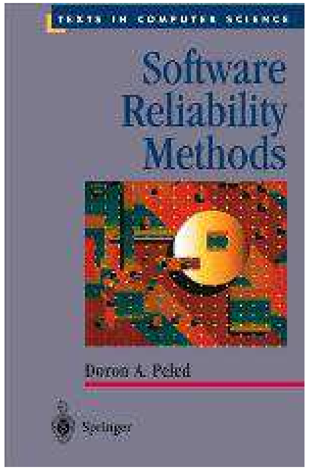
Analysis: model checking LTL

Prof.ssa Susanna Donatelli
Universita' di Torino
www.di.unito.it
susi@di.unito.it

Reference material books:



Prof. Doron A. Peled
(University of Warwick, UK)

Concepts, Algorithms, and Tools

for

Model Checking

Joost-Pieter Katoen

Lehrstuhl für Informatik VII

Friedrich-Alexander Universität Erlangen-Nürnberg

Lecture Notes of the Course

"Mechanised Validation of Parallel Systems"

(course number 10359)

Semester 1998/1999

Prof. Jost-Pieter Katoen
(University of Aachen, D)

Acknowledgements

Transparencies adapted from the course notes and trasparencies of

- Prof. Doron A. Peled, University of Warwick (UK) and Bar Ilan University (Israel) http://www.dcs.warwick.ac.uk/~doron/srm.html
- Prof. Jost-Pieter Katoen, University of Aachen (Germany)
- Dr. Jeremy Sproston, Universita' di Torino (Italy)

Our course - recall

Concentrate on distributed systems (as inherently protocols are)

Learn several formalisms to model system and properties (automata, process algebras, Petri Nets, temporal logic, timed automata).

Learn advantages and limitations, in order to choose the right methods and tools.

Learn how to combine existing formalisms and existing "solution" methods.

Flowchart of analysis material

1. Basic properties
2. RG analysis
3. Structural analysis (on PN)
4. Reduction rules (PN)
5. Equivalences (PA)
6. Model checking
   - definition of linear logic LTL and its model checking algorithm
   - definition of branching logic CTL and its model checking algorithm

Some important points

- *Reachable* states: obtained from an initial state through a sequence of enabled transitions.
- *Executions*: the set of maximal paths (finite or terminating in a node where nothing is enabled).
- *Nondeterministic choice*: when more than a single transition is enabled at a given state. We have a nondeterministic choice when at least one node at the state graph has more than one successor.

useful:The interleaving model

- An execution is a finite or infinite sequence of states $s_0, s_1, s_2, \ldots$
- The initial state satisfies the initial condition, I.e., $I(s_0)$.
- Moving from one state s_i to s_{i+1} is by executing a transition $e \rightarrow t$:
  - $e(s_i)$, I.e., s_i satisfies e.
  - s_{i+1} is obtained by applying t to s_i.
- Lets assume all sequences are **infinite** by extending finite ones by "**stuttering**" the last state.

Useful: A transition system

- A (finite) set of variables V.
- A set of states Σ.
- A (finite) set of transitions T, each transition e→t has
  - an enabling condition e and a transformation t.
- An initial condition *I*.
- Denote by R(s, s’) the fact that s’ is a successor of s.

Linear temporal logic (LTL)

- LTL has been introduced by Pnueli in 1977

- It is a logic to describe systems in terms of linear executions: total order between events

- Interpretation: over an **execution**, later over **all executions**.

- LTL is very popular in industry mainly thanks to the LTL model checker SPIN (by Holzmann et al. in the 90's)

LTL: Syntax

$\varphi ::= (\varphi) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi\, \mathcal{U} \varphi \mid \Box\varphi \mid \Diamond\varphi \mid O\varphi \mid p$

$p \in AP$

φ ::= (φ) | ¬φ | φ /\ φ | φ \/ φ | φ *U* φ |
[] φ | <>φ |O φ | *p*

[] φ (or Gφ)–– "box", "always", "forever"

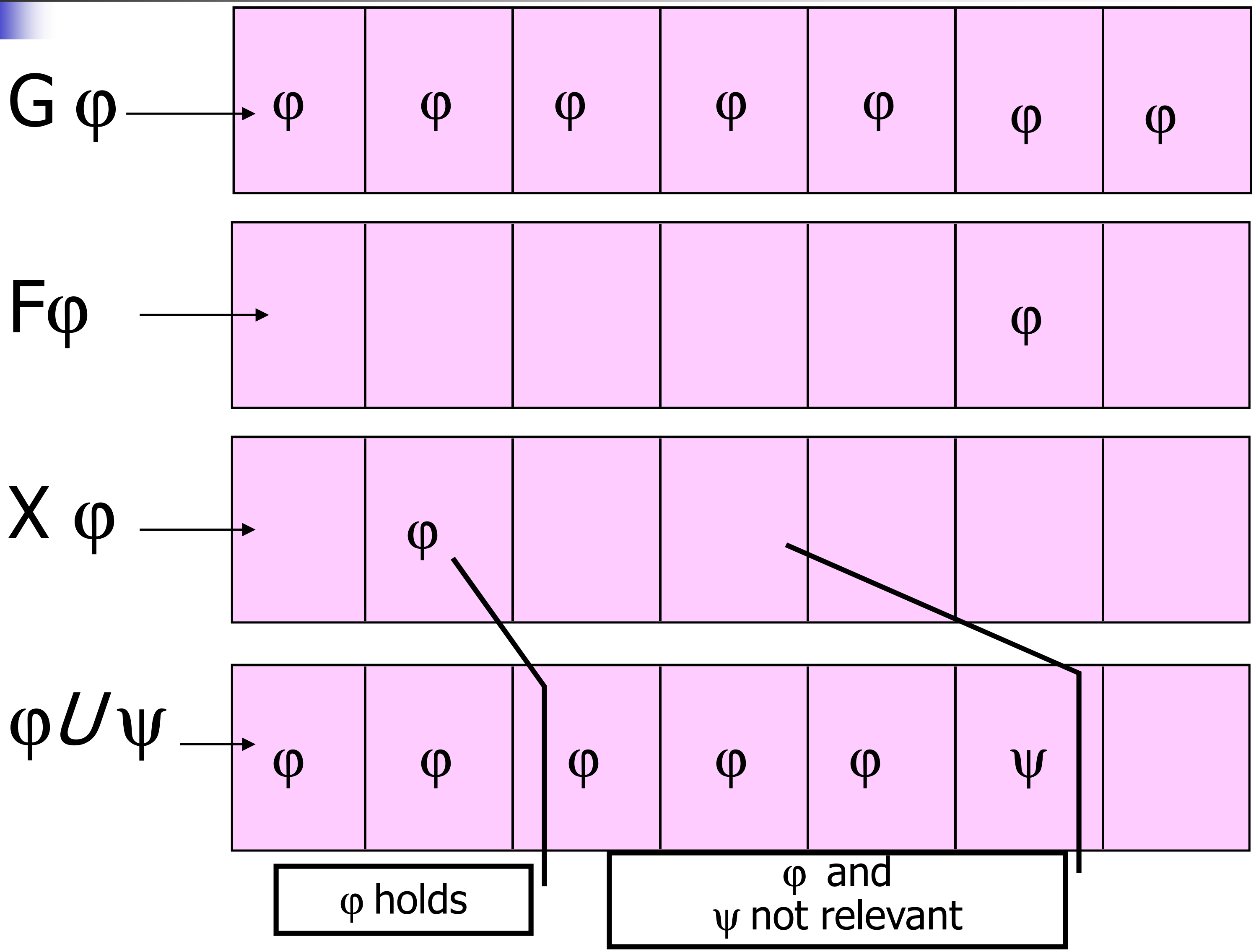
<>φ (or Fφ) –– "diamond", "eventually",sometimes"

O φ (or Xφ)–– "nexttime"

φ *U* ψ –– "until"

Propositions *p*, *q*, *r*, … Each represents some state property (x>y+1, z=t, at_CR, etc.)

Semantics *over **suffixes** of execution*



Can discard some operators

- Instead of Fp, write *true U p*.
- Instead of Gp, we can write ¬(F¬p), or ¬(*true U* ¬p).
  Because Gp=¬¬Gp.
  ¬Gp means it is not true that p holds forever, or at some point ¬p holds or F¬p.

Combinations

- $G\overset{a}{\underbrace{Fp}}$ "*p* will happen infinitely often"

  $Ga \quad \sigma \models Ga$ sse $\forall i \geq 0 \ \sigma^i \models a$ sse $\forall i \geq 0 \ \sigma^i \models Fp$ sse

  $\forall i \geq 0 \ \exists j \geq 0 : \sigma^{i+j} \models p$

- FG*p* "*p* will happen from some point forever".

- (GF*p*) → (GF*q*) "If *p* happens infinitely often, then *q* also happens infinitely often".

