Review of Temporal Logic and Buchi Automata

Computer Science and Artificial Intelligence Laboratory
MIT
Armando Solar-Lezama

Nov 25, 2015

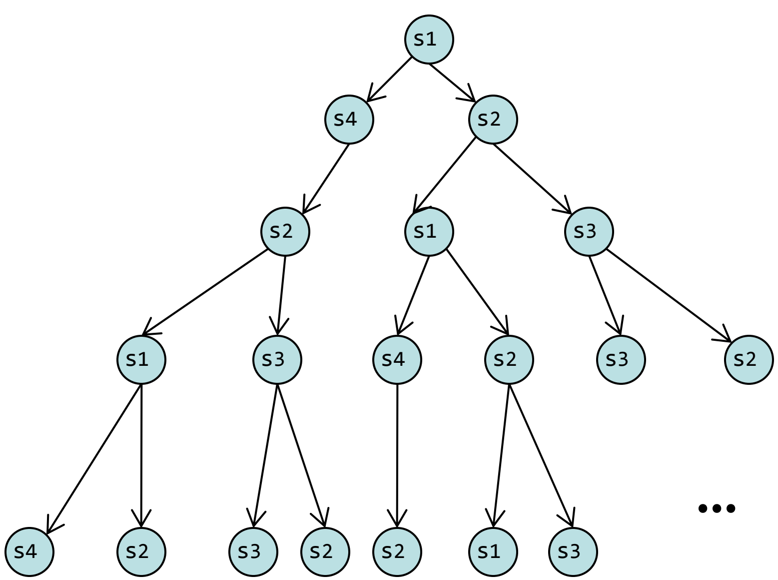
Relationship to Kripke structure

A Kripke structure represents a set of paths

- We want to establish the validity of a formula f under a Kripke structure M and a start state s

problem:

- formula is defined for a path, Kripke structure has many paths



CTL* Logic

Add two extra path quantifiers

- A f := for all paths, f
- E f := for some path, f

Two important subsets:

- LTL : all formulas of the form A f
  - Ex: A(FG p)
- CTL: there must be a path quantifier before every linear operator
  - Ex: AG (EF p)
- The two are different!

Example:

What does the following formula mean

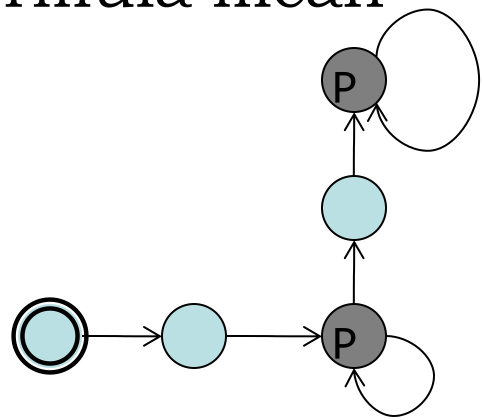
- A(F G p)

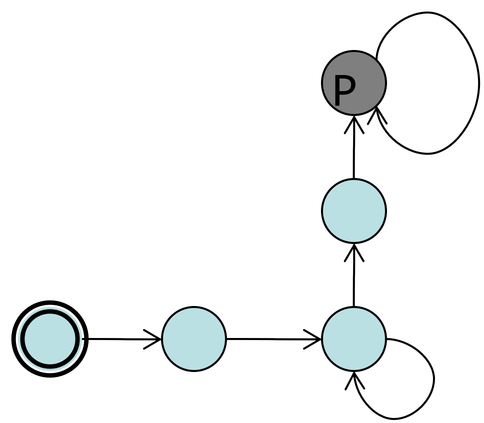
How about

- A(F A G p)

How about

- A(F E G p)

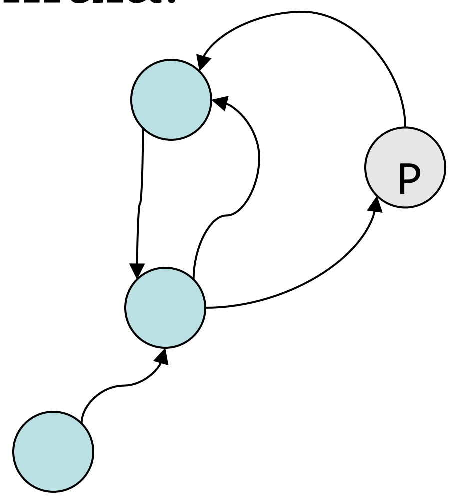




Review of Temporal Logic

What about the following formula:

