System Validation

Lecture 4: Linear Temporal Logic

Joost-Pieter Katoen

Formal Methods and Tools Group

E-mail: `katoen@cs.utwente.nl`

URL: `fmt.cs.utwente.nl/courses/systemvalidation/`

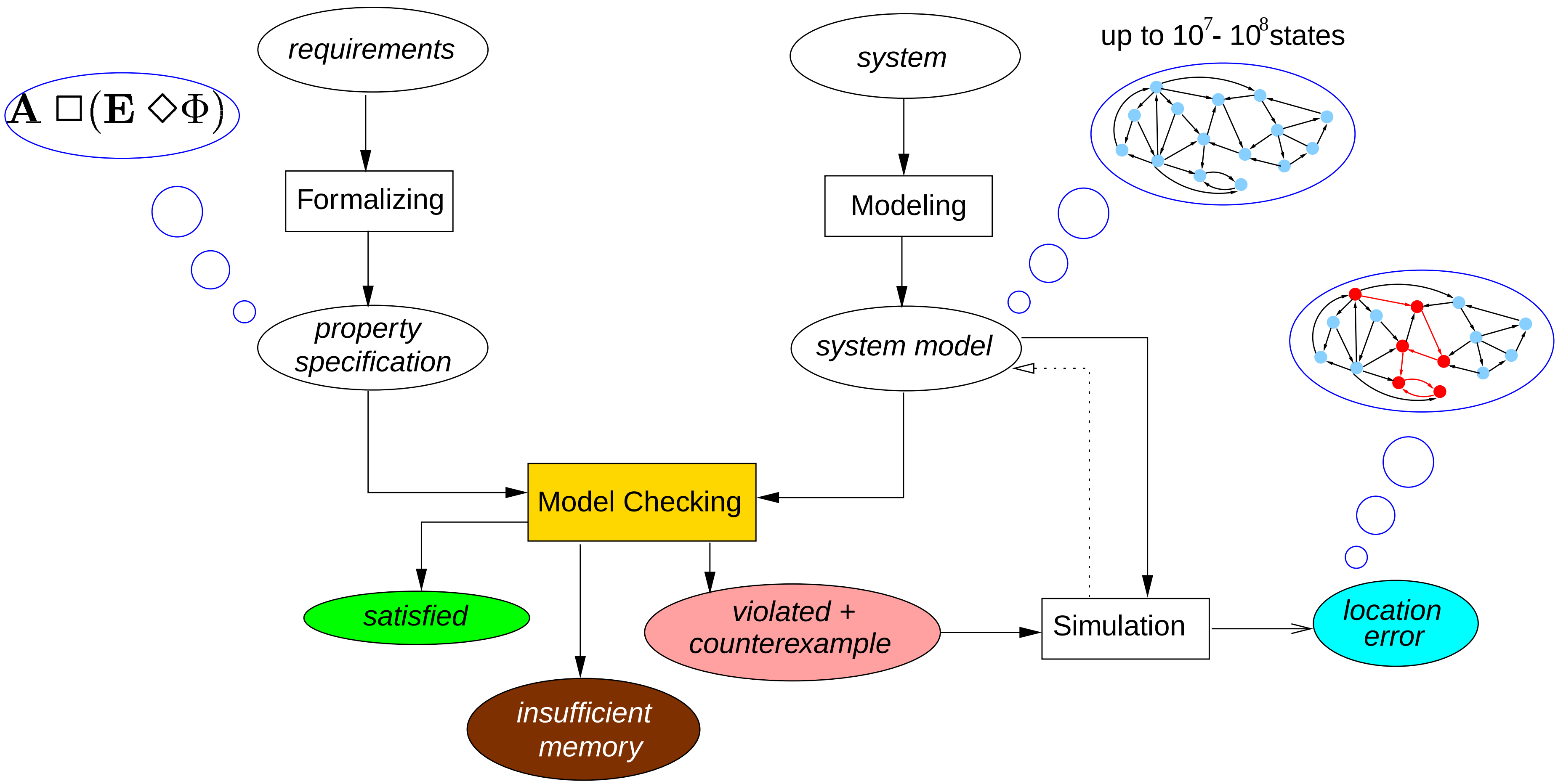
January 17, 2003

Overview of lecture

⇒ *Why temporal logic?*

- Propositional linear temporal logic
  - Syntax and semantics
  - Some formulas express the same
- Specifying properties in PLTL
- Model-checking PLTL in a nutshell
- How to model-check PLTL with SPIN?
- Practical use of PLTL

The model-checking approach



Properties of a mutual exclusion protocol

Typical properties of a mutual exclusion protocol

- it is never the case that two (or more) processes occupy their critical section at the same time

guarantee of mutual exclusion

- whenever a process wants to enter its critical section, it eventually will do so

no unbounded overtaking (absence of individual starvation)

How to specify these properties in an unambiguous and precise way?

