Linear Temporal Logic and Büchi Automata

Yih-Kuen Tsay
Department of Information Management
National Taiwan University
FLOLAC 2009

Introduction

- We have seen how automata, in particular Büchi automata, may be used to describe the behaviors of a concurrent system.
- Büchi automata “localize” temporal dependency between occurrences of events (represented by propositions) to relations between states and tend to be of lower level.
- We will study an alternative formalism, namely linear temporal logic.
- Temporal logic formulae describe temporal dependency without explicit references to time points and are in general more abstract.

Outline

- Introduction
- Propositional Temporal Logic (PTL)
- Quantified Propositional Temporal Logic (QPTL)
- Basic Properties
- From Temporal Formulae to Automata
  - On-the-fly Translation
  - Tableau Construction
- Concluding Remarks
- References

Introduction (cont.)

- The above Büchi automaton says that, whenever \( p \) holds at some point in time, \( q \) must hold at the same time or will hold at a later time.
- It may not be easy to see that this indeed is the case.
- In linear temporal logic, this can easily be expressed as \( \Box(p \rightarrow \Diamond q) \), which reads “always \( p \) implies eventually \( q \)”.

\[ s_0 \quad \text{~p~} \quad q \quad \text{~q~} \quad s_1 \]

\[ p \rightarrow q \quad \text{and} \quad q \quad \text{~q~} \quad s_1 \]
PTL: The Future Only

1. We first look at the future fragment of Propositional Temporal Logic (PTL).
2. Future operators include ⪪ (next), ⋄ (eventually), □ (always), Ո until), and ⊤ (wait-for).
3. With Ո replaced by Ռ (release), this fragment is often referred to as LTL (linear temporal logic) in the model checking community.
4. Let (Tree be a set of boolean variables.
5. The future PTL formulae are defined inductively as follows:
   - Every variable Ո is a PTL formula.
   - If Ո and Ո are PTL formulae, then so are Ո, Ո, Ո, Ո, Ո and Ո Ո Ո.
6. Examples: Ո Ո Ո Ո Ո.

PTL: The Future Only (cont.)

- A PTL formula is interpreted over an infinite sequence of states Ո = Ո Ո Ո, relative to a position in that sequence.
- A state is a subset of Ո, containing exactly those variables that evaluate to true in that state.
- If each possible subset of Ո is treated as a symbol, then a sequence of states can also be viewed as an infinite word over Ո.
- The semantics of PTL in terms of (Ո, i) = Ո (f holds at the i-th position of Ո) is given below.
- We say that a sequence Ո satisfies a PTL formula Ո or Ո is a model of Ո, denoted Ո = Ո, if (Ո, 0) = Ո.
The full PTL formulae are defined inductively as follows:

- Every variable \( p \in V \) is a PTL formula.
- If \( f \) and \( g \) are PTL formulae, then so are \( \neg f \), \( f \lor g \), \( f \land g \), \( \Diamond f \), \( \Box f \), \( f \limp g \), \( f \limp \Diamond g \), \( f \limp \Box g \), and \( f B g \).
- \( \neg \neg f \) is also written as \( f \limp \neg g \) and \( (f \limp g) \land (g \limp f) \) as \( f \iff g \).

**Examples:**

- \( \Box (p \limp \Diamond q) \) says “every \( p \) is preceded by a \( q \).”
- \( \Box (\neg p \limp \Diamond q) \) is another way of saying \( p \limp \Diamond q \).

**PTL: Adding the Past**

We now add the past fragment.

- Past operators include \( \Diamond \) (before), \( \Diamond \) (previous), \( \Diamond \) (once), \( \Box \) (so-far), \( S \) (since), and \( B \) (back-to).

- The full PTL formulae are defined inductively as follows:
  - Every variable \( p \in V \) is a PTL formula.
  - If \( f \) and \( g \) are PTL formulae, then so are \( \neg f \), \( f \lor g \), \( f \land g \), \( \Diamond f \), \( \Box f \), \( f \limp g \), \( f \limp \Diamond g \), \( f \limp \Box g \), and \( f B g \).
  - \( \neg \neg f \) is also written as \( f \limp \neg g \) and \( (f \limp g) \land (g \limp f) \) as \( f \iff g \).

