Software Model Checking: Theory and Practice

Lecture: *Specification Checking - 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., Bogor and SPIN – to specify simple properties of systems.
- Understand the formal semantics of LTL.

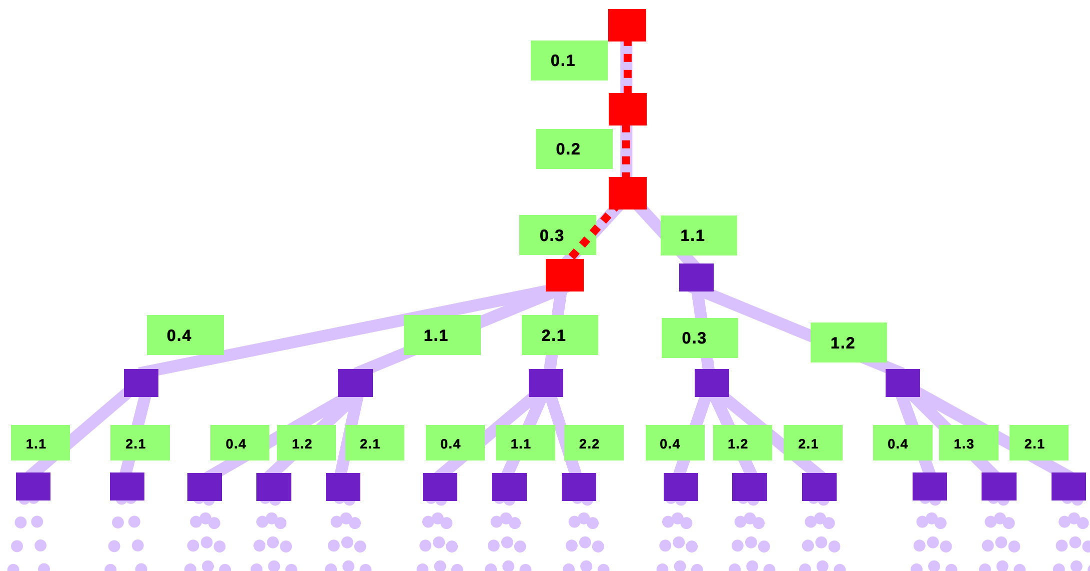
Outline

- CTL by example
- LTL by example
- LTL – formal definition
- Common properties to be stated for concurrent systems and how they can be specified using LTL
- Bogor’s support for 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 *All* paths p holds *Globally*

EG p …there *Exists* a path where p holds *Globally*

AF p …along *All* 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

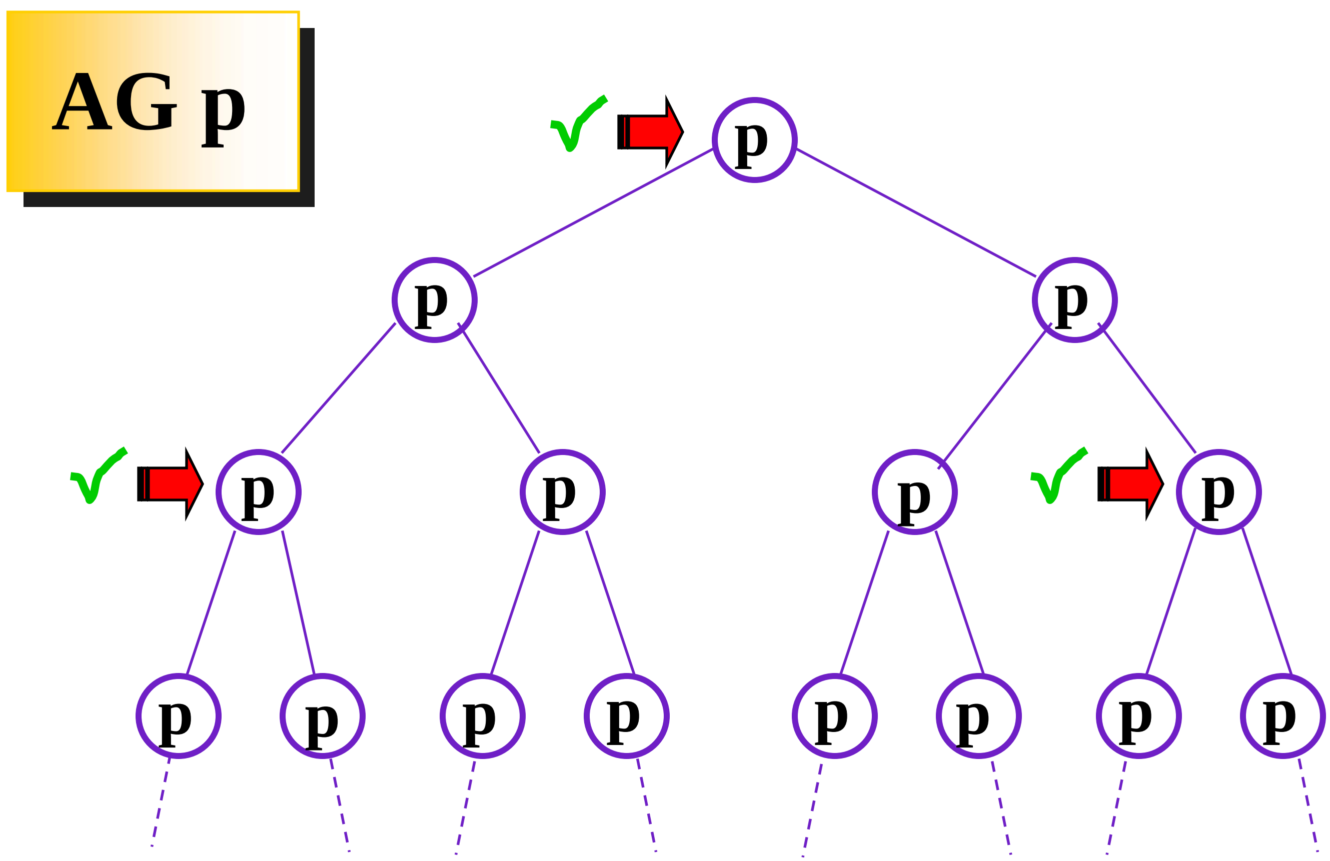
`AX p` …along *All* paths, p holds in the *neXt* state