Properties of a traffic light

Typical properties of a traffic light:

- once red, the light cannot become immediately green
- eventually the light will be green again
- once red, the light becomes green after being yellow for some time between being red and being green

How to specify these properties in an unambiguous and precise way?

using temporal logic

The need for temporal logic

How are sequential computer programs formally verified?

- property specification in propositional/predicate logic
- set of (compositional) proof rules (e.g., Hoare triples)

Example proof rule for iteration in sequential programs:

$$\frac{\{\, \Phi \wedge b \,\}\; S \;\{\, \Phi \,\}}{\{\, \Phi \,\}\; \mathbf{while}\; b\; \mathbf{do}\; S\; \mathbf{od}\; \{\, \Phi \wedge \neg b \,\}}$$

fine point: partial versus total correctness

how to find *invariants* like Φ?

The need for temporal logic (cont'd)

$$\frac{\{\,\Phi\,\}\,S\,\{\,\Psi\,\} \text{ and } \{\,\Phi'\,\}\,T\,\{\,\Psi'\,\}}{\{\,\Phi \wedge \Phi'\,\}\,S \;\textbf{par}\; T\,\{\,\Psi \wedge \Psi'\,\}}$$

- due to "interaction" of S and T this rule is **not** valid in general
- parallelism inherently leads to non-determinism:

$$x := x + 2 \;\textbf{par}\; x := 0 \text{ versus } (x := x + 1; x := x + 1) \;\textbf{par}\; x := 0$$

- not only begin- and end-states are of importance, but also what happens *during* the computation

pre- and postconditions – as for sequential programs – are insufficient

$\Longrightarrow$ use temporal logic!

Temporal and modal logics

- *modal logics* were originally developed by philosophers to study different modes of truth (“necessarily Φ” or “possibly Φ”)

- *temporal* logic (TL) is a special kind of modal logic where truth values of assertions vary over *time*

- typical modalities (temporal operators) are:
  - “*sometime* Φ” is true if property Φ holds at *some* future moment
  - “*always* Φ” is true if property Φ holds at *all* future moments

- TL is often used to specify and verify *reactive* systems, i.e. systems that continuously interact with the environment (Pnueli, 1977)

Overview of lecture

- Why temporal logic?

⇒ *Propositional linear temporal logic*

  - *Syntax and semantics*
  - *Some formulas express the same*

- Specifying properties in PLTL
- Model-checking PLTL in a nutshell
- How to model-check PLTL with SPIN?
- Practical use of PLTL

Atomic propositions

Atomic propositions – the basic elements of a temporal logic – are boolean expressions p, q, r over

- data variables (integers, lists, sets, etc.) and control variables (locations in programs),
- constants (the integers 0,1,2, . . ., the empty list `[]`, the empty set $\varnothing$, etc.)
- predicate symbols (like $\leqslant$ and $\geqslant$ over integers, `null` over lists, and $\in$ and $\subseteq$ over sets, etc.)

Atomic propositions are the *most elementary* properties one can state

Syntax of linear temporal logic

Propositional Linear Temporal Logic (PLTL) is the smallest set of formulas generated by the rules:

1. each atomic proposition p is a formula
2. if Φ and Ψ are formulas, then $\neg\,\Phi$ and $\Phi\ \vee\ \Psi$ are formulas
3. if Φ is a formula, then $\mathbf{X}\,\Phi$ ("next") is a formula
4. if Φ and Ψ are formulas, then $\Phi\,\mathbf{U}\,\Psi$ ("until") is a formula

$\mathbf{X}$ is sometimes denoted $\bigcirc$

Derived operators

$$\Phi \wedge \Psi \equiv \neg(\neg\Phi \vee \neg\Psi)$$

$$\Phi \Rightarrow \Psi \equiv \neg\Phi \vee \Psi$$

$$\Phi \Leftrightarrow \Psi \equiv (\Phi \Rightarrow \Psi) \wedge (\Psi \Rightarrow \Phi)$$

$$\text{true} \equiv \Phi \vee \neg\Phi$$

$$\text{false} \equiv \neg\,\text{true}$$

$$\mathbf{F}\,\Phi \equiv \text{true}\,\mathbf{U}\,\Phi$$

$$\mathbf{G}\,\Phi \equiv \neg\,\mathbf{F}\,\neg\Phi$$

$\mathbf{F}$ is called “future” (or “eventually”) and is sometimes denoted $\Diamond$

$\mathbf{G}$ is called “globally” (or “always”) and is sometimes denoted $\Box$

Some example PLTL formulas

let AP be the set of atomic propositions over variable x, boolean operators $<$, $\geqslant$ and $=$, and function $x + c$ for constant c

