Automated Program Verification and Testing 15414/15614 Fall 2016 Lecture 20: Temporal Properties

Matt Fredrikson
mfredrik@cs.cmu.edu

November 11, 2016

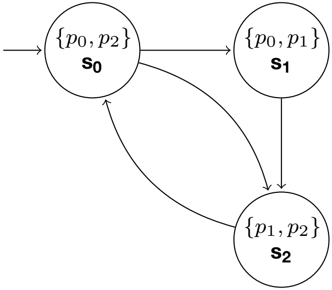
Modeling Computation (Review)

Computations are modeled using a *state transition graph*, also called a *Kripke structure*

Kripke structure

A Kripke structure $M = (P, S, I, L, R)$ consists of:

- Set of *atomic propositions* P
- States S
- Initial states $I \subseteq S$
- Labeling $L : S \mapsto 2^P$
- Transition relation $R \subseteq S \times S$



- $P = \{p_0, p_1, p_2\}$, $S = \{s_0, s_1, s_2\}$
- $I = \{s_0\}$
- $L = \{(s_0, \{p_0, p_2\}), \ldots\}$
- $R = \{(s_0, s_1), (s_1, s_2), \ldots\}$

Temporal Operators (Review)

There are five basic temporal operators we'll use

X p	p holds at the *next* point in time
F p	p holds at *some future* point in time
G p	p holds at *every point* in time
p **U** q	p holds *until* q holds
p **R** q	p *releases* q: q holds until p (if it ever does)

These operators describe properties of a path π

Linear Temporal Logic (Review)

These operators allow us to define **linear temporal logic** (LTL)

LTL contains **state formulas** and **path formulas**

State Formula

The syntax of state formulas is given by:

$$f ::= \quad \top \mid \bot \mid p \mid \neg f \mid f_1 \vee f_2 \mid f_1 \wedge f_2$$

State formulas correspond to facts that hold in a particular state.

Path Formula

An LTL formula is composed of the following elements:

$$g ::= \quad f \mid \neg g \mid g_1 \vee g_2 \mid g_1 \wedge g_2 \mid \mathbf{X}\ g \mid \mathbf{F}\ g \mid \mathbf{G}\ g \mid g_1\ \mathbf{U}\ g_2 \mid g_1\ \mathbf{R}\ g_s$$

Path formulas are evaluated along a particular path.

Path Quantifiers

We mentioned that LTL univerally quantifies over paths

It is also possible to have **existential** path quantification

We extend state formulas with two new forms:

- **E** f is now a state formula, where:

  $M, s \models \textbf{E}\ g \quad \Leftrightarrow \quad$ there exists a path π starting at s where $M, \pi \models g$

- **A** f is now a state formula, where:

  $$M, s \models \textbf{A}\ g \quad \Leftrightarrow \quad \text{for all paths } \pi \text{ starting at } s, M, \pi \models g$$

This defines the logic CTL*. LTL is a fragment of CTL*, where:

- Each formula implicitly starts with **A**
- State formulas don't have path quantifiers

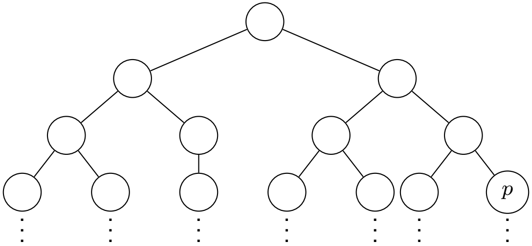
Branching Time

LTL formulas describe linear-time properties: single paths

CTL* formulas describe **branching-time** properties

Path quantifiers can talk about multiple **possible futures**

For example, consider **EF** p



Computation Tree Logic (CTL)

CTL is a fragment of CTL* where each temporal operator is preceded by a path quantifier

- If f, f_1, f_2 are **state** formulas, then **X** f, **F** f, **G** f, f_1 **U** f_2, and f_1 **R** f_2 are **path** formulas
- Importantly: path formulas are built directly from state formulas

