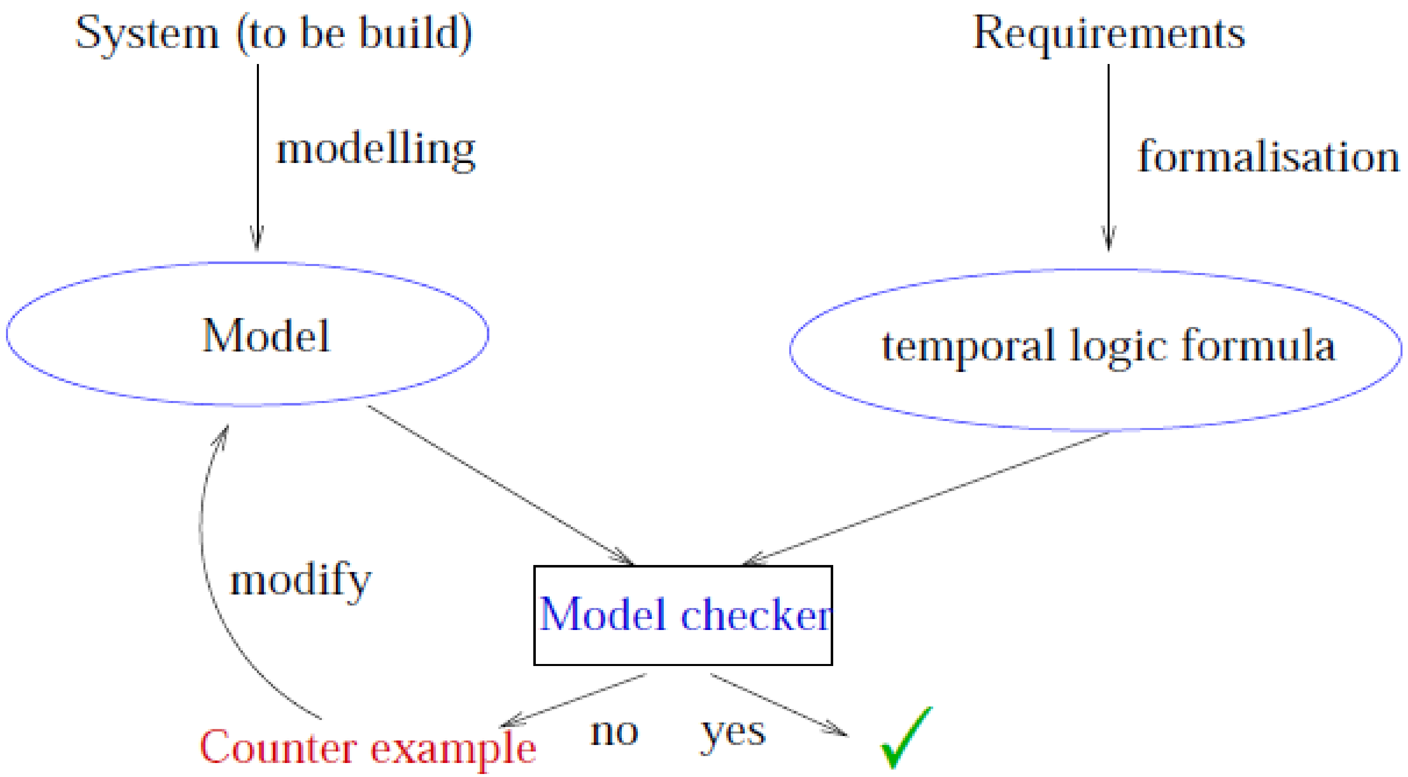
Temporal Logic : LTL

The Big Picture



Plan

- Temporal logic (LTL)
- LTL operators
- Semantics of LTL
- …
- Microwave oven example

LTL

(P)LTL - Propositional linear-time temporal logic

Basis:

atomic propositions

(assertions/predicates on states, also called state formulae)

additionally:

boolean connectives: $\vee, \wedge, \neg$

temporal operators: always, sometimes, tomorrow

LTL operators

- **Def..** AP a set of atomic propositions. The set of LTL-formulae over
- AP is inductively defined as follows
- p ∈ AP is an LTL formula
- if φ is an LTL formula, so is $\neg\varphi$,
- if φ, ψ are LTL formulae, so is $\varphi \vee \psi$,
- if φ is an LTL formula, so are X φ,G φ, F φ,
- if φ, ψ are LTL formulae, so is φ U ψ .

A formula without U,G , X , F is a state formula.

LTL operators

- Derived boolean connectives
- $false := \neg true$
- $\varphi \wedge \psi := \neg(\neg \varphi \vee \neg \psi)$
- $\varphi \Rightarrow \psi := \neg \varphi \vee \psi$
- $\varphi \Leftrightarrow \psi := (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$

LTL operators

Meaning of temporal operators

- X (next)

  $X\varphi$: φ holds in the next state

- G (globally, always)

  $G\varphi$: φ holds always

- F (eventually, in the future)

  $F\varphi$: φ holds sometimes in the future

- U (until)

  $\varphi\ U\ \psi$: φ holds until ψ holds (and ψ will eventually hold)

- Examples of LTL formulae: let $AP = \{x = 1, x < 2, x \geq 3\}$

$X(x = 1), \neg(x < 2), (x < 2)\ U (x \geq 3), F(x < 2) \vee G(x \geq 3)$

LTL operators (alternative notations)

- Alternative notations are used for temporal operators.

