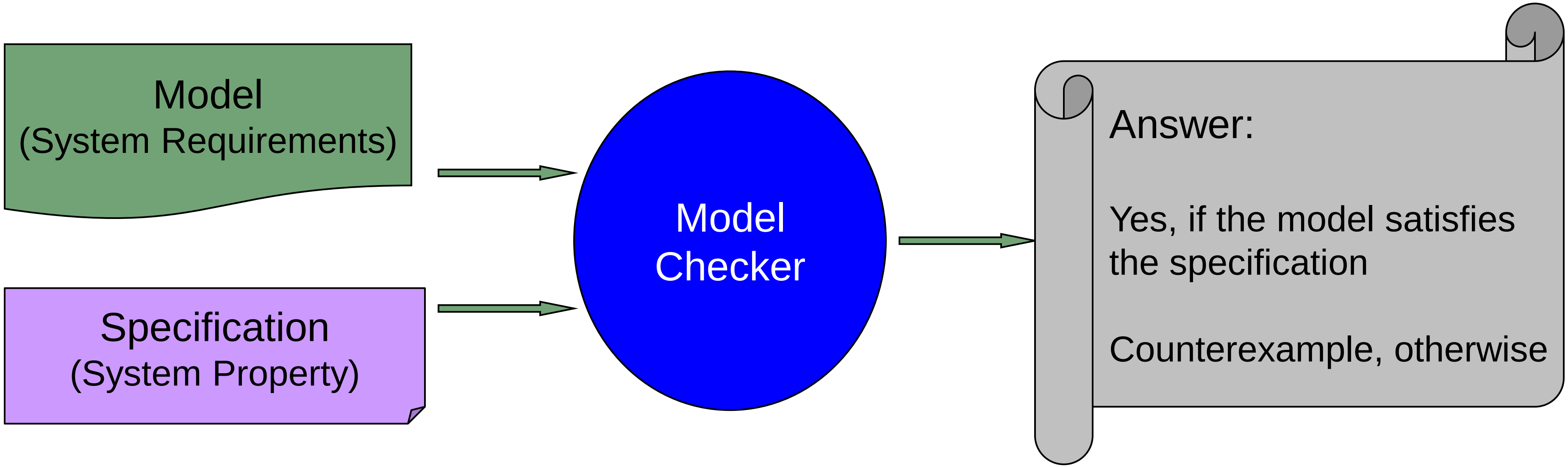
Model Checking

Principles


Model
(System Requirements)
Specification
(System Property)
Model
Checker
Answer:
Yes, if the model satisfies
the specification
Counterexample, otherwise

Kripke Model

- Kripke Structure + Labeling Function
  - Let AP be a non-empty set of atomic propositions.
  - Kripke Model: $M = (S, s_0, R, L)$

S	finite set of states
$s_0 \in S$	initial state
$R \subseteq S \times S$	transition relation
$L: S \rightarrow 2^{AP}$	labeling function

Specification

- Often expressed in temporal logic
  - Propositional logic with temporal aspect
  - Describes ordering of events without explicitly using the concept of time
  - Several variants: LTL, CTL, CTL*

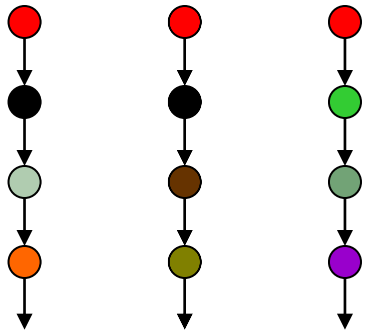
Why Use Temporal Logic?

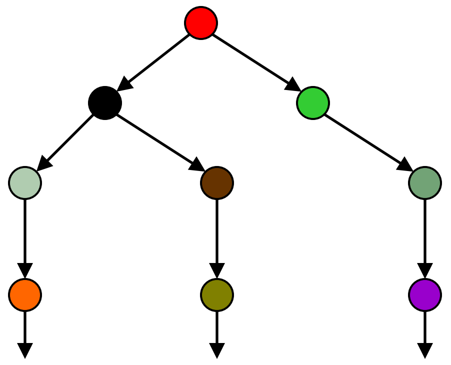
- Requirements of concurrent, distributed, and reactive systems are often phrased as constraints on *sequences of events or states* or constraints on *execution paths*.
- Temporal logic provides a formal, expressive, and compact notation for realizing such requirements.
- The temporal logics we consider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

Temporal Logics

- Express properties of event orderings in time

  - Linear Time
    - Every moment has a unique successor
    - Infinite sequences (words)
    - Linear Temporal Logic (LTL)
  - Branching Time
    - Every moment has several successors
    - Infinite tree
    - Computation Tree Logic (CTL)





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

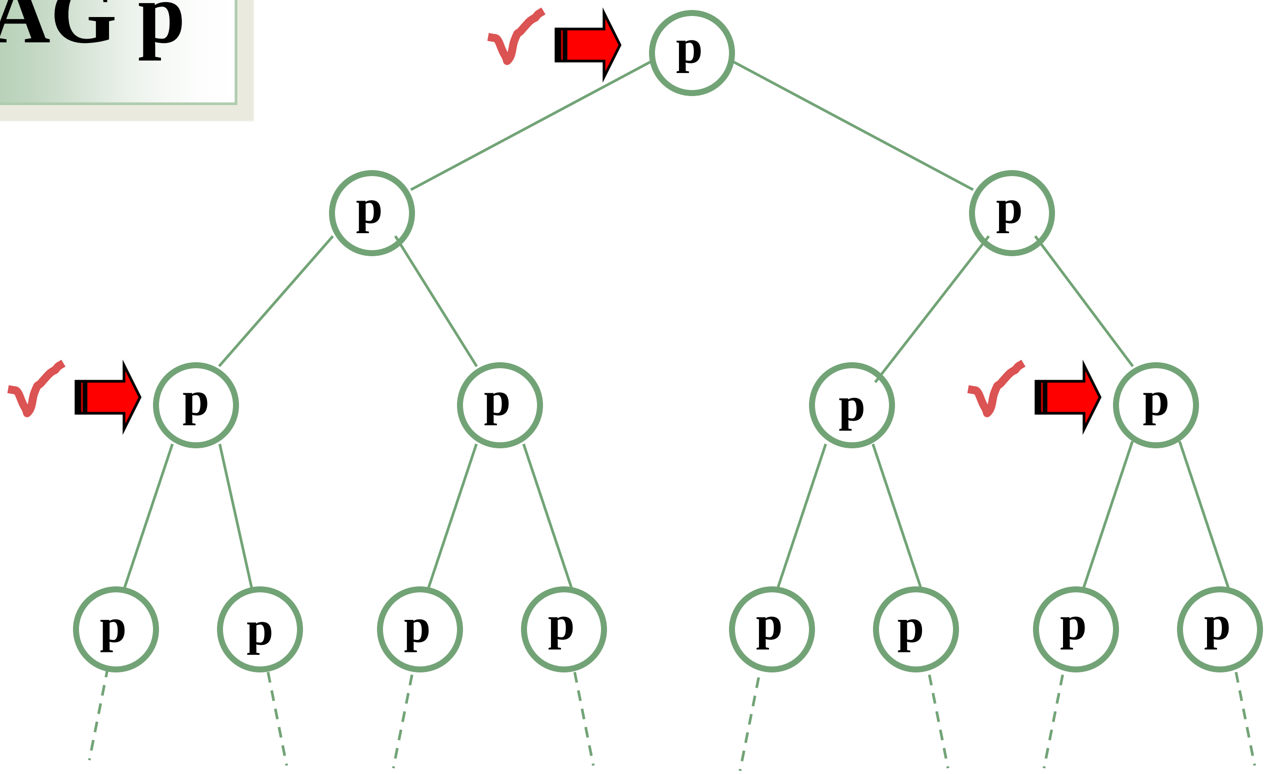
```
Φ ::=  P                                                  …primitive propositions
     | !Φ  | Φ && Φ | Φ || Φ | Φ -> Φ                     …propositional connectives
     | AG Φ | EG Φ | AF Φ | EF Φ                          …temporal operators
     | AX Φ | EX Φ | A[Φ U Φ] | E[Φ U Φ]
```

