CIS 842: Specification and Verification of Reactive Systems

Lecture SPIN-Temporal-Logic:
Introduction to Temporal Logic

Objectives

- Understand why temporal logic can be a useful formalism for specifying properties of concurrent/reactive systems.
- Understand the intuition behind Computation Tree Logic (CTL) – the specification logic used e.g., in the well-known SMV model-checker.
- Be able to confidently apply Linear Temporal Logic (LTL) – the specification logic used in e.g., SPIN – to specify simple properties of systems.
- Understand the formal semantics of LTL.

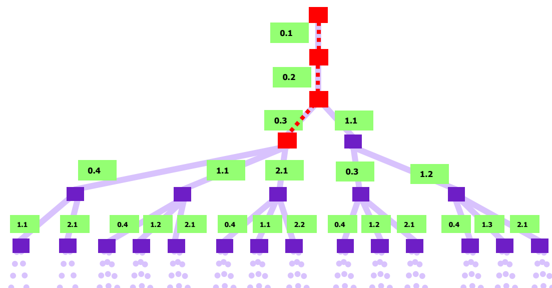
Outline

- CTL by example
- LTL by example
- Checking LTL specifications with SPIN
- LTL – formal definition
- Common properties to be stated for concurrent systems and how they can be specified using LTL

To Do

- Show never claims being generated from LTL formula
- For you to do's…

Reasoning about Executions



- We want to reason about execution trees
  - tree node = snap shot of the program's state
- Reasoning consists of two layers
  - defining predicates on the program states (control points, variable values)
  - expressing temporal relationships between those predicates

Why Use Temporal Logic?

- Requirements of concurrent, distributed, and reactive systems are often phrased as constraints on *sequences of events or states* or constraints on *execution paths.*
- Temporal logic provides a formal, expressive, and compact notation for realizing such requirements.
- The temporal logics we consider are also strongly tied to various computational frameworks (e.g., automata theory) which provides a foundation for building verification tools.

Computational Tree Logic (CTL)

Syntax

```
Φ ::=  P                                               ...primitive propositions
    | !Φ  | Φ && Φ | Φ || Φ | Φ -> Φ                   ...propositional connectives
    | AG Φ | EG Φ | AF Φ | EF Φ                        ...temporal operators
    | AX Φ | EX Φ | A[Φ U Φ] | E[Φ U Φ]
```

Computational Tree Logic (CTL)

Syntax

```
Φ ::=  P                                               ...primitive propositions
    | !Φ  | Φ && Φ | Φ || Φ | Φ -> Φ                   ...propositional connectives
    | AG Φ | EG Φ | AF Φ | EF Φ                        ...temporal operators
    | AX Φ | EX Φ | A[Φ U Φ] | E[Φ U Φ]
```


Semantic Intuition

path quantifier

temporal operator

AG p …along *A*ll paths p holds *G*lobally

EG p …there *Exists* a path where p holds *G*lobally

AF p …along *A*ll paths p holds at some state in the *Future*

EF p …there *Exists* a path where p holds at some state in the *Future*

Computational Tree Logic (CTL)

Syntax

```
Φ ::=  P                                              ...primitive propositions
    | !Φ  | Φ && Φ | Φ || Φ | Φ -> Φ                 ...propositional connectives
    | AG Φ | EG Φ | AF Φ | EF Φ                        ...temporal operators
    | AX Φ | EX Φ | A[Φ U Φ] | E[Φ U Φ]
```