`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

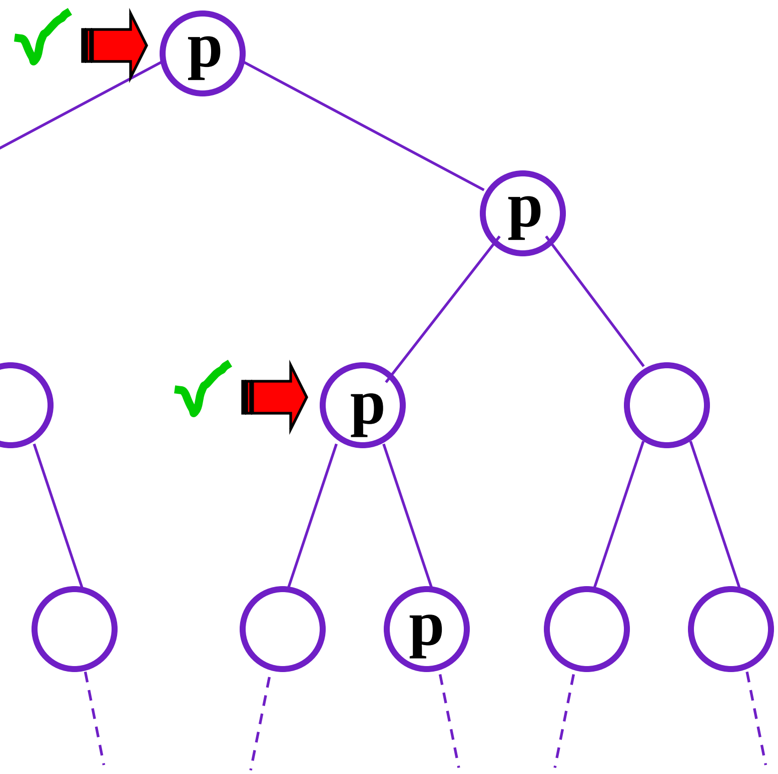
`E[p U q]` …there *Exists* a path where p holds *Until* q holds