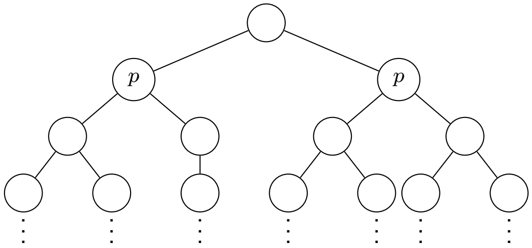
CTL Syntax

CTL formulas ϕ are built from the following grammar:

$$\begin{aligned}\phi ::= \quad & \top \mid \bot \mid p \in P \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \\ & \mid \textbf{EX}\ \phi \mid \textbf{EF}\ \phi \mid \textbf{EG}\ \phi \mid \textbf{E}\ [\phi_1\ \textbf{U}\ \phi_2] \mid \textbf{E}\ [\phi_1\ \textbf{R}\ \phi_2] \\ & \mid \textbf{AX}\ \phi \mid \textbf{AF}\ \phi \mid \textbf{AG}\ \phi \mid \textbf{A}\ [\phi_1\ \textbf{U}\ \phi_2] \mid \textbf{A}\ [\phi_1\ \textbf{R}\ \phi_2]\end{aligned}$$

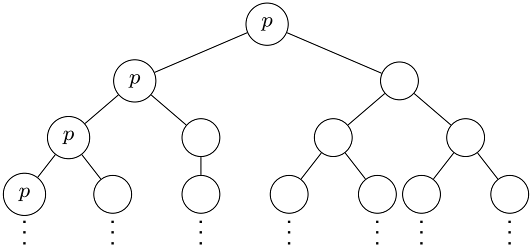
Example: Visualizing CTL (**AX** p)

What does **AX** p look like?



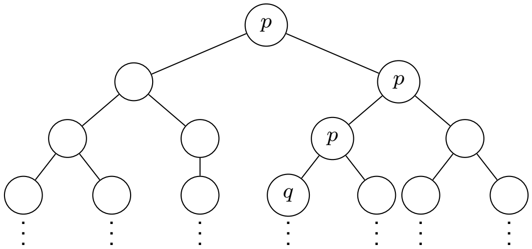
Example: Visualizing CTL (**EG** p)

What does **EG** p look like?



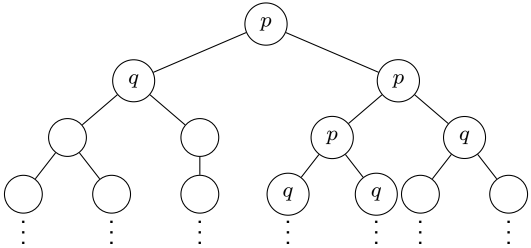
Example: Visualizing CTL (**E** $[p$ **U** $q]$)

What does **E** $[p$ **U** $q]$ look like?



Example: Visualizing CTL (**A** $[p$ **U** $q]$)

What does **A** $[p$ **U** $q]$ look like?



Example: CTL Properties

What do the following formulas mean?

- **EF** $($*start* $\wedge \neg$*ready*$)$

  Can get to a state where *start* holds by *ready* doesn't

- **AG** $($**EF** *restart*$)$

  It is always possible (from any state) to enter *restart*

- **AF** $($**AG** $p)$

  No matter what, p eventually always holds on all paths

- **AG** $($**AF** *crit*$)$

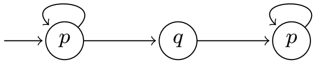
  On all paths at all times, enter *crit* infinitely often
  Or, *crit* is infintely often true on all paths

Expressiveness

Consider the last example from the previous slide:

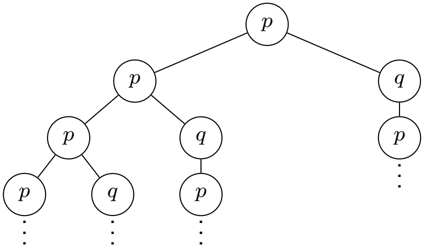
$$\textbf{AF}\ (\textbf{AG}\ p)$$

Couldn't we just write **F G** p? Consider the following:



$$\begin{tikzcd}
{} \arrow[r] & p \arrow[loop above] \arrow[r] & q \arrow[r] & p \arrow[loop above]
\end{tikzcd}$$

Does this satisfy **F G** p? **Yes.** What about **AF** $(\textbf{AG}\ p)$? **No.**



Expressiveness

Consider one of the examples from the previous slide:

$$\textbf{AG}\ (\textbf{EF}\ \textit{restart})$$

"It is always possible to enter *restart*."

Let's write this as an LTL formula

What about **G F** *restart*?

This says, "At all times, *restart* will eventually be entered".