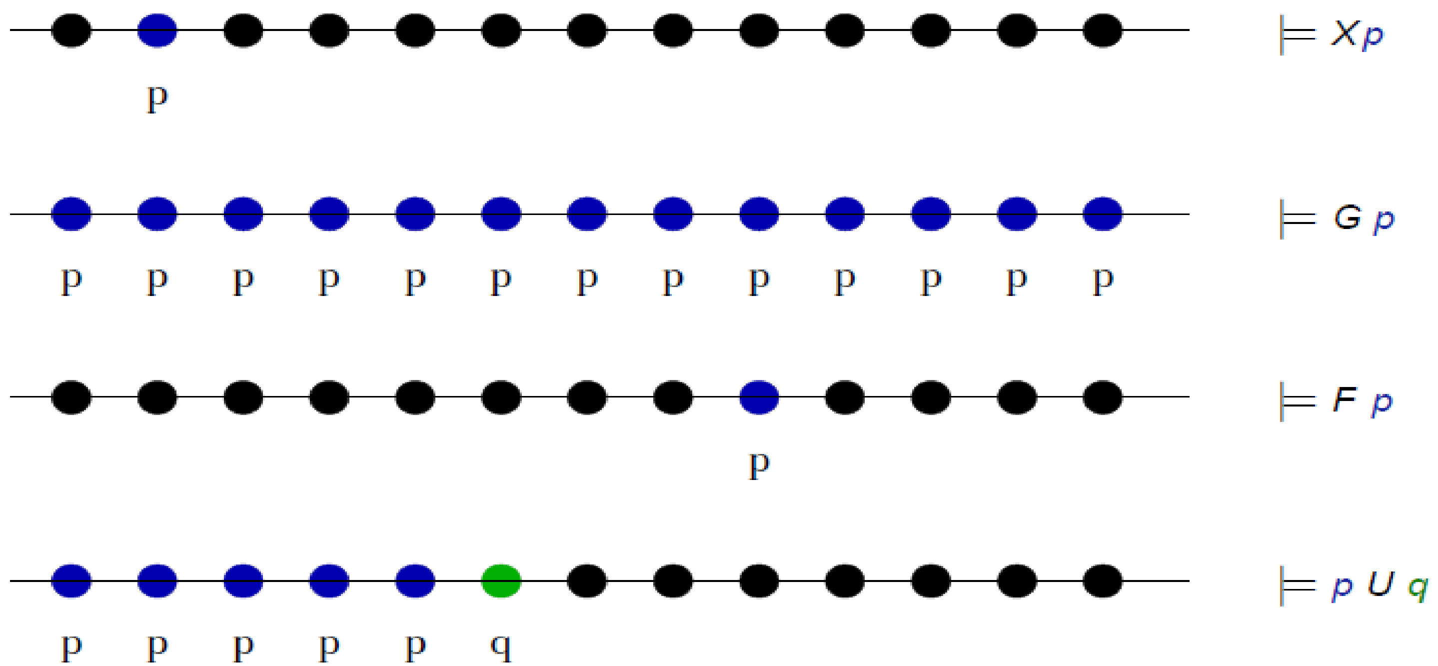
$\Diamond \rightsquigarrow$ F sometime in the Future

$\Box \rightsquigarrow$ G Globally in the future

$\bigcirc \rightsquigarrow$ X neXtime

Semantics: graphically

- Formulae are interpreted on paths of Kripke structures



Semantics (examples)

- Properties:

- p is always (eventually) followed by q

$$G\,(p \Rightarrow F\,q)$$

- p is always directly followed by q

$$G\,(p \Rightarrow X\,q)$$

- p will eventually be true forever

$$F\,G\,p$$

- p will always be true

$$G\,p$$

Semantics (examples)

- Atomic propositions:

  coffee_chosen, *tea_chosen*, *money_nserted*, *coffee_delivered*, *tea_delivered*

- once in a while someone chooses tea or coffee

  $G\, F\, (tea\_chosen \vee coffee\_chosen)$

- if coffee is chosen and next money is inserted coffee will be delivered

  $G\, ((coffee\_chosen \wedge X\, money\_inserted) \Rightarrow F\, coffee\_delivered)$

- when coffee has been chosen tea will not be delivered until tea is chosen

  $G\, (coffee\_chosen \Rightarrow (\neg tea\_delivered\ U\ tea\_chosen))$

Temporal Logic & Soft Eng.

- LTL has achieved a significant role in the formal specification and verification of concurrent reactive systems.
  Much of this popularity has been achieved as a number of useful concepts can be formally, and concisely, specified using temporal logics, e.g.
  - safety properties
  - liveness properties
  - fairness properties

Safety Properties

- Safety:

  "something bad will not happen"

- Typical examples:

  $G\neg(reactor\_temp > 1000)$

  $G\neg((x = 0) \wedge X\ (y = z/x))$

and so on.....

- Usually: $G\neg$....

Liveness Properties

- Liveness:

  "something good will happen"

- Typical examples:

  F rich , F $(x > 5)$, G (start $\Rightarrow$F terminate) G (Trying$\Rightarrow$F Critical)

- and so on.....
- Usually: F....

Fairness Properties

- Often only really useful when scheduling processes, responding to messages, etc.

  "if something is attempted/requested infinitely often, then it will be successful /allocated infinitely often"

- Typical example:

  $F \text{ ready} \Rightarrow F \text{ run}$

Expressiveness of LTL

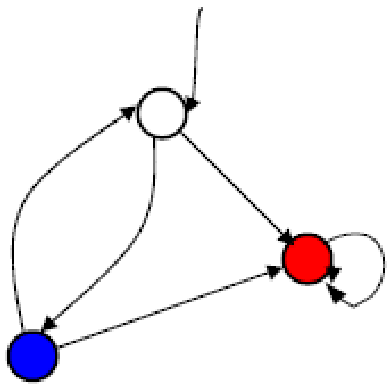
- Question: are there properties which cannot be expressed in LTL?

Answer: yes

-- properties which refer to the branching structure of the Kripke structure.