- the following formulas are *legal* PLTL-formulas over AP:
  - $\neg (x + 7 < 21) \ \vee \ (x = 64)$
  - $\mathbf{F}\,(x + 12 \geqslant 10)$
  - $\mathbf{G}\,(x \geqslant 0 \ \wedge \ x < 200)$
  - $x = 10 \Rightarrow \mathbf{X}\,(x \geqslant 10\,\mathbf{U}\,x = 0)$

- the following formulas are *illegal* PLTL-formulas over AP:
  - $\neg (x + x < 21) \ \vee \ (x^3 = 64)$
  - $(x \geqslant 10)\,\mathbf{U}\,(x = y)$

Traffic light properties

- once red, the light cannot become green immediately:

$$\mathbf{G}\,(red \;\Rightarrow\; \neg\,\mathbf{X}\,green)$$

- the green light becomes green eventually: $\mathbf{F}\,green$
- once red, the light becomes green eventually: $\mathbf{G}\,(red \;\Rightarrow\; \mathbf{F}\,green)$
- once red, the light always becomes green eventually after being yellow for some time inbetween:

$$\mathbf{G}\;(red \;\Rightarrow\; (red\,\mathbf{U}\,yellow)\,\mathbf{U}\,green)$$

Interpretation of PLTL

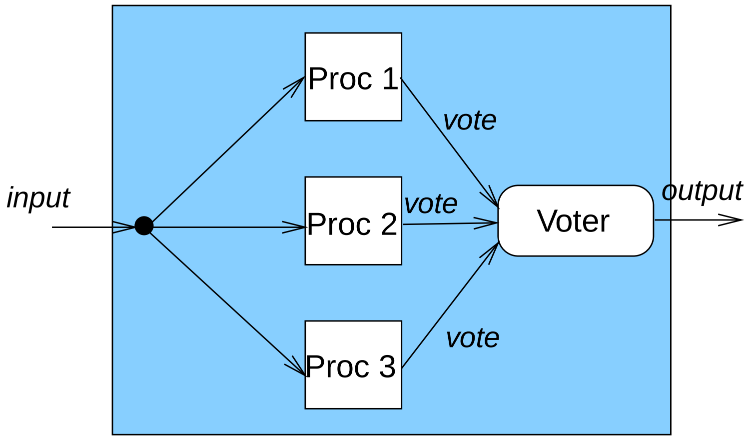
Formal interpretation of PLTL-formulas is defined in terms of a *Kripke structure* $\mathcal{M} = (S, I, R, Label)$ where

- S is a countable set of states,
- $I \subseteq S$ is a set of initial states,
- $R \subseteq S \times S$ is a transition relation with $\forall s \in S.\, (\exists s' \in S.\, (s, s') \in R)$
- $Label : S \longrightarrow 2^{AP}$ is an interpretation function on S.