Semantic Intuition

AX p …along *All* paths, p holds in the *neXt* state

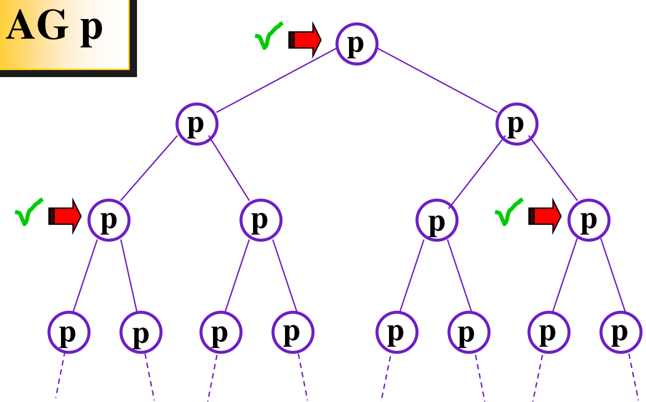
EX p …there *Exists* a path where p holds in the *neXt* state

A[p U q] …along *All* paths, p holds *Until* q holds

E[p U q] …there *Exists* a path where p holds *Until* q holds

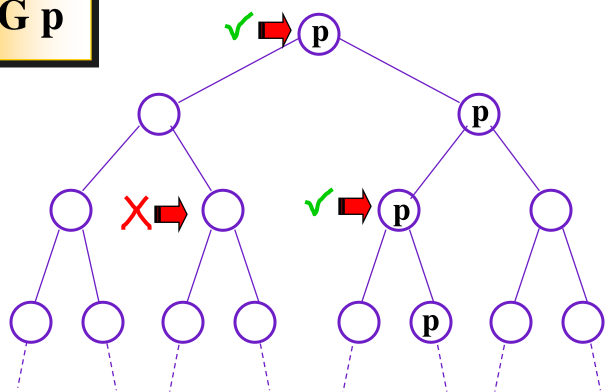
Computation Tree Logic

AG p



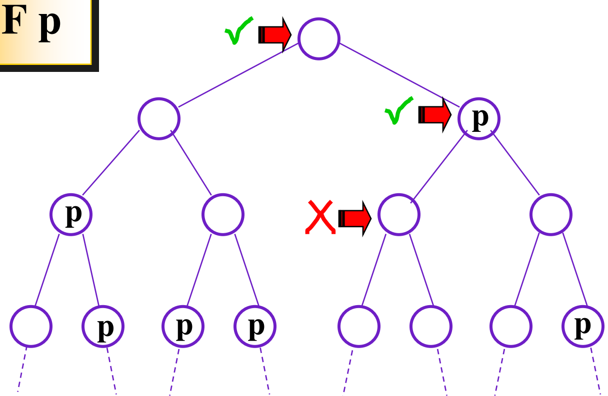
Computation Tree Logic

EG p



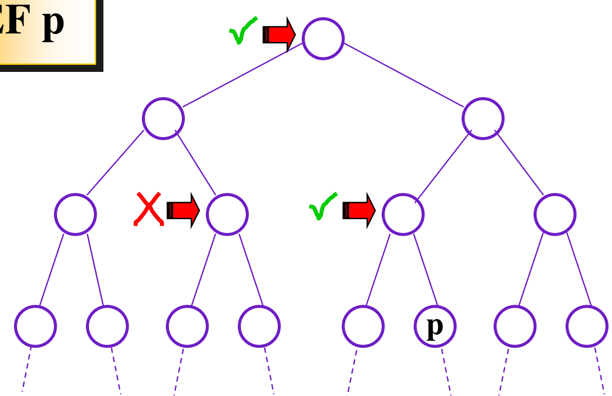
Computation Tree Logic

AF p



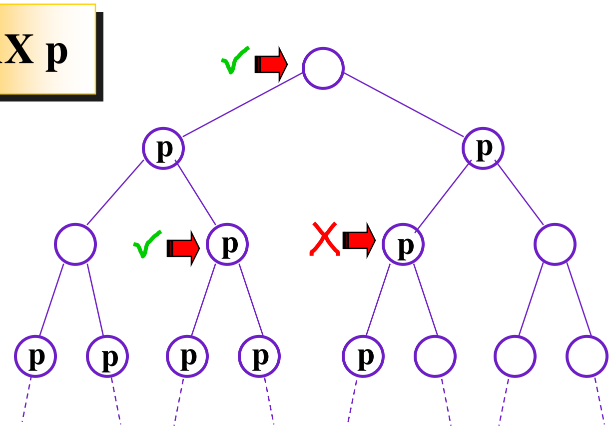
Computation Tree Logic

EF p



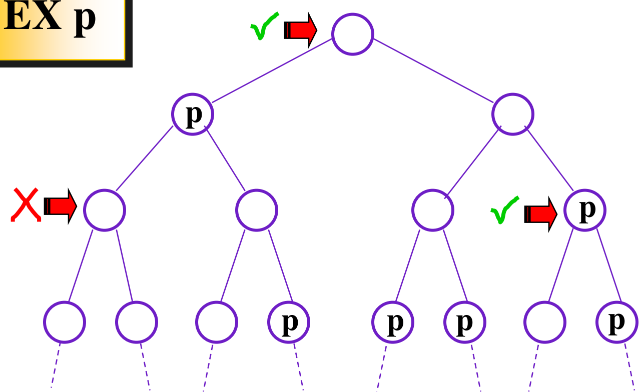
Computation Tree Logic

AX p



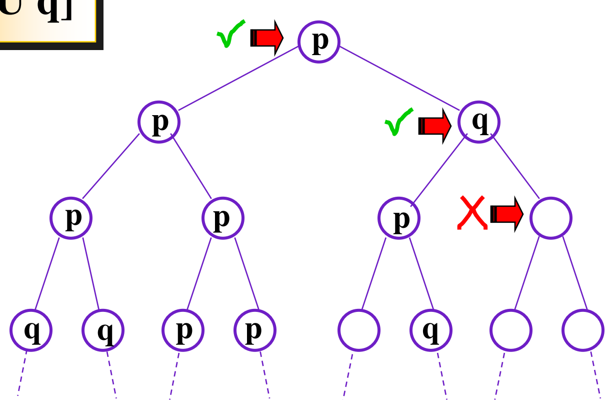
Computation Tree Logic

EX p



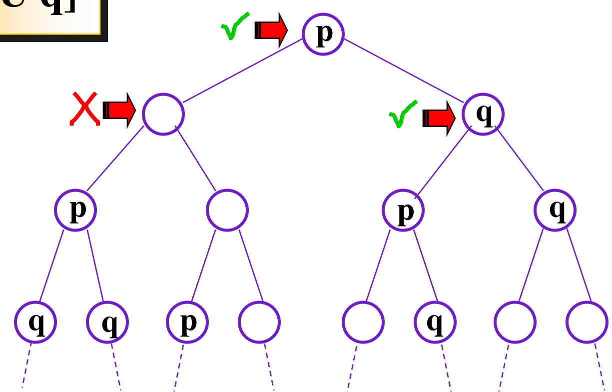
Computation Tree Logic

A[p U q]



Computation Tree Logic

E[p U q]



Example CTL Specifications

For any state, a request (e.g., for some resource) will eventually be acknowledged

```
AG(requested -> AF acknowledged)
```

Example CTL Specifications

From any state, it is possible to get to a restart state

```
AG(EF restart)
```

Example CTL Specifications

An upwards travelling elevator at the second floor does not changes its direction when it has passengers waiting to go to the fifth floor

```
AG((floor=2 && direction=up && button5pressed)
      -> A[direction=up U floor=5])
```

Semantics for CTL (excerpts)

- For $p \in AP$:
  $s \models p \Leftrightarrow p \in L(s)$ $s \models \neg p \Leftrightarrow p \notin L(s)$
- $s \models f \wedge g \Leftrightarrow s \models f$ and $s \models g$
- $s \models f \vee g \Leftrightarrow s \models f$ or $s \models g$
- $s \models EXf \Leftrightarrow \exists \pi = s_0 s_1 \ldots$ from s: $s_1 \models f$
- $s \models E(f\ U\ g) \Leftrightarrow \exists \pi = s_0 s_1 \ldots$ from s
  $\exists j \geq 0$ [$s_j \models g$ and $\forall i : 0 \leq i < j$ [$s_i \models f$]]
- $s \models EGf \Leftrightarrow \exists \pi = s_0 s_1 \ldots$ from s $\forall i \geq 0$: $s_i \models f$

