Introduction to Model-Checking

Theory and Practice

Beihang International Summer School 2018

Model Checking

Linear Temporal Properties

Model Checking

“Model checking is the method by which a desired *behavioral property* of a reactive system is verified over a given system (the model) through exhaustive enumeration (*explicit* or *implicit*) of all the states reachable by the system and the behaviors that traverse through them”

Amir Pnueli (2000)

Basic Properties

- *reachability* (something is possible)
- *invariant* (something is always true)
  the gate is closed when the train cross the road
- *safety* (something bad never happen: $\equiv \neg Reach$)
- *liveness* (something good may eventually happen)
  every process will (eventually) gain access to the printer
- *Fairness* (if it may happen ∞ often then it will happen)
  if messages cannot be lost ∞ often then I will received an ack

Model-Checking

- We have seen how to use a formal specification language to model the system (⇒ as a graph, but also as a language $\mathcal{L}$)

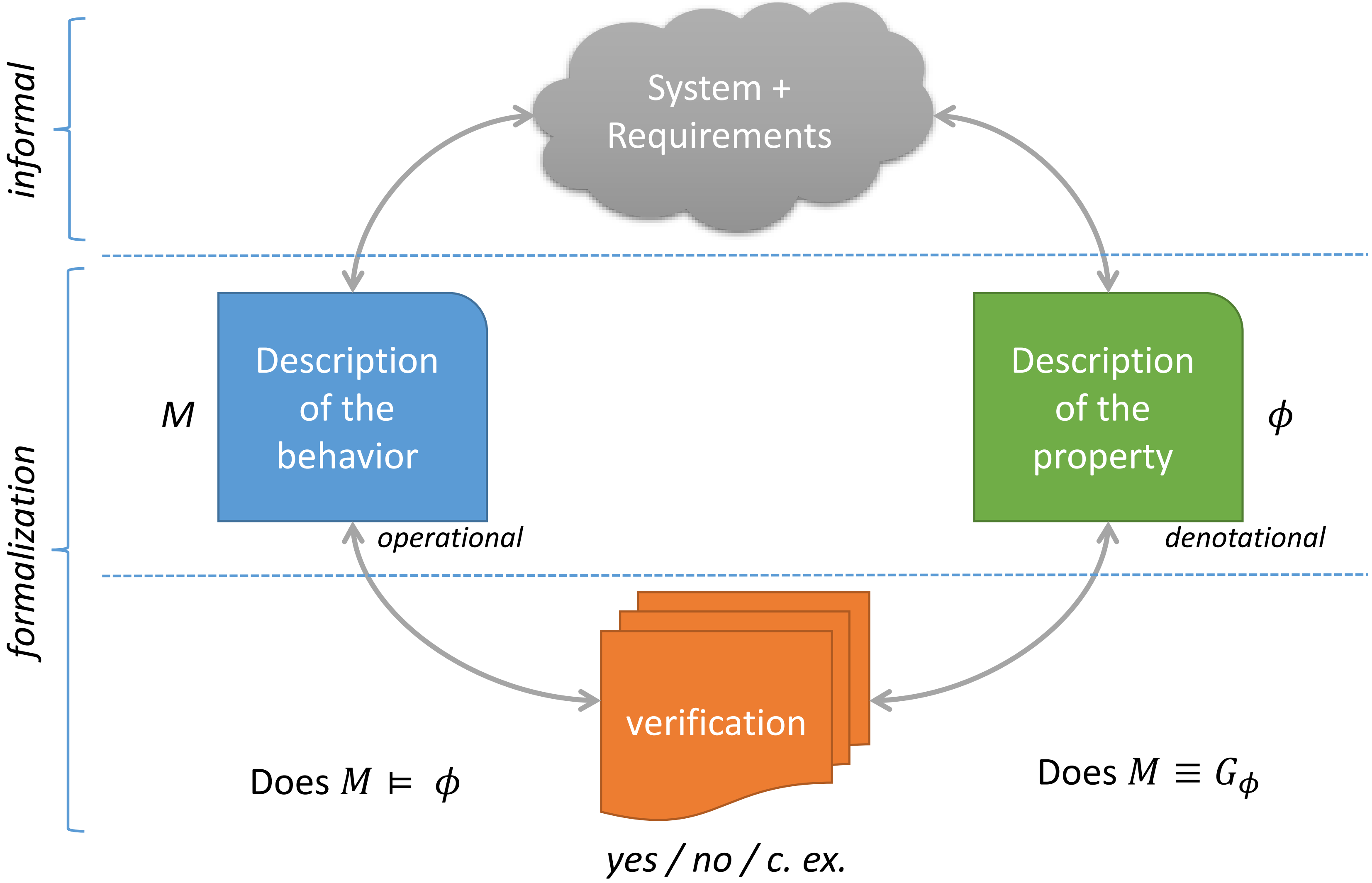
- We have seen how to check basic properties on the reachability graph (and that SCC can be useful)

- What if we want to check more general properties?

User-Defined Properties

- *present before* (we should see an a before the first b)
  the latch must be locked before take-off
- *infinitely often* (something will always happen, $\neq Always$)
  the timer should always be triggered
- *leadsto* (after every occurrence of a there should be a b)
  there is an audible signal after the alarm is activated
- *… see example of specification patterns here [Dwyer]*

Model-Checking



informal

System + Requirements

formalization

M

Description of the behavior

operational

ϕ

Description of the property

denotational

verification

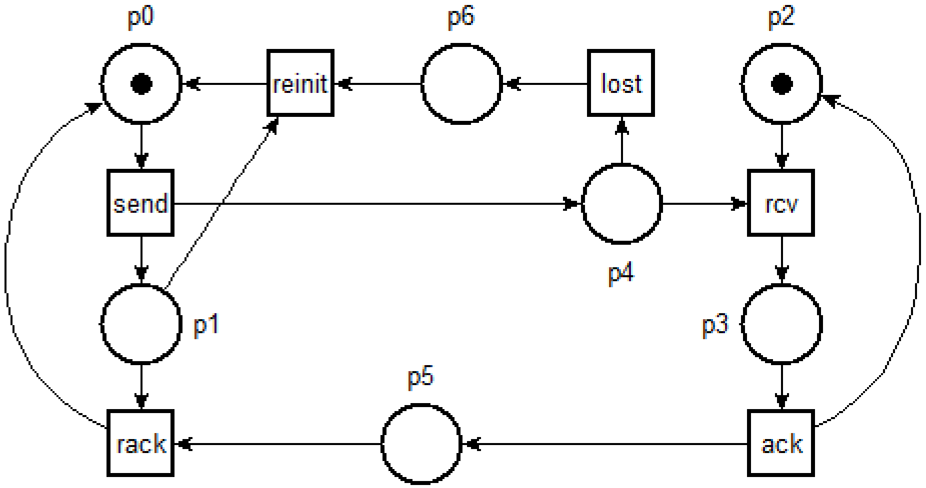
Does $M \models \phi$

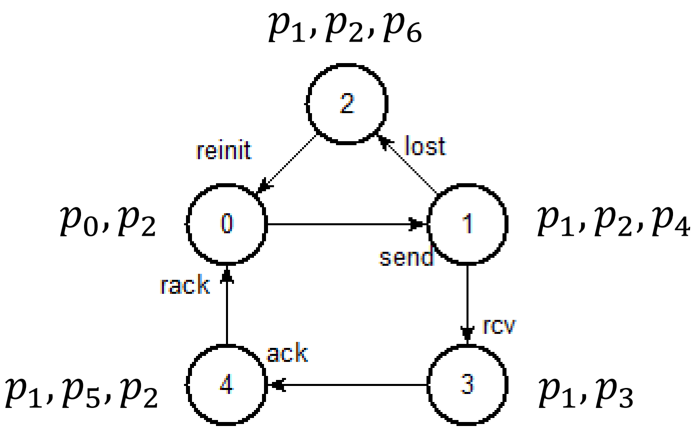
Does $M \equiv G_\phi$

yes / no / c. ex.