- **Examples:**
  - \( \Box (p \limp \Diamond q) \) says “every \( p \) is preceded by a \( q \).”
  - \( \Box (\neg p \limp \Diamond q) \) is another way of saying \( p \limp \Diamond q \).
Quantified Propositional Temporal Logic (QPTL) is PTL extended with quantification over boolean variables (so, every PTL formula is also a QPTL formula):

- If \( f \) is a QPTL formula and \( x \in V \), then \( \forall x: f \) and \( \exists x: f \) are QPTL formulae.
- Let \( \sigma = s_0 s_1 \cdots \) and \( \sigma' = s'_0 s'_1 \cdots \) be two sequences of states.
- We say that \( \sigma' \) is a \( x \)-variant of \( \sigma \) if, for every \( i \geq 0 \), \( s'_i \) differs from \( s_i \) at most in the valuation of \( x \), i.e., the symmetric set difference of \( s'_i \) and \( s_i \) is either \( \{x\} \) or empty.
- The semantics of QPTL is defined by extending that of PTL with additional semantic definitions for the quantifiers:
  - \((\sigma, i) \models \exists x: f \iff (\sigma', i) \models f \) for some \( x \)-variant \( \sigma' \) of \( \sigma \)
  - \((\sigma, i) \models \forall x: f \iff (\sigma', i) \models f \) for all \( x \)-variant \( \sigma' \) of \( \sigma \)

\[\]

**Equivalence and Congruence**

- A formula \( p \) is valid, denoted \( \models p \), if \( \sigma \models p \) for every \( \sigma \).
- Two formulae \( p \) and \( q \) are equivalent if \( \models p \leftrightarrow q \), i.e., \( \sigma \models p \) if and only if \( \sigma \models q \) for every \( \sigma \).
- Two formulae \( p \) and \( q \) are congruent, denoted \( p \cong q \), if \( \models \Box(p \leftrightarrow q) \).
- Congruence is a stronger relation than equivalence:
  - \( p \lor \neg p \) and \( \neg \Box(p \lor \neg p) \) are equivalent, as they are both true at position 0 of every model.
  - However, they are not congruent; \( p \lor \neg p \) holds at all positions of every model, while \( \neg \Box(p \lor \neg p) \) holds only at position 0.

\[\]

**Congruences**

- A minimal set of operators:
  \[\neg, \lor, \Box, W, \ominus, B\]
- Other operators could be encoded:
  \[\Box p \cong \neg \Box \neg p\]
  \[W p \cong \Box W \Box p\]
  \[B p \cong \Box B \Box p\]

**Weak vs. strong operators:**

\[\Box p \cong (\Box p \land \Box True)\]
\[\Box p \cong (\Box p \land \Box False)\]
\[p \lor q \cong (p W q \land \Box q)\]
\[W q \cong (p \lor q \land \Box p)\]
\[p S q \cong (p B q \land \Box q)\]
\[B q \cong (p S q \land \Box p)\]

**Congruences (cont.)**

- Duality:
  \[\neg \Box p \cong \Box \neg p\]
  \[\neg \Box p \cong \Box \neg p\]

**A formula is in the negation normal form** if negation only occurs in front of an atomic proposition.

**Every PTL/QPTL formula can be converted into an equivalent formula in the negation normal form.**
Congruences (cont.)

Expansion Formulae:

- \( \Box P \equiv P \land \Box P \)
- \( \Diamond P \equiv P \lor \Diamond P \)
- \( P \cup Q \equiv Q \lor (P \land (P \cup Q)) \)
- \( P \mid Q \equiv Q \lor (P \land (P \mid Q)) \)
- \( P \ast Q \equiv (P \land Q) \lor (Q \land (P \ast Q)) \)
- \( P \ast Q \equiv (P \land Q) \lor (Q \land (P \ast Q)) \)

These expansion formulae are essential in translation of a temporal formula into an equivalent Büchi automaton.

Idempotence:

- \( \Diamond \Diamond P \equiv \Diamond P \)
- \( \Box \Box P \equiv \Box P \)
- \( P \cup (P \cup Q) \equiv P \cup Q \)
- \( P \mid (P \mid Q) \equiv P \mid Q \)
- \( P \ast (P \ast Q) \equiv P \ast Q \)
- \( P \ast (P \ast Q) \equiv P \ast Q \)

These expansion formulae are essential in translation of a temporal formula into an equivalent Büchi automaton.

Expressiveness

Theorem

PTL is strictly less expressive than Büchi automata.

Proof.

1. Every PTL formula can be translated into an equivalent Büchi automaton.
2. “P holds at every even position” is recognizable by a Büchi automaton, but cannot be expressed in PTL.

Theorem

QPTL is expressively equivalent to Büchi automata (and hence \( \omega \)-regular expressions and S1S).