It's too strong: we just need the possibility of returning to *restart*

This formula can't be expressed in LTL

Expressiveness: CTL vs. LTL

CTL and LTL are **incomparable** in terms of expressiveness

There are formulas in both logics that can't be expressed in the other

We've seen an example CTL formula with no LTL equivalent (**AG** $($**EF** $p))$

We've also seen an LTL formula with no equivalence in CTL:

$$\mathbf{F}\ \mathbf{G}\ p$$

These differences are not merely academic: both types of formulas are useful in practice

- **AG** $($**EF** $p)$ says that it is possible to reach a state with p regardless of the current state
- **F G** p says that a state invariant will hold in the future

Useful Temporal Properties

Typically, the properties we want to check fall into a few categories

- **Safety:** Properties which require that “nothing bad should ever happen”
- **Liveness:** Require that “something good always happens” in the future
- **Fairness:** Precludes “degenerate” infinite behaviors

These are inherently linear-time properties

Can often be approximated by branching properties

Safety: Invariants

An important class of safety properties describes **invariant** behavior

Invariants state that some property over states holds at all times

- Mutual exclusion
- Deadlock freedom
- Well-formedness of data structures

If ϕ is a propositional formula over P, then the LTL formula

$$\mathbf{G}\ \phi$$

is an invariant property

Example: Mutual Exclusion

Consider two processes P_1, P_2 of the form:

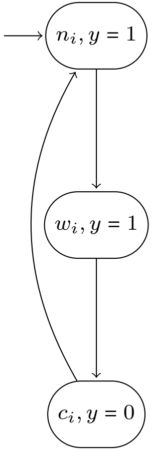
```
while (true) {
  ⋮
  (noncritical actions)
  ⋮
  while($y = 0$) skip;
  $y := y + 1$;
  critical section
  $y := y - 1$;
}
```

The variable y is **shared** between P_1 and P_2

We'll model each process using the predicates:

- c_i, that P_i is in its critical section
- n_i, that P_i is in non-critical section
- w_i, that P_i is waiting for $y = 0$
- $y = 0, y = 1$

Example: Mutual Exclusion


$$\begin{tikzcd}
\arrow[r] & \boxed{n_i, y = 1} \arrow[d] \\
& \boxed{w_i, y = 1} \arrow[d] \\
& \boxed{c_i, y = 0} \arrow[uu, bend left]
\end{tikzcd}$$

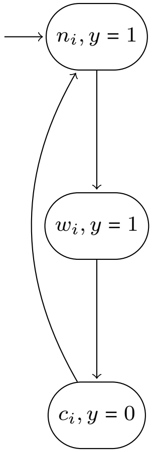
The variable y is **shared** between P_1 and P_2

We'll model each process using the predicates:

- c_i, that P_i is in its critical section
- n_i, that P_i is in non-critical section
- w_i, that P_i is waiting for $y = 0$
- $y = 0, y = 1$

Start by writing the TS for a single process

Modeling Concurrency



We'll model the concurrent execution of P_1, P_2

We'll create a **global** transition system

- Models state of both processes
- $P = \{c_1, c_2, n_1, n_2, w_1, w_2, y = 0, y = 1\}$
- Notice: shared state for $y = 0, y = 1$ is not duplicated
- Transitions account for all **interleavings**
- Intuitively, these are the outcome of nondeterministic choices between processes' steps

Example: Mutual Exclusion

$$\begin{tikzcd}
 & & {} \arrow[d] & & \\
 & & {n_1, n_1, y = 1} \arrow[dl] \arrow[dr] & & \\
 & {w_1, n_2, y = 1} \arrow[dl] \arrow[dr] & & {n_1, w_2, y = 1} \arrow[dl] \arrow[dr] & \\
{c_1, n_2, y = 0} \arrow[uurr, bend left] \arrow[dr] & & {w_1, w_2, y = 1} \arrow[dl] \arrow[dr] & & {n_1, c_2, y = 0} \arrow[uull, bend right] \arrow[dl] \\
 & {c_1, w_2, y = 0} \arrow[uurr, bend right] & & {w_1, c_2, y = 0} \arrow[uull, bend left] &
\end{tikzcd}$$

Example: Mutual Exclusion Formula

Recall that mutual exclusion is an invariant

$$\mathbf{G}\ \phi$$

What is ϕ? P_1 and P_2 can't be simultaneously critical

$$\phi \Leftrightarrow \neg c_1 \vee \neg c_2$$

Does the transition system satisfy this property?