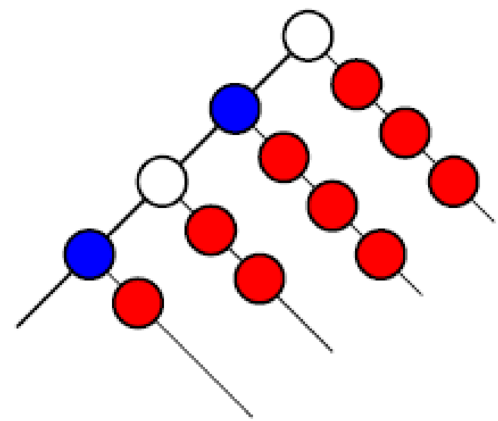
CTL can express such propertie (later)

Models of Computations

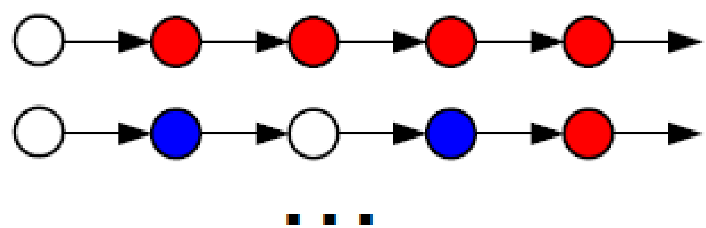


State transition graph
Kripke Structure

Computation tree



Paths

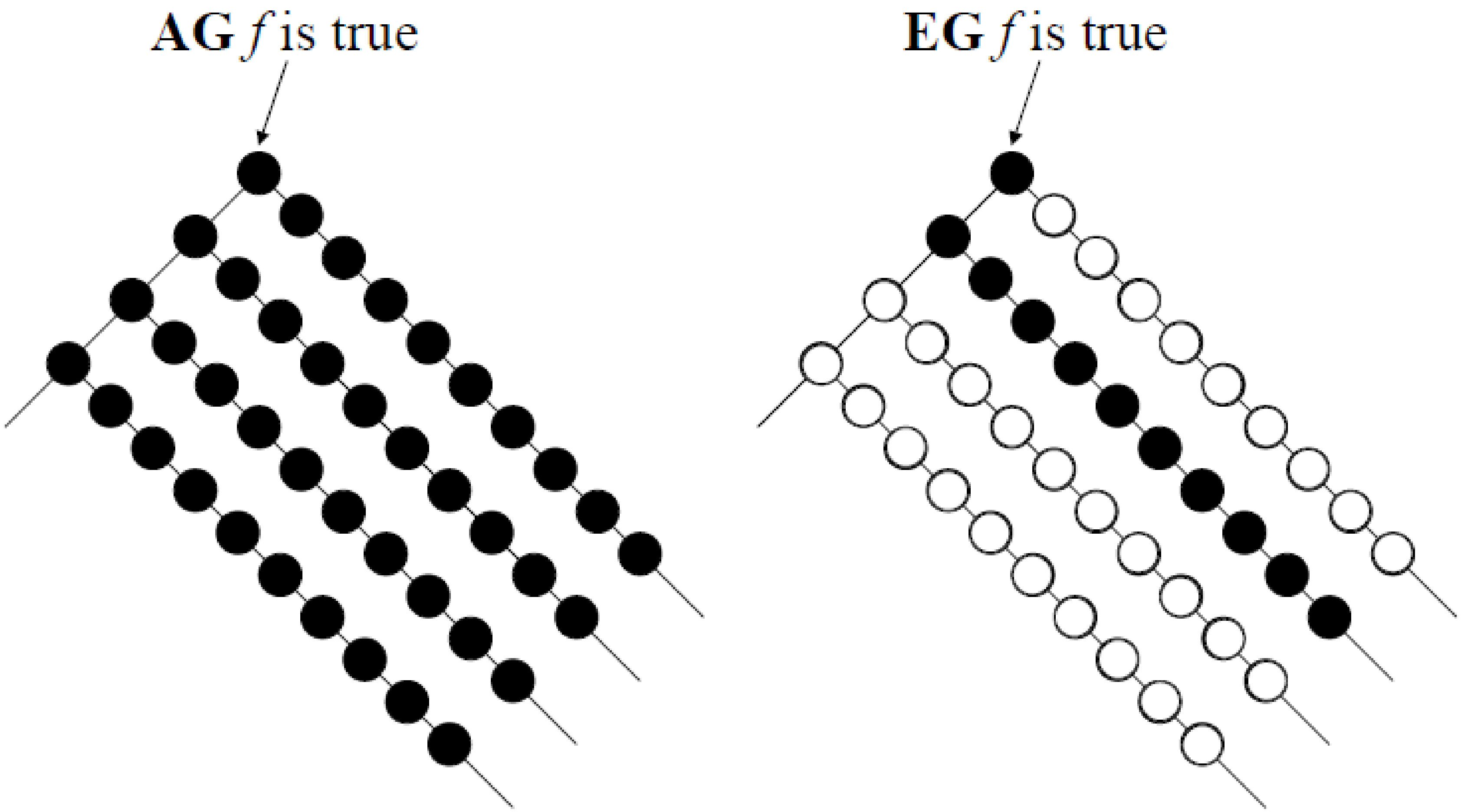


Computational Tree Logic CTL*

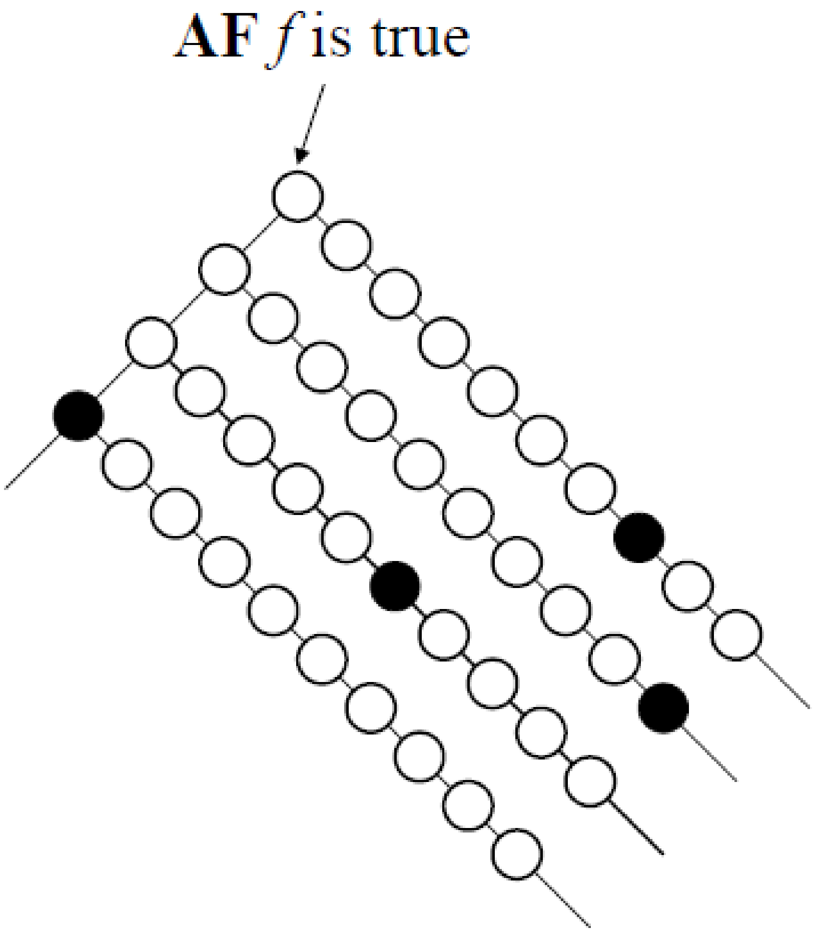
To describe paths from a given state.

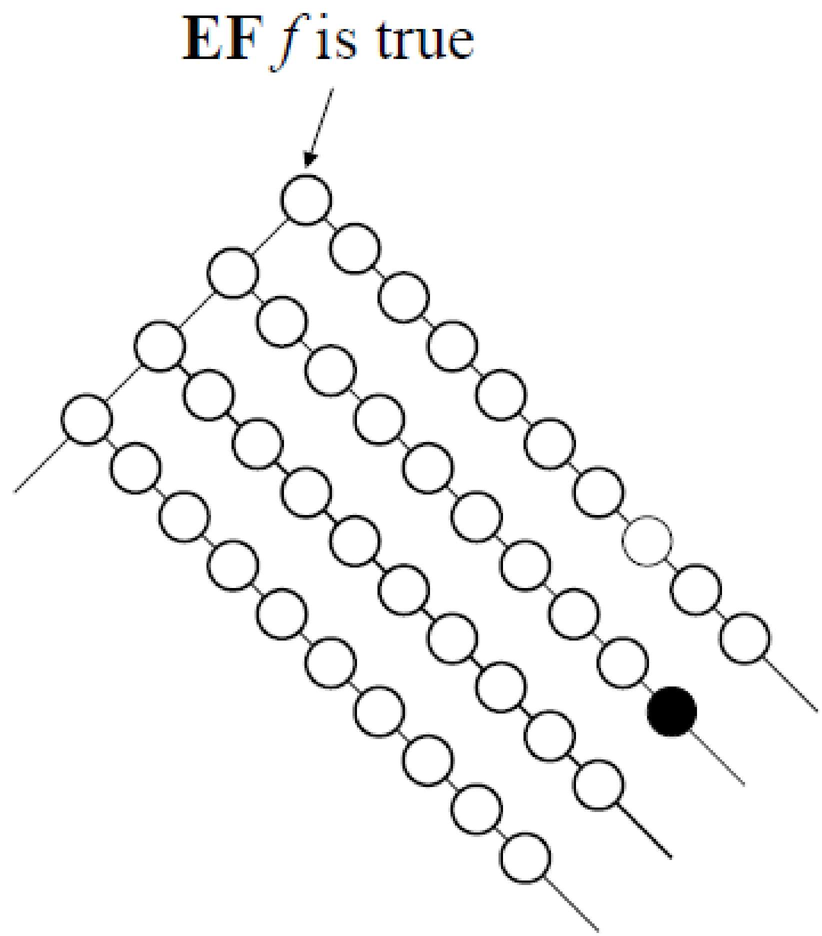
- Path quantifiers:
  - A: for all computation paths from a state.
  - E: for some computation path(s) from a state.
- Linear temporal operators: describe properties along a path.
  - Gp – p holds in every state on the path.
  - Fp – p holds in some state on the path.
  - Xp – p holds in the second state of the path
  - pUq – p holds until q holds in some state on the path.