Computation Tree Logic

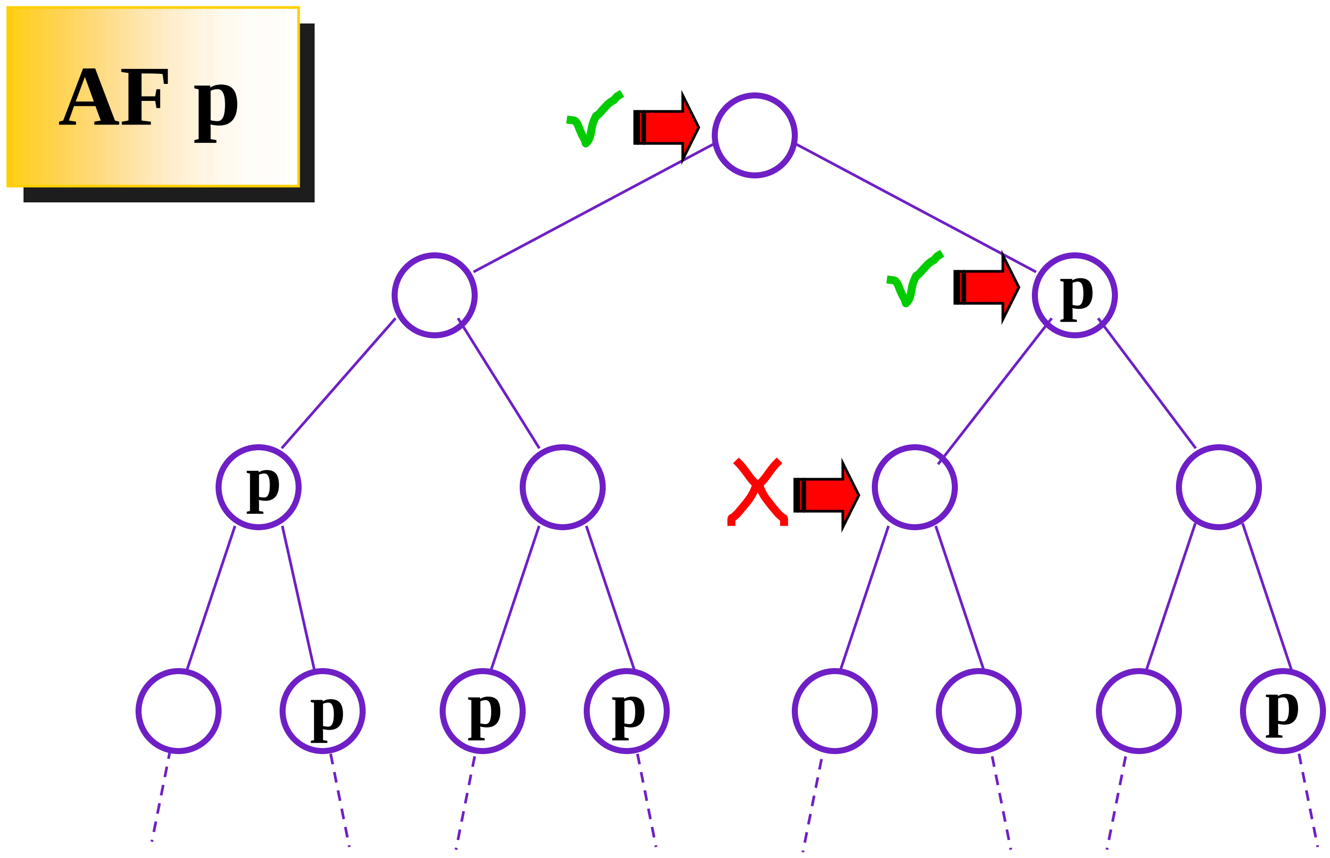


Computation Tree Logic

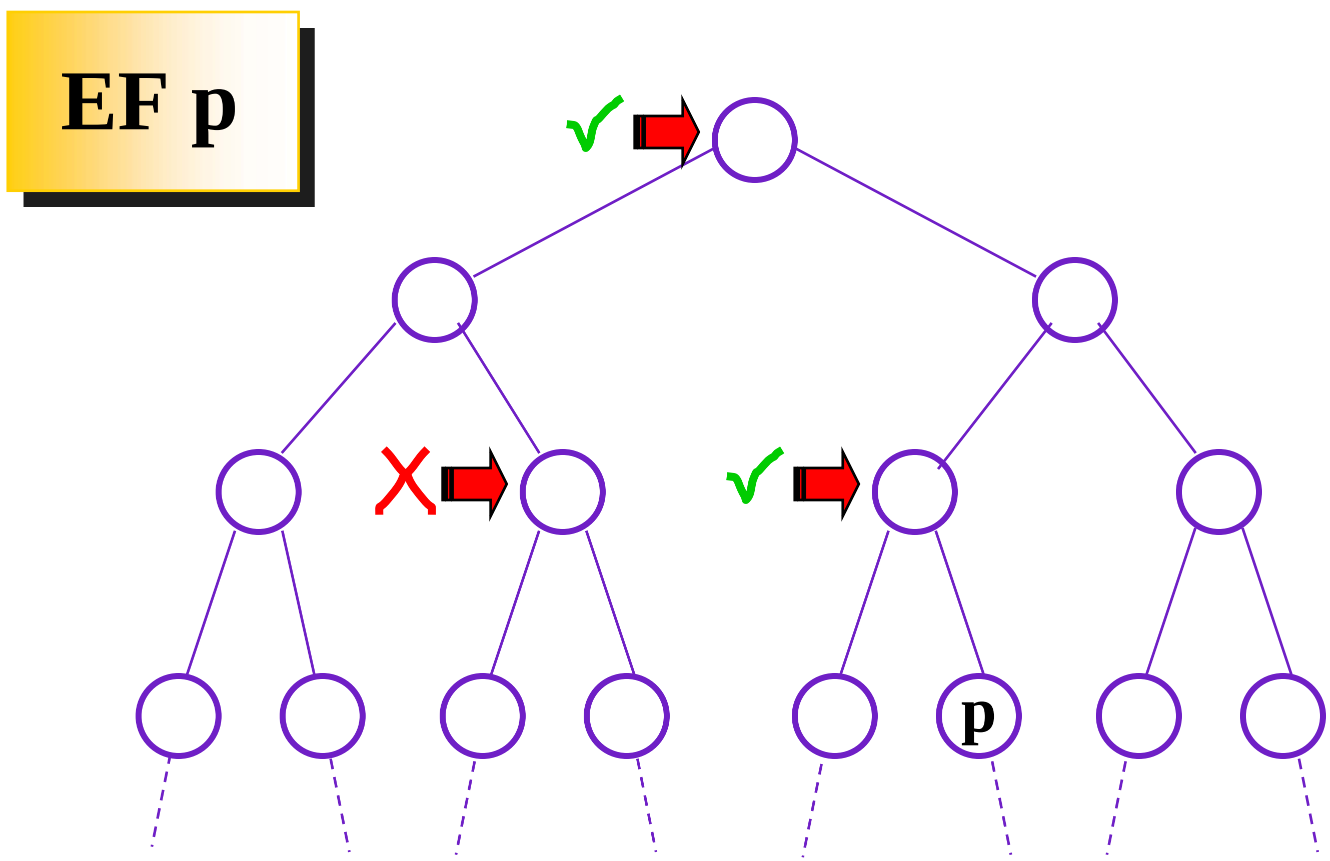