$Label(s)$ is the set of the atomic propositions $Label(s)$ that are valid in s

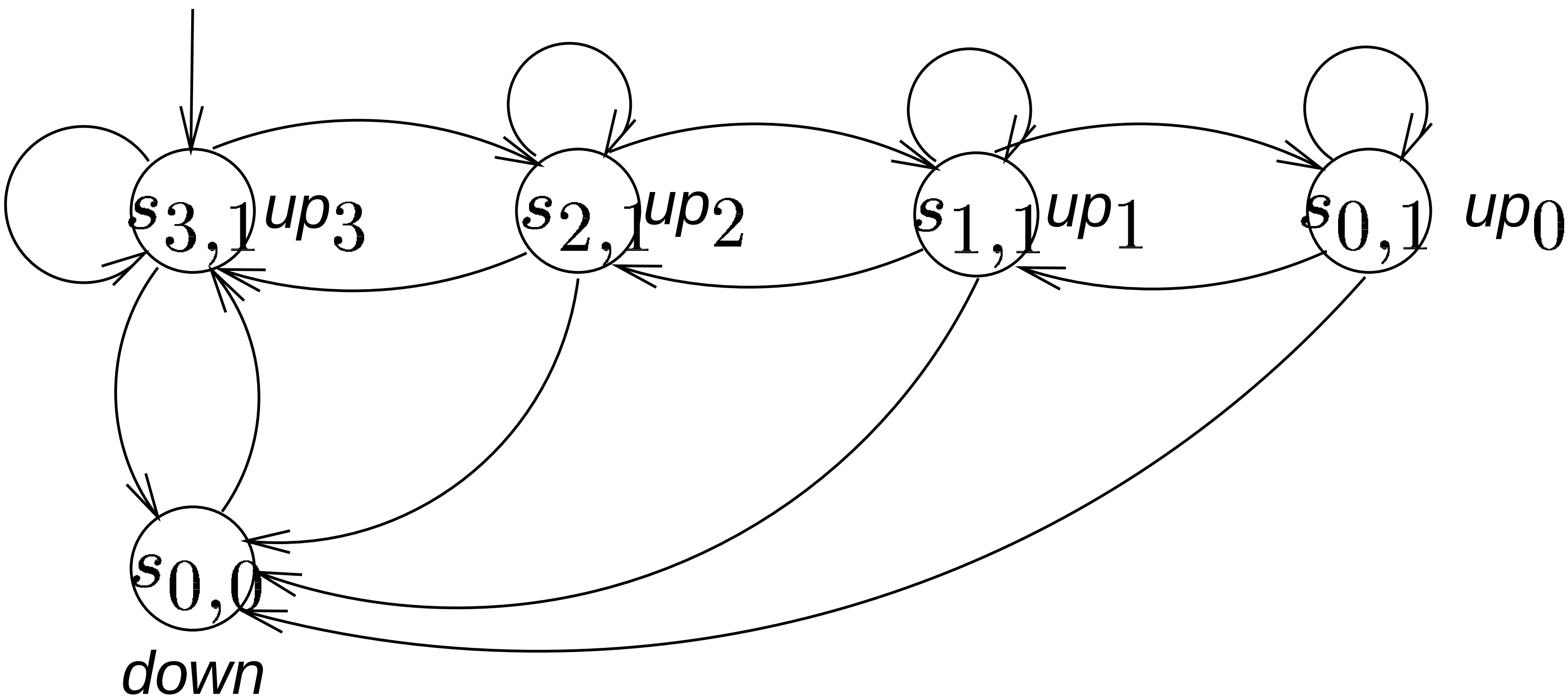
A triple modular redundant system

- 3 processors and a single voter:
  - processors run same program; voter takes a majority vote
  - each component (processor and voter) is failure-prone
  - there is a single repairman for repairing processors and voter



- Modelling assumptions:
  - if voter fails, entire system goes down
  - after voter-repair, system starts "as new"
  - state $=$ (#processors, #voters)

Example Kripke structure



Semantics of PLTL (cont'd)

Defined by a relation $\models$ such that:

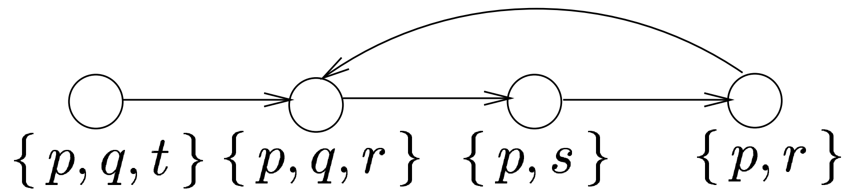
$\sigma \models \Phi$ if and only if formula Φ holds in path σ of structure $\mathcal{M}$

where a *path* in $\mathcal{M}$ is an infinite sequence of states $s_0\, s_1\, s_2 \ldots$ such that $s_0 \in I$ and $(s_i, s_{i+1}) \in R$ for all $i \geqslant 0$. We have:

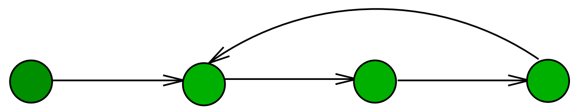
$$
\begin{array}{ll}
\sigma \models p & \text{iff } p \in \mathit{Label}(\sigma[0]) \\
\sigma \models \neg\, \Phi & \text{iff not } (\sigma \models \Phi) \\
\sigma \models \Phi \;\vee\; \Psi & \text{iff } (\sigma \models \Phi) \text{ or } (\sigma \models \Psi) \\
\sigma \models \mathbf{X}\, \Phi & \text{iff } \sigma^1 \models \Phi \\
\sigma \models \Phi\, \mathbf{U}\, \Psi & \text{iff } \exists\, j \geqslant 0.\ \big(\sigma^j \models \Psi \;\wedge\; (\forall\, 0 \leqslant k < j.\, \sigma^k \models \Phi)\big)
\end{array}
$$

where σ^i is the suffix of σ obtained by removing its first i states, i.e., $\sigma^i = s_i\, s_{i+1}\, s_{i+2} \ldots$.

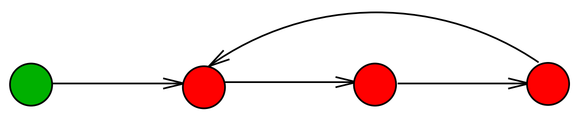
Example of semantics of PLTL



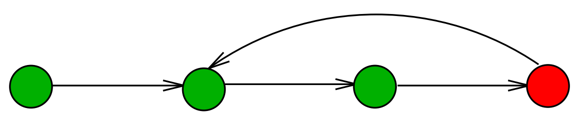
$\sigma \models \mathbf{G}\, p$?



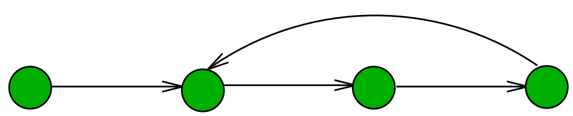
$\sigma \models \mathbf{F}\, t$?