Formal semantic definition - Peled's book

- Let σ be a sequence $s_0\ s_1\ s_2\ \ldots$
- Let σ^i be a suffix of σ: $s_i\ s_{i+1}\ s_{i+2}\ \ldots$ ($\sigma^0 = \sigma$)
- $\sigma^i \models p$, where p is a proposition, if $s_i \models p$.
- $\sigma^i \models \varphi /\backslash \psi$ if $\sigma^i \models \varphi$ and $\sigma^i \models \psi$.
- $\sigma^i \models \varphi \bigvee \psi$ if $\sigma^i \models \varphi$ or $\sigma^i \models \psi$.
- $\sigma^i \models \neg\varphi$ if it is not the case that $\sigma^i \models \varphi$.
- $\sigma^i \models X\varphi$ if $\sigma^{i+1} \models \varphi$.
- $\sigma^i \models F\varphi$ if for some $j \geq i$, $\sigma^j \models \varphi$.
- $\sigma^i \models G\varphi$ if for each $j \geq i$, $\sigma^j \models \varphi$.
- $\sigma^i \models \varphi\, U\, \psi$ if for some $j \geq i$, $\sigma^j \models \psi$.
  and for each $i \leq k < j$, $\sigma^k \models \varphi$.

Some relations:

- $G(\varphi \wedge \psi)=(G\varphi) \wedge (G\psi)$
- But $F(\varphi \wedge \psi) \neq (F\varphi) \wedge (F\psi)$

		ψ		φ		

- $F(\varphi \vee \psi)=(F\varphi) \vee (F\psi)$
- But $G(\varphi \vee \psi) \neq (G\varphi) \vee (G\psi)$

ψ	φ ψ	ψ	φ ψ	φ	ψ	φ

What about

- $(GF\varphi) \wedge (GF\psi) = GF(\varphi \wedge \psi)$? — No, just $\leftarrow$
- $(GF\varphi) \vee (GF\psi) = GF(\varphi \vee \psi)$? — Yes!!!
- $(FG\varphi) \wedge (FG\psi) = FG(\varphi \wedge \psi)$? — Yes!!!
- $(FG\varphi) \vee (FG\psi) = FG(\varphi \vee \psi)$? — No, just $\rightarrow$

Formal semantic definition - Peled's book

LTL formulas are interpreted over a linear model: infinite sequences over S

Given a sequence σ and a formula φ, we define the satisfaction relation $|=$, as $(\sigma, \varphi) \in |=$, and we write $\sigma \models \varphi$.

Formal semantic definition - Peled's book

- Let σ be a sequence $s_0\ s_1\ s_2\ \ldots$
- Let σ^i be a suffix of σ: $s_i\ s_{i+1}\ s_{i+2}\ \ldots\ (\sigma^0 = \sigma)$
- $\sigma^i \models p$, where p is a proposition, if $s_i \models p$.
- $\sigma^i \models \varphi \wedge \psi$ if $\sigma^i \models \varphi$ and $\sigma^i \models \psi$.
- $\sigma^i \models \varphi \vee \psi$ if $\sigma^i \models \varphi$ or $\sigma^i \models \psi$.
- $\sigma^i \models \neg\varphi$ if it is not the case that $\sigma^i \models \varphi$.
- $\sigma^i \models X\varphi$ if $\sigma^{i+1} \models \varphi$.
- $\sigma^i \models F\varphi$ if for some $j \geq i$, $\sigma^j \models \varphi$.
- $\sigma^i \models G\varphi$ if for each $j \geq i$, $\sigma^j \models \varphi$.
- $\sigma^i \models \varphi\, U\, \psi$ if for some $j \geq i$, $\sigma^j \models \psi$.
  and for each $i \leq k < j$, $\sigma^k \models \varphi$.

Formal semantic definition - Katoen's book

LTL formulas are interpreted over a linear model

$$\mathcal{M}(S, R, L)$$

where

- S is a set of states
- R:S-->S is a successor function (total function), assigning to s its unique successor R(s)
- L:S-->2^{AP}, is a labelling function

$\mathcal{M}$ can be seen as an infinite sequence over S

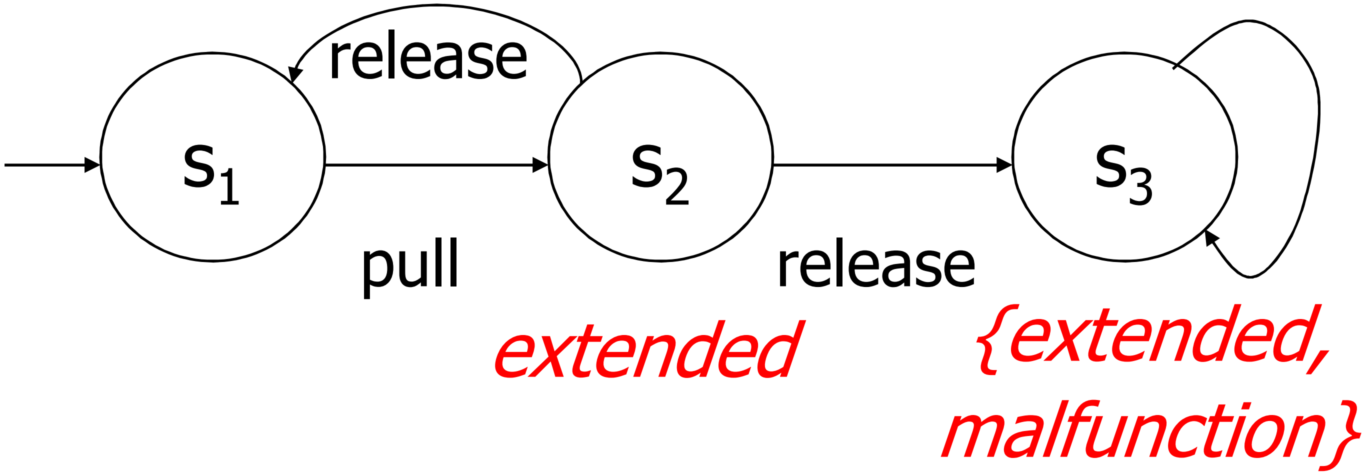
Given a model $\mathcal{M}$ and a formula φ, we define the satisfaction relation as $(\mathcal{M},s,\varphi) \in \;|=$, and we write $(\mathcal{M},s) \;|= \varphi$.

Formal semantic definition - Katoen's book

Let $R^0(s) = s$ and $R^{n+1}(s) = R(R^n(s))$, for any $n > 0$

- $s \models p$, where p a proposition, if $p \in L(s)$.
- $s \models \varphi \wedge \psi$ if $s \models \varphi$ and $s \models \psi$.
- $s \models \varphi \vee \psi$ if $s \models \varphi$ or $s \models \psi$.
- $s \models \neg\varphi$ if $\neg(s \models \varphi)$.
- $s \models F\varphi$ if $\exists\ j \geq 0$: $R^j(s) \models \varphi$.
- $s \models X\varphi$ if $R(s) \models \varphi$.
- $s \models G\varphi$ if for each $j \geq 0$, $R^j(s) \models \varphi$.
- $s \models \varphi\, U\, \psi$ if for some $j \geq 0$, $R^j(s) \models \psi$.
  and for each $0 \leq k < j$, $R^k(s) \models \varphi$.

Spring Example



$r_0 = s_1\ s_2\ s_1\ s_2\ s_1\ s_2\ s_1\ \ldots$ r0

$r_1 = s_1\ s_2\ s_3\ s_3\ s_3\ s_3\ s_3\ \ldots$ r1

$r_2 = s_1\ s_2\ s_1\ s_2\ s_3\ s_3\ s_3\ \ldots$ r2

…

Esempi dal testo di Katoen

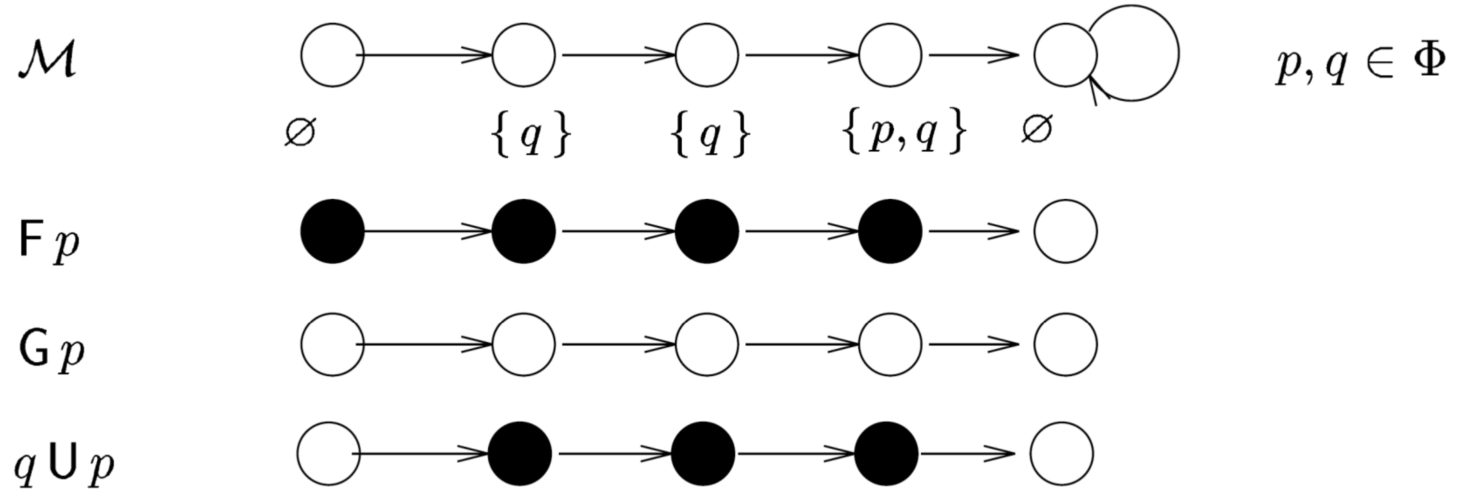


Figure 2.1: Example of interpretation of PLTL-formulas (I)

