Review of Temporal Logic and Buchi Automata

Computer Science and Artificial Intelligence Laboratory
MIT
Armando Solar-Lezama

Nov 25, 2011
Review of Temporal Logic

What about the following formula:

- $\text{AG } \text{EF } p$
Review of Temporal Logic

What does the following formula mean
1) $A( F G p)$

How about
2) $A( F A G p)$

How about
3) $A(F E G p)$
History Lesson

“Sometimes” and “Not Never” Revisited: On Branching versus Linear Time Temporal Logic
- Allen Emerson and Joseph Y. Halpern JACM Vol 33, 1986

Introduces CTL* as a way to unify branching time and linear time logics
Review of Temporal Logic

From any state, it is possible to return to the reset state along some execution.
  - AGEF reset

A request should stay asserted until an acknowledge is received. The acknowledge must eventually be received.
  - G req → req U ack

And, Ack must be received three cycles after request
  - G req → (req U ack ^ XXX ack)
Review of Temporal Logic

Engine starts and stops with button push
- If engine is off, it stays off until I push
  • If I never push it stays off forever
- If engine is on, it stays on until I push
  • If I never push it stays on forever
- If the engine is on, I should be able to stop it at any moment
- If it is off, I should be able to turn it back on, but not without identifying myself

\[ G \text{ off} \Rightarrow \text{off U push} \]
\[ G (\text{off} \Rightarrow (\text{off U push} \vee G \text{ off})) \]
\[ G (\text{on} \Rightarrow (\text{on U push} \vee G \text{ on})) \]
\[ AG (\text{on} \Rightarrow EF \text{ off}) \]
\[ AG (\text{off} \Rightarrow (EF \text{ on}) \land A((\text{offUid}) \lor G \text{ off})) \]
\[ A((\text{offUid}) \lor G \text{ off}) \equiv \neg E(\neg \text{idU}(\neg \text{off} \land \neg \text{id})) \]
Can the trains collide? \( \neg F (ph = 2 \land pv = 2) \)

\[
\text{while(*){}
\quad \text{if}(p=0)\{
\quad \quad p:=1;
\quad \}
\quad \text{if}(p=1)\{
\quad \quad \text{if}(g=\text{free})\{
\quad \quad \quad g:=\text{id};
\quad \quad \quad p:=2;
\quad \quad \}
\quad \}
\quad \text{if}(p=2)\{
\quad \quad \text{if}(p=1)\{
\quad \quad \quad p:=1;
\quad \quad \}
\quad \}
\quad \text{if}(p=2)\{
\quad \quad \text{if}(g=\text{free})\{
\quad \quad \quad g:=\text{id};
\quad \quad \quad p:=2;
\quad \quad \}
\quad \}
\quad \text{if}(p=3)\{
\quad \quad \text{if}(p=3)\{
\quad \quad \quad p:=0;
\quad \quad \}
\quad \}
\quad \text{if}(p=0)\{
\quad \quad p:=1;
\quad \}
\quad \}
\]

\[
\begin{align*}
ph &= \{0, 1, 2, 3\} \\
pv &= \{0, 1, 2, 3\} \\
g &= \{h, v, \text{free}\} \\
pch &= \{0, 1, \ldots, 9\} \\
pcv &= \{0, 1, \ldots, 9\}
\end{align*}
\]
Can the trains collide? \[ \neg F (ph = 2 \land pv = 2) \]

`while(*){
  if(p=0){
    p:=1;
    }
    if(p=1){
      if(g=free){
        g:=id;
        p:=2;
        }
        if(p=2){
          p:=3; g:=free
        }
        if(p=3){
          p:=0;
        }
  }
}`

` － F (ph = 2 \land pv = 2)`

**H train**

**V train**

\( ph = \{0, 1, 2, 3\} \)
\( pv = \{0, 1, 2, 3\} \)
\( g = \{h, \ v, \ free\} \)
\( pch = \{0, 1, \ldots, 9\} \)
\( pcv = \{0, 1, \ldots, 9\} \)

(ph, pv, g, pch, pcv)
Liveness Vs. Safety

Two terms you are likely to run into:

Safety:
- Something bad will never happen: $G \neg \text{bad}$
- If it fails to hold, it’s easy to produce a witness

Liveness:
- Something good will eventually happen: $F \text{ good}$
- What does a witness for this look like?
Automata for LTL properties