Simple On-the-fly Translation

This is a tableau-based algorithm for obtaining a Büchi automaton from an LTL (future PTL) formula.

The algorithm is geared towards being used in model checking in an on-the-fly fashion:

- It is possible to detect that a property does not hold by only constructing part of the model and of the automaton.
- The algorithm can also be used to check the validity of a temporal logic assertion.

To apply the translation algorithm, we first convert the formula \( \varphi \) into the negation normal form.
Data Structure of an Automaton Node

- **ID**: A string that identifies the node.
- **Incoming**: The incoming edges represented by the IDs of the nodes with an outgoing edge leading to the current node.
- **New**: A set of subformulae that must hold at the current state and have not yet been processed.
- **Old**: The subformulae that must hold in the node and have already been processed.
- **Next**: The subformulae that must hold in all states that are immediate successors of states satisfying the properties in **Old**.

The Algorithm

The algorithm starts with a single node, which has a single incoming edge labeled `init` (i.e., from an initial node) and expands the nodes in a DFS manner.

This starting node has initially one new obligation in **New**, namely $\phi$, and **Old** and **Next** are initially empty.

With the current node $N$, the algorithm checks if there are unprocessed obligations left in **New**.

If not, the current node is fully processed and ready to be added to **Nodes**.

If there already is a node in **Nodes** with the same obligations in both its **Old** and **Next** fields, the incoming edges of $N$ are incorporated into those of the existing node.

The Algorithm (cont.)

If no such node exists in **Nodes**, then the current node $N$ is added to this list, and a new current node is formed for its successor as follows:

- There is initially one edge from $N$ to the new node.
- **New** is set initially to the **Next** field of $N$.
- **Old** and **Next** of the new node are initially empty.

When processing the current node, a formula $\eta$ in **New** is removed from this list.

In the case that $\eta$ is a literal (a proposition or the negation of a proposition), then

- if $\neg \eta$ is in **Old**, the current node is discarded;
- otherwise, $\eta$ is added to **Old**.

When $\eta$ is not a literal, the current node can be split into two or not split, and new formulae can be added to the fields **New** and **Next**.

The exact actions depend on the form of $\eta$:

- $\eta = p \land q$, then both $p$ and $q$ are added to **New**.
- $\eta = p \lor q$, then the node is split, adding $p$ to **New** of one copy, and $q$ to the other.
- $\eta = p U q (\equiv q \lor (p \land \Box (p R q)))$, then the node is split.
  For the first copy, $p$ is added to **New** and $p U q$ to **Next**.
  For the other copy, $q$ is added to **New**.
- $\eta = p R q (\equiv (q \land p) \lor (q \land \Box (p R q)))$, similar to $U$.
- $\eta = \Diamond p$, then $p$ is added to **Next**.
Nodes to GBA

The list of nodes in \textit{Nodes} can now be converted into a \textit{generalized \Buchi automaton} \( B = (\Sigma, Q, q_0, \Delta, F) \):

\begin{itemize}
  \item \( \Sigma \) consists of sets of propositions from \( AP \).
  \item The set of states \( Q \) includes the nodes in \textit{Nodes} and the additional initial state \( q_0 \).
  \item \((r, \alpha, r') \in \Delta \) iff \( r \in \text{Incoming}(r') \) and \( \alpha \) satisfies the conjunction of the negated and nonnegated propositions in \( \text{Old}(r') \)
  \item \( q_0 \) is the initial state, playing the role of \textit{init}.
  \item \( F \) contains a separate set \( F_i \) of states for each subformula of the form \( p \mathrel{U} q \); \( F_i \) contains all the states \( r \) such that either \( q \in \text{Old}(r) \) or \( p \mathrel{U} q \notin \text{Old}(r) \).
\end{itemize}

Tableau Construction

\begin{itemize}
  \item We next study the Tableau Construction as described in [Manna and Pnueli 1995], which handles both future and past temporal operators.
  \item More efficient constructions exist, but this construction is relatively easy to understand.
  \item A \textit{tableau} is a graphical representation of all models/sequences that satisfy the given temporal logic formula.
  \item The construction results in essentially a GBA, but leaving propositions on the states (rather than moving them to the incoming edges of a state).
  \item Our presentation will be slightly different, to make the resulting GBA more apparent.
\end{itemize}