Esempi dal testo di Katoen

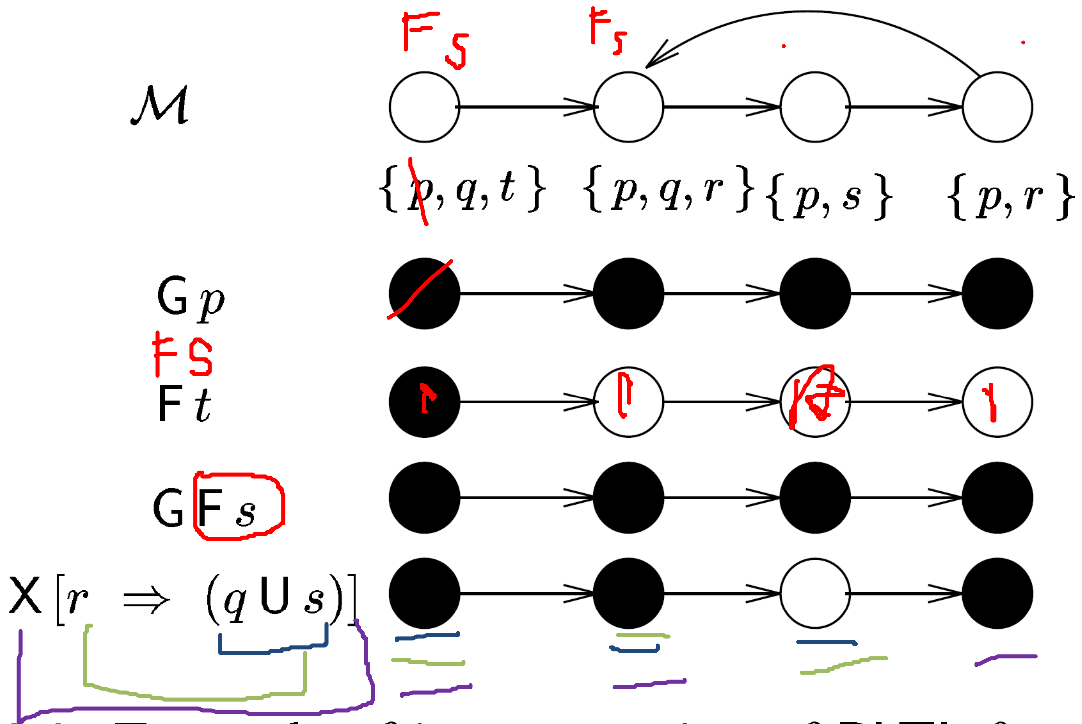
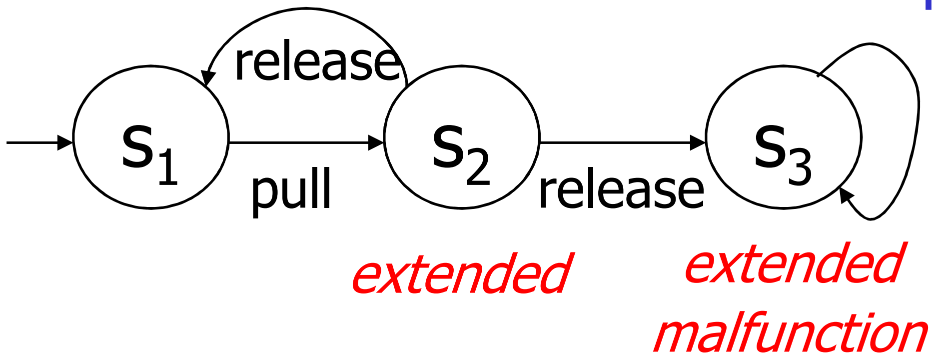


Figure 2.2: Example of interpretation of PLTL-formulas (II)

LTL satisfaction by a single sequence

$r_2 = s_1\ s_2\ s_1\ s_2\ s_3\ s_3\ s_3\ \ldots$



$r_2 \models$ extended ??

$r_2 \models$ X extended ??

$r_2 \models$ X X extended ??

$r_2 \models$ F extended ??

$r_2 \models$ G extended ??

$r_2 \models$ FG extended ??

$r_2 \models \neg$ FG extended ??

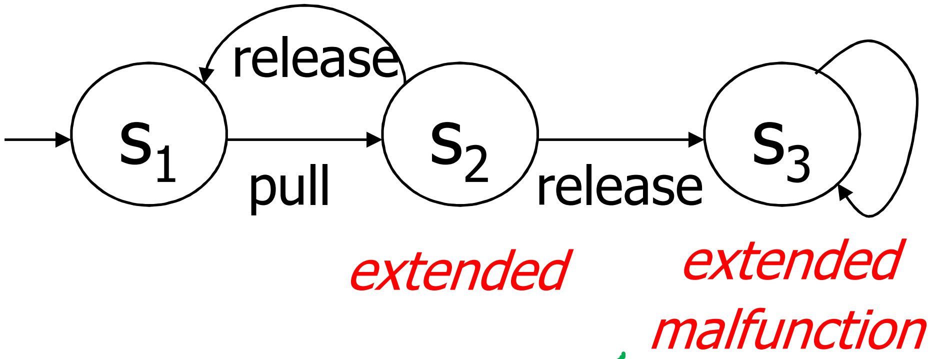
$r_2 \models$ ($\neg$extended) *U* malfunction ??

$r_2 \models$ G($\neg$extended->X extended)

G(extended $\bigvee$ X extended)

LTL satisfaction by a system

$\forall \sigma \in L(S_1)$



P |= extended ?? ✗

P |= X extended ?? ✓

P |= X X extended ?? ✗

P |= F extended ?? ✓

P|= G extended ?? ✗

P |= FG extended ?? ✗

P |= ¬ FG extended ?? ✗

P |= (¬extended) *U* malfunction ?? ✗

P |= G(¬extended->Xextended) ?? ✓

Exercise

Try at home over Dekker's algorithm:

- The processes alternate in entering their critical sections.
- Each process that tries to enter the critical section will eventually be allowed to enter it (responsiveness).
- Each process enters its critical section infinitely often.
- When a process enters its trying section, it will remain there, unless it progresses to its critical section

Traffic light example

Green → Yellow → Red

Always has exactly one light:

$G(gr \vee ye \vee re)$

Correct specification?

$G(\neg(gr \wedge ye) \wedge \neg(ye \wedge re) \wedge \neg(re \wedge gr) \wedge (gr \vee ye \vee re))$

Correct change of color:

$G((gr\, U\, ye) \vee (ye\, U\, re) \vee (re\, U\, gr))$

Correct specification?

What if colour does not change?

Another kind of traffic light

Green → Yellow → Red → Yellow

First attempt:

$$\cancel{G(((gr \vee re)\ U\ ye) \vee (ye\ U\ (gr \vee re)))}$$

Correct specification:

$$
\begin{aligned}
G(\ & (gr \rightarrow (gr\ U\ (ye \wedge (ye\ U\ re)))) \\
& \wedge (re \rightarrow (re\ U\ (ye \wedge (ye\ U\ gr)))) \\
& \wedge (ye \rightarrow (ye\ U\ (gr \vee re))))
\end{aligned}
$$

LTL properties and PN

We can specify the traffic light as a (very) simple PN, and then check the previous properties.

What is needed: a language for the definition of AP

La specifica del semaforo è equivalente a:

G((gr /\ X ye) \/ (ye /\ X re) \/ (re /\ X gr)) ?