present a before b

- In every execution, action a should occur before b or there should be no b

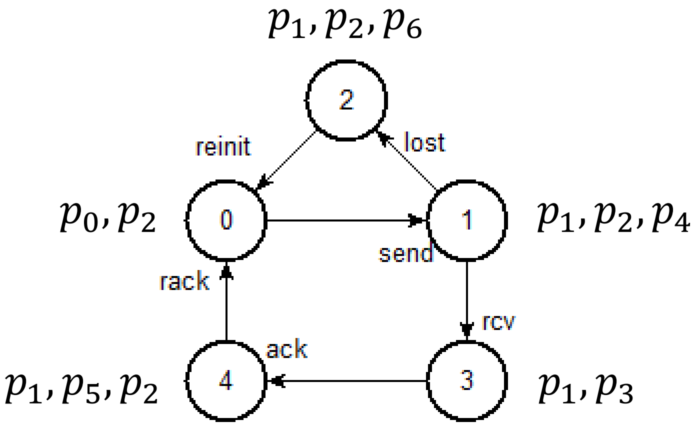




present rcv before ack

Infinitely often a

- every "maximal execution" should have an unbounded number of a
  - equivalently: from every state, it is unavoidable to do an a

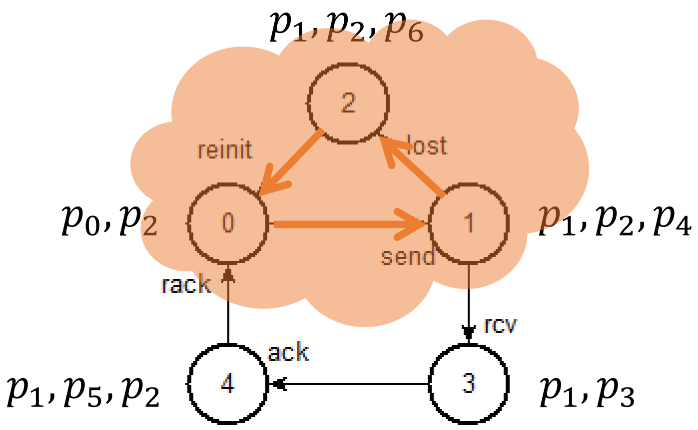


infinitely rcv infinitely $send$

Infinitely often a

- every “maximal execution” should have an unbounded number of a
  - equivalently: from every state, it is unavoidable to do an a

Infinitely rcv

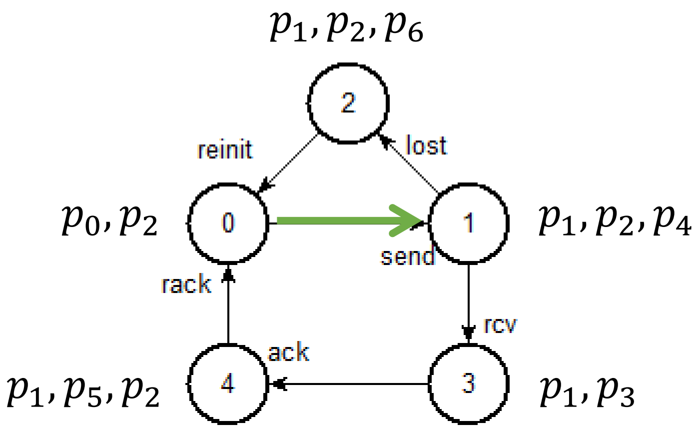


NO!

Infinitely often a

- every "maximal execution" should have an unbounded number of a
  - equivalently: from every state, it is unavoidable to do an a

Infinitely $send$

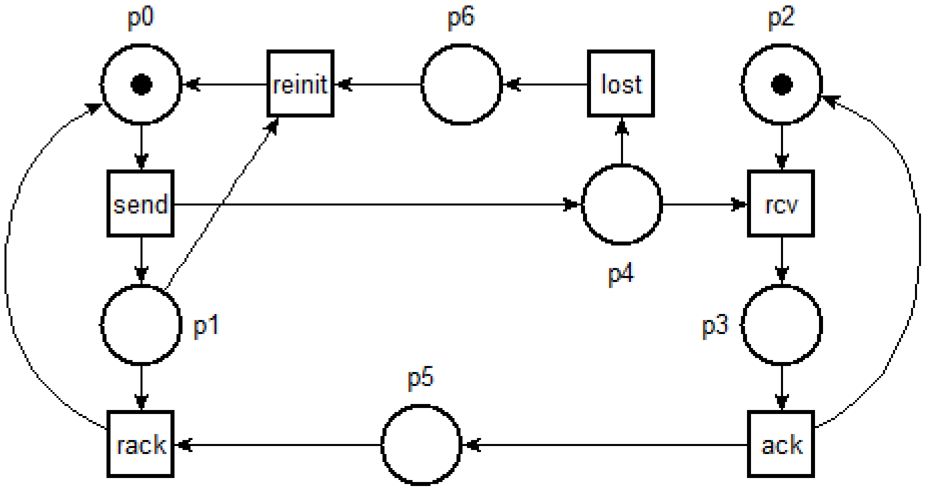


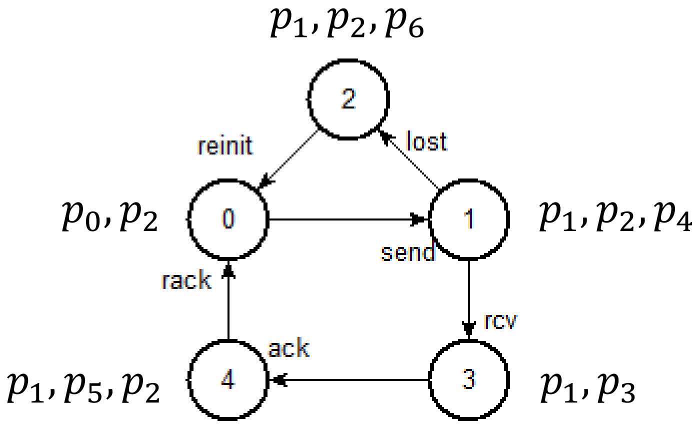
YES!