EG p



Computation Tree Logic

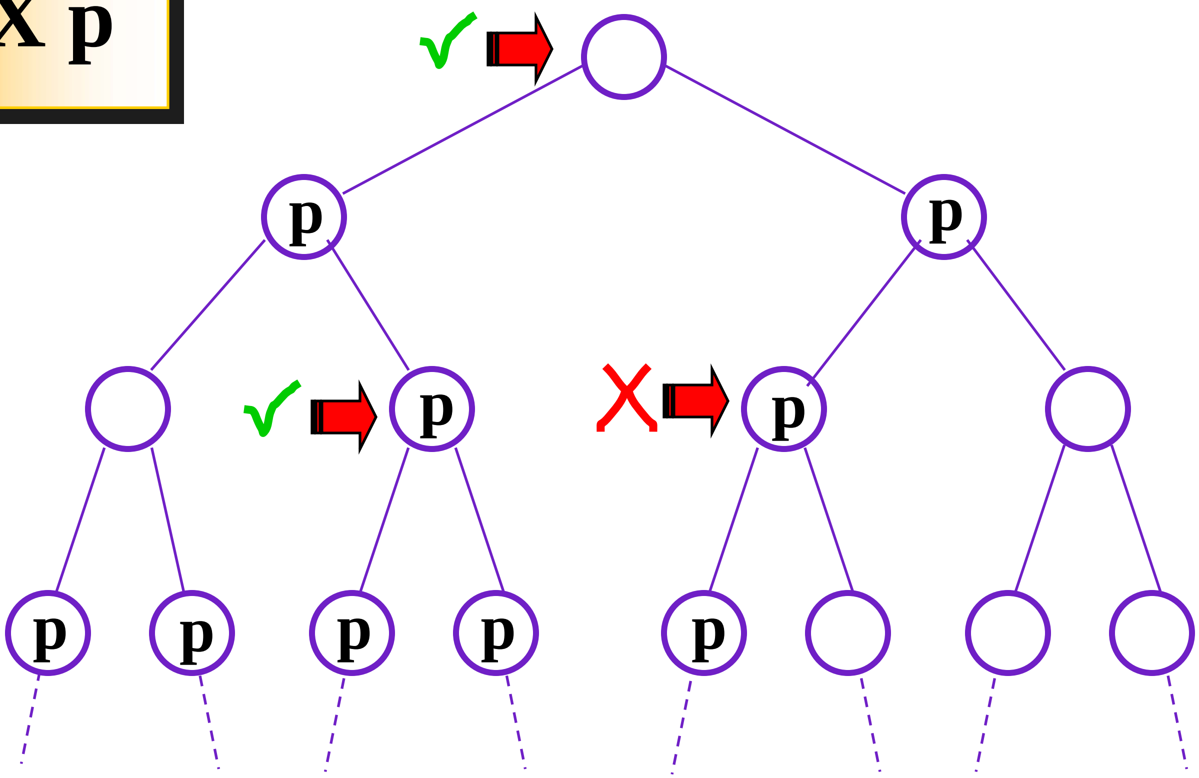


Computation Tree Logic

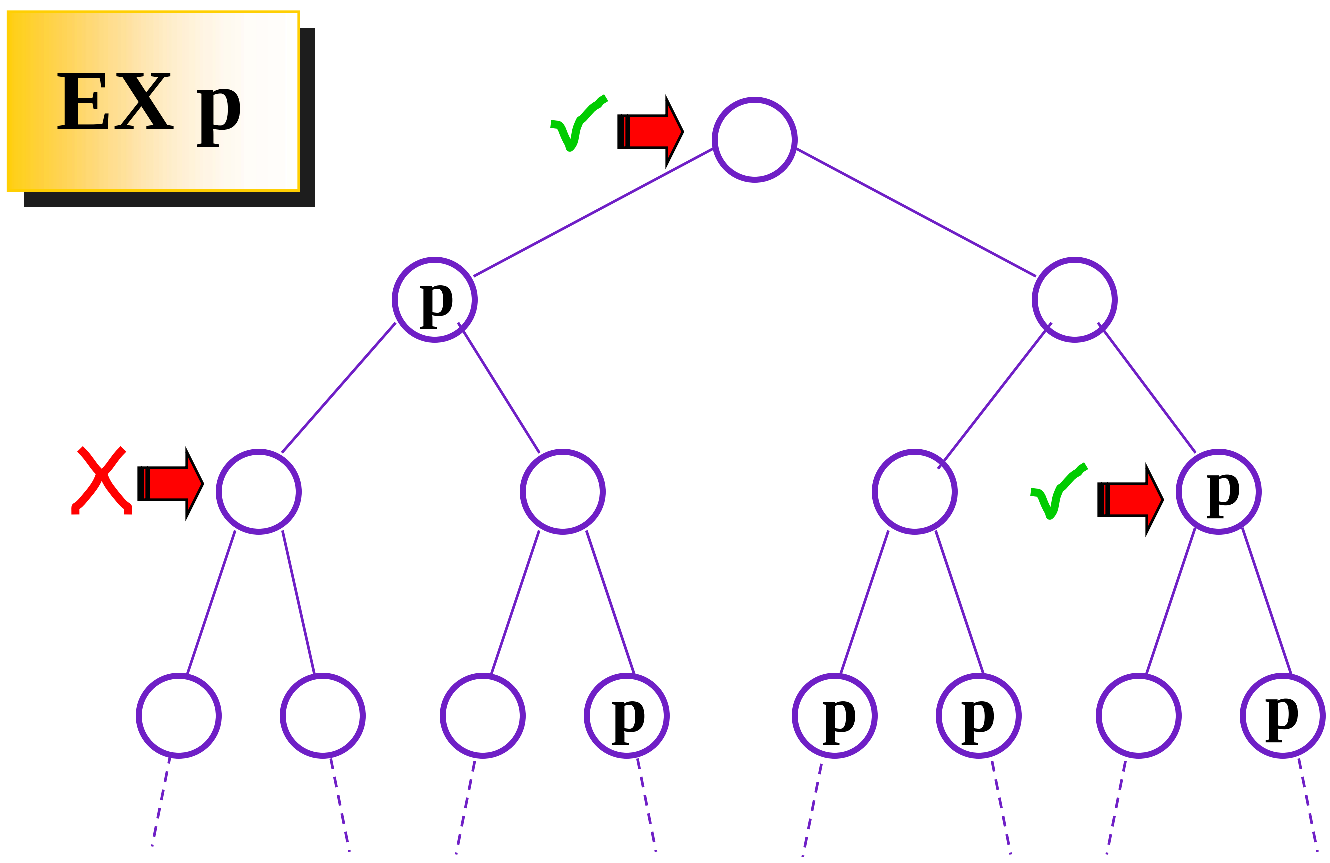