- AG EF p



Review of Temporal Logic

What does the following formula mean

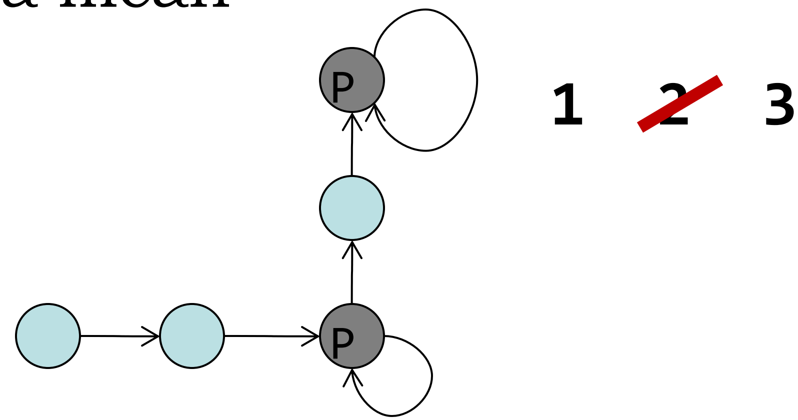
1) A(F G p)

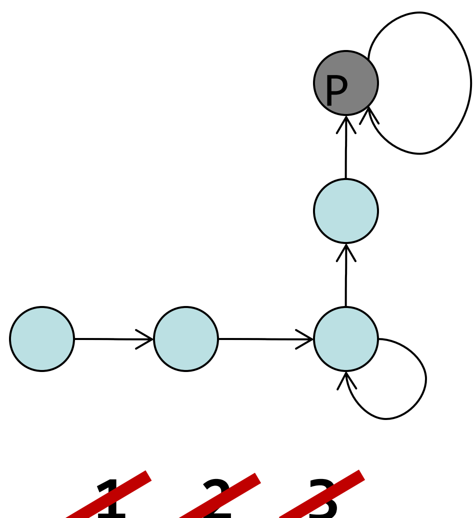
How about

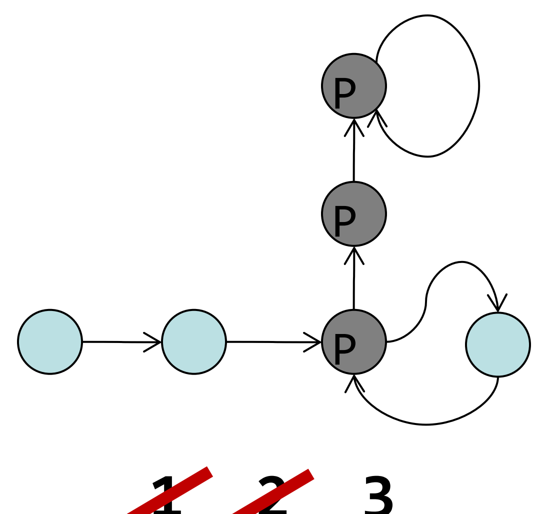
2) A(F A G p)

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

```
on, off, push, id
```

$$G\ off \Rightarrow off\ U\ push$$

$$G\ (off \Rightarrow (off\ U\ push \vee G\ off))$$

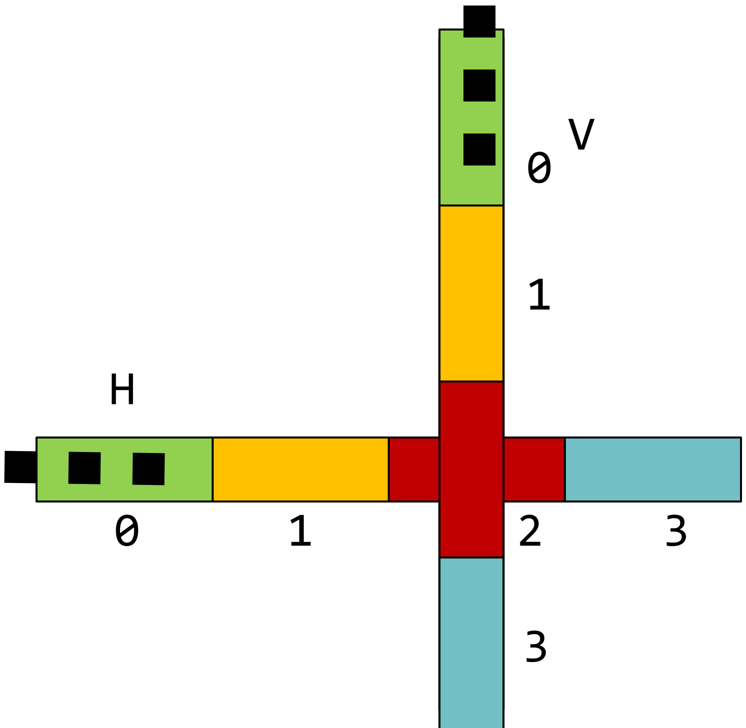
$$G\ (on \Rightarrow (on\ U\ push \vee G\ on))$$