a leadsto b

- after every occurrence of action a we should eventually find an occurrence of action b

  or there is no a



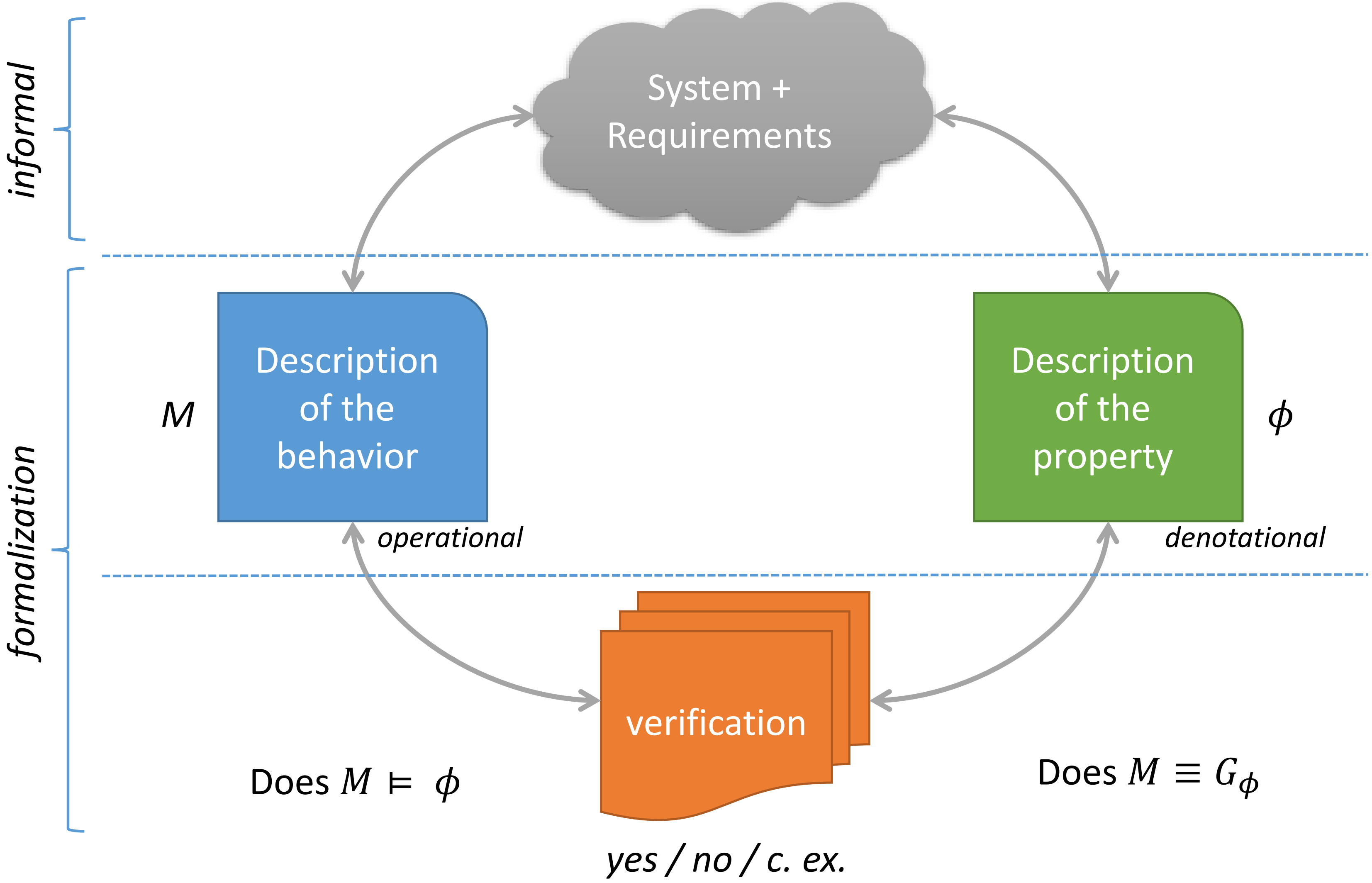


snd leadsto rcv

Model Checking

Linear Temporal Properties using Language Inclusion

Model-Checking



informal

formalization

System + Requirements

M

Description of the behavior

operational

Description of the property

ϕ

denotational

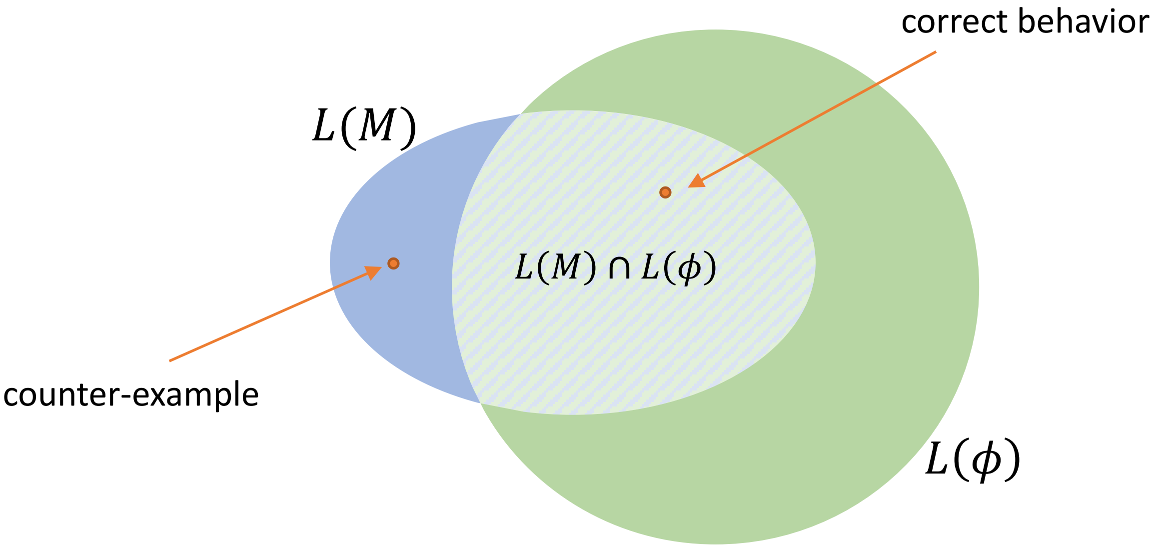
verification

Does $M \vDash \phi$

Does $M \equiv G_{\phi}$

yes / no / c. ex.

Model-Checking



M Description of the behavior

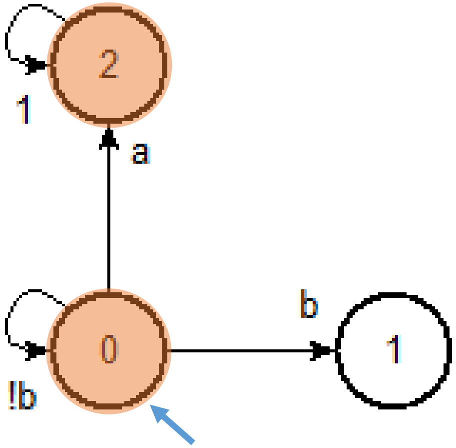
Does $L(M) \subseteq L(\phi)$?