Properties of sequential programs

- init-when the program starts and satisfies the initial condition.
- finish-when the program terminates and nothing is enabled.
- q: the correct function has been computed
- Partial correctness: init/\G(finish→q)
- Termination: init/\F finish
- Total correctness: init/\F(finish/\ q)
- Invariant: init/\Gp

The communication channel

- Sender S, output buffer S.out, input buffer R.in, Receiver R

- prop1: a message cannot be in both buffers at the same time

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

- prop2: the channel does not loose messages (whatever is in S.out will be in R.in)

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

The communication channel

- Prop 2, cont.: since m can't be in both,

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

- prop3: the channel is order preserving

$$\mathsf{G}\ (m \in S.out \ \wedge\ \neg\, m' \in S.out \ \wedge\ \mathsf{F}\,(m' \in S.out)$$
$$\Rightarrow\ \mathsf{F}\,(m \in R.in \ \wedge\ \neg\, m' \in R.in \ \wedge\ \mathsf{F}\,(m' \in R.in)))$$

- prop4: the channel does not spontaneously generate messages

$$\mathsf{G}\ \left(m \in R.in \ \Rightarrow\ \mathsf{F}^{-1}\,(m \in S.out)\right)$$

$$\mathsf{G}\ ((\neg\, m \in R.in)\,\mathsf{U}\,(m \in S.out))$$

Correct specification?

Model-Checking LTL

The model-checking problem is:

given a (finite) model M, a state s, and a property ψ, do we have $s \models \psi$?

It is different from satisfiability: given a formula ψ, does it exists a model and a state s, such that: $(M,s) \models \psi$?

Satisfiability is decidable for LTL
--> model-checking is decidable

Model-Checking LTL

The validity problem is:

given a property ψ, do we have for all models M, and for all states s in these models,that $(M,s) \models \psi$?

Logically this is equivalent to the satisfiability of $\neg\psi$

Note: Valid formula are the basis for re-writing rules

Model-Checking LTL

Validity can be based on the semantics, or we can use the syntax and a set of proof rules that allows the re-writing, at a syntactical level, of LTL formulas into semantically equivalent LTL formula

Rewriting rules are of the form $\psi=\varphi$, and they need to be valid (*sound*)

(for all M and s: $(M,s)|=\psi$ iff $(M,s)|=\varphi$?

Ex: $GG\varphi = G\varphi$, or $FGF\varphi = GF\varphi$

Some sound rules for LTL

Duality axioms:

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

Idempotency axioms:

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

Absorption axioms:

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

Commutation axiom:

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

Expansion axioms:

$$\Phi\,\mathsf{U}\,\Psi \equiv \Psi \lor (\Phi \land \mathsf{X}\,(\Phi\,\mathsf{U}\,\Psi))$$
$$\mathsf{F}\,\Phi \equiv \Phi \lor \mathsf{X}\,\mathsf{F}\,\Phi$$
$$\mathsf{G}\,\Phi \equiv \Phi \land \mathsf{X}\,\mathsf{G}\,\Phi$$

Used for recursive model checking

Provate con il tool SPOT https://spot.lrde.epita.fr/app/ a tradurre formule in Automi di Buchi. Per ogni formula il traduttore produce un automa (visibile in modo grafico) che accetta tutte e sole le sequenze che soddisfano la formula. Sono automi di Buchi, quindi la regola di accettazione non è quella degli automi a stati finiti, ma si definisce che una sequenza infinita è accettata da un automa di Buchi se il cammino di accettazione della sequenza nell'automa passa infinitamente spesso dagli stati accettanti.

Osservate che le coppie di formule della pagina precedente producono lo stesso Automa di Buchi. La sintassi di SPOT usa ! e & per OR e AND (rispettivamente)

Provate anche coppie di formule che volete confrontare. Per esempio potremmo chiederci se F (p U q) e F (s U q) sono equivalenti. Lo sono??

Model-Checking LTL

Commonly used formulas:

pattern	*category*	*PLTL-formula*	*frequency*
response	liveness	$\mathsf{G}\,(\Phi \;\Rightarrow\; \mathsf{F}\,\Psi)$	43.4 %
universality	safety	$\mathsf{G}\,\Phi$	19.8 %
absence	negated reachability	$\mathsf{G}\,\neg\,\Phi$	7.4 %
precedence	liveness	$\mathsf{G}\,(\neg\,\Phi\,\mathsf{W}\,\Psi)$	4.5 %
absence		$\mathsf{G}\,((\Phi \;\wedge\; \neg\,\Psi \;\wedge\; \mathsf{F}\,\Psi)$	
		$\Rightarrow \;(\neg\,\Phi'\,\mathsf{U}\,\Psi))$	3.2 %
absence	safety	$\mathsf{G}\,(\Psi \;\Rightarrow\; \mathsf{G}\,\neg\,\Phi)$	2.1 %
existence	liveness	$\mathsf{F}\,\Phi$	2.1 %
			$\approx$ 80 %

Unless operator:

$\varphi \mathsf{W} \psi == \mathsf{G}\varphi \vee \varphi \mathsf{U} \psi$

Practical properties in LTL

- Reachability
  - Negated reachability $F \neg \psi$
    - in tutti i cammini non riesco a raggiungere q (quindi q non e' mai raggiungibile)
  - Conditional reachability $\varphi U \psi$
  - Reachability (exists a path, as for home states)

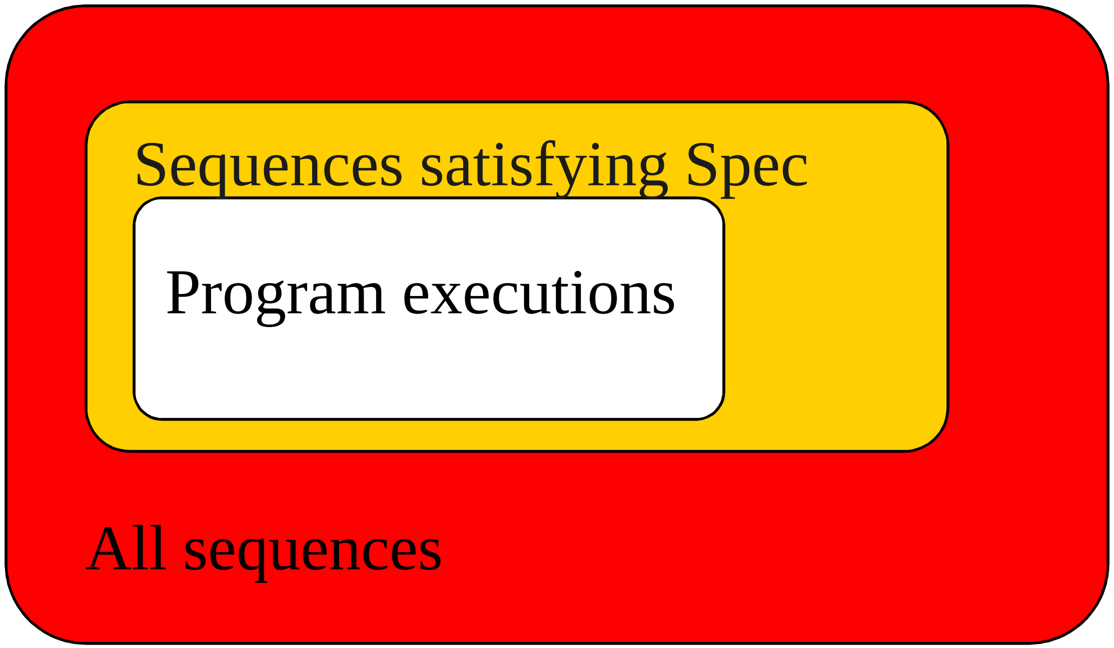
    not expressible

    posso solo dire che phi e' raggiungibile in tutte le esecuzioni
- Safety
  - Simple safety $G \neg \psi$
  - Conditional safety $\varphi U \psi \vee F \varphi$
- Liveness $G (\varphi \Rightarrow F \psi)$ *and others*
- Fairness $GF \psi$ *and others*