Yes. Easy to see by visual inspection in this case

Algorithmically, invariant checking is a reachability problem

Safety properties, beyond invariants

Violating an invariant is one type of “bad thing” that could happen

Generally, we’re interested in more than simple reachability

General safety properties are defined in terms of **bad path prefixes**

- Let ϕ be a property over paths
- Let $\hat{\sigma}$ be a prefix such that no path $\sigma = \hat{\sigma} \cdots$ satisfies ϕ
- Note: we require $\hat{\sigma}$ to be finite

ϕ is a safety property iff every path that violates ϕ has a bad prefix

Think of a bad prefix as a witness of a violating instance

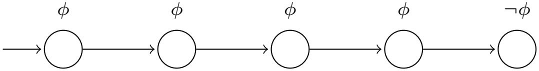
Bad prefixes: example

The "bad prefix" definition shows that invariants are safety properties

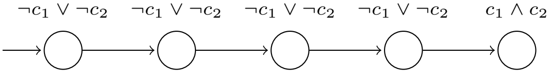
What are the bad prefixes of an invariant property?

$$\mathbf{G}\ \phi$$

Any finite sequence ending with $\neg\phi$:



What about in the mutual exclusion case?



Example: Safety Property

We want to model a property of a standard traffic light:

> "a red light must be preceeded immediately by a yellow"

What are the atomic propositions?

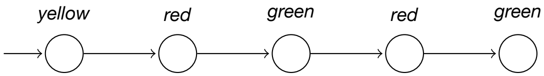
$$P = \{\mathit{red}, \mathit{yellow}, \mathit{green}\}$$

What is an LTL formula?

$$\mathbf{G}\ ((\mathit{yellow} \rightarrow \mathbf{X}\ \mathit{red}) \wedge ((\mathit{green} \vee \mathit{red}) \rightarrow \neg\mathbf{X}\ \mathit{red}))$$

Is this an invariant? **No.**

What is a bad prefix for this property?



Liveness Properties

Safety properties prevent bad things from happening

It’s easy to design systems that don’t do bad things: **do nothing**

Usually this isn’t what we want, and **liveness** properties address this

Intuitively, “something good” will always happen in the future

This intuition reveals a key distinction from safety properties:

- Safety properties are violated in **finite time**
- Liveness properties can only be violated in **infinite time**

Liveness, Formally

Liveness properties are defined in terms of **infinite path extensions**

- Let ϕ be a property over paths
- Let $\hat{\sigma}$ be a finite path prefix
- σ is an infinite path extension if $\hat{\sigma}\sigma$ satisfies ϕ

ϕ is a liveness property iff every finite prefix has an infinite extension

Put differently: a liveness property does not rule out any finite prefix

This implies that there are no finite witnesses for liveness

Liveness: Examples

Think of the mutual exclusion example

Desirable liveness property: each P_i enters *critical*$_i$ infinitely often

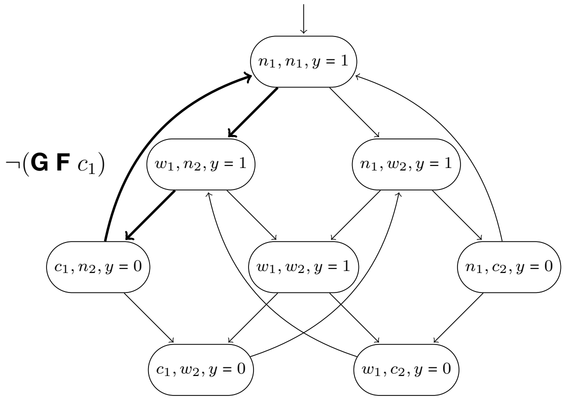
$$\textbf{G}\ \textbf{F}\ c_1 \wedge \textbf{G}\ \textbf{F}\ c_2$$

Another: each waiting process eventually enters its *critical*$_i$

$$\textbf{G}\ (w_i \rightarrow \textbf{F}\ c_i)$$

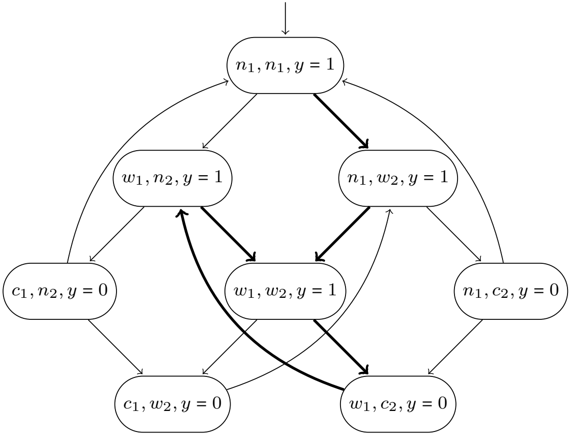
Any finite prefix can always be extended to satisfy these properties

Liveness: "Repeated Eventually"



Does this satisfy $\mathbf{G}\ \mathbf{F}\ c_1 \wedge \mathbf{G}\ \mathbf{F}\ c_2$? **No.** What's a counterexample?