Description of the property ϕ

present a before b

- In every execution, action a should occur before b

  or there should be no $b$present a before b

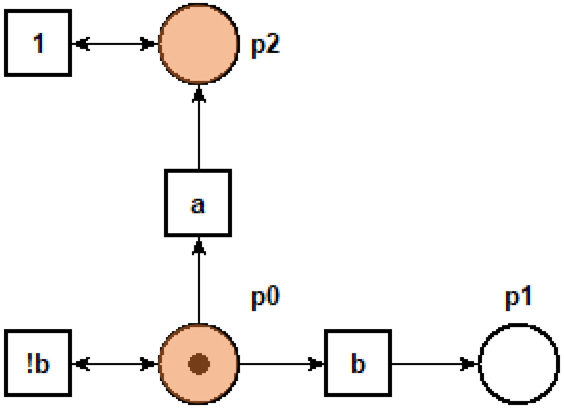


a = {a}
1 = {a, b, c, …}
!a = {b, c, …}

- **idea**: use set of words (language) of *infinite length* that go through a *"red state"* infinitely often

present a before b

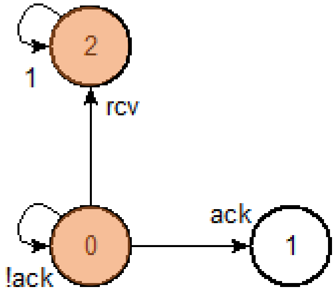
- In every execution, action a should occur before b
  or there should be no $b$present a before b

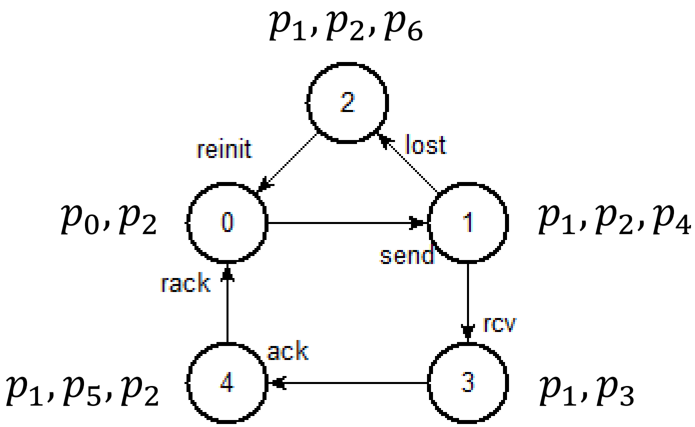


a = {a}
1 = {a, b, c, …}
!a = {b, c, …}

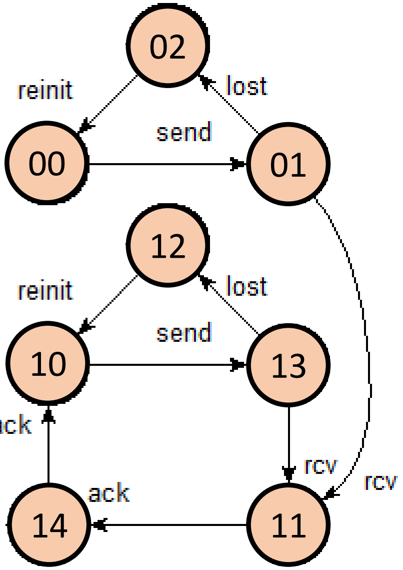
- Equivalently: find traces where *“red places” are infinitely* marked.

present a before b





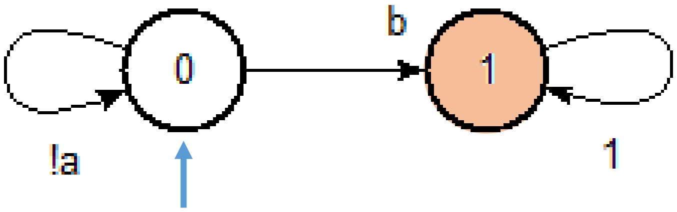
present rcv before ack



Büchi Automata

Büchi automata

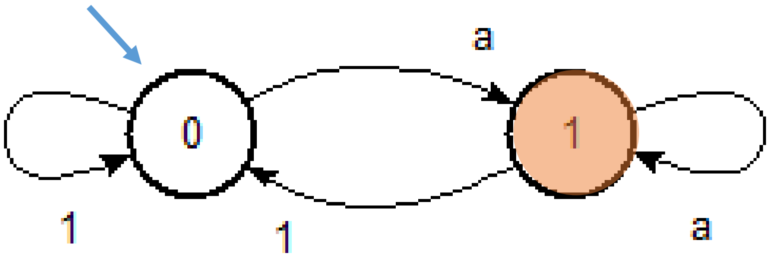
- In here we consider automata with a set of final, or accepting states $F \subseteq Q$
  - we use “red states” to mean accepting
- It is a finite state automata with a different *acceptance condition*. An (infinite) word is accepted if it corresponds to an infinite run that contains infinitely many accepting states



a = {a}
1 = {a, b, c, …}
!a = {b, c, …}

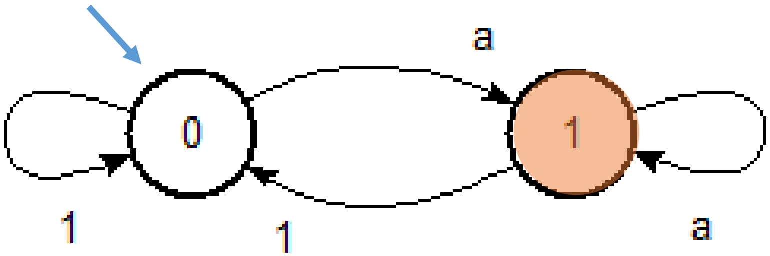
Example

$a.a.a.\ldots a^{\omega}$	is accepted
$a.a.b.a.b.a.\ldots(b.a)^{\omega}$	is accepted
$a.a.b.b.b.\ldots(b)^{\omega}$	is rejected
$a.b.a.b^{2}.a.b^{3}.\ldots(b^{n}.a)^{\omega}$	is accepted



Büchi automata

- To find if there exist a word that is not accepted by a Büchi automata, it is enough to find a cycle without accepting state
  we know how to do it (remember Tarjan's algorithm)
- Same thing if we want to test if $\mathcal{L} = \emptyset$

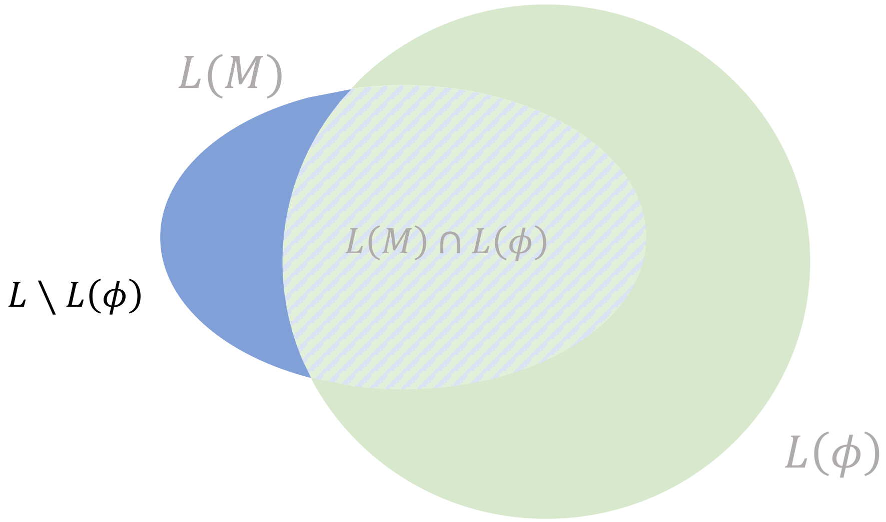


Model Checking

Linear Temporal Properties using Language Inclusion

a better idea

Model-Checking



M

Description of the behavior

Does $L(M) \setminus L(\phi) = \emptyset$?