Semantics: State Formulas

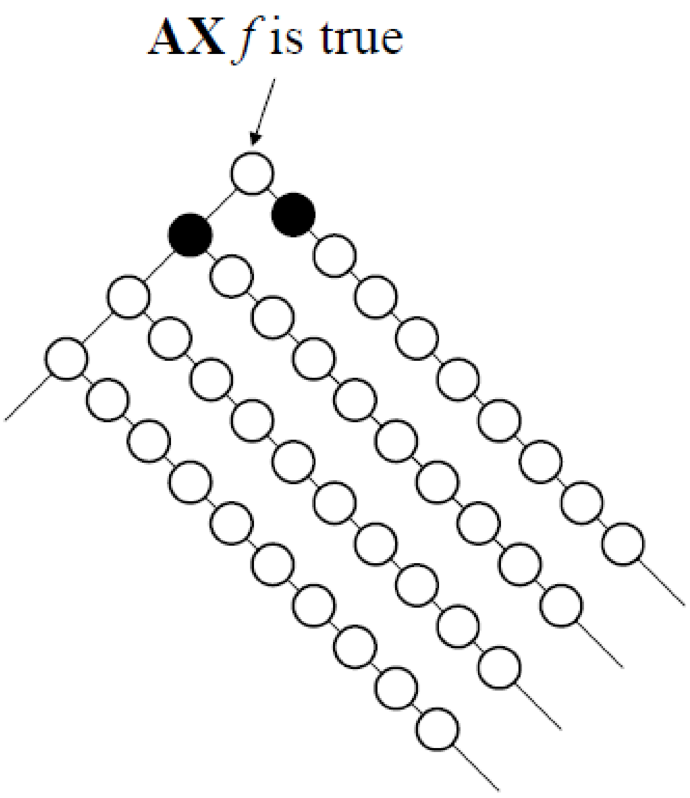


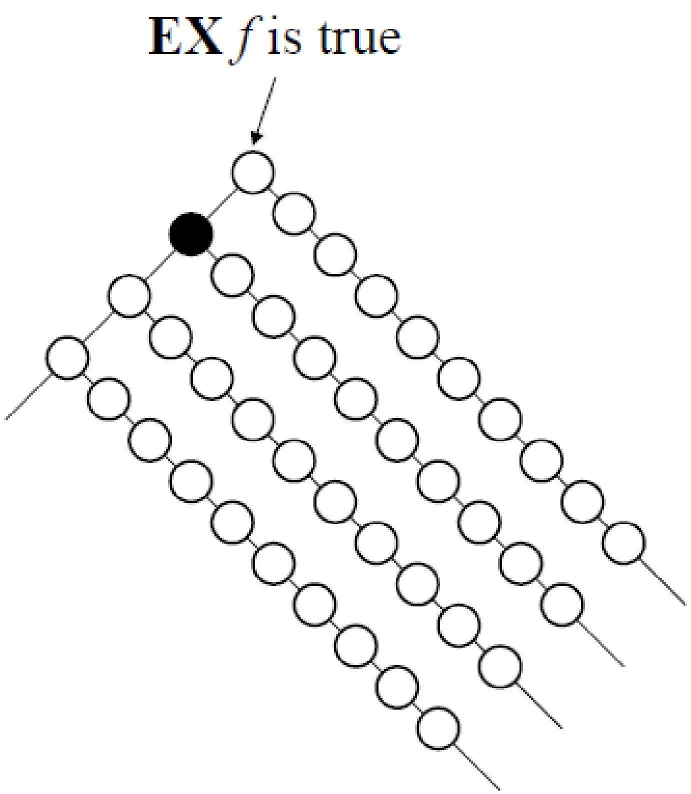
Semantics: State Formulas



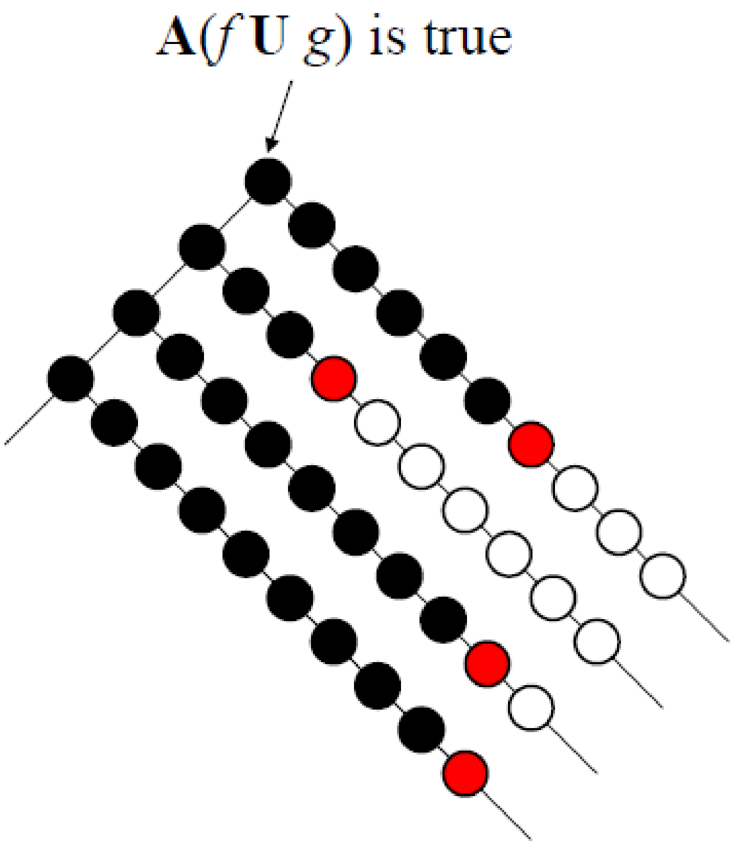


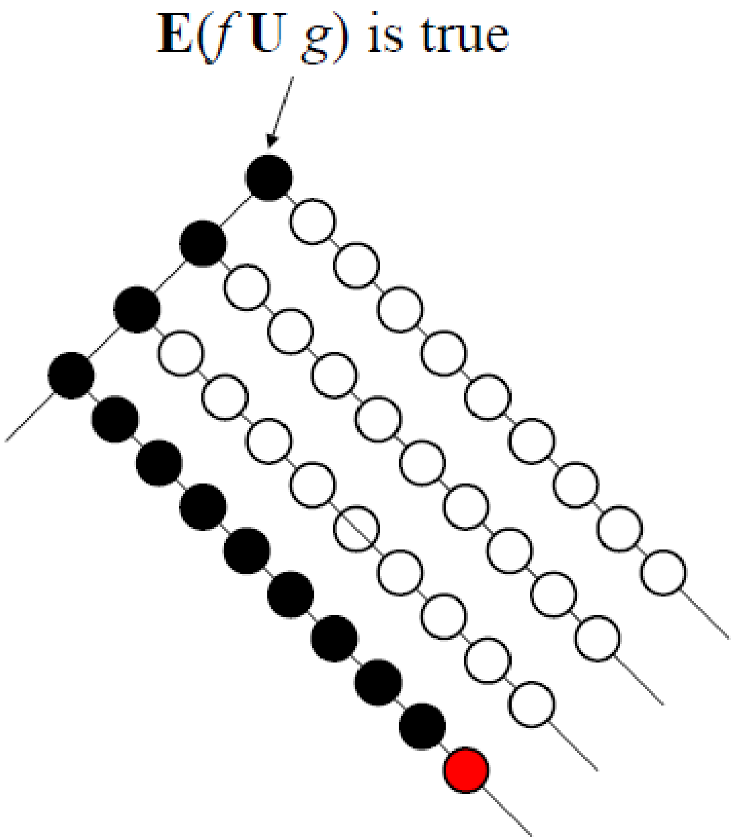
Semantics: State Formulas





Semantics: State Formulas

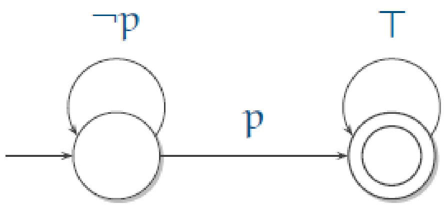




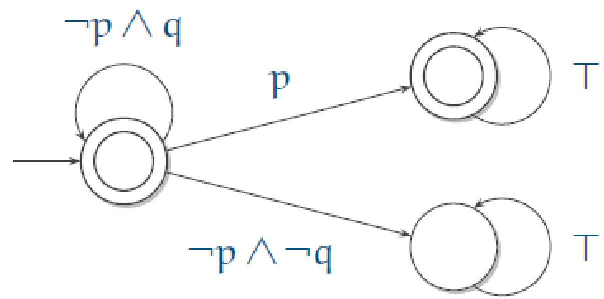
Equivalences

- $f \wedge g \equiv \neg(\neg f \vee \neg g)$
- $\mathbf{A}\, f \equiv \neg\mathbf{E}\ (\neg \mathrm{f}\ )$
- $\mathbf{G}\, f \equiv \neg\mathbf{F}\ (\neg \mathrm{f})$
- $\mathbf{F}\, f \equiv (\mathit{true}\ \mathbf{U}\ f)$
- $\mathbf{F}(f \vee g) \equiv \mathbf{F}f \vee \mathbf{F}g$
  - What about $\mathbf{F}(f \wedge g) \equiv \mathbf{F}f \wedge \mathbf{F}g$?