Expansion Formulae

\begin{itemize}
  \item The requirement that a temporal formula holds at a position \( j \) of a model can often be decomposed into requirements that:
    \begin{itemize}
      \item a simpler formula holds at the same position and
      \item some other formula holds either at \( j + 1 \) or \( j - 1 \).
    \end{itemize}
  \item For this decomposition, we have the following expansion formulae:
    \begin{align*}
      \Box p &\equiv p \land \Box \Box p \\
      \Diamond p &\equiv p \lor \Diamond \Diamond p \\
      p \mathrel{U} q &\equiv q \lor (p \land (p \mathrel{U} q)) \\
      p \mathrel{S} q &\equiv q \lor (p \land (p \mathrel{S} q)) \\
      p \mathrel{W} q &\equiv q \lor (p \land (p \mathrel{W} q)) \\
      p \mathrel{B} q &\equiv q \lor (p \land (p \mathrel{B} q))
    \end{align*}
  \item Note: this construction does not deal with \( R \).
\end{itemize}

Closure

\begin{itemize}
  \item We define the \textit{closure} of a formula \( \varphi \), denoted by \( \Phi_\varphi \), as the smallest set of formulae satisfying the following requirements:
    \begin{itemize}
      \item \( \varphi \in \Phi_\varphi \).
      \item For every \( p \in \Phi_\varphi \), if \( q \) a subformula of \( p \) then \( q \in \Phi_\varphi \).
      \item For every \( p \in \Phi_\varphi \), \( \neg p \in \Phi_\varphi \).
      \item For every \( \psi \in \{ \Box p, \Diamond p, p \mathrel{U} q, p \mathrel{W} q \} \), if \( \psi \in \Phi_\varphi \) then \( \Diamond \psi \in \Phi_\varphi \).
      \item For every \( \psi \in \{ \Box p, p \mathrel{S} q \} \), if \( \psi \in \Phi_\varphi \) then \( \Box \psi \in \Phi_\varphi \).
      \item For every \( \psi \in \{ \Diamond p, p \mathrel{B} q \} \), if \( \psi \in \Phi_\varphi \) then \( \Diamond \psi \in \Phi_\varphi \).
    \end{itemize}
  \item So, the closure \( \Phi_\varphi \) of a formula \( \varphi \) includes all formulae that are relevant to the truth of \( \varphi \).
\end{itemize}
Classification of Formulae

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$K(\alpha)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p \land q$</td>
<td>$p$, $q$</td>
</tr>
<tr>
<td>$\Box p$</td>
<td>$p$, $\Box p$</td>
</tr>
<tr>
<td>$\Diamond p$</td>
<td>$p$, $\Diamond p$</td>
</tr>
<tr>
<td>$p \cup q$</td>
<td>$q$, $\Diamond (p \cup q)$</td>
</tr>
<tr>
<td>$p \wedge q$</td>
<td>$q$, $\Diamond (p \wedge q)$</td>
</tr>
<tr>
<td>$p \vee q$</td>
<td>$q$, $\Diamond (p \vee q)$</td>
</tr>
<tr>
<td>$p \setminus q$</td>
<td>$q$, $\Diamond (p \setminus q)$</td>
</tr>
<tr>
<td>$p \setminus \neg q$</td>
<td>$q$, $\Diamond (p \setminus \neg q)$</td>
</tr>
<tr>
<td>$p \Box q$</td>
<td>$q$, $\Diamond (p \Box q)$</td>
</tr>
</tbody>
</table>

An $\alpha$-formula $\varphi$ holds at position $j$ iff all the $K(\varphi)$-formulae hold at $j$.

A $\beta$-formula $\psi$ holds at position $j$ iff either $K_1(\psi)$ or all the $K_2(\psi)$-formulae (or both) hold at $j$.

Mutually Satisfiable Formulae

A set of formulae $S \subseteq \Phi_\varphi$ is called mutually satisfiable if there exists a model $\sigma$ and a position $j \geq 0$, such that every formula $p \in S$ holds at position $j$ of $\sigma$.

The intended meaning of an atom is that it represents a maximal mutually satisfiable set of formulae.

Claim (atoms represent necessary conditions)

Let $S \subseteq \Phi_\varphi$ be a mutually satisfiable set of formulae. Then there exists a $\varphi$-atom $A$ such that $S \subseteq A$.

Basic Formulae

A formula is called basic if it is either a proposition or has the form $\Diamond p$, $\neg \Diamond p$, or $\neg \Diamond p$.

Basic formulae are important because their presence or absence in an atom uniquely determines all other closure formulae in the same atom.

Let $\Phi_\varphi^+$ denote the set of formulae in $\Phi_\varphi$ that are not of the form $\neg \Diamond \psi$.