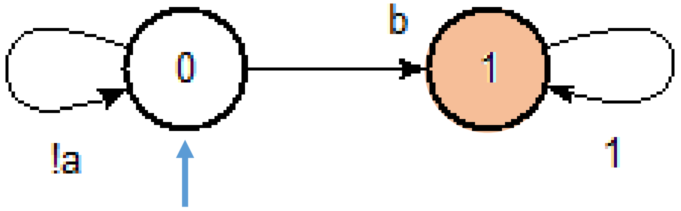
Description of the property

ϕ

present a before b

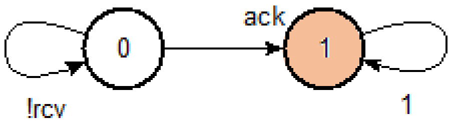
- We disprove the property if we find a trace where b can be reached without firing action a

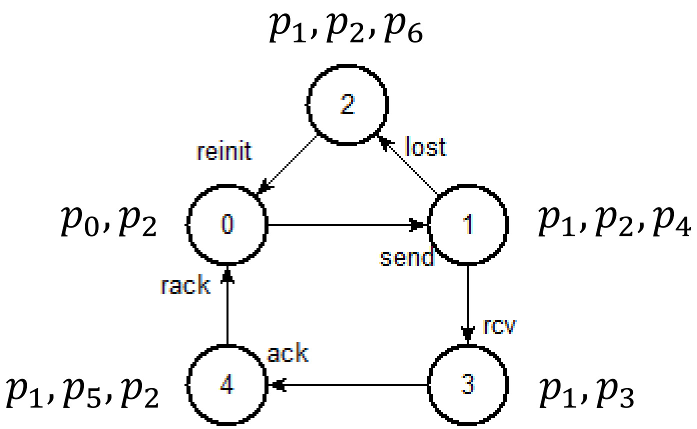
$L(\neg \phi) \equiv$



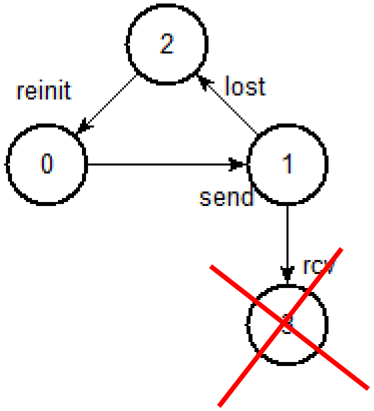
a = {a}
1 = {a, b, c, …}
!a = {b, c, …}

present a before b



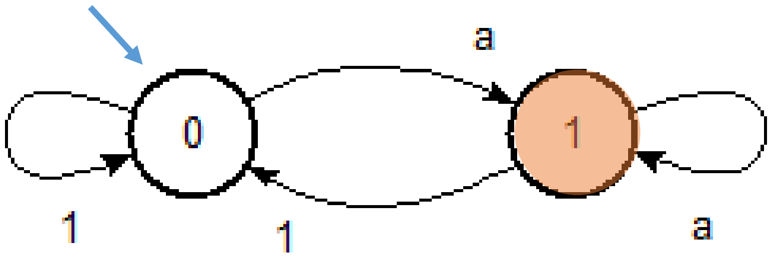


present rcv before ack



Infinitely often a

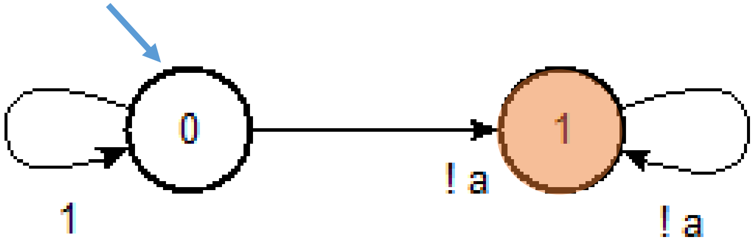
- From every state, it is unavoidable to do an a



$\equiv L(\phi)$

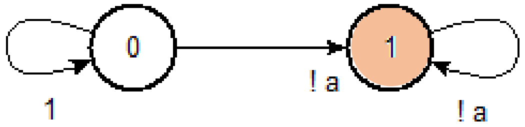
Infinitely often a

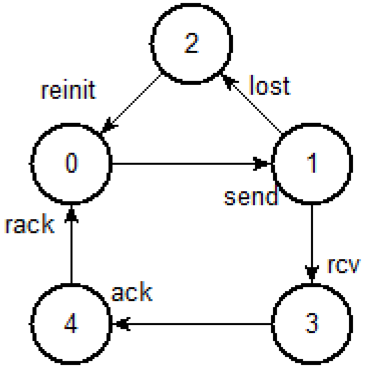
- We disprove the property if we find a trace without a single a (maybe after some other transitions fire)