- $\mathbf{G}(f \wedge g) \equiv \mathbf{G}f \wedge \mathbf{G}g$
  - What about $\mathbf{G}(f \vee g) \equiv \mathbf{G}f \vee \mathbf{G}g$.
- $f\,\mathbf{U}\ g \equiv \neg(\neg g\ \mathbf{U}\ (\neg f \wedge \neg g)) \wedge \mathbf{F}\ g$

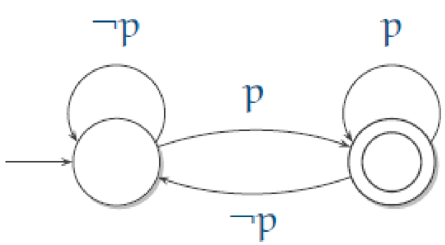
LTL Formulas as Automata



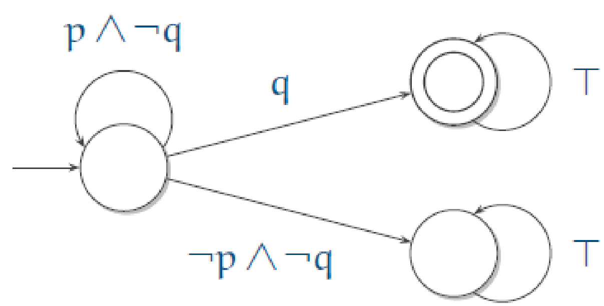
F p



p R q



GF p



p U q

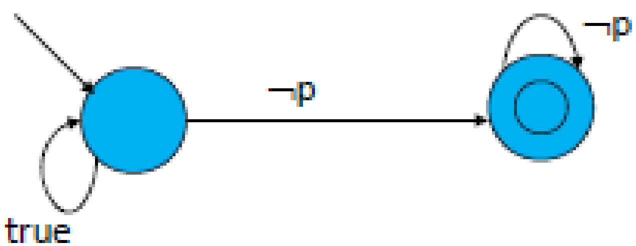
Model-Checking LTL

- Problem: “is formula Φ true of model M?”
  1 Convert M to automaton Am
  2 Convert to automaton A_{Φ}

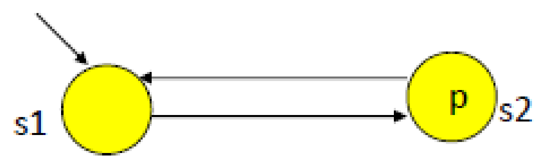
- Question is then: “is $L(Am) \subseteq L(A_{\Phi})$?”
- Which is the same as: “is $L(Am) \cap L(A_{\Phi}) = \emptyset$?”