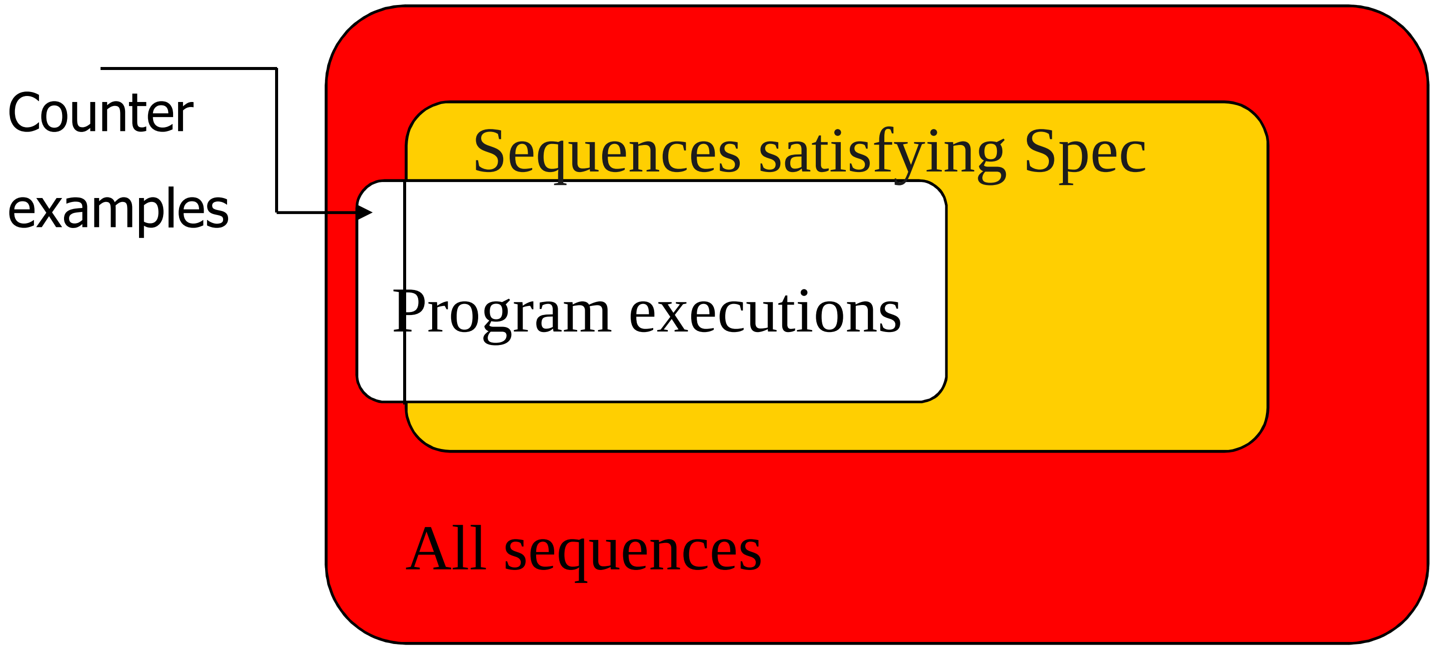
Model checking LTL

- We want to find a correctness condition for a model to satisfy a specification.
- Language of a model: L(Model)
- Language of a specification: L(Spec).
- We need: L(Model) $\subseteq$ L(Spec).

Correctness



Incorrectness



How to prove correctness?

- Show that L(Model) $\subseteq$ L(Spec).
- Equivalently:
  Show that L(Model) $\cap$ $\overline{\text{L(Spec)}}$ = Ø.
- Model is specified as a Buchi automata, Spec can be specified as a Buchi automata *automatically translated* from LTL

Model checking schema

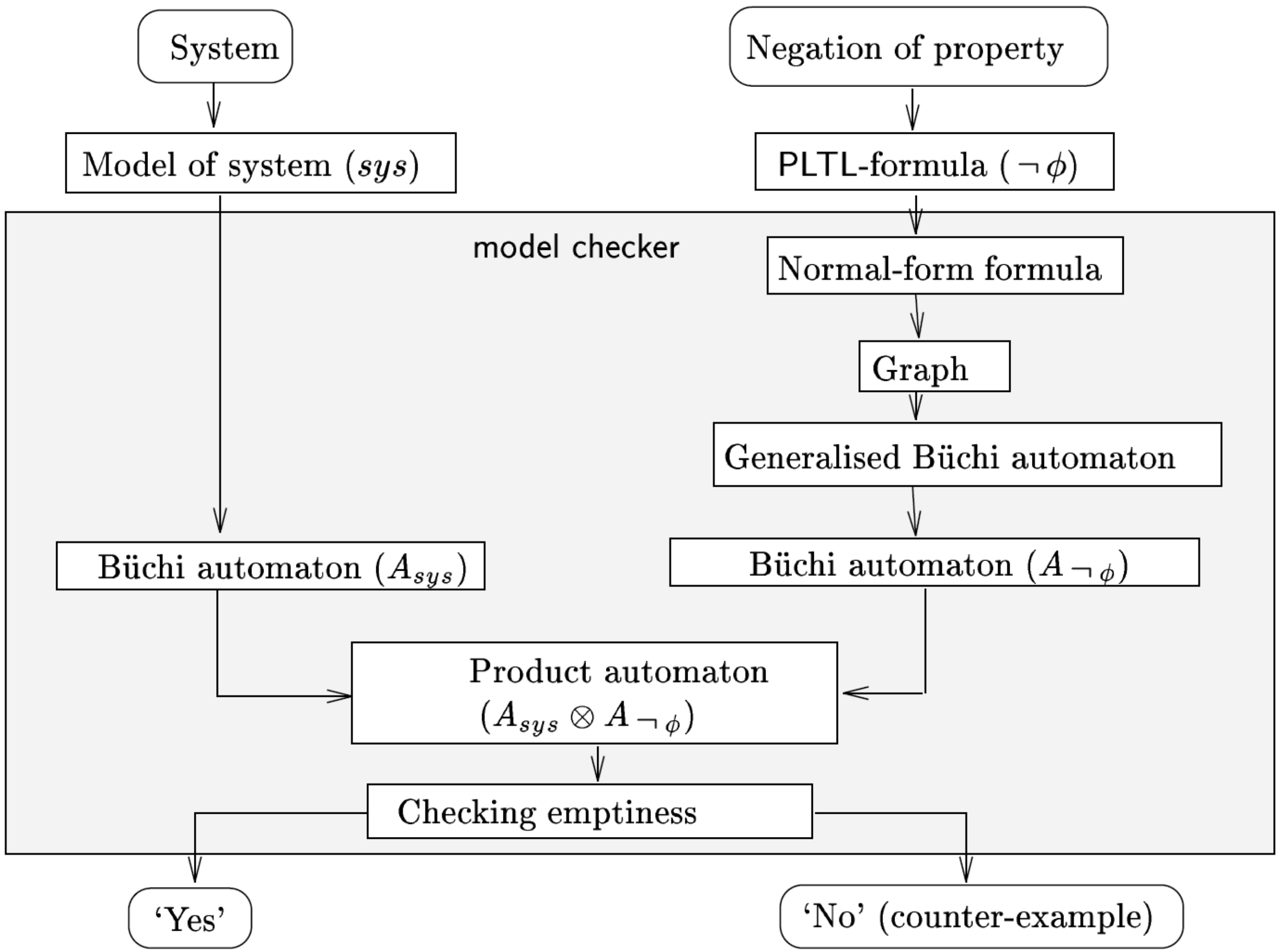
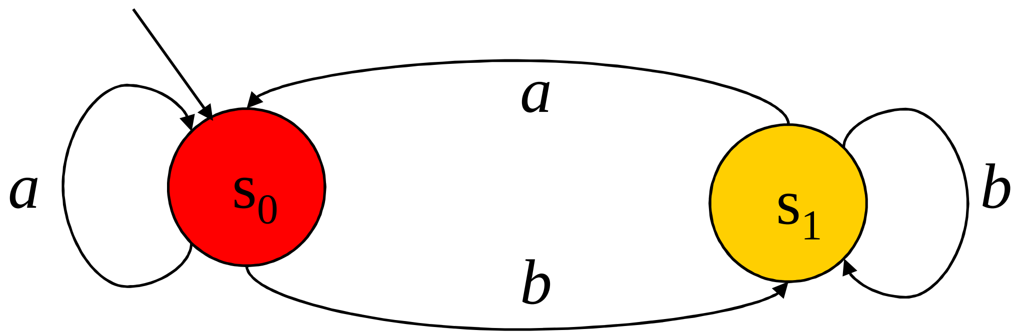


Figure 2.8: Overview of model-checking PLTL

Automata over finite words

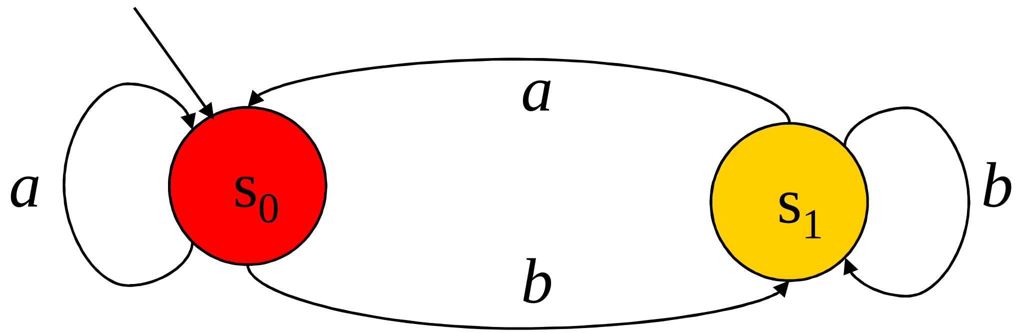
- A=<Σ, S, Δ, I, F>
- Σ (finite) - the alphabet.
- S (finite) - the states.
- $\Delta \subseteq S \times \Sigma \times S$ - the transition relation.
- $I \subseteq S$ - the starting states. *(depicted with an incomig edge from nohere)*
- $F \subseteq S$ - the accepting states. *(depicted in red)*



A *run* over a word

- A word over Σ, e.g., *abaab*.
- A sequence of states, e.g. s_0 s_0 s_1 s_0 s_0 s_1.
- Starts with an initial state.
- Follows the transition relation (s_i, c_i, s_{i+1}).