$$AG\ (on \Rightarrow EF\ off)$$

$$AG\ (off \Rightarrow (EF\ on)\ \wedge A((off U id) \vee G off))$$

$$A\big((off U id) \vee G off\big) \equiv \neg E(\neg id U(\neg off \wedge \neg\, id))$$

Can the trains collide? $\neg F\ (ph = 2 \wedge pv = 2)$



ph={0,1,2,3}

pv={0,1,2,3}

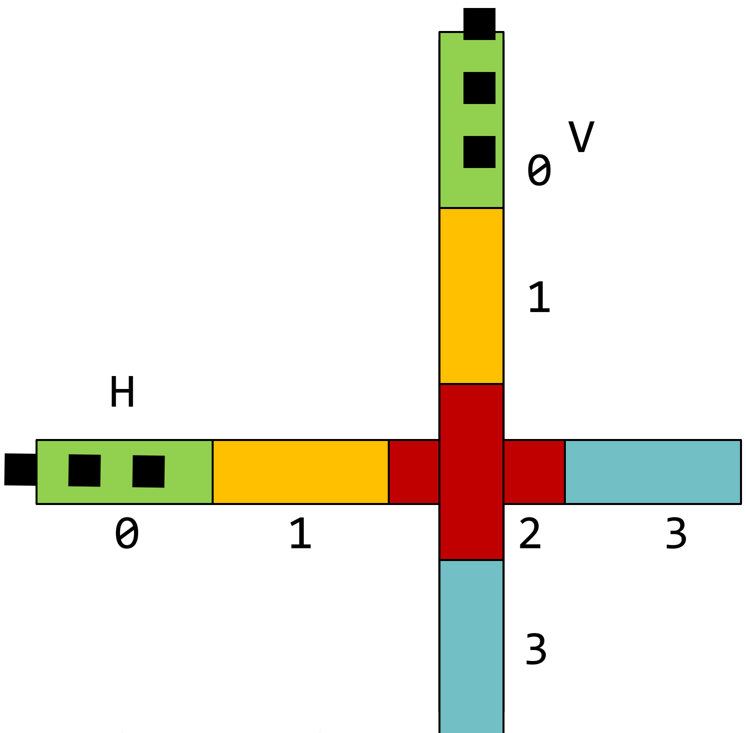
g={h, v, free}

pch={0, 1, ..., 9}

pcv={0, 1, ..., 9}

```
            while(*){
pc=0          if(p=0){
pc=1             p:=1;
              }
pc=2          if(p=1){
pc=3            if(g=free){
pc=4              g:=id;
pc=5              p:=2;
                }
              }
pc=6          if(p=2){
pc=7            p:=3; g:=free
              }
pc=8          if(p=3){
pc=9            p:=0;
              }
            }
```

Can the trains collide? $\neg F\ (ph = 2 \wedge pv = 2)$




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

```
      while(*){
pc=0   if(p=0){
pc=1       p:=1;
       }
pc=2   if(p=1){
pc=3     if(g=free){
pc=4         g:=id;
pc=5         p:=2;
         }
       }
pc=6   if(p=2){
pc=7     p:=3; g:=free
       }
pc=8   if(p=3){
pc=9     p:=0;
       }
      }
```

H train

```
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;
  }
}
```

V train

Liveness Vs. Safety

Two terms you are likely to run into:

Safety:

- Something bad will never happen: $G\ \neg bad$
- If it fails to hold, it's easy to produce a witness

Liveness:

- Something good will eventually happen: $F\ good$
- 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