LTL defines properties over a trace

Given a trace, we want to know whether it satisfies the property

Problem:
- we need to build an automata to recognize infinite strings!
- $\omega$ -- Regular Languages
Buchi Automata

Similar to a DFA
- but with a stronger notion of acceptance

In DFA, you have an accept state
- when you reach accept state, you are done
- this means you only accept finite strings

In Buchi automata you also have accepting states
- but you only accept strings that visit the accept state infinitely often
Buchi Automata

A Buchi Automaton is a 4-tuple \( \langle \Sigma, S, I, \delta, F \rangle \)
- \( \Sigma \) is an alphabet
- \( S \) is a finite set of states
- \( I \subseteq S \) is a set of initial states
- \( \delta \subseteq S \times \Sigma \times S \) is a transition relation
- \( F \subseteq S \) is a set of accepting states

Non-deterministic Buchi Automata are not equivalent to deterministic ones
Example

G req $\rightarrow$ F ack
Example
From LTL to automata

Any LTL formula can be expressed as a Buchi automata
- but the construction of the automata is complicated
  • exponential on the size of the formula

- See Vardi and Wolper, *Reasoning about infinite computations*, 1983

Not all Buchi automata correspond to LTL formulas
Explicit State Model checking

The basic Strategy

- Temporal Logic Formula
  - Buchi Automata
  - Product Automata
  - Kripke structure

- Model checker

- OK

- Counterexample trace
Buchi Automaton from Kripke Structure

Given a Kripke structure:
- \( M = (S, S_0, R, L) \)

Construct a Buchi Automaton
- \((\Sigma, S \cup \{Init\}, \{Init\}, T, S \cup \{Init\})\)

- \(T\) is defined s.t.
  - \(T(s, \sigma, s') \iff R(s, s') \text{ and } \sigma \in L(s')\)
  - \(T(Init, \sigma, s) \iff s \in S_0 \text{ and } \sigma \in L(s)\)
Buchi Automaton from Kripke Structure

- $(\Sigma, S \cup \{\text{Init}\}, \{\text{Init}\}, T, S \cup \{\text{Init}\})$
- $T$ is defined s.t.
  - $T(s, \sigma, s')$ iff $R(s, s')$ and $\sigma \in L(s')$
  - $T(\text{Init}, \sigma, s)$ iff $s \in S_0$ and $\sigma \in L(s)$
Given a Kripke structure:
- \( M = (S, S_0, R, L) \)

Construct a Buchi Automaton
- \( (\Sigma, S \cup \{\text{Init}\}, \{\text{Init}\}, T, S \cup \{\text{Init}\} ) \)

- \( T \) is defined s.t.
  - \( T(s, \sigma, s') \) iff \( R(s, s') \) and \( \sigma \in L(s') \)
  - \( T(\text{Init}, \sigma, s) \) iff \( s \in S_0 \) and \( \sigma \in L(s) \)

What about missing transitions?
- Need to add a dummy “error state”
Explicit State Model checking

The basic Strategy

- Temporal Logic Formula
- Buchi Automata
- Product Automata
- Model checker
- Counterexample trace

System Description

Kripke structure
Negated Properties

Given a good property $P$, you can define a bad property $P'$:
- If the system has a trace that satisfies $P'$, then it is buggy.

Example
- Good property: $G(\text{req} \implies \text{F ack})$
- Bad property: $F(\text{req} \& (G \neg \text{ack}))$

We are going to ask whether $M$ satisfies $P'$:
- If it does, then we found a bug

Why are we doing the negation?
Computing the Product Automata

Given Buchi automata A and B’
- A = (Σ, S_A, T_A, {Init_A}, S_A)
- B’ = (Σ, S_B, T_B, {Init_B}, F’)
- A x B’ = (Σ, S_A x S_B, T, {(Init_A, Init_B)}, F)

Where
- T((s_1, s_2), σ, (s_1’, s_2’)) iff T_A(s_1, σ, s_1’) and T_B(s_2, σ, s_2’)
- (s_1, s_2) F iff s_2 ∈ F’
Check if a state is visited infinitely often

Check for a cycle with an accepting state

Cycle must be reachable from the initial state

Simple algorithm
- Do DFS to find an accepting state
- Do a DFS from that accepting state to see if it can reach itself