$\sigma \models q\, \mathbf{U}\, s$?



$\sigma \models \mathbf{G}\, \mathbf{F}\, (q\, \mathbf{U}\, s)$?



Model checking, satisfiability and validity

$\mathcal{M} \models \Phi$ if and only if all paths (that start in some initial state) satisfy Φ

The model-checking *problem is: given a Kripke structure $\mathcal{M}$, and a property Φ, do we have $\mathcal{M} \models \Phi$?*

- *Satisfiability problem:* given a property Φ, does there *exist* a model $\mathcal{M}$ such that $\mathcal{M} \models \Phi$?
  - $p \Rightarrow \mathbf{F}\, q$ and $\mathbf{G}\,(p \Rightarrow \mathbf{X}\, q)$ are satisfiable

- *Validity problem:* given a property Φ, do we have for *all* models $\mathcal{M}$ that $\mathcal{M} \models \Phi$?
  - $(p \ \wedge \ \mathbf{G}\,(p \Rightarrow \mathbf{X}\, p)) \ \Rightarrow \ \mathbf{G}\, p$ is valid
  - $p \Rightarrow \mathbf{F}\, q$ and $\mathbf{G}\,(p \Rightarrow \mathbf{X}\, q)$ are not valid

Some important validities for PLTL

Duality rules:

$$\neg\,\mathbf{G}\,\Phi \equiv \mathbf{F}\,\neg\,\Phi$$
$$\neg\,\mathbf{F}\,\Phi \equiv \mathbf{G}\,\neg\,\Phi$$
$$\neg\,\mathbf{X}\,\Phi \equiv \mathbf{X}\,\neg\,\Phi$$

Idempotency rules:

$$\mathbf{G}\,\mathbf{G}\,\Phi \equiv \mathbf{G}\,\Phi$$
$$\mathbf{F}\,\mathbf{F}\,\Phi \equiv \mathbf{F}\,\Phi$$
$$\Phi\,\mathbf{U}\,(\Phi\,\mathbf{U}\,\Psi) \equiv \Phi\,\mathbf{U}\,\Psi$$

Absorption rules:

$$\mathbf{F}\,\mathbf{G}\,\mathbf{F}\,\Phi \equiv \mathbf{G}\,\mathbf{F}\,\Phi$$
$$\mathbf{G}\,\mathbf{F}\,\mathbf{G}\,\Phi \equiv \mathbf{F}\,\mathbf{G}\,\Phi$$

Commutation rule:

$$\mathbf{X}\,(\Phi\,\mathbf{U}\,\Psi) \equiv (\mathbf{X}\,\Phi)\,\mathbf{U}\,(\mathbf{X}\,\Psi)$$

Expansion rules:

$$\Phi\,\mathbf{U}\,\Psi \equiv \Psi \vee (\Phi \wedge \mathbf{X}\,(\Phi\,\mathbf{U}\,\Psi))$$
$$\mathbf{F}\,\Phi \equiv \Phi \vee \mathbf{X}\,\mathbf{F}\,\Phi$$
$$\mathbf{G}\,\Phi \equiv \Phi \wedge \mathbf{X}\,\mathbf{G}\,\Phi$$

Overview of lecture

- Why temporal logic?
- Propositional linear temporal logic
  - Syntax and semantics
  - Some formulas express the same

$\Rightarrow$ *Specifying properties in PLTL*

- Model-checking PLTL in a nutshell
- How to model-check PLTL with SPIN?
- Practical use of PLTL

Specifying properties in PLTL

atomic propositions: variables $m, m', S.out$ and $R.in$ and predicate $\in$

- A message cannot be in both buffers at the same time

$$\mathbf{G}\ \neg\,(m \in S.out\ \wedge\ m \in R.in)$$

- The channel does not lose any messages

$$\mathbf{G}\ (m \in S.out\ \Rightarrow\ \mathbf{F}\,(m \in R.in))$$

what if we would replace $\mathbf{F}$ by $\mathbf{X}\,\mathbf{F}$?

Specifying properties in PLTL (cont'd)

- The channel does not spontaneously generate messages

$$\mathbf{G}\ ((m \notin R.in)\,\mathbf{U}\,(m \in S.out))$$

- The channel is order-preserving, i.e. messages are received in the same order as they were sent

$$\begin{aligned} \mathbf{G}\ (&m \in S.out\ \wedge\ m' \notin S.out\ \wedge\ \mathbf{F}\,(m' \in S.out) \\ &\Rightarrow\ \mathbf{F}\,(m \in R.in\ \wedge\ m' \notin R.in\ \wedge\ \mathbf{F}\,(m' \in R.in)) \\ &) \end{aligned}$$

can we replace $m' \notin S.out\ \wedge\ \mathbf{F}\,(m' \in S.out)$ by $\mathbf{X}\,\mathbf{F}\,(m' \in S.out)$?