Computation Tree Logic

AX p

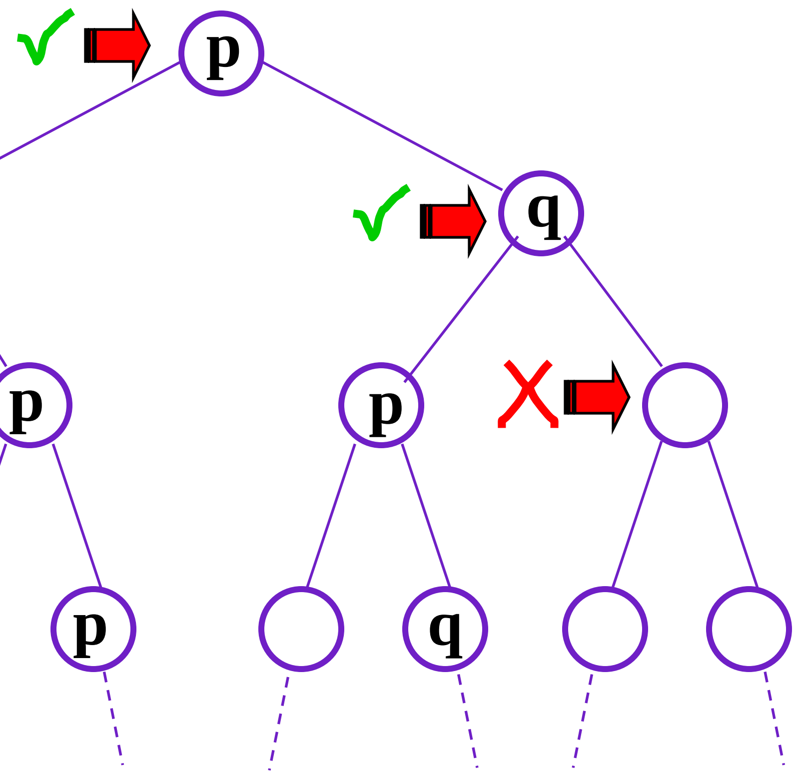


Computation Tree Logic



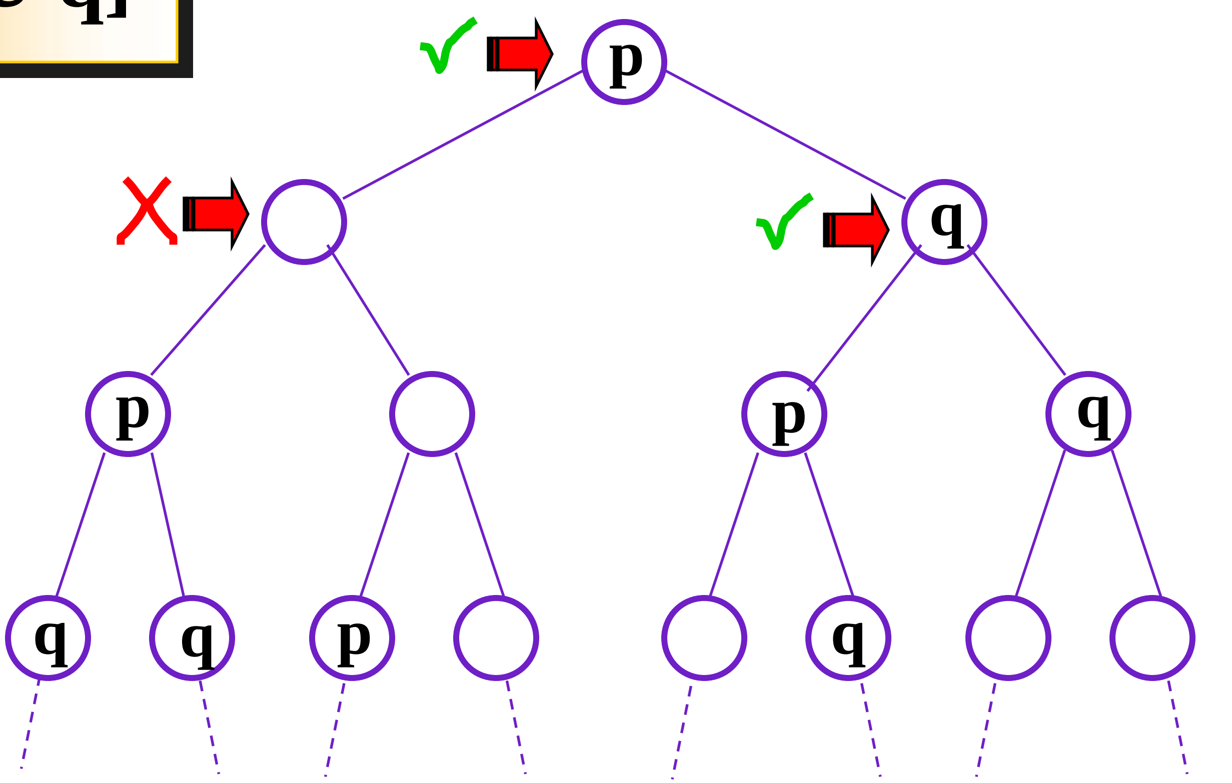
Computation Tree Logic

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 $\forall i \geq 0$: $s_i \models f$