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

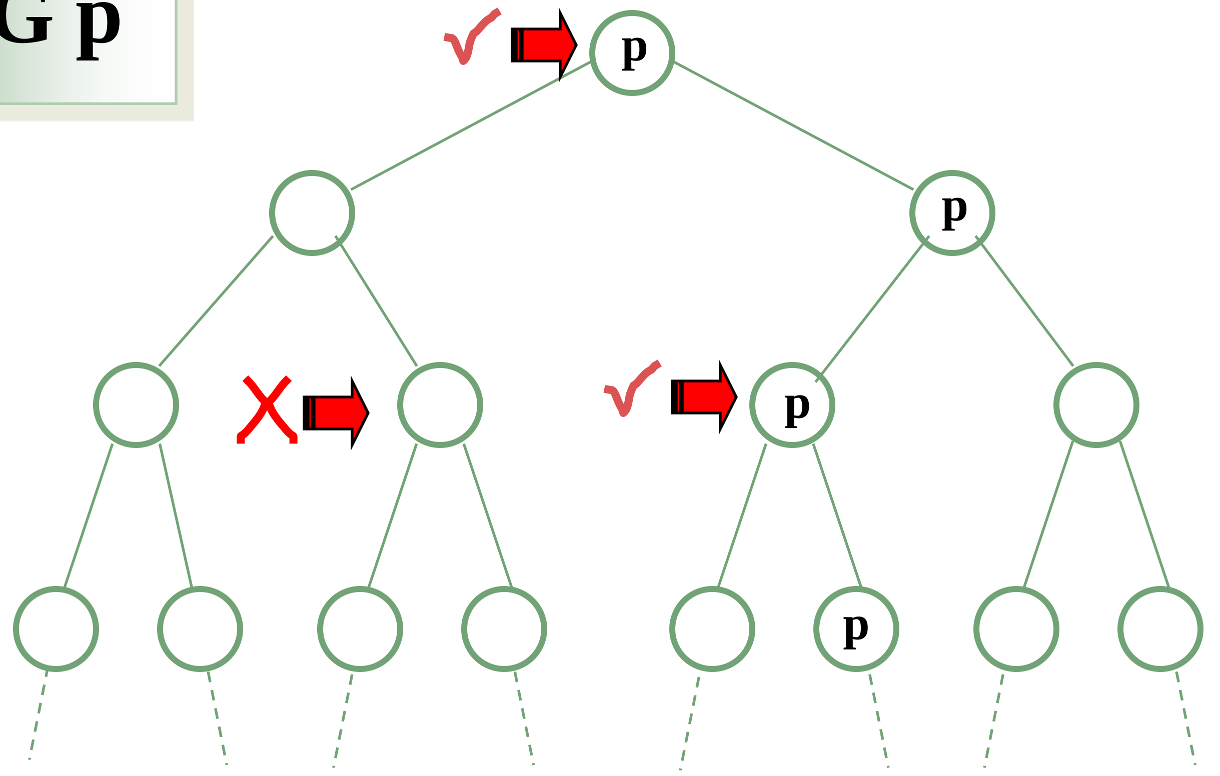
Computation Tree Logic

AG p



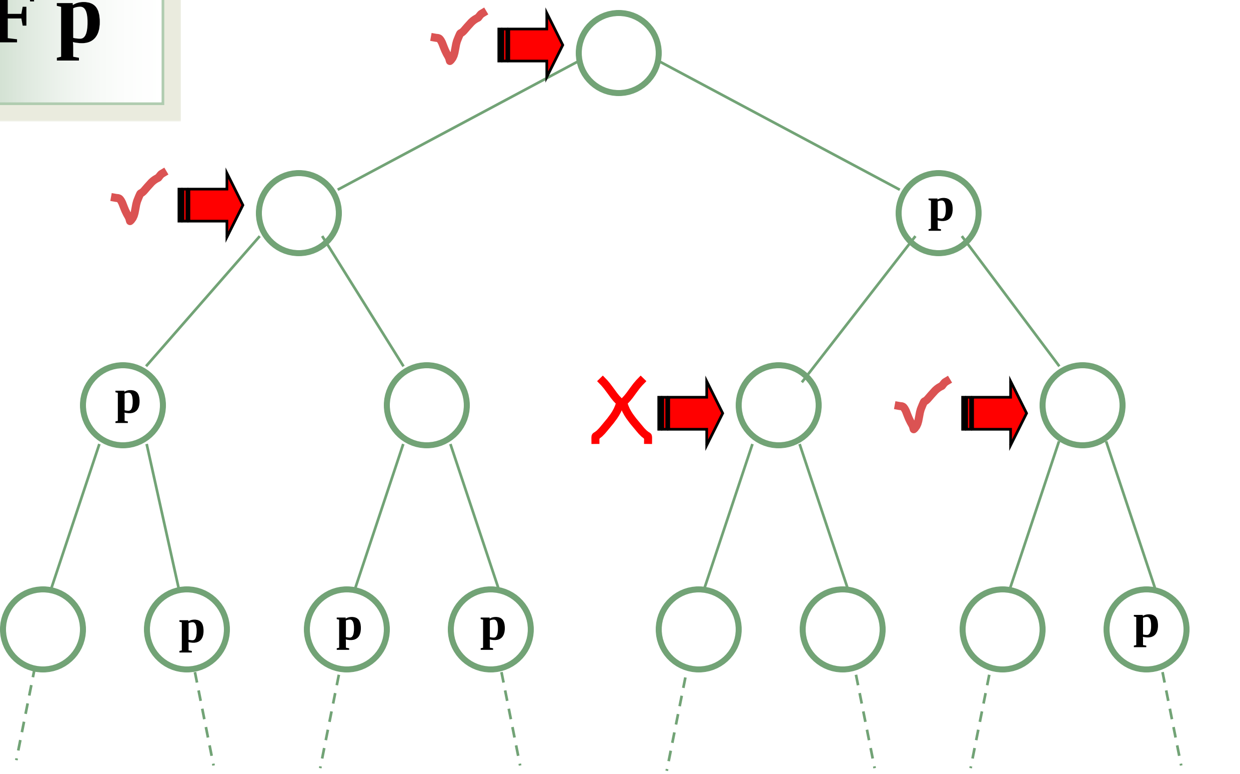
Computation Tree Logic

EG p



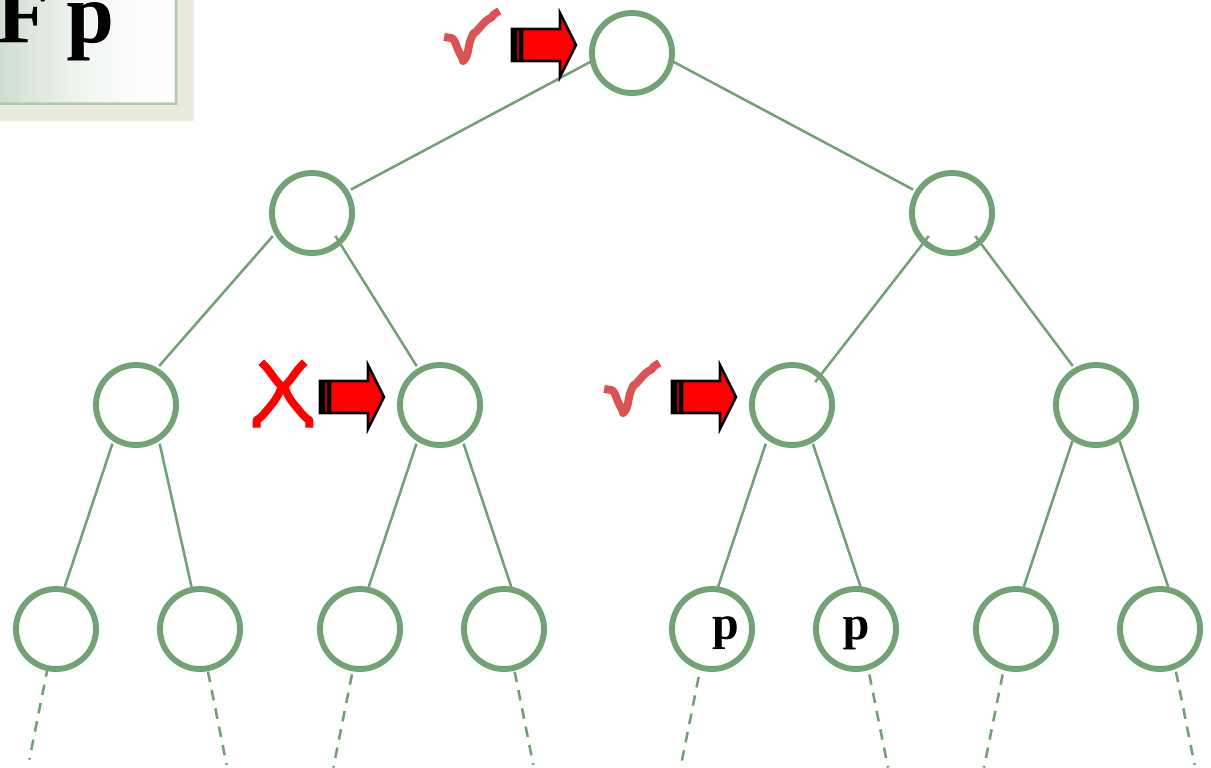
Computation Tree Logic

AF p