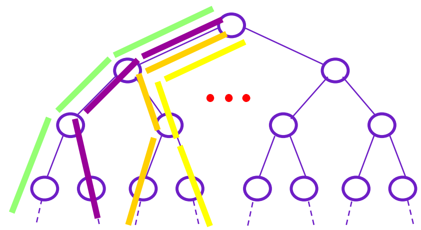
Source: Orna Grumberg

CTL Notes

- Invented by E. Clarke and E. A. Emerson (early 1980's)
- Specification language for Symbolic Model Verifier (SMV) model-checker
- SMV is a *symbolic* model-checker instead of an *explicit-state* model-checker
- Symbolic model-checking uses Binary Decision Diagrams (BDDs) to represent boolean functions (both transition system and specification

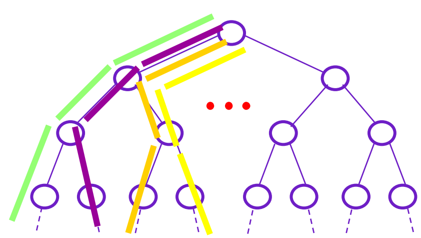
Linear Time Logic

Restrict path quantification to *"ALL" (no "EXISTS")*

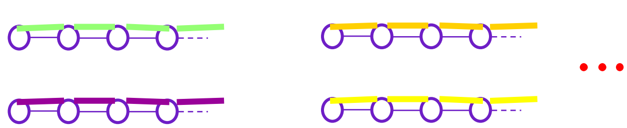


Linear Time Logic

Restrict path quantification to *"ALL" (no "EXISTS")*



Reason in terms of branching traces instead of branching trees



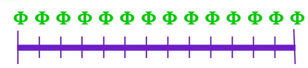
Linear Time Logic (LTL)

Syntax

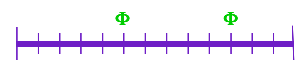
```
Φ ::=  P                                        ...primitive propositions
    | !Φ   | Φ && Φ | Φ || Φ | Φ -> Φ           ...propositional connectives
    | []Φ | <>Φ    | Φ U Φ   | X Φ              ...temporal operators
```

Semantic Intuition

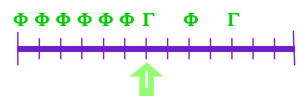
[]Φ ...*always* **Φ**



<>Φ ...*eventually* **Φ**

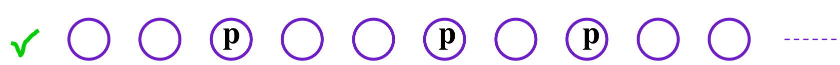


Φ U Γ ...**Φ** *until* **Γ**



Linear Time Logic

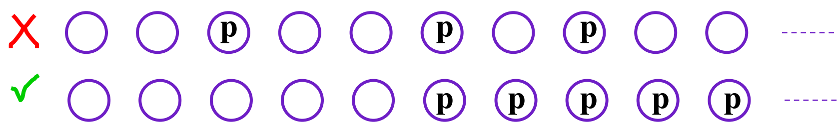
[]<>p



- "Along all paths, it must be the case that globally (I.e., in each state we come to) eventually p will hold"
- Expresses a form of fairness
  - p must occur infinitely often along the path
  - To check Φ under the assumption of fair traces, check []<>p -> Φ