Algorithm (atom construction)

1. Find all basic formulae $p_1, \ldots, p_b \in \Phi_\varphi^+$.
2. Construct all $2^b$ combinations.
3. Complete each combination into a full atom.

Atoms

We define an atom over $\varphi$ to be a subset $A \subseteq \Phi_\varphi$ satisfying the following requirements:

- $R_{\text{sat}}$: the conjunction of all state formulae in $A$ is satisfiable.
- $R_\varphi$: for every $p \in \Phi_\varphi$, $p \in A$ iff $\neg p \notin A$.
- $R_\alpha$: for every $\alpha$-formula $p \in \Phi_\varphi$, $p \in A$ iff $K(p) \subseteq A$.
- $R_\beta$: for every $\beta$-formula $p \in \Phi_\varphi$, $p \in A$ iff either $K_1(p) \in A$ or $K_2(p) \subseteq A$ (or both).

For example, if atom $A$ contains the formula $\neg \Diamond p$, it must also contain the formulae $\neg p$ and $\neg \Diamond \neg p$. 

Mutually Satisfiable Formulae

A set of formulae $S \subseteq \Phi_\varphi$ is called mutually satisfiable if there exists a model $\sigma$ and a position $j \geq 0$, such that every formula $p \in S$ holds at position $j$ of $\sigma$.

The intended meaning of an atom is that it represents a maximal mutually satisfiable set of formulae.

Claim (atoms represent necessary conditions)

Let $S \subseteq \Phi_\varphi$ be a mutually satisfiable set of formulae. Then there exists a $\varphi$-atom $A$ such that $S \subseteq A$.

It is important to realize that inclusion in an atom is only a necessary condition for mutual satisfiability (e.g., $\{\Diamond p \vee \Diamond \neg p, \Diamond p, \Diamond \neg p, p\}$ is an atom for the formula $\Diamond p \vee \Diamond \neg p$).
Example

Consider the formula \( \varphi_1 : \Box p \land \diamond \neg p \) whose basic formulae are

\[ p, \bigcirc \Box p, \Box \Diamond \neg p. \]

Following is the list of all atoms of \( \varphi_1 \):

- \( A_0 : \{ \neg p, \neg \Box \neg p, \bigcirc \Diamond \neg p, \neg p, \Box \neg p, \neg \varphi_1 \} \)
- \( A_1 : \{ p, \neg \Box \neg p, \bigcirc \Diamond \neg p, \neg p, \Box \neg p, \neg \varphi_1 \} \)
- \( A_2 : \{ \neg p, \neg \Box \neg p, \bigcirc \Diamond \neg p, \neg p, \Box \neg p, \neg \varphi_1 \} \)
- \( A_3 : \{ p, \neg \Box \neg p, \bigcirc \Diamond \neg p, \neg p, \Box \neg p, \neg \varphi_1 \} \)
- \( A_4 : \{ \neg p, \bigcirc \Box p, \bigcirc \Diamond \neg p, \neg p, \Box \neg p, \neg \varphi_1 \} \)
- \( A_5 : \{ p, \bigcirc \Box p, \bigcirc \Diamond \neg p, \neg p, \Box \neg p, \neg \varphi_1 \} \)
- \( A_6 : \{ \neg p, \bigcirc \Box p, \bigcirc \Diamond \neg p, \neg p, \Box \neg p, \varphi_1 \} \)
- \( A_7 : \{ p, \bigcirc \Box p, \bigcirc \Diamond \neg p, \neg p, \Box \neg p, \varphi_1 \} \)

The Tableau

Given a formula \( \varphi \), we construct a directed graph \( T_\varphi \), called the tableau of \( \varphi \), by the following algorithm.

Algorithm (tableau construction)

1. The nodes of \( T_\varphi \) are the atoms of \( \varphi \).
2. Atom \( A \) is connected to atom \( B \) by a directed edge if all of the following are satisfied:
   - \( R_\bigcirc : \text{For every } \bigcirc p \in \Phi_\varphi, \bigcirc p \in A \iff p \in B \).
   - \( R_\Box : \text{For every } \Box p \in \Phi_\varphi, p \in A \iff \Box p \in B \).
   - \( R_\Diamond : \text{For every } \Diamond p \in \Phi_\varphi, p \in A \iff \Diamond p \in B \).
3. An atom is called initial if it does not contain a formula of the form \( \bigcirc p \) or \( \neg \bigcirc p \) (\( \equiv \bigcirc \neg p \)).