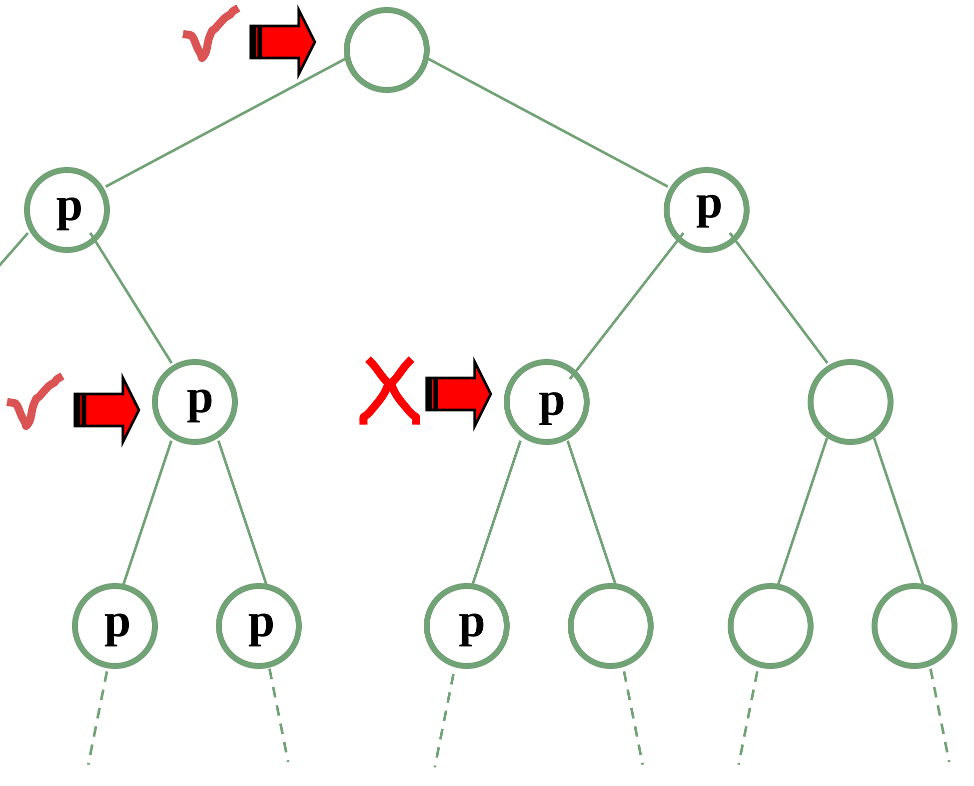
Computation Tree Logic

EF p



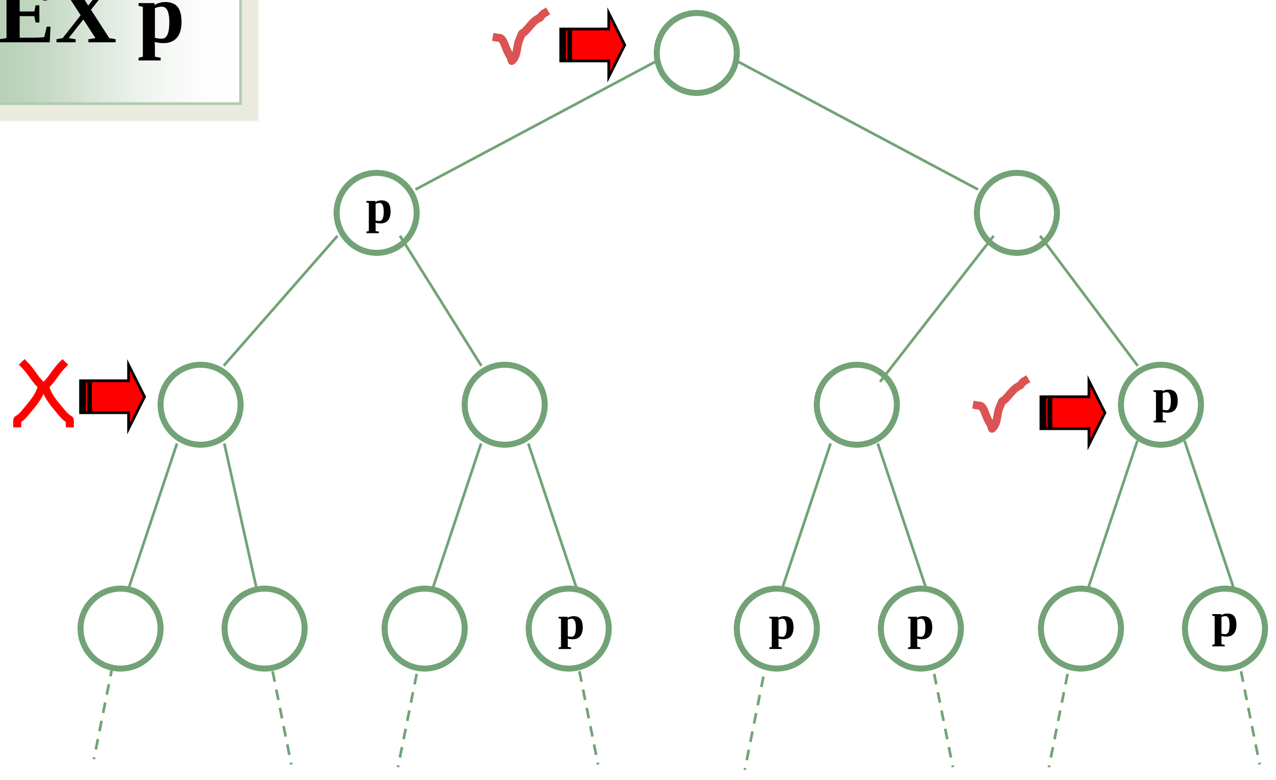
Computation Tree Logic

AX p



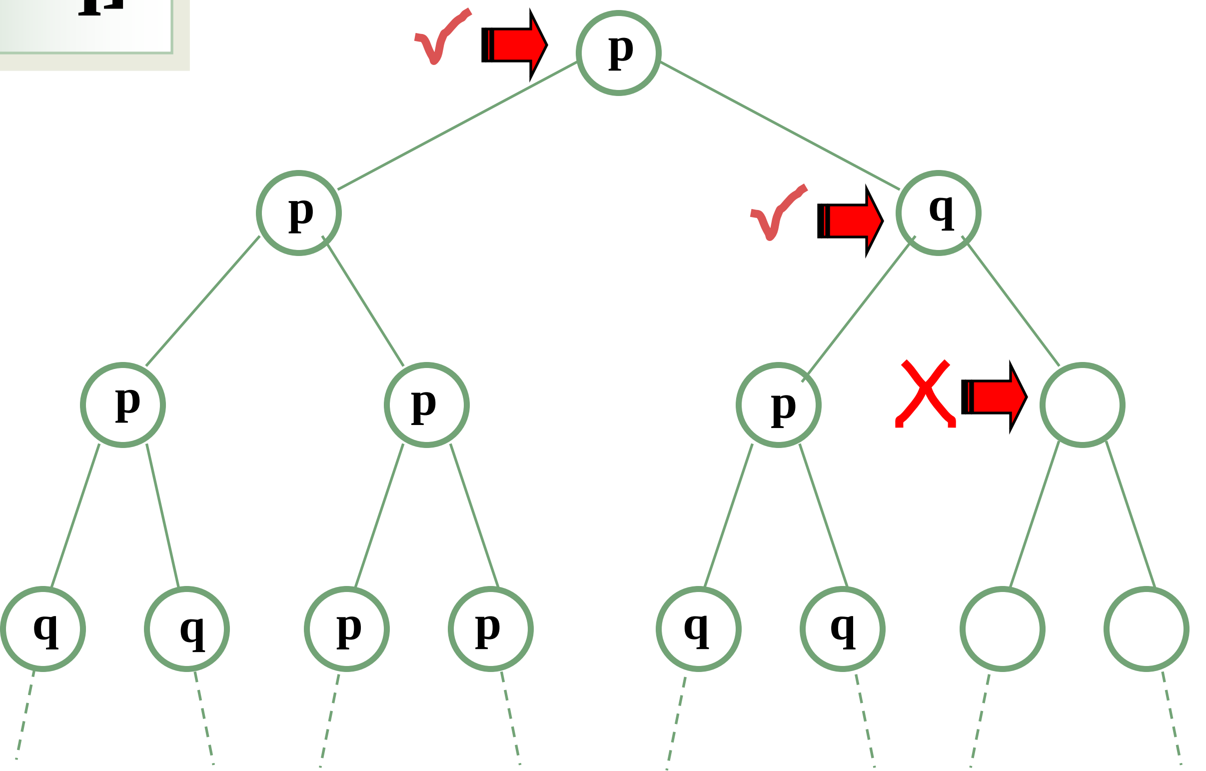
Computation Tree Logic

EX p



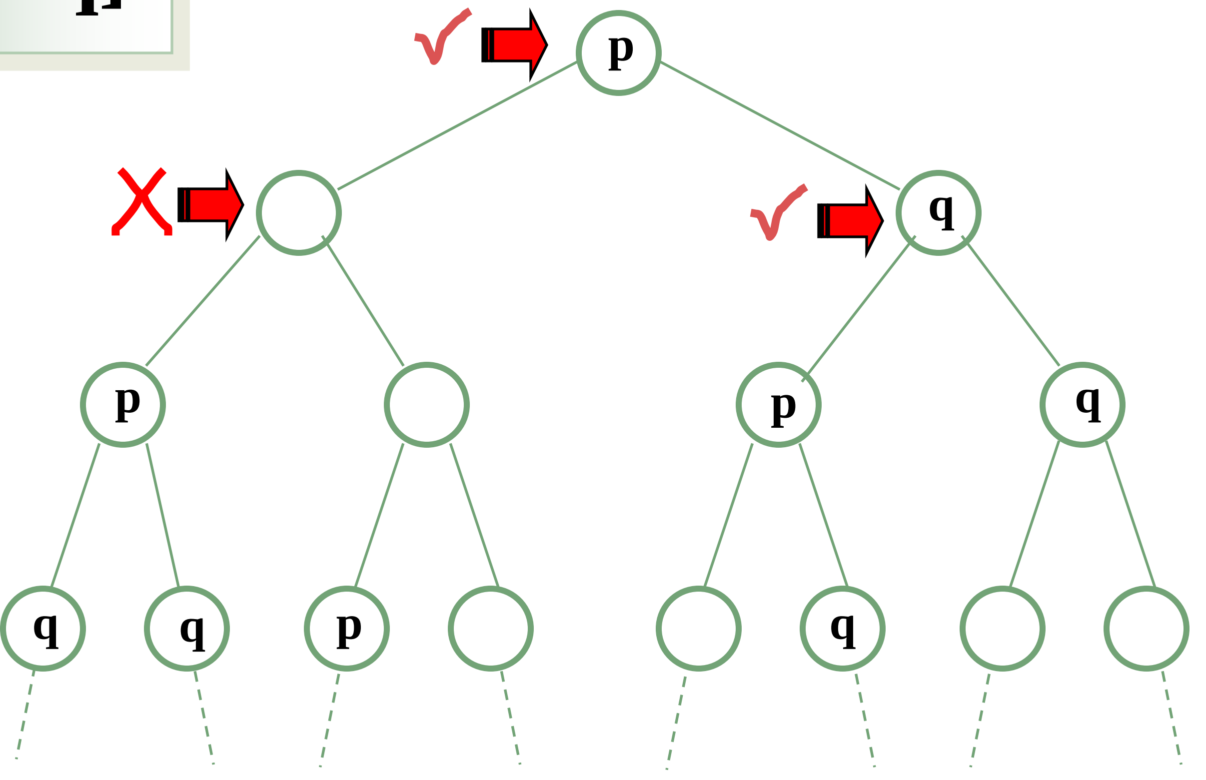
Computation Tree Logic

