the strong and weak variants are solutions of the same fixpoint equation, where either a strong or weak previous operator is used. As the weak and strong variants do however differ at the initial point of time, we can choose between them by adding appropriate initial conditions. There is a further technical problem with the past temporal operators: the above equations only allow to abbreviate occurrences where a past temporal
operator is preceded by a previous operator. However, this is sufficient, since any formula that starts with a past temporal operator can be unrolled by the recursion laws so that the temporal operator is then preceded by a previous operator.

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 \( \omega \)-automaton. For example, the definition \( \ell_1 = \text{Ga} \) is replaced with the formula:

\[
[\forall t. \ell_1(t) = a(t) \land \ell_1(t+1)] \land \forall t_1. \exists t_2. a(t_1 + t_2) \rightarrow \ell_1(t_1 + t_2)
\]

and similarly, the equation \( \ell_1 = \overline{X} \text{Ga} \) would be replaced by

\[
[\ell_1(0) = T] \land [\forall t. \ell_1(t+1) = a(t) \land \ell_1(t)]
\]

As the introduced variables \( \ell_i \) occur under an existential quantifier in the definition normal form, this formula corresponds directly to a generalized Büchi automaton. Hence, the following theorem holds:

**Theorem 2 (Translating LTL to \( \omega \)-Automata)** Given any LTL formula \( \Phi \), the algorithm given in Figure 1.4 computes \( \{\ell_1 = \varphi_1, \ldots, \ell_n = \varphi_n\}, \Psi = \text{ElemForms}(\Phi) \) such that the following holds:

- \( \text{ElemForms}(\Phi) \) terminates in time \( O(|\Phi|) \)
- \( \Psi \) is a propositional formula
- each \( \varphi_i \) contains exactly one temporal operator that occurs on top of \( \varphi_i \)
- \( \varphi_i \) contains at most the variables that occur in \( \Phi \) plus the variables \( \ell_1, \ldots, \ell_{i-1} \)
- \( \Phi = \exists \ell_1 \ldots \ell_n. \Psi \land \bigwedge_{i \in I} G[\ell_i = \varphi_i] \)

The result of the function \( \text{Automaton} \) results in an equivalent generalized Büchi automaton with the initial states \( \Psi(0) \land \bigwedge_{i=1}^n \left[ \text{initial}(\ell_i, \varphi_i) \right](0) \), transition relation \( \bigwedge_{i=1}^n \text{trans}(\ell_i, \varphi_i) \) and acceptance condition \( \bigwedge_{i=1}^n \forall t_1. \exists t_2. \text{fair}(\ell_i, \varphi_i)^{(t_1 + t_2)} \). To be precise, the following equation holds for any model \( \pi \) at initial time \( (t = 0) \):

\[
\Phi = \left( \begin{array}{c}
\exists \ell_1, \ldots, \ell_n. \\
\Psi(0) \land \bigwedge_{i=1}^n \left[ \text{initial}(\ell_i, \varphi_i) \right](0) \\
\land \bigwedge_{i=1}^n \forall t. \text{trans}(\ell_i, \varphi_i)^{(t)} \\
\land \bigwedge_{i=1}^n \forall t_1. \exists t_2. \text{fair}(\ell_i, \varphi_i)^{(t_1 + t_2)} \end{array} \right)
\]

The construction yields in an automaton with \( 2^{O(|\Phi|)} \) states and an acceptance condition of length \( O(|\Phi|) \), although the computed automaton formula is of size \( O(|\Phi|) \) and is computed in time \( O(|\Phi|) \). Note further that the constructed \( \omega \)-automaton is in general nondeterministic. This can not be avoided, since deterministic Büchi automata are not as expressive as nondeterministic ones [43]. Finally, note that
function ElemForms(\( \Phi \))
case \( \Phi \) of
  is_prop(\( \varphi \)) : return (\{ \}, \( \varphi \));
  \( \neg \varphi \) : (\( E_1, \varphi_1 \)) \( \equiv \) ElemForms(\( \varphi \));
      return (\( E_1, \neg \varphi_1 \));
  \( \varphi \land \psi \) : (\( E_1, \varphi_1 \)) \( \equiv \) ElemForms(\( \varphi \)); (\( E_2, \psi_1 \)) \( \equiv \) ElemForms(\( \psi \));
      return (\( E_1 \cup E_2, \varphi_1 \land \psi_1 \));
  \( \varphi \lor \psi \) : (\( E_1, \varphi_1 \)) \( \equiv \) ElemForms(\( \varphi \)); (\( E_2, \psi_1 \)) \( \equiv \) ElemForms(\( \psi \));
      return (\( E_1 \cup E_2, \varphi_1 \lor \psi_1 \));
  \( X \varphi \) : (\( E_1, \varphi_1 \)) \( \equiv \) ElemForms(\( \varphi \)); \( \ell = \text{new\_var} \);
      return (\( E_1 \cup \{ \ell = X \varphi_1 \}, \ell \));
  \( \neg X \varphi \) : (\( E_1, \varphi_1 \)) \( \equiv \) ElemForms(\( \varphi \)); \( \ell = \text{new\_var} \);
      return (\( E_1 \cup \{ \ell = \neg X \varphi_1 \}, \ell \));
  \([ \varphi \cup \psi ] \) : (\( E_1, \varphi_1 \)) \( \equiv \) ElemForms(\( \varphi \)); (\( E_2, \psi_1 \)) \( \equiv \) ElemForms(\( \psi \)); \( \ell = \text{new\_var} \);
      return (\( E_1 \cup E_2 \cup \{ \ell = [ \varphi_1 \cup \psi_1 ] \}, \ell \));
  \([ \varphi \uplus \psi ] \) : (\( E_1, \varphi_1 \)) \( \equiv \) ElemForms(\( \varphi \)); (\( E_2, \psi_1 \)) \( \equiv \) ElemForms(\( \psi \)); \( \ell = \text{new\_var} \);
      return (\( E_1 \cup E_2 \cup \{ \ell = [ \varphi_1 \uplus \psi_1 ] \}, \psi_1 \lor \varphi_1 \land \ell \));