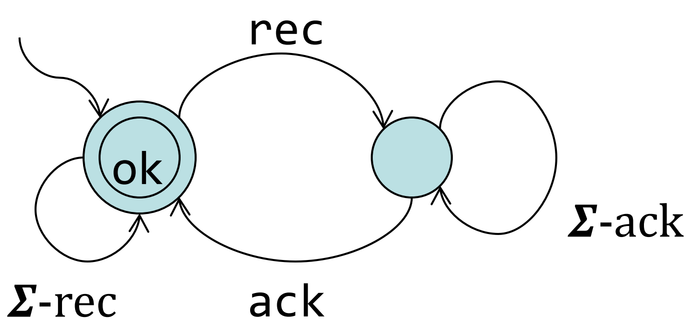
A Buchi Automaton is a 5-tuple $\langle \Sigma, S, I, \delta, F \rangle$

- Σ 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

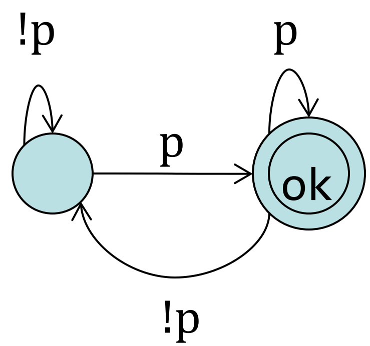
Example

G req → F ack



Example

G F p



From LTL to automata

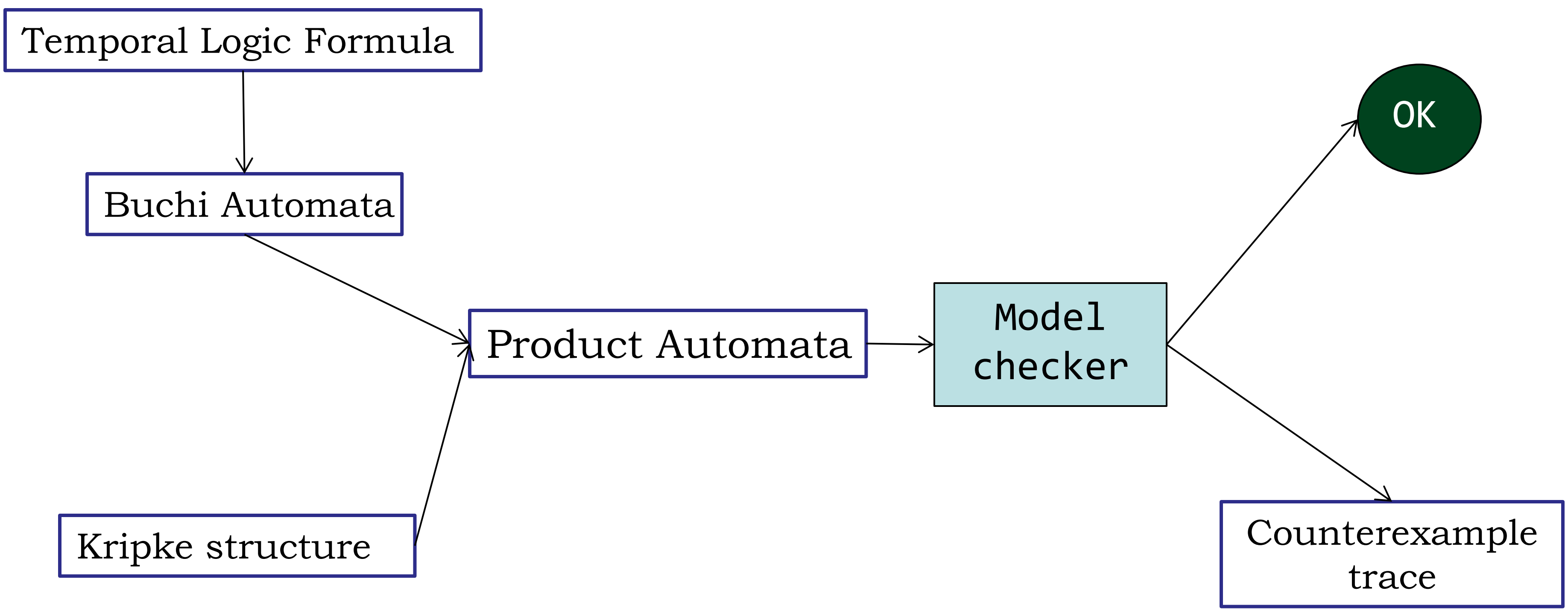
Any LTL formula can be expressed as a buchi automata

- but the construction of the automata is complicated
  - exponential on the size of the formula

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

Explicit State Model checking

The basic Strategy



6.820 Fundamentals of Program Analysis
Fall 2015