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)
```

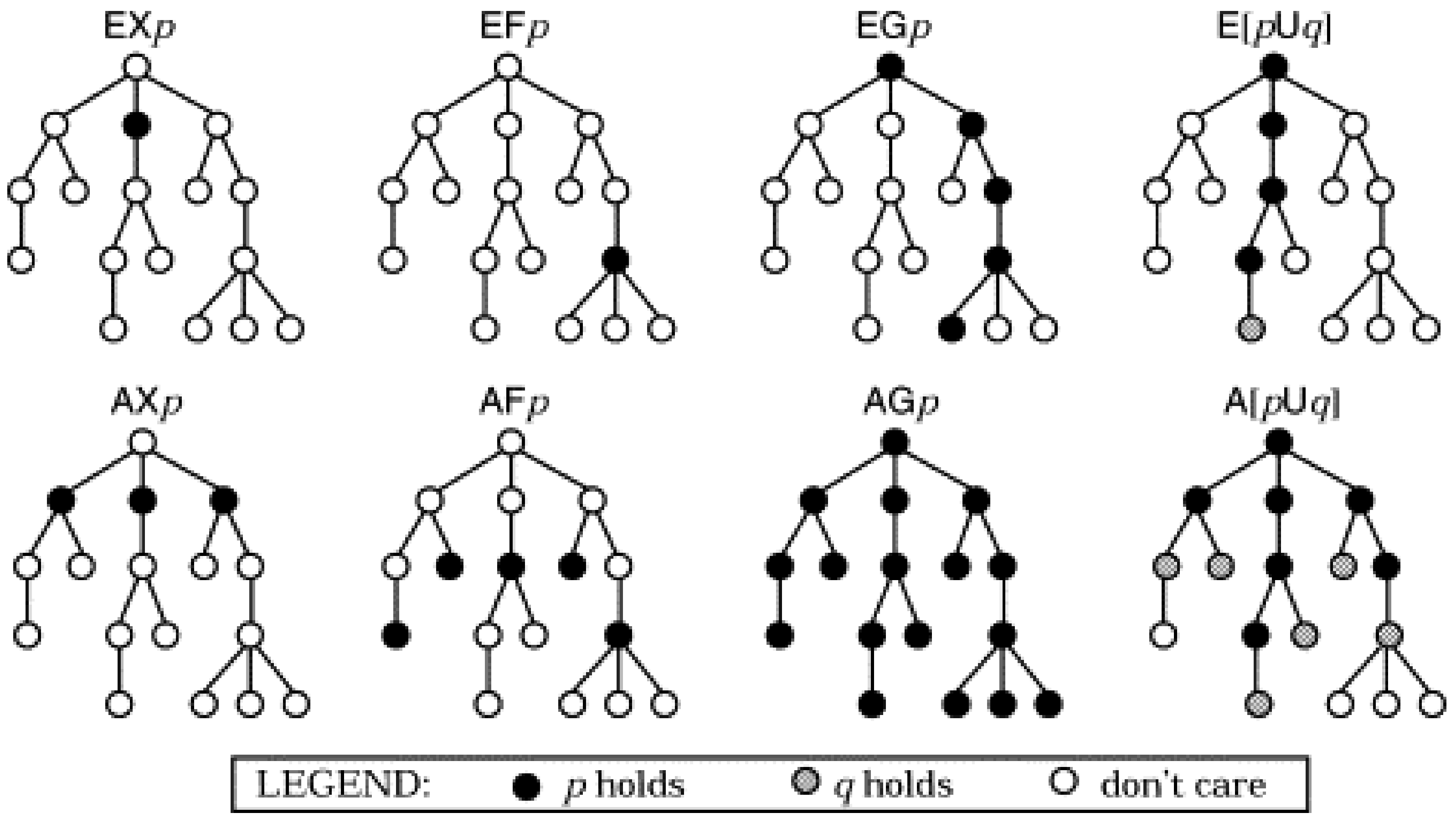
From any state, it is possible to get to a restart state

```
AG(EF restart)
```

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])
```

CTL Example



EXp
EFp
EGp
E[pUq]
AXp
AFp
AGp
A[pUq]
LEGEND:
p holds
q holds
don't care

CTL Semantics

- $M, s \models p$ if $p \in L(s)$
- $M, s \models \neg p$ if not $M, s \models p$
- $M, s \models p \wedge q$ if $M, s \models p$ and $M, s \models q$
- $M, s \models p \vee q$ if $M, s \models p$ or $M, s \models q$