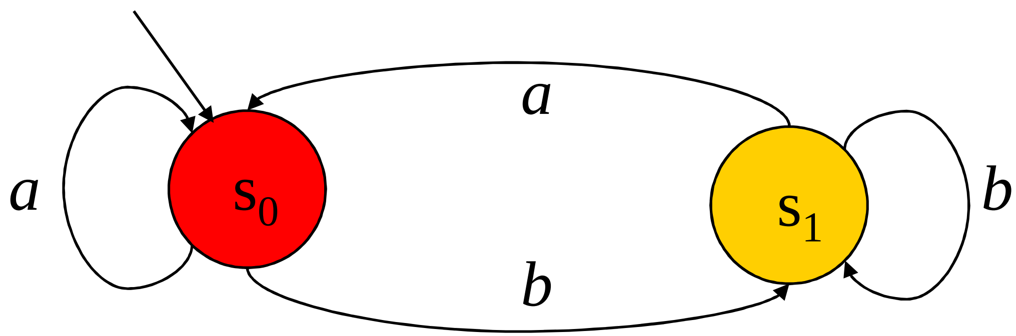
- Accepting if ends at accepting state.



The *language* of an automaton

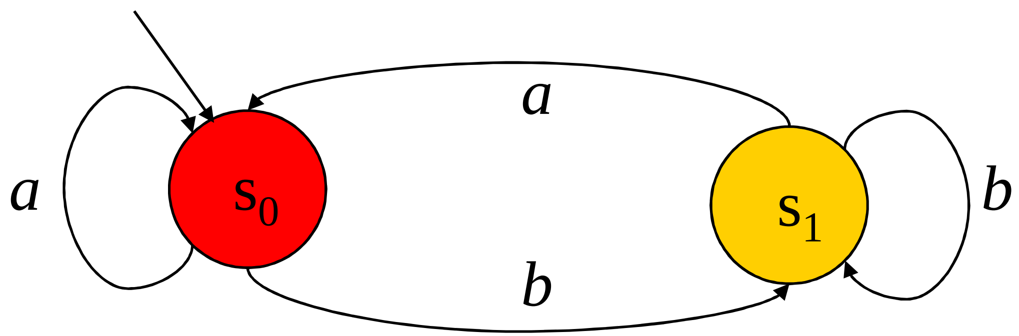
- The words that are accepted by the automaton.
- Includes *aabbba*, *abbbba*.
- Does not include *abab*, *abbb*.

- What is the language?



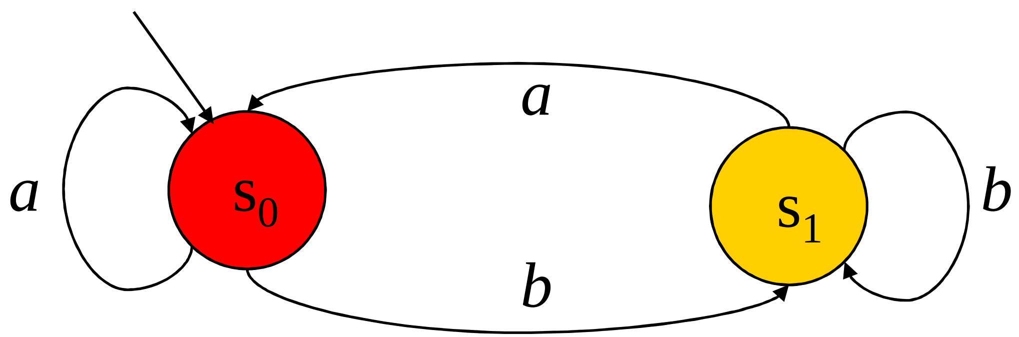
Automata over infinite words

- Similar definition.
- Runs on infinite words over Σ.
- Accepts when an accepting state occurs infinitely often in a run.



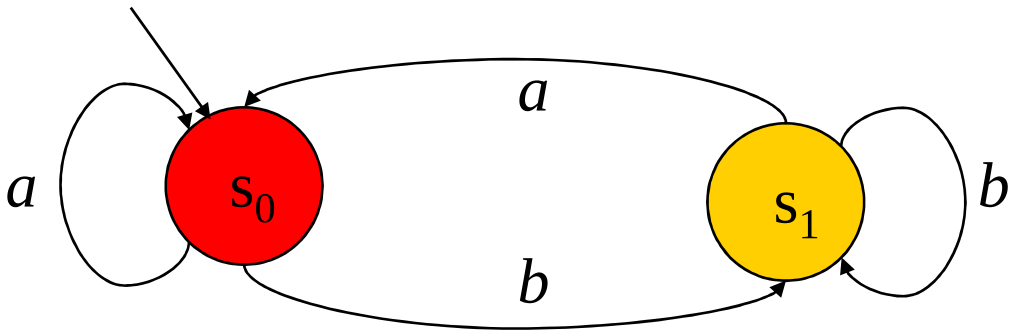
Automata over infinite words

- Consider the word *abababab*…
- There is a run $s_0 s_0 s_1 s_0 s_1 s_0 s_1$ …
- For the word *bbbbb*… the run is s_0 s_1 s_1 s_1 s_1… and is not accepting.
- For the word *aaabbbbb* …, the run is s_0 s_0 s_0 s_0 s_1 s_1 s_1 s_1 …
- What is the run for *ababbabbb* …?



Specification using Automata

- Let each letter correspond to some propositional property.
- Example: *a* -- P0 enters critical section,
  b -- P0 does not enter section.



Generalized Büchi automata

- Acceptance condition F is a set $F=\{f_1, f_2, \ldots, f_n\}$ where each f_i is a set of states.
- To accept, a run needs to pass infinitely often through a state from every set f_i.

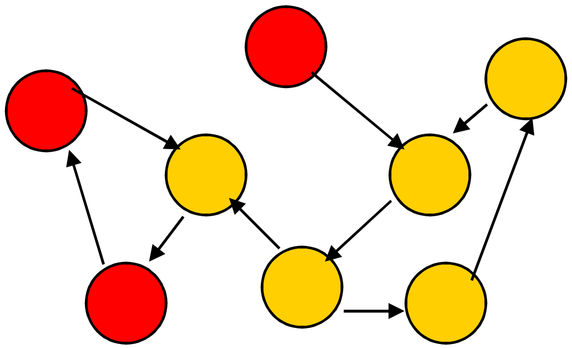
Finding accepting runs

If there is an accepting run, then at least one accepting state repeats on it forever.

Look at a suffix of this run where *all the states appear infinitely often*.

These states form a strongly connected component on the automaton graph, including an accepting state.

Find a component like that and form an accepting cycle including the accepting state.

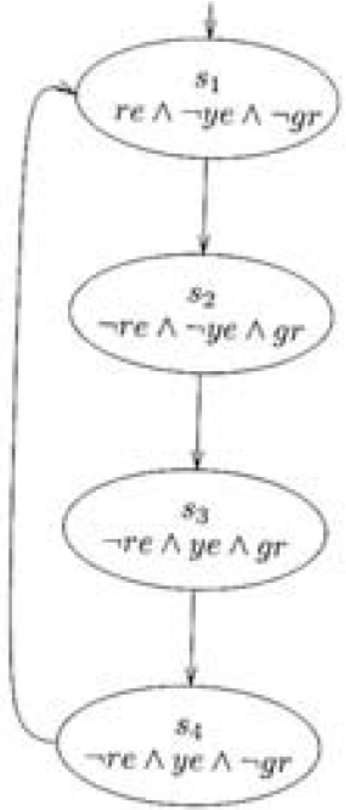


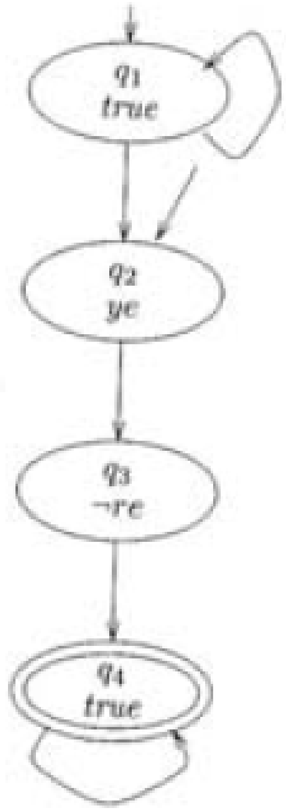
Model checking LTL on an example

Consider the traffic light example, we want to model check an LTL formula against the implementation of a traffic light specified as a Buchi automata. Also the formula is specified as a Buchi automata. These automata are as in the Peled's book, with proposition associated to states