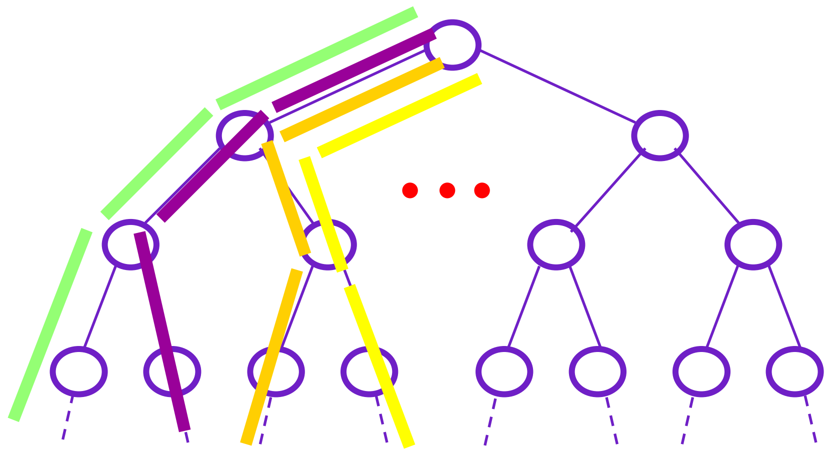
Source: Orna Grumberg

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

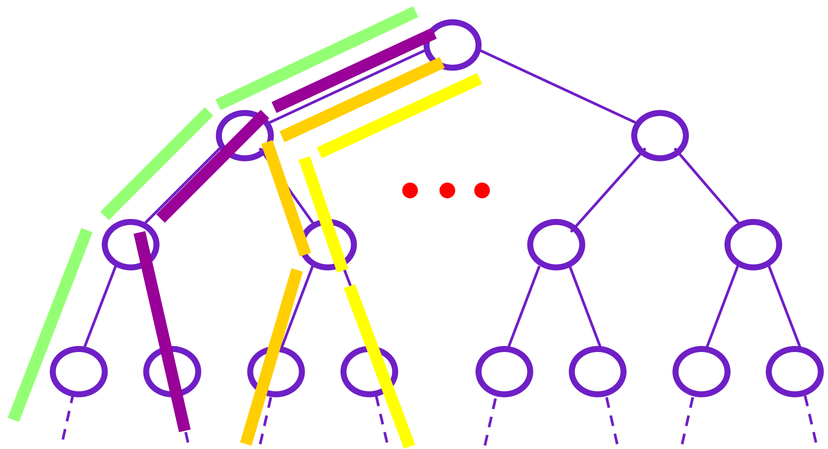
Linear Time Logic

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

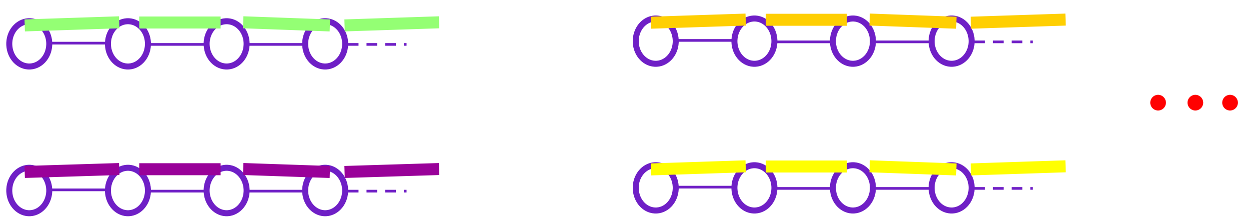


Linear Time Logic

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



Reason in terms of branching traces instead of branching trees



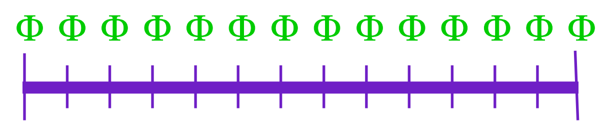
Linear Time Logic (LTL)

Syntax

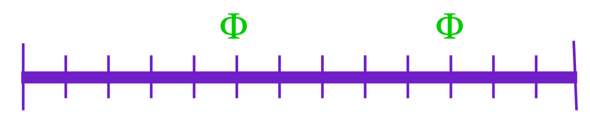
```
Φ ::=  P                                      …primitive propositions
    | !Φ  | Φ && Φ | Φ || Φ | Φ -> Φ          …propositional connectives
    | []Φ | <>Φ    | Φ U Φ  | X Φ             …temporal operators
```

Semantic Intuition

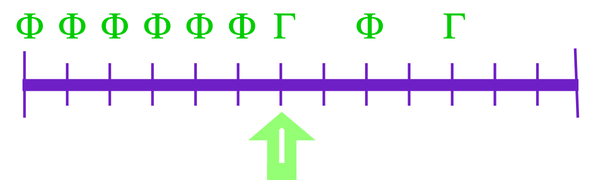
[]Φ ... *…always* Φ



<>Φ ... *…eventually* Φ



Φ **U** Γ ... *…*Φ *until* Γ



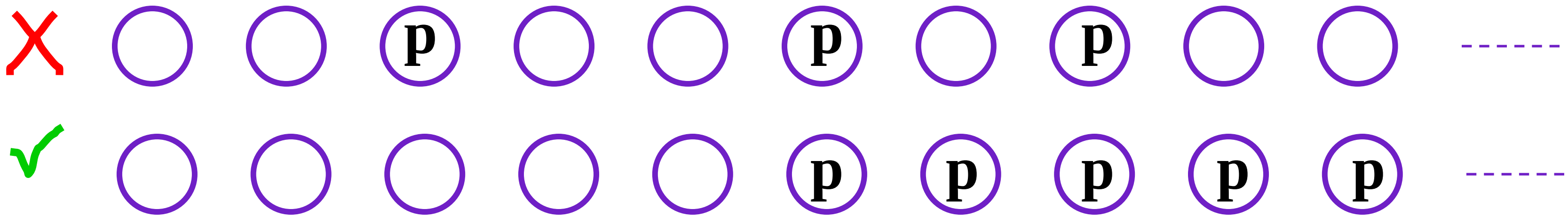
Linear Time Logic

[]<>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

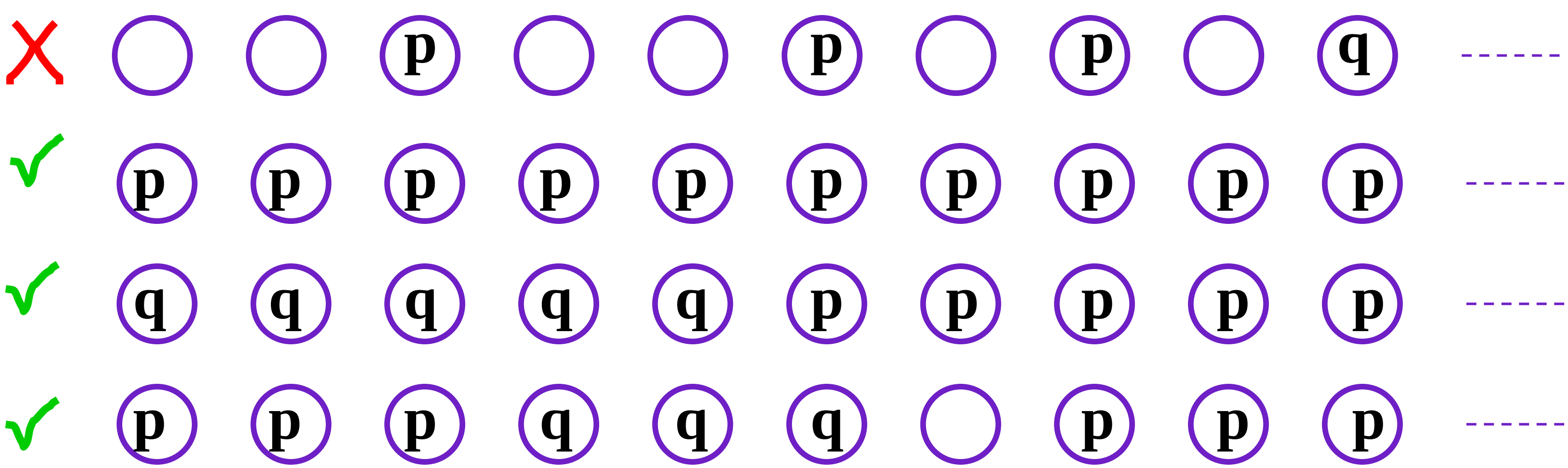
<>[]p



- "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”

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.