Variants of Linear Temporal Logic

Variants can be constructed from PLTL by, for instance:

- allowing finite paths besides infinite paths
- adding past temporal operators, like
  - $\underline{\mathbf{X}}\,\Phi$ is true if Φ holds in the previous state (if any)
  - $\underline{\mathbf{G}}\,\Phi$ is true if Φ holds in all previous states
- adding real-time (i.e., continuous-time) operators, like
  - $\mathbf{F}^{<t}\,\Phi$ is true if Φ holds in some future state within t time units
- adding *first-order* ($\exists$ and $\forall$ over logical variables) or higher-order constructs

Overview of lecture

- Why temporal logic?
- Propositional linear temporal logic
  - Syntax and semantics
  - Some formulas express the same
- Specifying properties in PLTL

$\Rightarrow$ *Model-checking PLTL in a nutshell*

- How to model-check PLTL with SPIN?
- Practical use of PLTL

Conversion of PLTL into automata: theory

A Büchi automaton for PLTL-formulas is a

- a finite-state automaton with transitions labelled with atomic propositions (and negations thereof)
- *accept* states that should be visited infinitely often by a legal computation

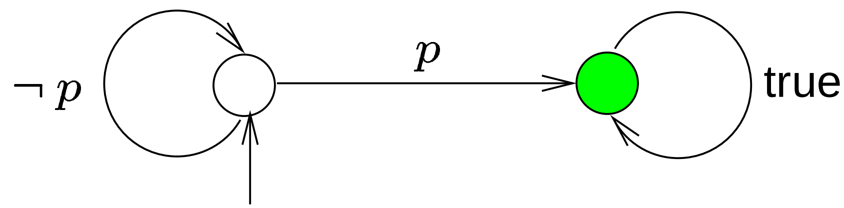
Theorem: (Wolper, Vardi & Sistla, 1983)

For any PLTL-formula Φ a "corresponding" Büchi automaton can be constructed with at most $2^{\vert\Phi\vert}$ states

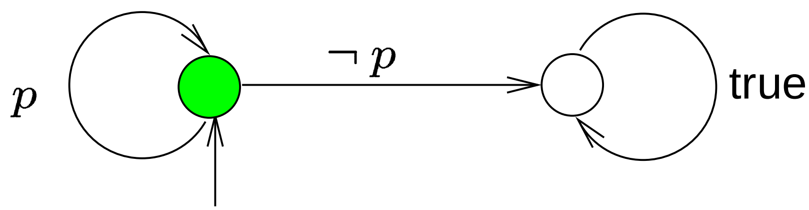
an efficient algorithm for this conversion is implemented in SPIN

Conversion of PLTL into automata: examples

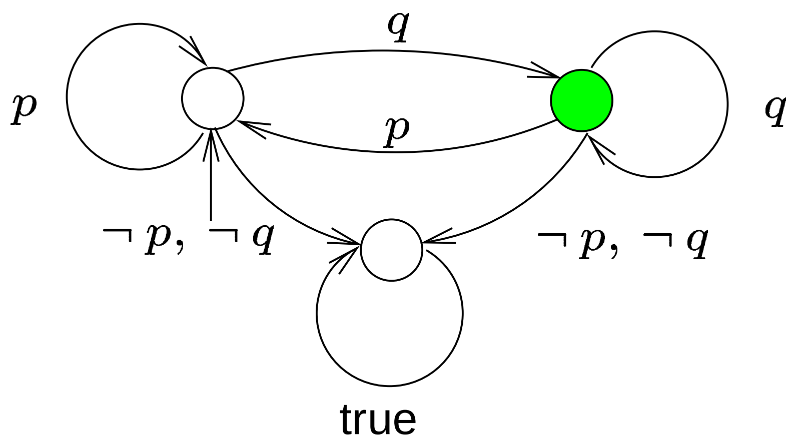
$\mathbf{F}\, p$



$\mathbf{G}\, p$



$\mathbf{G}\,(p\,\mathbf{U}\,q)$



Overview of lecture

- Why temporal logic?
- Propositional linear temporal logic
  - Syntax and semantics
  - Some formulas express the same
- Specifying properties in PLTL
- Model-checking PLTL in a nutshell

$\Rightarrow$ *How to model-check PLTL with SPIN?*

- Practical use of PLTL

PLTL syntax in SPIN

Syntax of PLTL in SPIN property manager:

```
p ::= boolean_expression | proctype[pid]@label

H ::=  p           /* atomic proposition */
    |  !H          /* negation */
    |  H && H      /* conjunction */
    |  H || H      /* disjunction */
    |  H -> H      /* implication */
    |  <>H         /* eventually */
    |  []H         /* always     */
    |  H U H       /* until      */
```