- $M, s \models Ap$ if $\forall \pi \in \pi(s)$: $M, \pi \models p$
- $M, s \models Ep$ if $\exists \pi \in \pi(s)$: $M, \pi \models p$

CTL Semantics

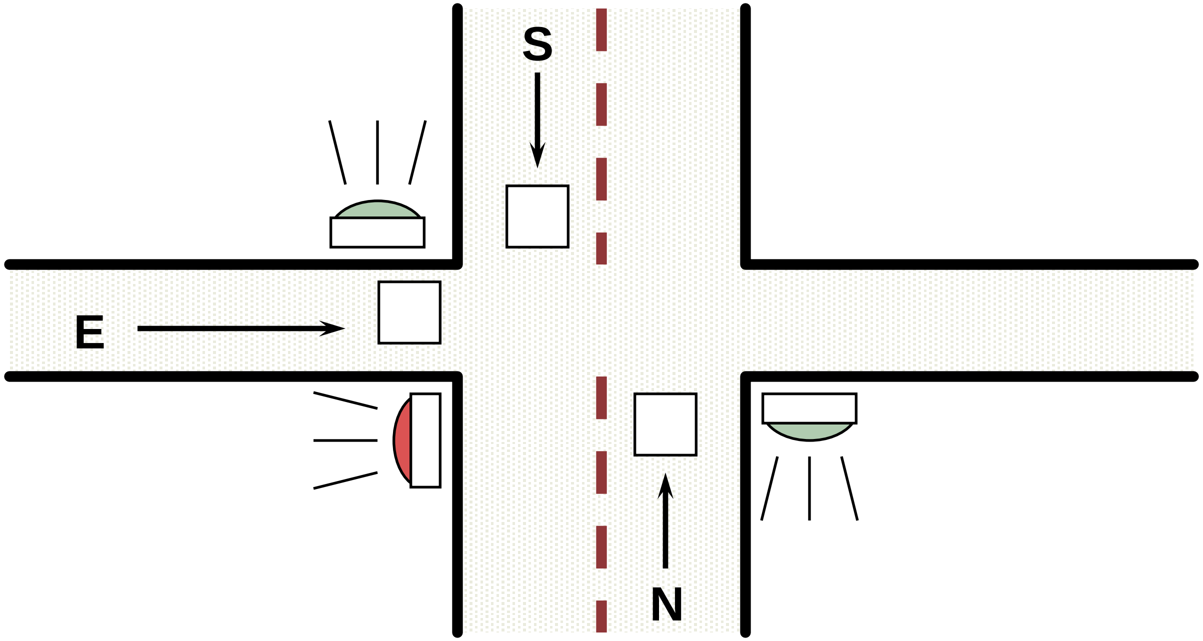
- $M, \pi \models Xp$ if $M, \pi_1 \models p$
- $M, \pi \models Fp$ if $\exists i \geq 0$: $M, \pi_i \models p$
- $M, \pi \models Gp$ if $\forall i \geq 0$: $M, \pi_i \models p$
- $M, \pi \models pUq$ if $\exists i \geq 0$: $M, \pi_i \models q$ and $\forall j < i$: $M, \pi_j \models p$

$M \models p$ if $M, s_0 \models p$

CTL Satisfiability

- If a CTL formula is satisfiable, then the formula is satisfiable by a finite Kripke model.
- CTL Model Checking: $O(|p|\cdot(|S|+|R|))$

Example: traffic light controller



- Guarantee no collisions
- Guarantee eventual service

Specifications

- Safety (no collisions)

$$AG \neg (E\_Go \wedge (N\_Go \mid S\_Go));$$

- Liveness

$$AG (\neg N\_Go \wedge N\_Sense \Rightarrow AF\ N\_Go);$$

$$AG (\neg S\_Go \wedge S\_Sense \Rightarrow AF\ S\_Go);$$

$$AG (\neg E\_Go \wedge E\_Sense \Rightarrow AF\ E\_Go);$$

- Fairness constraints

$$AF \neg(N\_Go \wedge N\_Sense);$$

$$AF \neg(S\_Go \wedge S\_Sense);$$

$$AF \neg(E\_Go \wedge E\_Sense);$$

Equivalence

EXp	EGp	E(pUq)