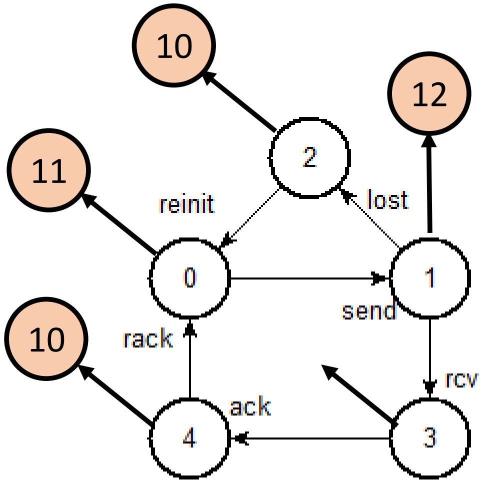
$\equiv L(\neg\, \phi)$

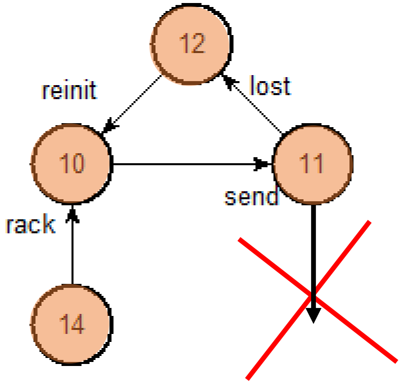
Infinitely often a



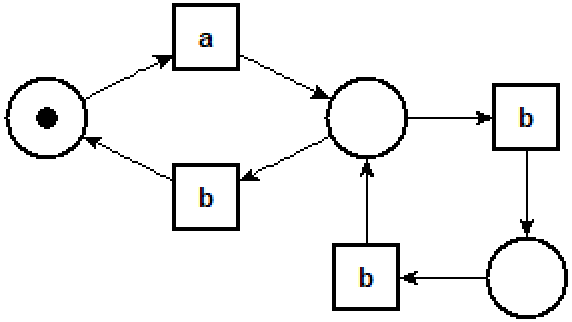


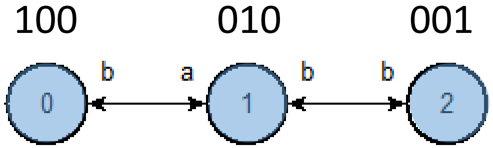
infinitely rcv

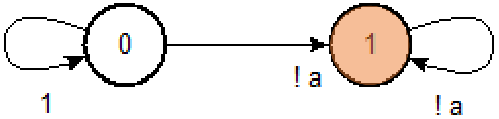


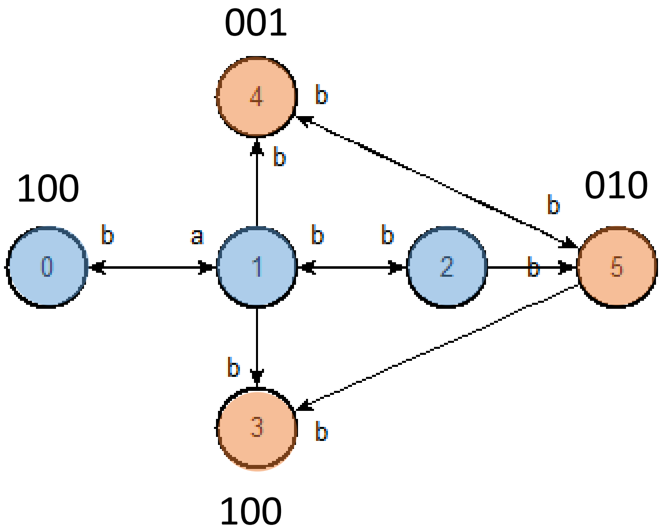


Infinitely often a





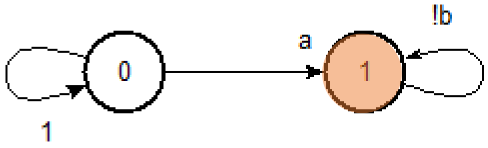


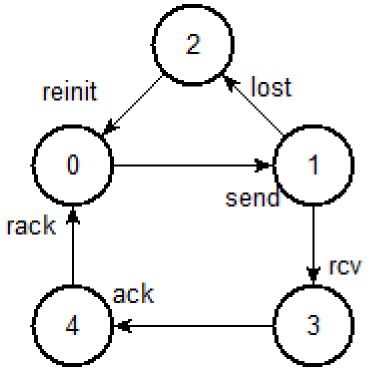


a leadsto b

- after every a we eventually find a b

  We disprove the property if we find an a followed by an infinite sequence (a SCC) without b





snd leadsto rcv

What you need to remember

- We can check more complex properties using an automata-theoretic approach
- It is often easier to try to disprove the property

$$A(Sys) \otimes A(\neg\phi) = \emptyset$$

- We need automata that accept infinite words → different acceptance criterion (Büchi-automaton)
- There is a link with SCC (infinitely often ≈ cycle)

Model Checking

LTL, a Principled Approach

Model-Checking

- We have seen how to use “language inclusion” (product of automata and search for an infinite path) to express *temporal properties* on a system
- What if we want to check more general properties? Is there a more friendly way to define temporal properties ?
- How can we derive an automaton from it

Classical logic

- logic is the systematic study of the *form of valid inference*
  - Aristotle (322 BC)
  - Clarence Lewis (~1910) for its actual form
  - A formula is *valid* iff it is true under every interpretation.
  - An argument form (or schema) is valid iff every argument is valid

wikipedia

Non-classical logics

- Example of valid argument form (or schema)

  $((A \Rightarrow B) \land A) \Rightarrow B$

- Predicate logic has *propositional variables* ($A, B, \ldots$) and connectives ($\land, \neg, \Rightarrow, \ldots$)

- There are also non-classical logics, such as modal logics, that extend logic with operators (modalities) expressing the fact that the truth may depends on the context

  examples are beliefs: *I am certain vs it is possible*; permissions (deontic logic): *it is permissible vs it is obligatory*; and time: *always vs eventually*

Temporal logic

Amir Pnueli (1941—2009)

"In mathematics, logic is static. It deals with connections among entities that exist in the same time frame. When one designs a dynamic computer system that has to react to ever changing conditions, … one cannot design the system based on a static view. It is necessary to characterize and describe dynamic behaviors that connect entities, events, and reactions at different time points. Temporal Logic deals therefore with a dynamic view of the world that evolves over time."

Atomic proposition

- We start by defining *atomic propositions*
  statements about the here and now !
- We assume a set of propositional variables $\{p_1, p_2, \dots, p_n\}$
- We will use the language of logic. *Atomic Formulas* are built from P. V. and from connectives

$$a, b, \dots ::= p \mid \neg a \mid a \wedge b \mid \dots$$

- e.g.: $p_1 \wedge (p_2 \vee \neg p_3) \qquad p_1 \Rightarrow p_2 \equiv (\neg p_1) \vee p_2$

Atomic Prop.: event/state-based

- If we want to deal with events (transitions), we can choose atomic propositions $a, b, \ldots$ that corresponds to event names:
  - t_1
  - $dead$

- We can also choose to deal with states (markings)
  - $p_1 + 2.p_3 \leq 4$ $p_1 \equiv (p_1 \geq 1)$
  - $Firable(t)$
  - $dead$
  - $initial\ (`0)$

(from now on we consider only “state-based” formulas)

Linear Temporal Logic

- The logic has two main connectives

  $F\ \phi$: reads "finally" (eventually) ϕ is true

  $G\ \phi$: reads "globally" (always) ϕ is true

$$\phi, \psi, \ldots ::= a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid F\ \phi \mid G\ \phi$$

- So you can write formulas such as:

  $G\ (p_1 \Rightarrow F\ (p_2 + p_3 \leq p_4))$ $\quad\neg\ (F\ dead) \vee G\ (\neg p_2)$

  $F\ G\ p_1 \wedge G\ F\ p_2$ $\quad G\ ((G\ t_1) \vee (G\ t_2))$

Linear Temporal Logic

- The logic has two main connectives

  $F\ \phi$: reads “finally” (eventually) ϕ is true, also $[]\phi$

  $G\ \phi$: reads “globally” (always) ϕ is true, also $\langle\rangle\phi$

- There is another possible presentation based on two additional connectives

  $\phi\ U\ \psi$: reads “ϕ holds until ψ”

  $X\phi$: reads “next” ϕ holds, also written $()\phi$

LTL—syntax equivalence

$F\ \phi$	$\langle\ \rangle\phi$	finally
$G\ \phi$	$[\ ]\phi$	globally
$X\ \phi$	$(\ )\phi$	next
$\phi\ U\ \psi$	$\phi\ U\ \psi$	until
$!\ \phi$	$\neg\phi$	negation
SPIN syntax	selt syntax	Pinyin name