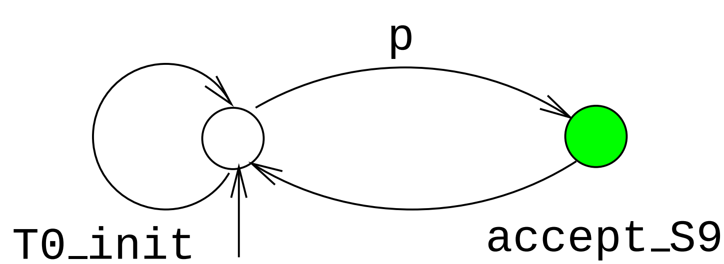
There is no next operator ($\mathbf{X}$) in SPIN

Generating Büchi automata with SPIN

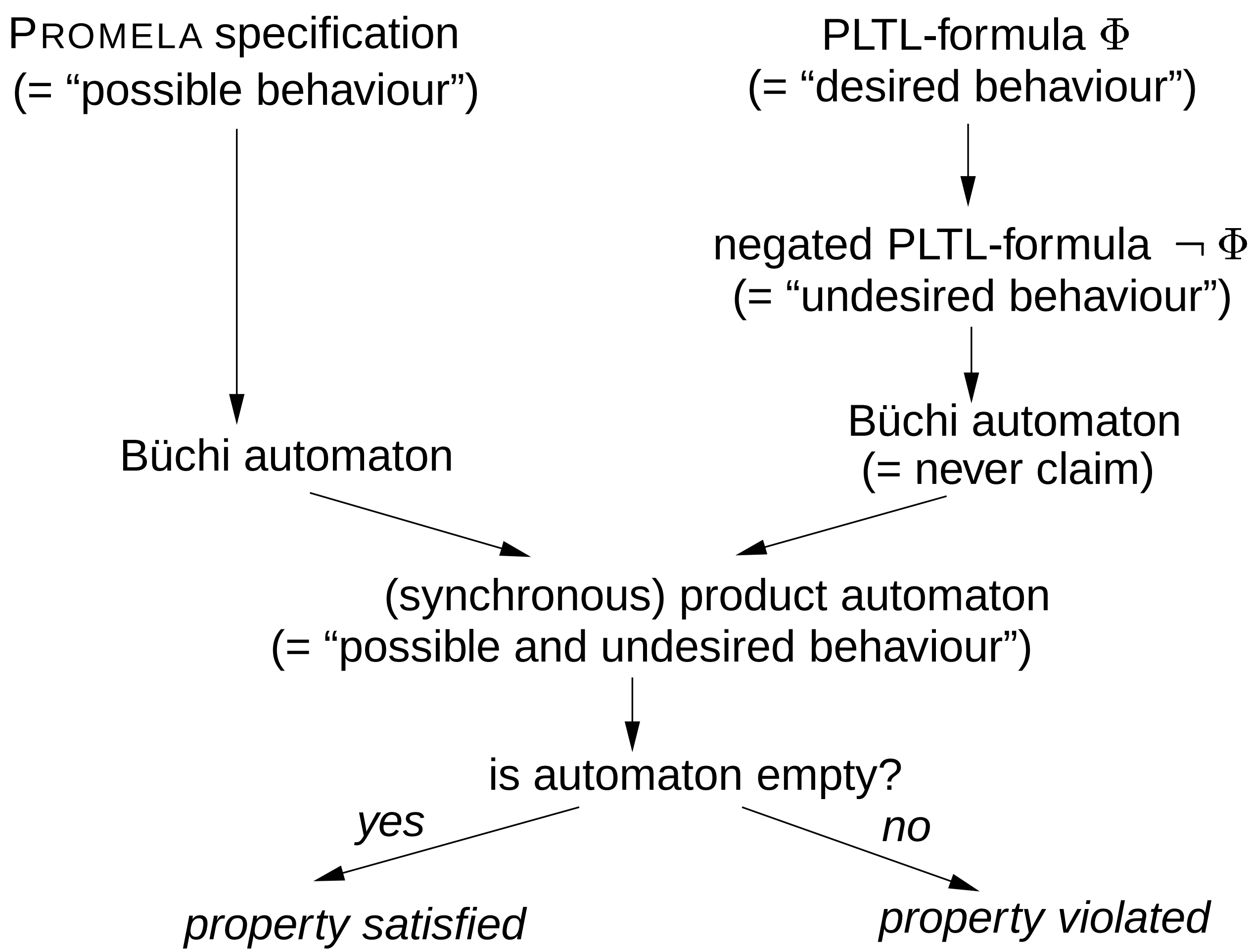
- SPIN automatically converts PLTL-formula Φ into an automaton for $\neg\Phi$

- this is called a *never claim*; for instance for Φ = `![] <> p`:

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



How does SPIN model check PLTL-formulas?



Overview of lecture

- Why temporal logic?
- Propositional linear temporal logic
  - Syntax and semantics
  - Some formulas express the same
- Specifying properties in PLTL
- Model-checking PLTL in a nutshell
- How to model-check PLTL with SPIN?