System

(all states accepting)





Formula: G(ye->Xre):

always move from ye to re

not G(ye->Xre) =

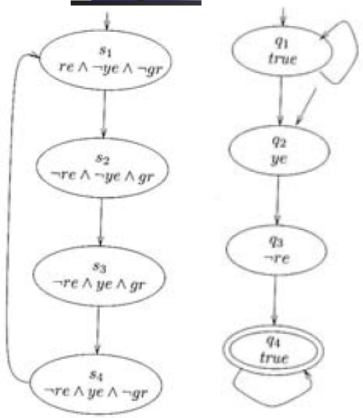
not G (not ye or Xre) =

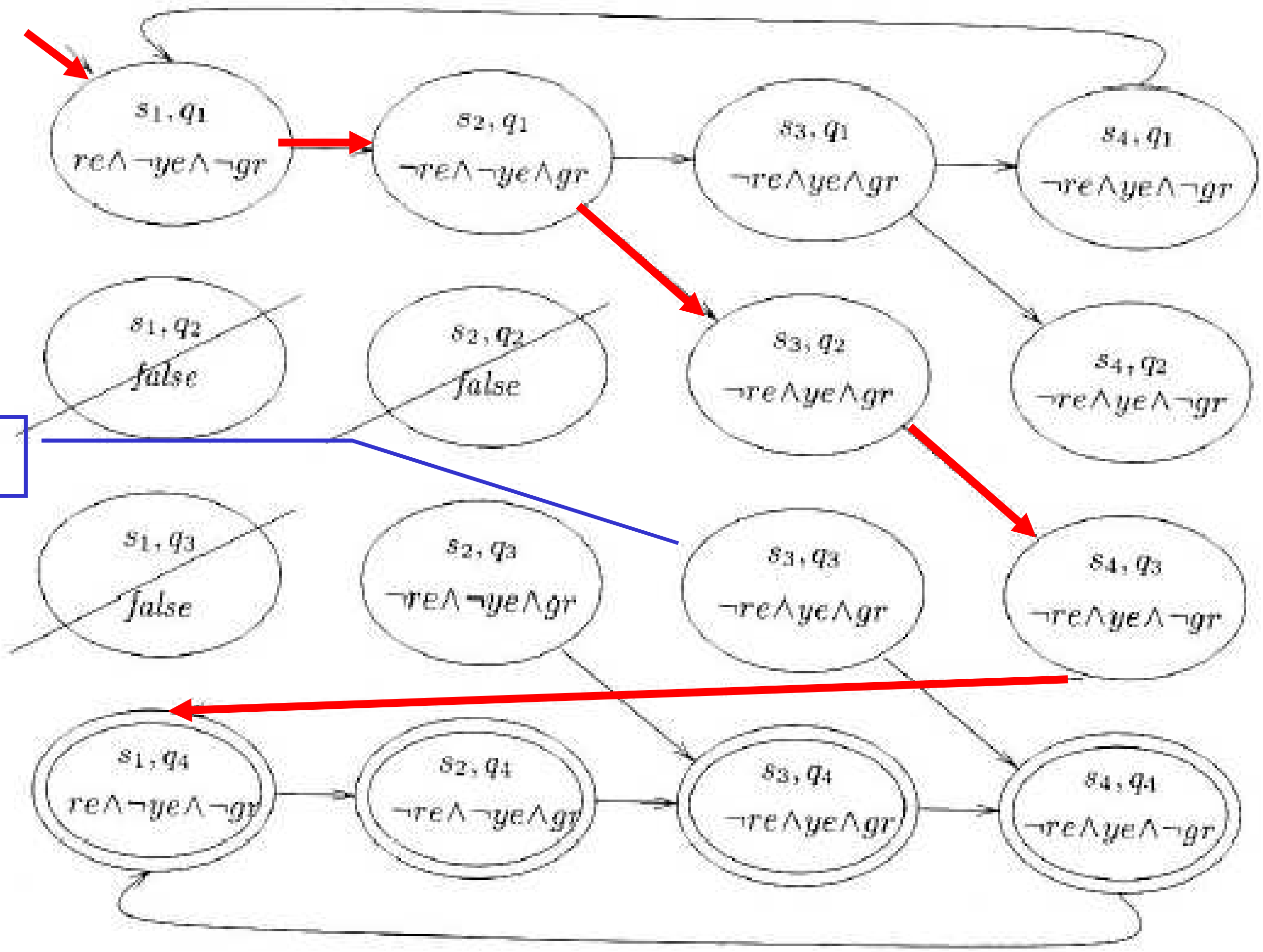
F (ye and not Xre) =

F (ye and X not re)

(2 initial states, 1 acc.)

Model checking LTL on an example - intersection





Not reachable

Intersection is not empty, and red path is a counter example

Model checking LTL - complexity (from JPK)

The automata of the formula φ has a size that depends on the number of subsets of the formula $O(2^{|\varphi|})$

The worst state space complexity of the product is $O(|Sys| * 2^{|\varphi|})$, where Sys is the size of the system (number of nodes + number of transitions)

Checking emptiness is linear in number of states and transitions, and we finally get that:

The worst case time complexity of checking whether Sys satisfies the LTL formula φ is $O(|Sys| * 2^{|\varphi|})$

Fairness

Fairness is used generically to refer to semantics contraints imposed on interleaved executions of concurrent systems.

E.g. P1 and P2, independent programs, that execute forever. On a real cpu they alternate into cpu, depending on the scheduler policy. We do not want to insert the scheduler policy in the model (too detailed), but we want to rule out interleaved executions that ignore enabled transitions of one process forever, since they do not correspond to any realistic scheduler.

Fair executions: motivations

Consider the following piece of code:

$$\textbf{process}\ \mathsf{Inc} \quad = \quad \textbf{while}\ \langle\, x \geqslant 0\ \textbf{do}\ x := x + 1\,\rangle\ \textbf{od}$$
$$\textbf{process}\ \mathsf{Reset} \quad = \quad x := -1$$

where $\langle .. \rangle$ means "atomic execution".

Does the program satisfies "F terminates"? No, since there is an execution in which only Inc is executed.

This situation is not possible if the OS schedule is fair, and we would like to rule-out from the model checking whose executions that are not fair

Fair executions: solutions

We want to consider only execution with fair behaviour.

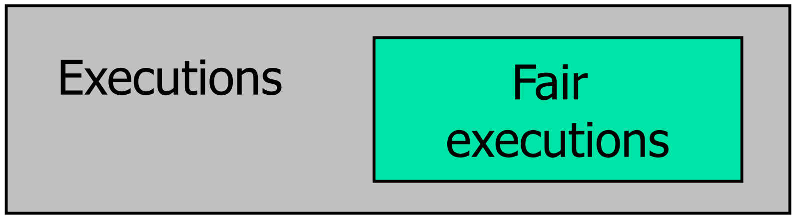
Can be done:

• enforcing fairness in the formula: instead of verifying that the program satisfies φ, verify it satisfies *fair-constraint* $\Rightarrow \varphi$

$$(\mathsf{G}\,\mathsf{F}\,\mathsf{Inc}.running \;\wedge\; \mathsf{G}\,\mathsf{F}\,\mathsf{Reset}.running) \;\Rightarrow\; \mathsf{F}\, terminate$$

OR

• modifying the MC algorithm as to consider only fair executions



Some fairness definitions (JPK)

- Si tratta della definizione della parte di fairness constraint in

$$\textit{fair-constraint} \Rightarrow \varphi$$

- Vogliamo che il fair constraint sia abbastanza ampio (nel senso che deve essere soddisfatto in molte esecuzioni).
- Esempi di casi limite per la determinazione delle esecuzioni fair in una proprieta' di terminazione, del tipo

$$\textit{fair-constraint} \Rightarrow \text{F terminate}$$

  - *fair-constraint* = true : il programma deve terminare su tutte le esecuzioni
  - fair-constraint = false: anche se il programma non termina la proprieta' e' soddisfatta

$\text{true} \Rightarrow \text{F terminata} \equiv \text{false} \vee \text{F terminata} \equiv \text{F terminata}$

Some definitions (JPK) for *fairness-constraint*

- *Unconditional fairness:*
  Un unconditional fairness constraint is an LTL formula of the form:
  $GF\ \psi$ also stated as $true \Rightarrow GF\ \psi$

- *Weak fairness (justice):*
  A weak fairness constraint is an LTL formula of the form:
  $$FG\ \varphi \Rightarrow GF\ \psi$$
  as in: FG enabled(a) $\Rightarrow$ GF executed(a)