Liveness: Starvation Freedom



Does this satisfy **G** $(w_i \rightarrow$ **F** $c_i)$? **No.** What's a counterexample?

Fairness

Liveness violated because processes aren't scheduled **fairly**

Unfairness arises from nondeterminism

- Choices that are globally biased against certain options
- Often these choices are unrealistic in practice
- Dealt with in the modelling, rather than checking, phase

To prove liveness, we want to rule out unfair paths from the model

Fairness constraints allow us to do this

Fairness Constraints

There are several different types of fairness

- **Unconditional fairness**: Every process is executed infinitely often.

  $$\text{for all } i, \mathbf{G}\ \mathbf{F}\ P_i$$

- **Strong fairness**: Every process that is enabled infinitely often gets its turn infinitely often when it is enabled

  $$\text{for all } i, (\mathbf{G}\ \mathbf{F}\ \textit{enabled}_i) \rightarrow (\mathbf{G}\ \mathbf{F}\ P_i)$$

- **Weak fairness**: Every process that is at some point continuously enabled gets its turn infinitely often

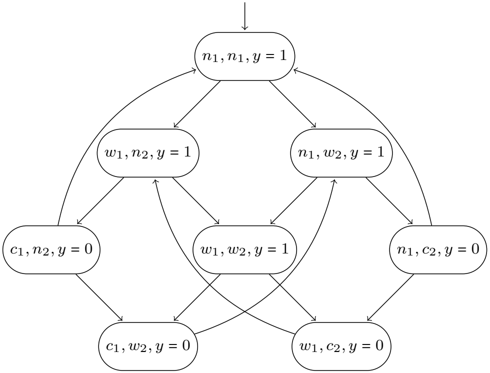
  $$\text{for all } i, (\mathbf{F}\ \mathbf{G}\ \textit{enabled}_i) \rightarrow (\mathbf{G}\ \mathbf{F}\ P_i)$$

“Enabled” means “able to execute” a particular transition

Example: Weak Fairness

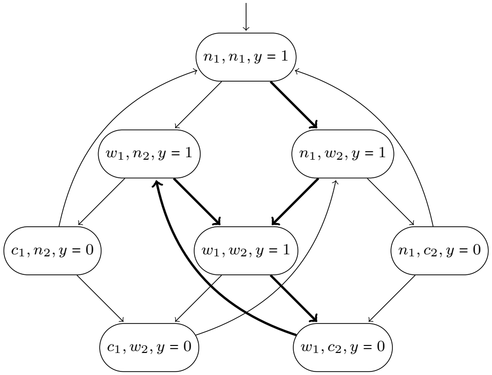
Consider the mutual exclusion example

A process is **enabled** if it is able to enter its critical section



Does this satisfy **G** $(w_i \rightarrow$ **F** $c_i)$ under weak fairness?

Example: Weak Fairness

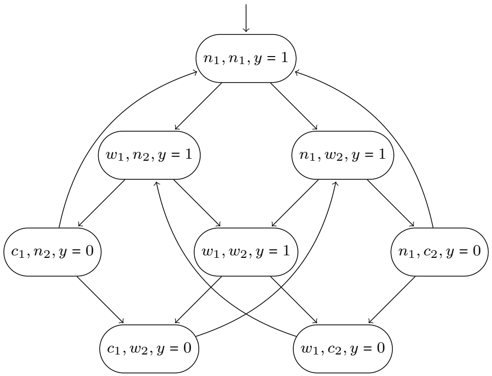


This doesn't satisfy starvation freedom under weak fairness

Whenever P_2 enters *critical*, P_1 is not enabled

The antecedent that P_i be enabled continuously is false

Example: Strong Fairness



Does this satisfy **G** $(w_i \rightarrow$ **F** $c_i)$ under strong fairness?

Yes, both processes enabled infinitely often along all paths

So a strongly-fair scheduler lets both execute infinitely often

Example: Unconditional Fairness

Does starvation freedom hold if the scheduler is unconditionally fair?

Unconditional Fairness : for all i, **G F** P_i

Yes. It held for strong fairness, which was:

Strong Fairness : for all i, (**G F** *enabled*$_i$) $\rightarrow$ (**G F** P_i)

So we see that unconditional fairness implies strong fairness

In fact, we can say that:

Unconditional Fairness $\Longrightarrow$ Strong Fairness $\Longrightarrow$ Weak Fairness

Next Lecture

- Continue discussing model checking
- Model checking techniques
- Fifth assignment goes out today
- Fourth assignment due tonight at 11pm