$\Rightarrow$ *Practical use of PLTL*

Classification of temporal properties

Three main categories of properties: (Lamport, 1977)

1. *Safety* properties state “nothing bad can happen”

   there are never two (or more) processes in their critical section at the same time

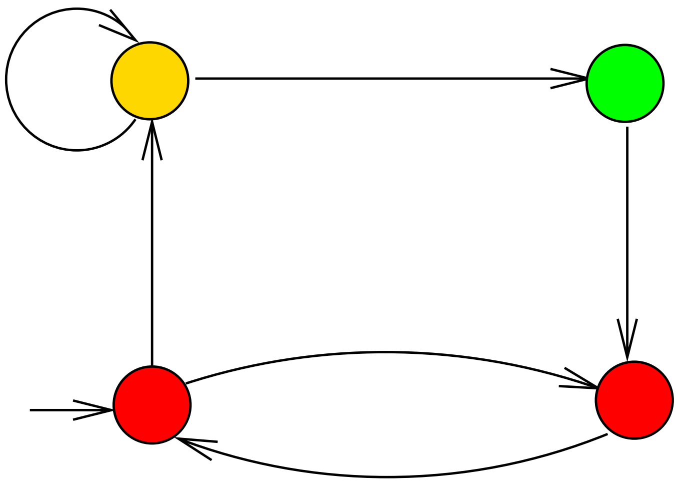
2. *Liveness* properties state “something good will eventually happen”

   if a process wants to enter its critical section, it eventually will do so

3. *Fairness* properties state, for instance, “every (potentially repeating) request is eventually granted”

   if I continuously buy a lottery ticket, I eventually will win a prize

A non-standard traffic light



Classification of example properties

- Safety properties:
  - once red, the light cannot become green immediately

$$\mathbf{G}\,(red \;\Rightarrow\; \neg\,\mathbf{X}\,green)$$

- Liveness properties:
  - once red, the light becomes green eventually: $\mathbf{G}\,(red \;\Rightarrow\; \mathbf{F}\,green)$

- Fairness properties:
  - the light is infinitely often green: $\mathbf{G}\,\mathbf{F}\,green$
  - if the light is red infinitely often, it should be yellow infinitely often

$$\mathbf{G}\,\mathbf{F}\,red \;\Rightarrow\; \mathbf{G}\,\mathbf{F}\,yellow$$

Practical properties in PLTL

- Reachability ("there exists a path such that ... is reached")
  - negated reachability $\mathbf{F}\,\neg\Psi$
  - conditional reachability $\Phi\,\mathbf{U}\,\neg\Psi$
  - reachability from any state not expressible
- Safety ("something bad never happens")
  - simple safety $\mathbf{G}\,\neg\Phi$
  - conditional safety $(\Phi\,\mathbf{U}\,\Psi) \;\vee\; \mathbf{F}\,\Phi$
- Liveness $\mathbf{G}\,(\Phi \;\Rightarrow\; \mathbf{F}\,\Psi)$
- Fairness $\mathbf{G}\,\mathbf{F}\,\Phi$ and others

How to use PLTL in practice?

Capture commonly-used types of formulas in specification patterns

- *Specification pattern*: generalized description of a commonly occurring requirement on the permissable paths in a model
  - parameterizable: only state-formulas to be instantiated
  - high-level: no detailed knowledge of TL is required
  - formalism-independent: by mappings onto TL at hand
- *Scope* of a pattern: the extent of the computation over which the pattern must hold, such as
  - global: the entire computation
  - after: the computation after a given state
  - between: any part of the computation from one state to another

Most commonly used specification patterns for PLTL

Investigation of 555 requirement specifications reveals that the following patterns are most widely used for P, Q and R state-formulas: (Dwyer et al, 1998)

pattern	*scope*	*PLTL-formula*	*frequency*
response	global	$\mathbf{G}\,(P \;\Rightarrow\; \mathbf{F}\,Q)$	43.4 %
universality	global	$\mathbf{G}\,P$	19.8 %
absence	global	$\mathbf{G}\,\neg\,P$	7.4 %
precedence	global	$\mathbf{G}\,\neg\,P \;\vee\; \neg\,P\,\mathbf{U}\,Q$	4.5 %
absence	between	$\mathbf{G}\,((P \;\wedge\; \neg\,Q \;\wedge\; \mathbf{F}\,Q)$	
		$\Rightarrow\; (\neg\,R\,\mathbf{U}\,Q))$	3.2 %
absence	after	$\mathbf{G}\,(Q \;\Rightarrow\; \mathbf{G}\,\neg\,P)$	2.1 %
existence	global	$\mathbf{F}\,P$	2.1 %
			$\approx$ 80 %

more info at: `www.cis.ksu.edu/santos/spec-patterns/`