function trans(\( \ell, \Phi \))
case \( \Phi \) of
  \( X \varphi \) : return \( \ell = X \varphi \);
  \( \neg X \varphi \) : return \( X \ell = \varphi \);
  \([ \varphi \cup \psi ] \) : return \( \ell = \psi \lor \varphi \land X \ell \);
  \([ \varphi \uplus \psi ] \) : return \( X \ell = \psi \lor \varphi \land \ell \);

function fair(\( \ell, \Phi \))
case \( \Phi \) of
  \([ \varphi \cup \psi ] \) : return \( \neg \ell \lor \psi \);
  otherwise : return T;

function initial(\( \ell, \Phi \))
case \( \Phi \) of
  \( X \varphi \) : return \( \ell = T \);
  \([ \varphi \uplus \psi ] \) : return \( \ell = F \);
  otherwise : return T;

function Automaton(\( \Phi \))
\( \{ \ell_1 = \varphi_1, \ldots, \ell_n = \varphi_n \}, \Psi \) := ElemForms(\( \Phi \));
\( \exists \ell_1, \ldots, \ell_n. \Psi(0) \land \bigwedge_{i=1}^n \text{initial}(\ell_i, \varphi_i)(0) \land \bigwedge_{i=1}^n \forall \ell_i. \text{trans}(\ell_i, \varphi_i)(t) \land \bigwedge_{i=1}^n \forall \ell_1, \exists \ell_2. \text{fair}(\ell_i, \varphi_i)(t_1 + t_2) \)

Figure 1.4: Algorithm for translating LTL to \( \omega \)-automata
the algorithm of Figure 1.4 also applies recursion laws so that occurrences of past temporal operators are preceded by previous operators.

To illustrate the translation, consider the formula \((F \overline{G} a) \rightarrow (GF a)\). The construction of \(E\) and \(\Psi\) results in \(E = \{\ell_1 = \overline{G} a, \ell_2 = F \ell_1, \ell_3 = Fa, \ell_4 = G \ell_3\}\) and \(\Psi = \ell_2 \rightarrow \ell_4\). The next step is now to construct a transition relation out of \(E\) for an \(\omega\)-automaton with the state variables \(\ell_i\) such that each \(\ell_i\) behaves as \(\varphi_i\). For our example, we derive the following \(\omega\)-automaton:

\[
\exists \ell_1 \ell_2 \ell_3 \ell_4. \\
\left[ \ell_2^{(0)} \rightarrow \ell_4^{(0)} \right] \land \ell_1^{(0)} \land \\
\forall t. \\
\left[ \begin{array}{l}
\ell_1^{(t+1)} = a^{(t)} \land \ell_1^{(t)} \\
\ell_2^{(t)} = \ell_1^{(t)} \lor \ell_2^{(t+1)} \\
\ell_3^{(t)} = a^{(t)} \lor \ell_3^{(t+1)} \\
\ell_4^{(t)} = \ell_3^{(t)} \land \ell_4^{(t+1)} \\
\end{array} \right] \land \\
\left( \begin{array}{l}
\forall t_1 \exists t_2. \ell_2^{(t_1+1)} \rightarrow \ell_1^{(t_1+t_2)} \\
\forall t_1 \exists t_2. \ell_3^{(t_1+1)} \rightarrow a^{(t_1+t_2)} \\
\forall t_1 \exists t_2. \ell_3^{(t_1+1)} \rightarrow \ell_4^{(t_1+t_2)} \\
\end{array} \right)
\]

The above translation is the basis for LTL model checking, LTL satisfiability checking, and LTL theorem proving, since it allows to reduce these problems to nonemptiness problems of corresponding \(\omega\)-automata. To see this, consider first how LTL theorem proving is reduced to LTL satisfiability checking: Given that for a LTL formula \(\Phi\) over the variables \(x_0, \ldots, x_m\), it has to be checked whether any model \(\pi\) satisfies the formula \(\Phi\) at any position \(t \in \mathbb{N}\), i.e., that the relation \(\pi, t \models \Phi\) holds. Due to the used HOL embedding, this is equivalent to prove the HOL formula \(\forall x_0. \ldots \forall x_m. \forall t. \Phi^{(t)}\), which is due to the semantics of the temporal operators the same as \(\forall x_0. \ldots \forall x_m. [G \Phi]^{(0)}\).