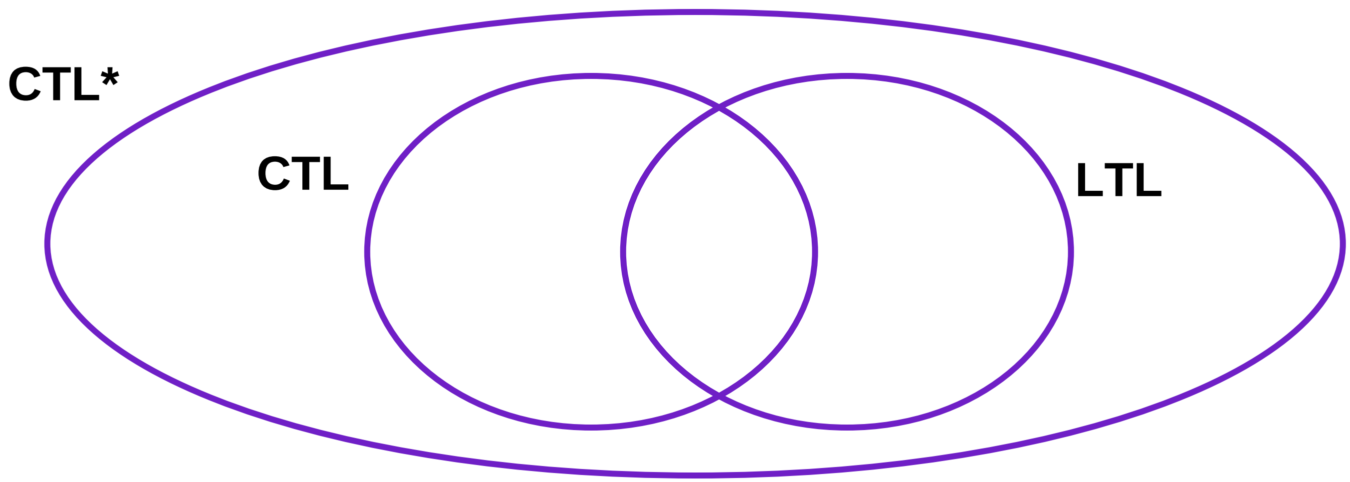
Semantics of LTL

- 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

Bogor Support

- As for regular properties, Bogor defines an extension for LTL properties
  - Property extension is the same
- LTL extension
  - Implemented by

  …bogor.module.property.ltl.LinearTemporalLogicModule
  - Supports
    - Atomic propositions and literals (e.g., true/false)
    - Propositional connectives (e.g., and, or)
    - Temporal operators (e.g., always, eventually)

LTL extension

```
extension LTL for edu.ksu.cis.projects.bogor.module.property.ltl.LinearTemporalLogicModule
   {
          typedef Formula;

          expdef LTL.Formula prop(string);
          expdef LTL.Formula literal(boolean);
          expdef LTL.Formula always(LTL.Formula);
          expdef LTL.Formula eventually(LTL.Formula);
          expdef LTL.Formula negation(LTL.Formula);
          expdef LTL.Formula until(LTL.Formula, LTL.Formula);
          expdef LTL.Formula release(LTL.Formula, LTL.Formula);
          expdef LTL.Formula equivalence(LTL.Formula, LTL.Formula);
          expdef LTL.Formula implication(LTL.Formula, LTL.Formula);
          expdef LTL.Formula conjunction(LTL.Formula, LTL.Formula);
          expdef LTL.Formula disjunction(LTL.Formula, LTL.Formula);

          expdef boolean temporalProperty(Property.ObservableDictionary,
                                                   LTL.Formula);
   }
```

An Example

Mutual exclusion in ReadersWriters

```
fun mutualExclusion() returns boolean
        = LTL.temporalProperty(
                    Property.createObservableDictionary(
                        Property.createObservableKey(
                            "someReading", activeReaders>0),
                        Property.createObservableKey(
                            "someWriting", activeWriters>0)
                    ),
             LTL.always(
                   LTL.implication(
                       LTL.prop("someReading"),
                       LTL.negation(LTL.prop("someWriting"))
                   )
             )
         );
```

Bogor Configuration

Use the defaults except for these settings

```
edu.ksu.cis.projects.bogor.module.IStateFactory=
   edu.ksu.cis.projects.bogor.module.property.fsa.FSAStateFactory

edu.ksu.cis.projects.bogor.ast.transform.ISystemTransformer=
   edu.ksu.cis.projects.bogor.module.property.ltl.LtlSystemTransformer

edu.ksu.cis.projects.bogor.module.ISearcher=
   edu.ksu.cis.projects.bogor.module.property.buechi.NestedFSASearcher

edu.ksu.cis.projects.bogor.module.IStateManager.stateAugmenter=
   edu.ksu.cis.projects.bogor.module.property.fsa.FSAStateAugmenter

ltlFunId=mutualExclusion
```