From the Tableau to a GBA

1. Create an initial node and link it to every initial atom that contains \( \varphi \).
2. Label each directed edge with the atomic propositions that are contained in the ending atom.
3. Add a set of atoms to the accepting set for each subformula of the following form:
   - \( \Diamond q \): atoms with \( q \) or \( \neg \diamond q \).
   - \( p \cup q \): atoms with \( q \) or \( \neg (p \cup q) \).
   - \( \neg \Box q (\equiv \Diamond \neg q) \): atoms with \( q \) or \( \neg q \).
   - \( \neg (\neg q W p) (\equiv \neg p \cup (q \land \neg p)) \): atoms with \( q \) or \( \neg q \).
   - \( \neg q (\equiv \Diamond \neg q) \): atoms with \( q \) or \( \neg q \).
   - \( \neg (q W p) (\equiv \neg p \cup (\neg q \land \neg p)) \): atoms with \( q \) or \( \neg q \).

Figure: Tableau \( T_{\varphi_1} \) for \( \varphi_1 = \Box p \land \diamond \neg p \). Source: [Manna and Pnueli 1995].
Correctness: Models vs. Paths

- For a model $\sigma$, the infinite atom path $\pi_\sigma : A_0, A_1, \cdots$ in $T_\varphi$ is said to be induced by $\sigma$ if, for every position $j \geq 0$ and every closure formula $p \in \Phi_\varphi$,

  $$(\sigma, j) \models p \text{ iff } p \in A_j.$$ 

Claim (models induce paths)

Consider a formula $\varphi$ and its tableau $T_\varphi$. For every model $\sigma : s_0, s_1, \cdots$, there exists an infinite atom path $\pi_\sigma : A_0, A_1, \cdots$ in $T_\varphi$ induced by $\sigma$.

Furthermore, $A_0$ is an initial atom, and if $\sigma \models \varphi$ then $\varphi \in A_0$.

Correctness: Promising Formulae

- A formula $\psi \in \Phi_\varphi$ is said to promise the formula $r$ if $\psi$ has one of the following forms:

  $$\Diamond r, \ p \mathrel{\mathcal{U}} r, \ \neg \Box \neg r, \ \neg (\neg r \mathrel{\mathcal{W}} p).$$

  or if $r$ is the negation $\neg q$ and $\psi$ has one of the forms:

  $$\neg \Box q, \ \neg (q \mathrel{\mathcal{W}} p).$$

Claim (promise fulfillment by models)

Let $\sigma$ be a model and $\psi$, a formula promising $r$. Then, $\sigma$ contains infinitely many positions $j \geq 0$ such that

$$(\sigma, j) \models \neg \psi \text{ or } (\sigma, j) \models r.$$ 

Correctness: Fulfilling Paths

- Atom $A$ fulfills a formula $\psi$ that promises $r$ if $\neg \psi \in A$ or $r \in A$.

- A path $\pi : A_0, A_1, \cdots$ in the tableau $T_\varphi$ is called fulfilling:

  - $A_0$ is an initial atom.
  - For every promising formula $\psi \in \Phi_\varphi$, $\pi$ contains infinitely many atoms $A_j$ that fulfill $\psi$.

Claim (models induce fulfilling paths)

If $\pi_\sigma : A_0, A_1, \cdots$ is a path induced by a model $\sigma$, then $\pi_\sigma$ is fulfilling.

Correctness: Fulfilling Paths (cont.)

Claim (fulfilling paths induce models)

If $\pi : A_0, A_1, \cdots$ is a fulfilling path in $T_\varphi$, there exists a model $\sigma$ inducing $\pi$, i.e., $\pi = \pi_\sigma$ and, for every $\psi \in \Phi_\varphi$ and every $j \geq 0$,

$$(\sigma, j) \models \psi \text{ iff } \psi \in A_j.$$ 

Proposition (satisfiability and fulfilling paths)

Formula $\varphi$ is satisfiable iff the tableau $T_\varphi$ contains a fulfilling path $\pi = A_0, A_1, \cdots$ such that $A_0$ is an initial $\varphi$-atom.
Concluding Remarks

PTL can be extended in other ways to be as expressive as Büchi automata, i.e., to express all $\omega$-regular properties.

For example, the industry standard IEEE 1850 Property Specification Language (PSL) is based on an extension that adds classic regular expressions.

Regarding translation of a temporal formula into an equivalent Büchi automaton, there have been quite a few algorithms proposed in the past.

How to obtain an automaton as small as possible remains interesting, for both theoretical and practical reasons.

References