LTL—semantics

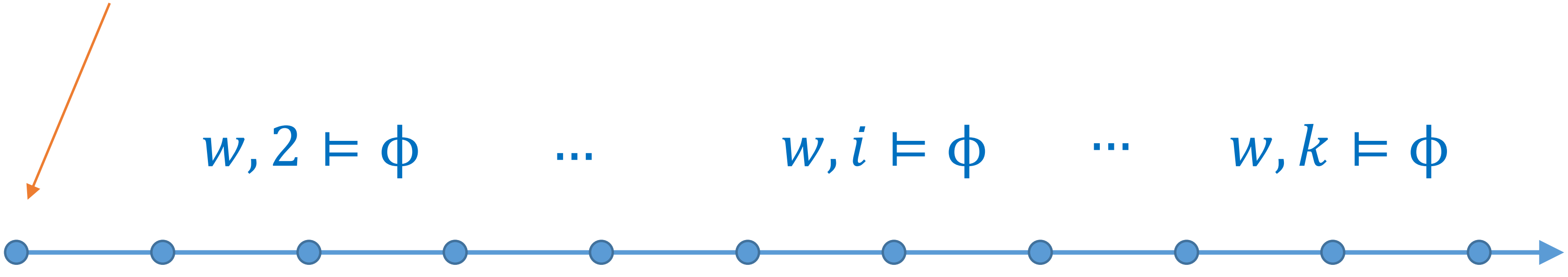
- LTL formulas are interpreted on (maximal) traces, $w = w_0.w_1.\dots.w_i.\dots$

  for "state-based" properties, w_i is a state ≡ the set of atomic propositions true in w_i
- We call $w(i)$ the i^{th} element in w
- We use the notation $w, i \vDash \phi$ to say that ϕ holds for w from position i

$$w \vDash \phi \quad \Leftrightarrow \quad w, 0 \vDash \phi$$

satisfaction relation ⊨

LTL—semantics



$w \vDash \phi$?

$w, 2 \vDash \phi$ … $w, i \vDash \phi$ … $w, k \vDash \phi$

w_0 w_1 w_2 … w_i … w_k

LTL—atomic propositions

- For atomic propositions, a, we can say whether it holds for w_i or not (we write $w_i \vDash a$ if it holds).

$$w, i \vDash a \qquad \text{iff} \qquad w_i \vDash a$$

- For example, if w_i is the marking $(p_1, p_2 .2, p_4)$—say $(1, 2, 0, 1)$—then we have that:

$$w, i \vDash p_2$$

$$w, i \vDash (p_1 + p_4 \leq \ p_2)$$

$$w, i \vDash \neg(p_1 \land p_3)$$

LTL—other connectives

$w, i \vDash \phi \vee \psi$ iff $(w, i \vDash \phi)$ or $(w, i \vDash \psi)$

$w, i \vDash \neg\phi$ iff not $w, i \vDash \phi$

etc.

$w, i \vDash \langle\,\rangle\phi$ iff $\exists\, \mathrm{j} \geq \mathrm{i}\,.\ (w, j \vDash \phi)$

ϕ holds at some “instant” j in the future

$w, i \vDash [\,]\phi$ iff $\forall k \geq i\,.\ (w, k \vDash \phi)$

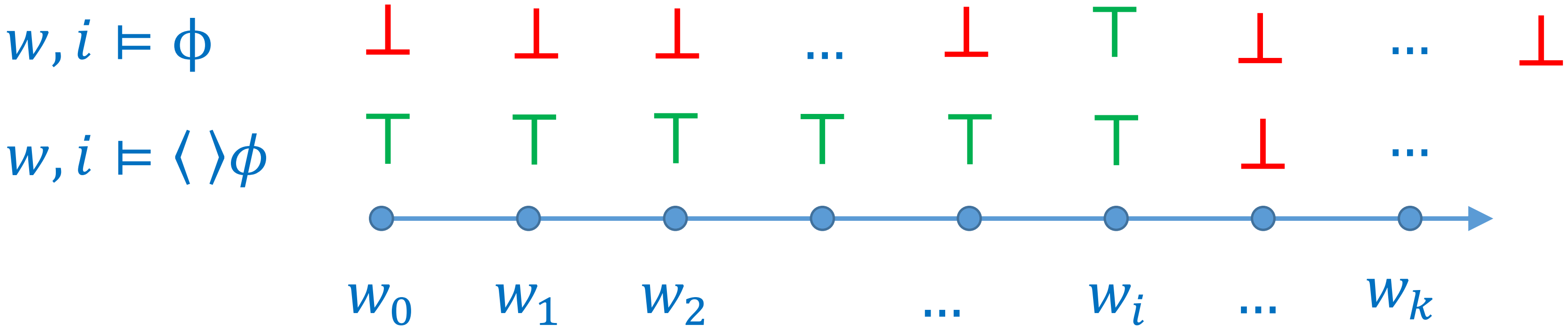
ϕ holds at all instants after i

LTL—semantics

True : $\top$

False : $\bot$

$w \vDash \langle\ \rangle\phi$?



$w, i \vDash \phi$	$\bot$	$\bot$	$\bot$	…	$\bot$	$\top$	$\bot$	…	$\bot$
$w, i \vDash \langle\ \rangle\phi$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\bot$	…	
	w_0	w_1	w_2		…	w_i	…	w_k	

LTL—semantics

True : $\top$

False : $\bot$

$w \vDash [\,]\phi$?

$w, i \vDash \phi$	$\top$	$\top$	$\top$	…	$\top$	$\bot$	$\top$	…	$\top$
$w, i \vDash [\,]\phi$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$	…	
	w_0	w_1	w_2		…	w_i	…		w_k

LTL—other connectives

We can define the semantics for the two extra op.

$w, i \vDash ()\phi$ iff $w, i + 1 \vDash \phi$

ϕ holds at the next “instant”

$w, i \vDash \phi \; U \; \psi$ iff $\exists j \geq i \, . \, (w, j) \vDash \psi$
and $\forall k \in [i, j[\, . \, (w, k) \vDash \phi$

there is an instant j in the future such
that ψ holds and ϕ holds until that time

LTL—semantics

True : $\top$

False : $\bot$

$w \vDash \phi \, U \, \psi$?

$w, i \vDash \phi$	$\top$	$\top$	$\top$	…	$\top$	…		
$w, i \vDash \psi$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$	…	
$\phi \, U \, \psi$	$\top$	$\top$	$\top$	…	$\top$	$\top$		
	w_0	w_1	w_2		…	w_i	…	w_k

LTL—other connectives

We can use the until connective to define F and G

$$\langle\,\rangle\phi \;\equiv\; True\; U\; \phi$$

$$[\,]\phi \;\equiv\; \neg(True\; U\; (\neg\phi))$$

Actually, it is true that

$$[\,]\phi \;\equiv \neg\langle\,\rangle(\neg\phi)$$

Think De Morgan's laws:
$\neg(a \vee b) \equiv \neg a \wedge \neg b$

LTL—other connectives

We can also use next to (recursively) define F and G

$$\langle\,\rangle\phi \;\equiv\; \phi \lor (\,)\langle\,\rangle\phi \;\equiv\; \mu X.(\phi \lor (\,)X)$$

$$[\,]\phi \;\equiv\; \phi \land (\,)[\,]\phi \;\equiv\; \mu X.(\phi \land (\,)X)$$