$AXp \equiv \neg EX \neg p$

$AFp \equiv \neg EG \neg p$

$AGp \equiv \neg EF \neg p$

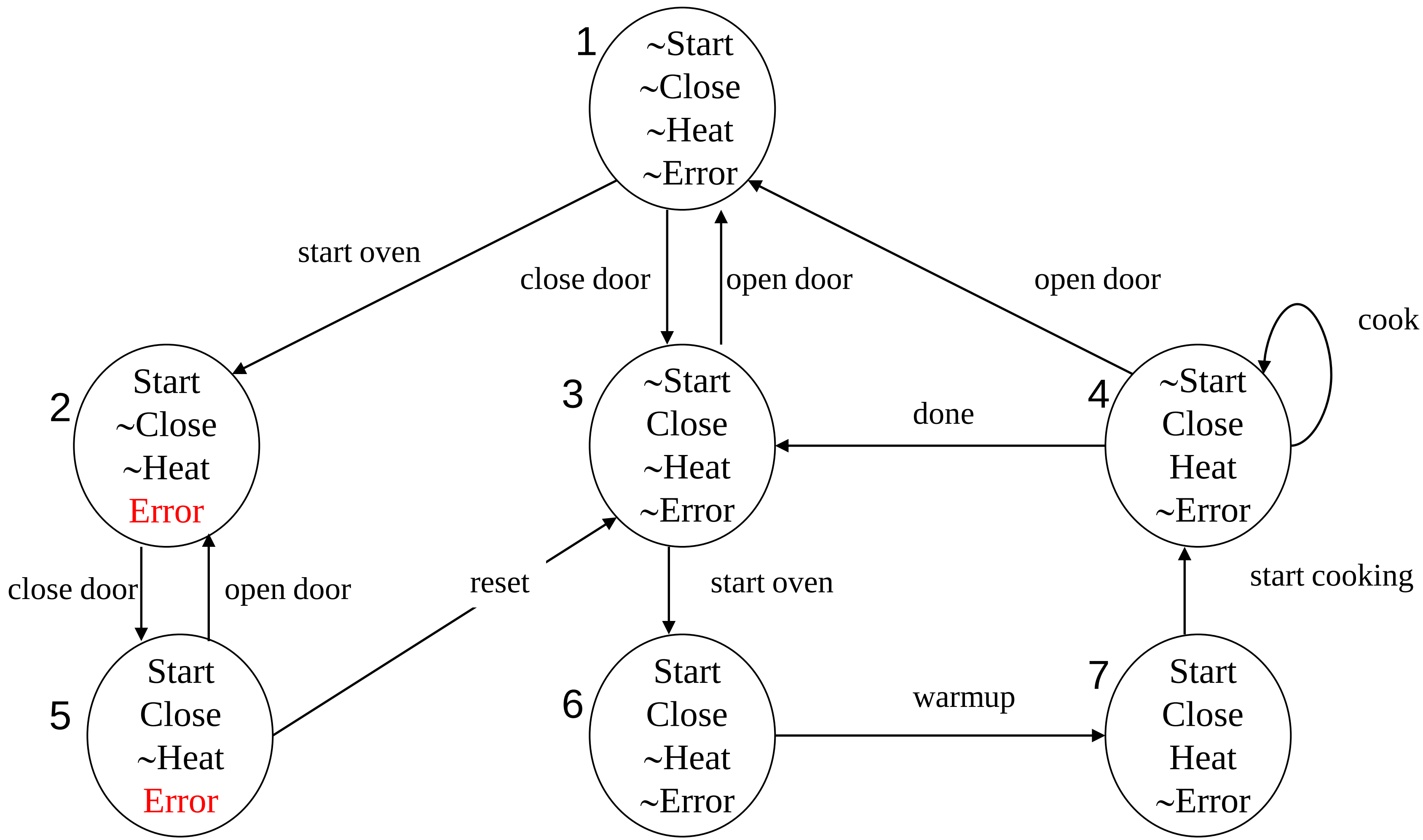
$A(pUq) \equiv \neg E(\neg pR \neg q)$

$EFp \equiv E(true\ U\ p)$

CTL Model Checking

- Six Cases:
  - p is an atomic proposition
  - $p = \neg q$
  - $p = q \vee r$
  - $p = EXq$
  - $p = EGq$
  - $p = E(qUr)$

Example: Microwave Oven



CTL Specification

- We would like the microwave to have the following properties (among others):
  - No heat while door is open
    - $\mathbf{AG}(\, \mathit{Heat} \rightarrow \mathit{Close})$:
  - If oven starts, it will eventually start cooking
    - $\mathbf{AG}\ (\mathit{Start} \rightarrow \mathbf{AF}\ \mathit{Heat})$
  - It must be possible to correct errors
    - $\mathbf{AG}(\, \mathit{Error} \rightarrow \mathbf{AF}\ \neg\ \mathit{Error})$:
- Does it? How do we prove it?

CTL Model Checking Algorithm

- Iterate over subformulas of *f* from smallest to largest
  - For each $s \in S$, if subformula is true in *s*, add it to *labels(s)*
- When algorithm terminates
  - $M,s \models f$ iff $f \in labels(s)$

Checking Subformulas

- Any CTL formula can be expressed in terms of:

¬, ∨, EX, EU, and **EG,** therefore must consider 6 cases:

Atomic proposition

if ap $\in$ L(s), add to *labels(s)*

$\neg f_1$

if $f_1 \notin$ *labels(s)*, add $\neg f_1$ to *labels(s)*

$f_1 \vee f_2$

if $f_1 \in$ *labels(s)* or $f_1 \in$ *labels(s)*, add $f_1 \vee f_2$ *to labels(s)*

EX f_1

add **EX** f_1 to *labels(s)* if successor of *s,* s', has $f_1 \in$ *labels*(s')

Checking Subformulas

- $\mathbf{E}[f_1\ \mathbf{U}\ f_2]$
  - Find all states s for which $f_2 \in$ *labels(s)*
  - Follow paths backwards from s finding all states that can reach s on a path in which every state is labeled with f_1
  - Label each of these states with $\mathbf{E}[f_1\ \mathbf{U}\ f_2]$

Checking Subformulas

- **EG** f_1 Basic idea – look for one infinite path on which f1 holds.
- Decompose M into nontrivial strongly connected components
  - A strongly connected component (SCC) C is
    - a maximal subgraph such that every node in C is reachable by every other node in C on a directed path that contained entirely within C.
  - C is nontrivial iff either
    - it has more than one node or
    - it contains one node with a self loop
- Create M' = (S’,R’,L’) from M by removing all states s $\in$ S in which $f_1 \notin$ labels(s) and updating S, R, and L accordingly

Checking Subformulas

- Lemma $M,s \models$ **EG** f_1 iff

1. $s \in S'$

2. There exists a path in M' that leads from s to some node t in a nontrivial strongly connected component of the graph (S', R', L').

- Proof left as exercise, but basic idea is
  - Can't have an infinite path over finite states without cycles
  - So if we find a path from *s* to a cycle and f_1 holds in every state (by construction) , then we've found an infinite path over which f_1 holds

Checking EG f_1

procedure CheckEG(f_1)

$S' = \{s \mid f_1 \in labels(s)\};$

$SCC = \{C \mid C \text{ is a nontrivial SCC of } S'\};$

$T = \cup_{C \in SCC} \{s \mid s \in C\};$

for all $s \in T$ *do* $labels(s) = labels(s) \cup \{\mathbf{EG}\ f_1\};$

while $T \neq \emptyset$ **do**

choose $s \in T$;

$T = T \setminus \{s\};$

for all t **such that** $t \in S'$ **and** $R(t,s)$ **do**

if EG $f_1 \notin labels(t)$ **then**

$labels(t) = labels(t) \cup \{\mathbf{EG}\ f_1\};$

$T = T \cup \{t\};$

end if;

end for all;

end while;

end procedure;

Checking a Property

- Checking AG(Start $\rightarrow$ AF Heat)
  - Rewrite as $\neg$EF(Start $\wedge$ EG $\neg$Heat)
  - Rewrite as $\neg$ E[true U (Start $\wedge$ EG $\neg$Heat)]
- Compute labels for smallest subformulas
  - Start, Heat
  - $\neg$ Heat

Formulas/States	1	2	3	4	5	6	7
Start		x			x	x	x
Heat				x			x
$\neg$ *Heat*	x	x	x		x	x	
EG $\neg$*Heat*							
Start $\wedge$ **EG** $\neg$*Heat*							
E[true U(*Start* $\wedge$ **EG** $\neg$*Heat*)]							
$\neg$ **E[true U**(*Start* $\wedge$ **EG** $\neg$*Heat*)]							

Checking a Property

- Compute labels for **EG** ¬*Heat*
- *S'* = {1,2,3,5,6}
- *SCC* = {{1,2,3,5}}
- *T* = {1,2,3,5}
- No other state in *S'* can reach a state in *T* along a path in *S'*.
- Computation terminates. States 1,2,3, and 5 labelled with **EG** ¬*Heat*