The validity problem is dual to the satisfiability problem, where for a given LTL formula \(\Phi\) over the variables \(x_0, \ldots, x_m\), it has to be checked whether a model \(\pi\) and a position \(t \in \mathbb{N}\) exists so that \(\pi, t \models \Phi\) holds. Hence, it is to be checked whether \(\exists x_0. \ldots \exists x_m. \exists t. \Phi^{(t)}\) holds, which is equivalent to \(\exists x_0. \ldots \exists x_m. [F \Phi]^{(0)}\). Hence, \(\Phi\) is valid iff \(\neg \Phi\) is unsatisfiable, since \(\forall x_0. \ldots \forall x_m. [G \Phi]^{(0)}\) is equivalent to \(\neg \exists x_0. \ldots \exists x_m. [F \neg \Phi]^{(0)}\).

The satisfiability problem for a given LTL formula over the variables \(x_0, \ldots, x_m\) in turn is solved as follows: To check whether \(\exists x_0. \ldots \exists x_m. [F \Phi]^{(0)}\) holds, the formula \(F \Phi\) is translated by the algorithm of Figure 1.4. By the above theorem, this yields an \(\omega\)-automaton so that the following equation holds:
Therefore, the \textit{LTL satisfiability problem} is reduced to a nonemptiness check of the language accepted by the $\omega$-automaton given on the right hand side of the above equation. The latter problem can be efficiently solved by symbolic model checkers in that they must check whether there is a fair path through the transition system with initial states $\Psi^{(0)} \land \bigwedge_{i=1}^{n} \text{initial}(\ell_i, \varphi_i)^{(0)}$ and transition relation $\forall t. \bigwedge_{i=1}^{n} \text{trans}(\ell_i, \varphi_i)^{(t)}$. The \textit{‘fairness’} of the path is thereby determined by the fairness constraints $\text{fair}(\ell_1, \varphi_1), \ldots, \text{fair}(\ell_n, \varphi_n)$.

Finally, \textit{LTL model checking is reduced to an automaton problem}: In general, any finite state transition system can be described by an $\omega$-automaton with an empty acceptance condition, so that the following goal is to be proved where $\Phi$ is an LTL formula over the variables $x_0, \ldots, x_m$ (that occur as free variables below):

$$\left( \exists q_0 \ldots q_s. \Phi_I(q_0^{(0)}, \ldots, q_s^{(0)}) \land \left[ \forall t. \Phi_R(q_0^{(t)}, \ldots, q_s^{(t)}, x_0^{(t)}, \ldots, x_m^{(t)}, q_0^{(t+1)}, \ldots, q_s^{(t+1)}) \right] \right) \rightarrow [G\Phi]^{(0)}$$

This is obviously equivalent to the following goal:

$$\neg \left( \exists q_0 \ldots q_s. \Phi_I(q_0^{(0)}, \ldots, q_s^{(0)}) \land \left[ \forall t. \Phi_R(q_0^{(t)}, \ldots, q_s^{(t)}, x_0^{(t)}, \ldots, x_m^{(t)}, q_0^{(t+1)}, \ldots, q_s^{(t+1)}) \right] \right) \land \neg[G\Phi]^{(0)}$$

If $\neg[G\Phi]^{(0)}$ is then converted into an $\omega$-automaton, we end up with the following automaton problem:

$$\left( \exists x_0 \ldots x_m. \exists q_0 \ldots q_s. \exists \ell_1, \ldots, \ell_n. \left[ \Phi_I(q_0^{(0)}, \ldots, q_s^{(0)}) \land \Psi^{(0)} \land \bigwedge_{i=1}^{n} \text{initial}(\ell_i, \varphi_i)^{(0)} \right] \land \forall t. \left[ \Phi_R(q_0^{(t)}, \ldots, q_s^{(t)}, x_0^{(t)}, \ldots, x_m^{(t)}, q_0^{(t+1)}, \ldots, q_s^{(t+1)}) \right] \land \bigwedge_{i=1}^{n} \text{trans}(\ell_i, \varphi_i)^{(t)} \land \bigwedge_{i=1}^{n} \forall t_2. \text{fair}(\ell_i, \varphi_i)^{(t_2+t_2)} \right)$$

Hence, again a nonemptiness check of the language of an $\omega$-automaton is obtained.
Bibliography


[37] 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.