  For considering only paths in which, from a certain point on, if you keep asking, you get it infinitely often

- *Strong fairness:*
  A strong fairness constraint is an LTL formula of the form:
  $$GF\ \varphi \Rightarrow GF\ \psi$$

  For considering only paths in which, if you ask infinitely often, you get it infinitely often

Som

Usually unconditional and strong are useful for solving contentions, and weak is often sufficient to resolve non-determinism due to interleaving semantics

- *Unconditional fairness:*
  Un unconditional fairness constraint is an LTL formula of the form:

  $$GF\ \psi \qquad \text{also stated as} \qquad true \Rightarrow GF\ \psi$$

- *Weak fairness (justice):*
  A weak fairness constraint is an LTL formula of the form:

  $$FG\ \varphi \Rightarrow GF\ \psi$$

  as in: FG enabled(a) $\Rightarrow$ GF executed(a)

- *Strong fairness:*
  A strong fairness constraint is an LTL formula of the form:

  $$GF\ \varphi \Rightarrow GF\ \psi$$

Sono ``fairness assumptions" gli AND di fairness constraints

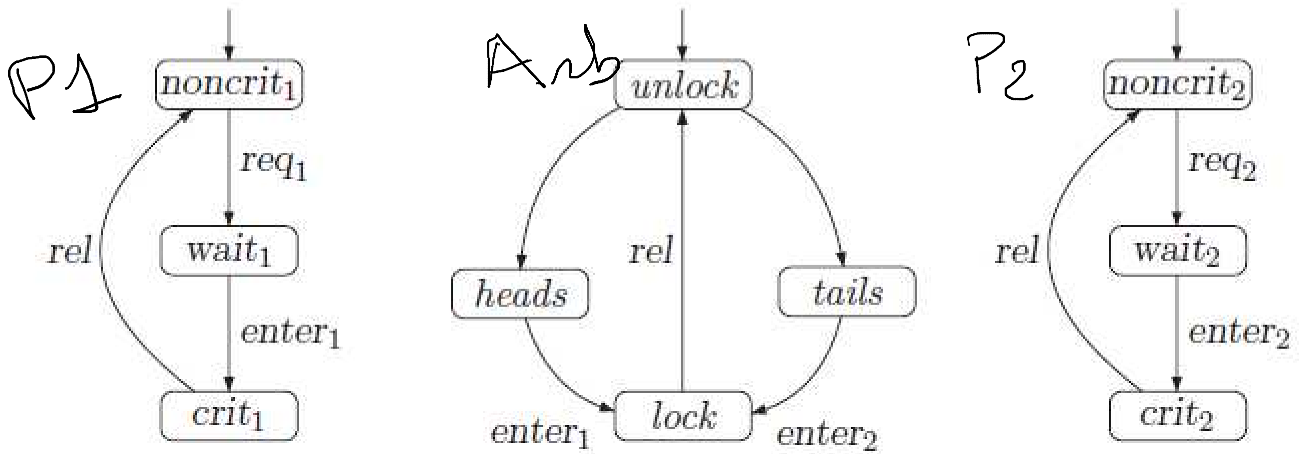
Esempio di fairness

Sys = P1 || P2 || Arb su {enter1 e enter2}

Sys |= GF crit1 ?? Questo si traduce in

$\forall \sigma \in$ Lang(Sys), s |= GF crit1

La formula LTL è falsa perché esiste un'esecuzione in cui Arb sceglie sempre tail



Esempio di fairness

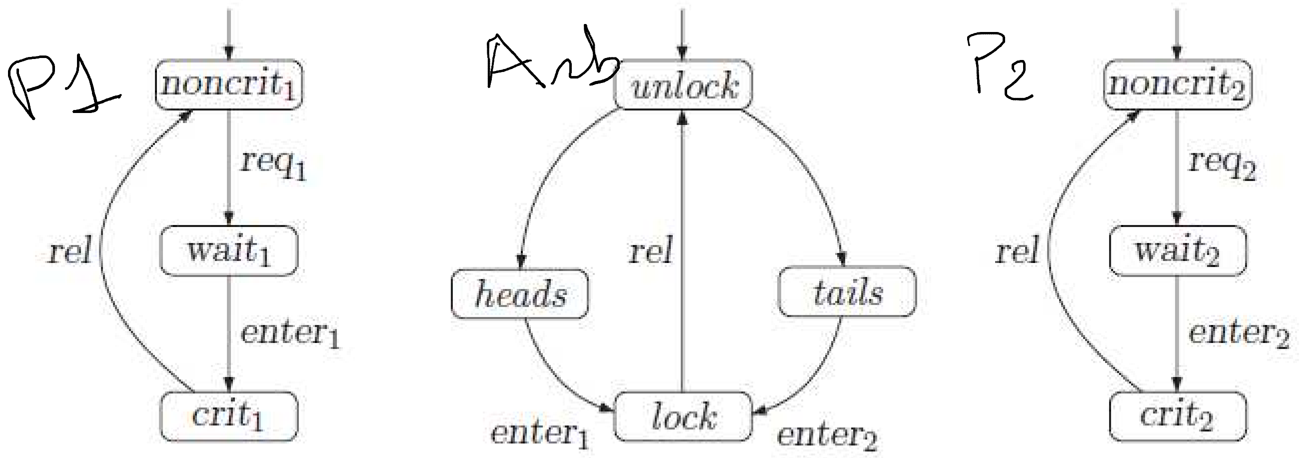
Sys = P1 || P2 || Arb su {enter1 e enter2}

Fairness constraint (unconditional):

GF heads AND GF tail

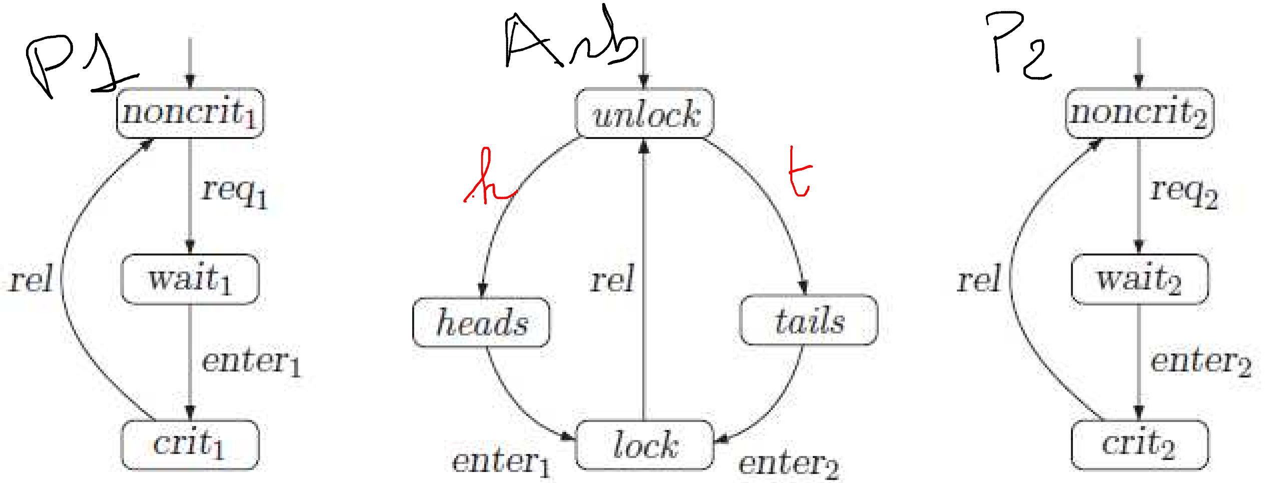
ed è vero che:

Sys |= (GF heads AND GF tail) --> GF crit1



Action-based vs. transition based fairness

Di fatto nell'esempio precedente la condizione di fairnes è espressa sugli stati locali (heads, tails) ma forse sarebbe più naturale esprimerla sulle azioni h e t che esprimono la scelta non deterministica. È possibile provare che si può tradurre la specifica action-base in specifica state-based (modificando gli stati del sistema)



If MC is so good, why deductive verification methods exists?

- Model checking works only for finite state systems. Would not work with
  - Unconstrained integers.
  - Unbounded message queues.
  - General data structures:
    - queues
    - trees
    - stacks
  - parametric algorithms and systems.

The state space explosion

- Need to represent the state space of a program in the computer memory.
  - Each state can be as big as the entire memory!
  - Many states:
    - Each integer variable has 2^32 possibilities. Two such variables have 2^64 possibilities.
    - In concurrent protocols, the number of states usually grows exponentially with the number of processes.

If MC is so constrained, is it of any use?

- Many protocols are finite state.
- Many programs or procedure are finite state in nature. Can use abstraction techniques.
- Sometimes it is possible to decompose a program, and prove part of it by model checking and part by theorem proving.
- Many techniques to reduce the state space explosion.