Formulas/States	1	2	3	4	5	6	7
Start		x			x	x	x
Heat				x			x
¬ *Heat*	x	x	x		x	x	
EG ¬*Heat*	x	x	x		x		
Start ∧ **EG** ¬*Heat*							
E[true U(*Start* ∧ **EG** ¬*Heat*)]							
¬ **E[true U**(*Start* ∧ **EG** ¬*Heat*)]							

Checking a Property

- Compute labels for *Start* ∧ **EG** ¬*Heat*

Formulas/States	1	2	3	4	5	6	7
Start		x			x	x	x
Heat				x			x
¬ *Heat*	x	x	x		x	x	
EG ¬*Heat*	x	x	x		x		
Start ∧ **EG** ¬*Heat*		x			x		
E[true U(*Start* ∧ **EG** ¬*Heat*)]							
¬ **E[true U(***Start* ∧ **EG** ¬*Heat*)]							

Checking a Property

- **E[*true* U**(*Start* ∧ **EG** ¬*Heat*)]
- Start with set of states in which *Start* ∧ **EG** ¬*Heat* holds i.e., {2,5}
- Work backwards marking every state in which ***true*** holds

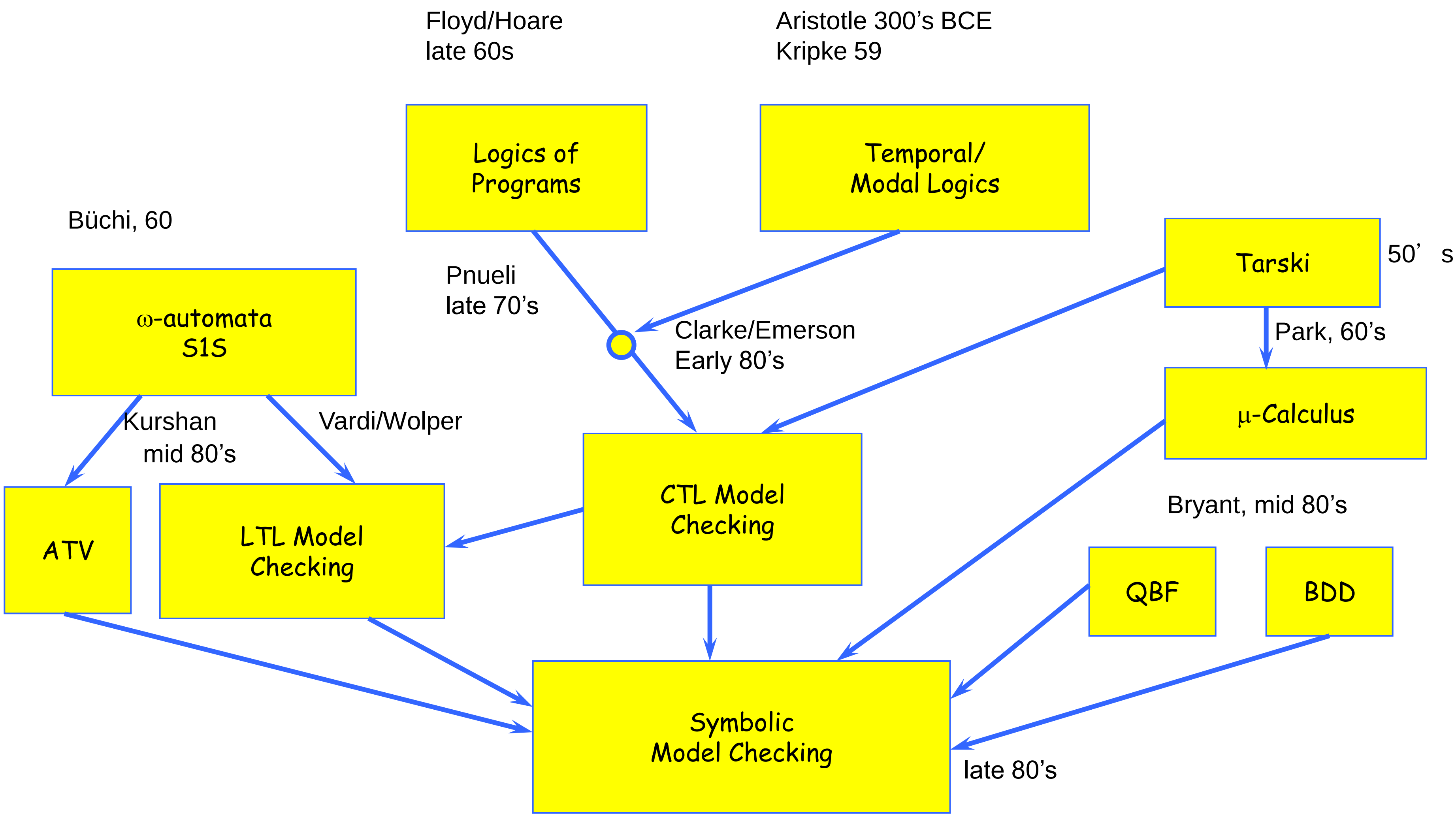
Formulas/States	1	2	3	4	5	6	7
Start		x			x	x	x
Heat				x			x
¬ *Heat*	x	x	x		x	x	
EG ¬*Heat*	x	x	x		x		
Start ∧ **EG** ¬*Heat*		x			x		
E[true U(*Start* ∧ **EG** ¬*Heat*)]	x	x	x	x	x	x	x
¬ **E[true U**(*Start* ∧ **EG** ¬*Heat*)]							

Checking a Property

- Check ¬ E[true U(Start ∧ EG ¬Heat)]
- Leaves us with the empty set, so this property doesn't hold over our microwave oven

Formulas/States	1	2	3	4	5	6	7
Start		x			x	x	x
Heat				x			x
¬ *Heat*	x	x	x		x	x	
EG ¬*Heat*	x	x	x		x		
Start ∧ **EG** ¬*Heat*		x			x		
E[true U(*Start* ∧ **EG** ¬*Heat*)]	x	x	x	x	x	x	x
¬ **E[true U**(*Start* ∧ **EG** ¬*Heat*)]							

Genealogy



Floyd/Hoare
late 60s
Aristotle 300's BCE
Kripke 59
Logics of Programs
Temporal/ Modal Logics
Büchi, 60
Tarski
50' s
Pnueli
late 70's
ω-automata
S1S
Clarke/Emerson
Early 80's
Park, 60's
μ-Calculus
Kurshan
mid 80's
Vardi/Wolper
CTL Model Checking
ATV
LTL Model Checking
Bryant, mid 80's
QBF
BDD
Symbolic Model Checking
late 80's

Turing Awards in Verification

1. Amir Pnueli (1996)

   Temporal logics for specifying system behavior

2. Edmund Clarke, Allen Emerson, and Joseph Sifakis (2007)

   Development of model checking