Example: Verify GFp

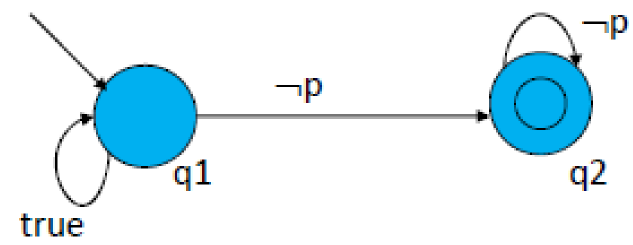
- Construct negation of the property
  - $\neg GFp \equiv FG\neg p$
- Construct automata accepting infinite length traces satisfying $FG\neg p$



Example: Verify GFp

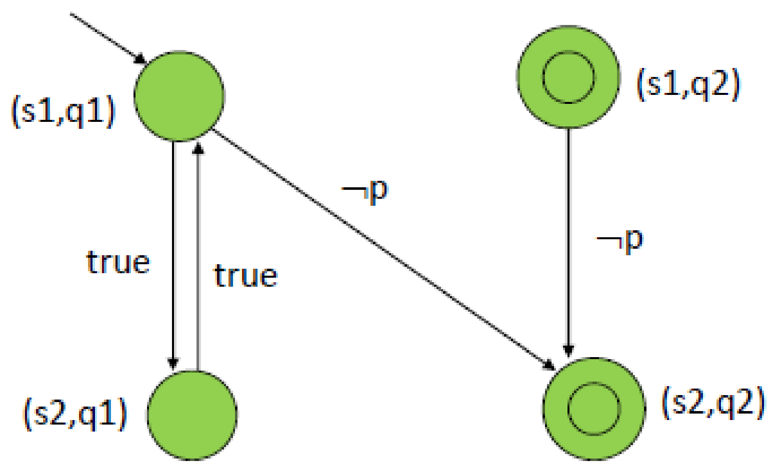


(i) System Model M



(ii) Property Automata A

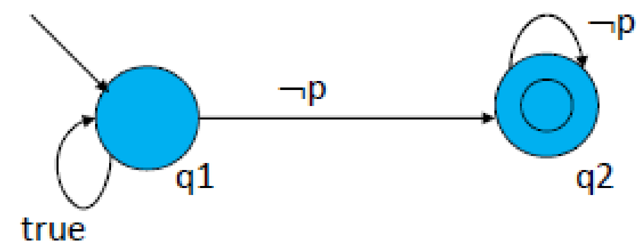
M |= GFp



(iii) Product Automata M × A

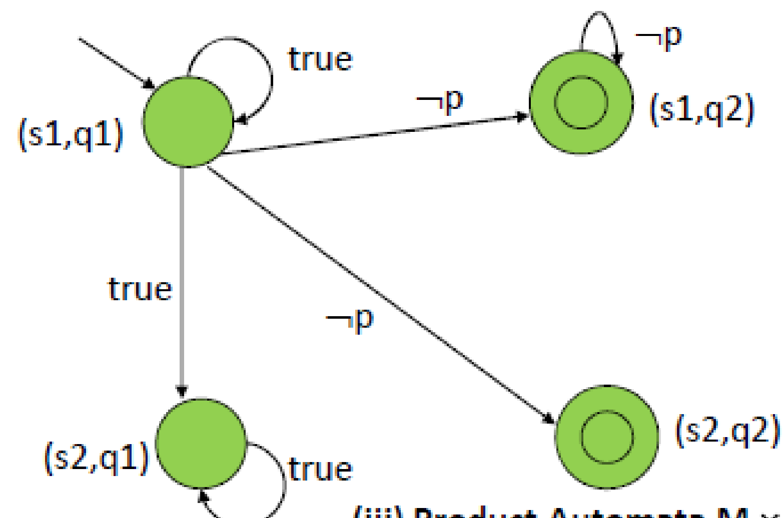
Example: Verify GFp

(i) System Model M



(ii) Property Automata A

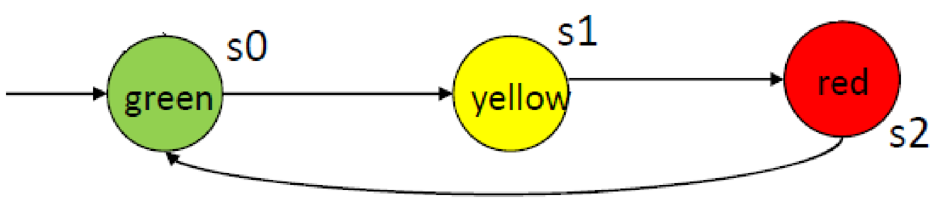
$M \not\models GF\ p$



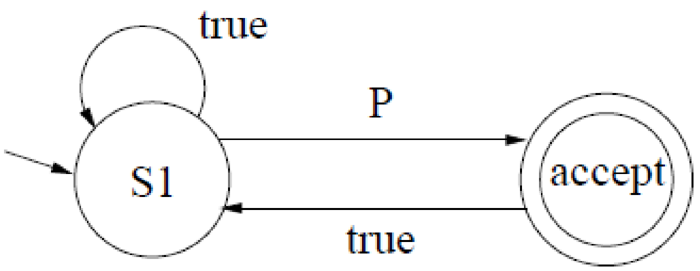
(iii) Product Automata M × A

SPIN & "Promela" language

```
active proctype TrafficLightController() {
        byte color = green;
        do
         :: (color == green) -> color = yellow;
         :: (color == yellow) -> color = red;
         :: (color == red) -> color = green;
         od;
}
```

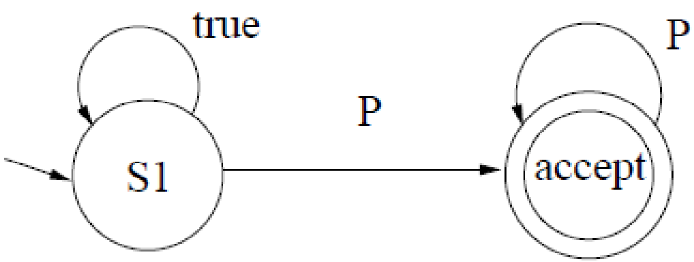


SPIN & “Promela” language




```
never {    /* []<> p */
T0_init:
       if
       :: ((p)) -> goto accept_S9
       :: (1) -> goto T0_init
       fi;
accept_S9:
       if
       :: (1) -> goto T0_init
       fi;
}
```