Linear Time Logic

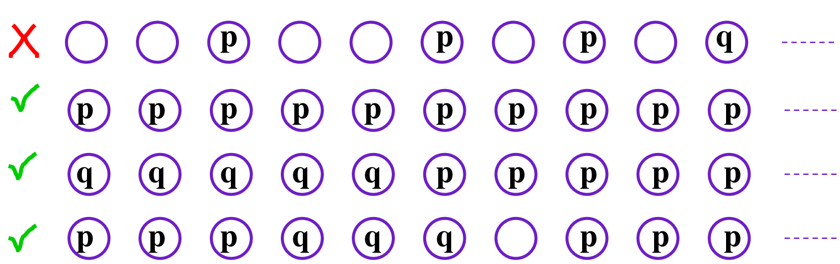
<>[]p



- "Along all paths, eventually it is the case that p holds at each state)" (i.e., "eventually permanently p")
- "Any path contains only finitely many !p states"

Linear Time Logic

p W q = **[]p || (p U q)**



- "p unless q", or "p waiting for q", or "p weak-until q"

Checking LTL Specs in SPIN

- Define the predicates/propositions using `#define` in `sys.prom` file (using lowercase letters to begin predicate names)
  - e.g., `#define bigx   x > 1000`
- Formalize requirement as an LTL formula
  - e.g. "eventually x is greater than 1000" becomes `<>(bigx)`
- Put the negation of the desired LTL property in file `req.ltl`
  - e.g., `(!<>(bigx))`
- Run SPIN to create a verifier based on the property
  - `spin -a -F req.ltl sys.prom`
- Compile
  - `gcc -o pan.exe pan.c`
- Run with command-line option (-a) specifying that a liveness property is being checked
  - `pan.exe -a`
- Display error trail
  - `spin -t sys.prom`

Semantics for LTL

- Semantics of LTL is given with respect to a (usually infinite) path or trace
  - $\pi = s_1\ s_2\ s_3 \ldots$
- We write π_i for the suffix starting at s_i, e.g.,
  - $\pi_3 = s_3\ s_4\ s_5 \ldots$
- A system satisfies an LTL formula f if each path through the system satisfies f.

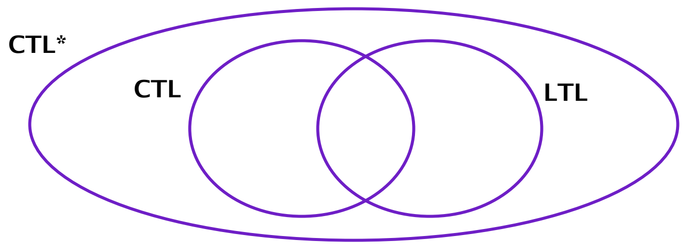
Semantics of LTL

- For $p \in AP$:
- $\pi \models p \Leftrightarrow p \in L(s_1)$ $\pi \models \neg p \Leftrightarrow p \notin L(s_1)$
- $\pi \models f \wedge g \Leftrightarrow \pi \models f$ and $\pi \models g$
- $\pi \models f \vee g \Leftrightarrow \pi \models f$ or $\pi \models g$
- $\pi \models Xf \Leftrightarrow \pi_2 \models f$
- $\pi \models \langle\rangle f \Leftrightarrow \exists i >= 1.\ \pi_i \models f$
- $\pi \models []f \Leftrightarrow \forall i >= 1.\ \pi_i \models f$
- $\pi \models (f\ U\ g) \Leftrightarrow \exists i >= 1.\ \pi_i \models g$
  and $\forall j : 1 \leq j < i\text{-}1.\ \pi_j \models f$

LTL Notes

- Invented by Prior (1960's), and first use to reason about concurrent systems by A. Pnueli, Z. Manna, etc.
- LTL model-checkers are usually explicit-state checkers due to connection between LTL and automata theory
- Most popular LTL-based checker is SPIN (G. Holzman)

Comparing LTL and CTL



- CTL is not strictly more expression than LTL (and vice versa)
- CTL* invented by Emerson and Halpern in 1986 to unify CTL and LTL
- We believe that almost all properties that one wants to express about software lie in intersection of LTL and CTL