$$\phi\ U\ \psi \;\equiv\; \mu X.(\psi \lor (\phi \land (\,)X))$$

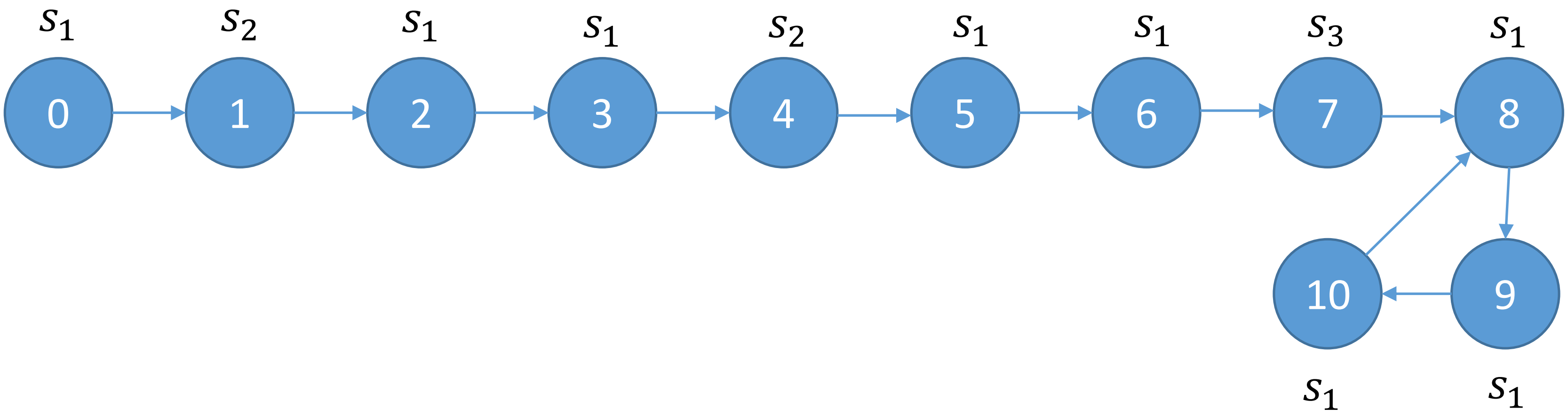
$\mu X.f(X)$ means the “smallest fixpoint” for the functional f, i.e. V such that $f(V) = V$.

LTL—semantics

- A property is true (it holds) for a trace w if it is true "at the beginning" ($w, 0 \vDash \phi$)
- A property holds for a system if it is true for all its (maximal) trace
  - how many traces can there be ?

LTL—semantics

We consider finite-state systems, hence a maximal trace either ends in a deadlock or it has a cycle



LTL—semantics

Example: $F\ G\ s_1$ (that is $\langle\,\rangle[\,]s_1$)

	0	1	2	3	4	5	6	7	8	…
s_1 =	$\top$	$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$
$[\,]s_1$ =	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$
$\langle\,\rangle[\,]s_1$ =	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
w =	s_1	s_2	s_1	s_1	s_2	s_1	s_1	s_3	s_1	… s_1 *

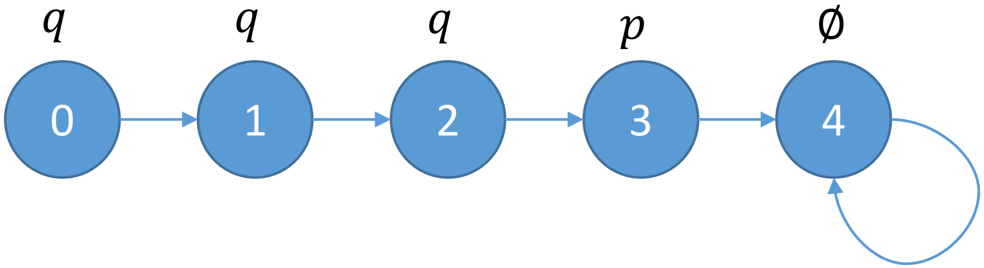
LTL—semantics

Example: $F\ (s_3 \vee G\ s_2)$ (that is $\langle\,\rangle(s_3 \vee [\,]s_2)$)

	0	1	2	3	4	5	6	7	8	…
$s_3 =$	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊤	⊥	⊥
$[\,]s_2 =$	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥
$s_3 \vee [\,]s_2 =$	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊤	⊥	⊥
$\langle\,\rangle(\ldots) =$	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊥	⊥
$w =$	s_1	s_2	s_1	s_1	s_2	s_1	s_1	s_3	s_1	$\ldots\ s_1{}^*$

Exercise

Check $w, i \vDash \phi$



	0	1	2	3	4	5	…
$q\ U\ p$							
$F\ G\ \neg p$							
$F(q\ U\ p)$							
$F\neg(q\ U\ p)$							
$\neg G(q\ U\ p)$							
$\neg G\neg(q\ U\ p)$							

Model Checking

LTL specifications

Example specification

- Mutual exclusion

  never more than one process can be in state working at any given time

- No starvation

  a process that wants to work (in state waiting) should eventually reach state working

- Bounded usage time

  a process in state working should eventually be idle

Example specification

atomic prop. are: $idle_i$, $wait_i$, and $work_i$.

- Mutual exclusion

$$[\,]\neg(work_i \land work_j)$$

- No starvation

$$[\,](wait_i \Rightarrow \langle\,\rangle work_i)$$

$$[\,](wait_i \Rightarrow (wait_i \; U \; work_i))$$

unnecessary

- Bounded usage time

$$[\,](work_i \Rightarrow \langle\,\rangle \neg work_i)$$

Exercise

Additional specification

We say that ϕ precedes ψ holds for w, at k (written $w, k \vDash \phi\ P\ \psi$) when:

$$\forall j \geq k\,.\,(w, j \vDash \psi) \Rightarrow \exists i \in [k, j].\,(w, i \vDash \phi)$$

that is, $w \vDash \phi\ P\ \psi$ as soon as:

$$\forall j.\,(w, j \vDash \psi) \Rightarrow \exists i \leq j.\,(w, i \vDash \phi)$$

can you express this new modality in LTL or should we add it to the logic ?

Additional specification

We can write the following requirement as follows: “*access to the critical section is allowed only to the workers that asked for it*”

$$[\,](\neg work_i \Rightarrow (wait_i \; P \; work_i))$$

that is, before working, process i must have asked it.

Could you express the stronger requirement that: “*access to the critical section is granted in the order where workers asked for it*” ?

Model Checking

using Tina selt

tina > selt

- The tina toolbox has a LTL model-checker called selt
- The program takes as input a reachability graph (either in AUT format, or in a compressed format called KTZ)
- LTL formulas include:

negation	—	implication	=>
conjunction	/\	disjunction	\/
always	[]	eventually	<>
constants	T (true), F (false), dead		

Some examples of formulas

[] (p1 /\ p2) ;

p1 and p2 always true (everywhere)

<> (p1 \/ p2) ;

means either p1 or p2 is true in every trace

[] (<> p) ;

means p true infinitely often

<> ([] p) ;

means p will become always true

How to use selt

1. use nd to draw/open a Petri net
2. use tool > reachability to generate the marking graph in compressed (ktz) format
3. you can either
   A. invoke selt directly from the nd window (right click then choose “model check LTL”)
   B. save in a file, say xx.ktz, and invoke “selt xx.ktz” in the command line
4. Every input must end with a semi-colon: “;”
5. When a property is false, a counter-example is printed

see http://projects.laas.fr/tina/manuals/selt.html

How to use selt

- Counter-examples can be replayed in the simulator (if it is already open)
- There are several levels of details for printing the counter-examples:

```
output fullproof ;
```

- To quit selt, simply enter “quit;”
- There are other commands, go see: http://projects.laas.fr/tina/manuals/selt.html