SPIN & “Promela” language



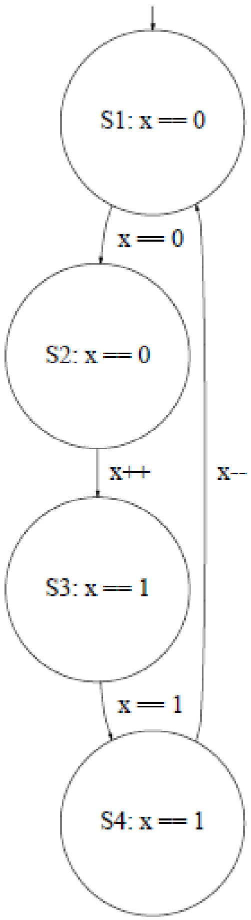

```
never {     /* <>[] p */
T0_init:
      if
      :: ((p)) -> goto accept_S4
      :: (1) -> goto T0_init
      fi;
accept_S4:
      if
      :: ((p)) -> goto accept_S4
      fi;
}
```

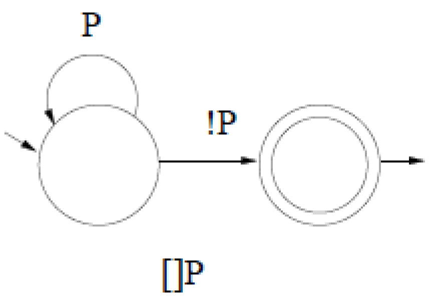
SPIN & “Promela” language

```
bit x=0;
proctype A(){
    do
    :: (x==0) -> x++
    od
}
proctype B(){
    do
    :: (x==1) -> x--
    od
}
init {atomic{ run A(); run B()}}
```

```
[](x == 0 || x == 1)
```

SPIN & “Promela” language





P defined as (x == 0 || x ==1)

Microwave Oven Example

Model: **M = (S, S0, R, L)**

- S = (S1, S2, S3, S4)
- S1 is the initial state
- R = ({S1, S2} {S2, S1}, {S1, S4}, {S4, S2}, {S2, S3}, {S3, S2}, {S3, S3}
- L (S1) = {¬close, ¬ start, ¬ cooking}

  L (S2) = {close, ¬ start, ¬ cooking}

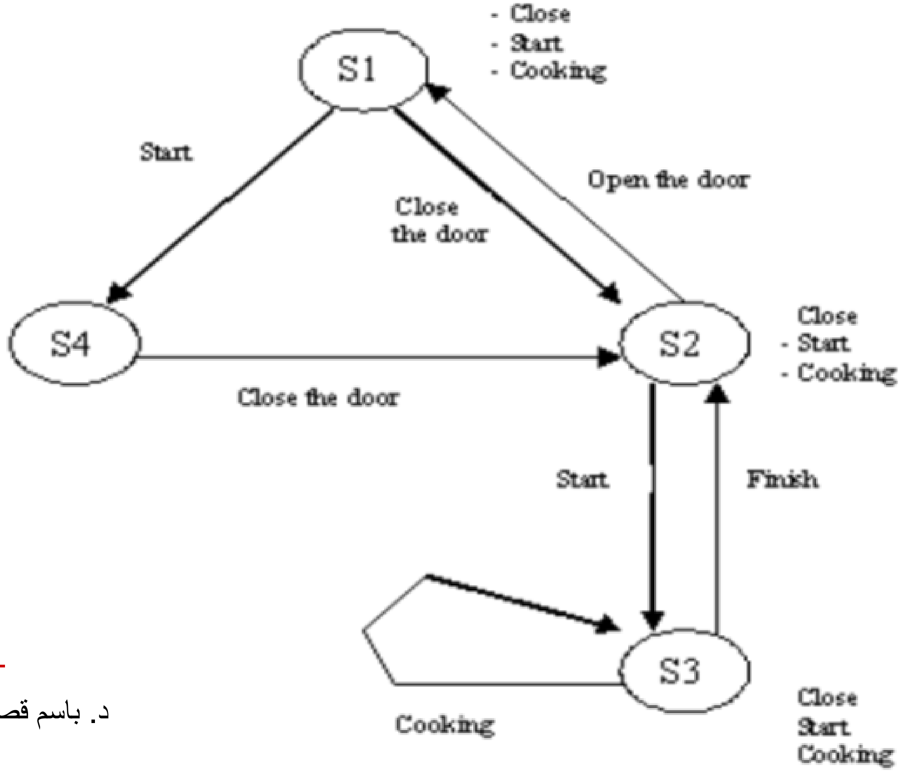
  L (S3) = {close, start, cooking}

  L (S4) = {¬close, start, ¬ cooking}

Specification:

1.AG (start $\rightarrow$ AF cooking)

2. AG ((close $\wedge$ start) $\rightarrow$ AF cooking)



Microwave Oven Example

AG (start → AF cooking)
1) Change formal to ¬EF (start ∧ EG ¬ cooking))
2) From simple partial formulas to the more complicated formulas, until all of the formulas are true.

- S (start) = {S3, S4}
- S (¬cooking) = {S1, S2, S4}
- S (EG ¬ cooking) = {S1, S2, S4} (all conditions lie on a path)
- S (start ∧ EG ¬ cooking) = {S4}
- S (EF (start Ù EG ¬ cooking)) = {S1, S2, S3, S4} (can be followed with S4)
- S (¬ (EF (start ∧ EG ¬ cooking))) = {}

2. AG ((close ∧ start) → AF cooking)
1) change formal to ¬ EF(close ∧ start ∧ EG ¬ cooking)
2) Now the algorithm can be applied to the formula

- S (close)= {S2, S3}
- S (start)= {S3, S4}
- S (¬ cooking) = {S1, S2, S4}
- S (EG ¬ cooking) = {S1, S2, S4}
- S (close ∧ start ∧ EG ¬ cooking) = {}
- S (EF (close ∧ start ∧ EG ¬ cooking) = {}
- S (¬ (EF (close ∧ start v EG ¬ cooking)) = {S1, S2, S3, S4