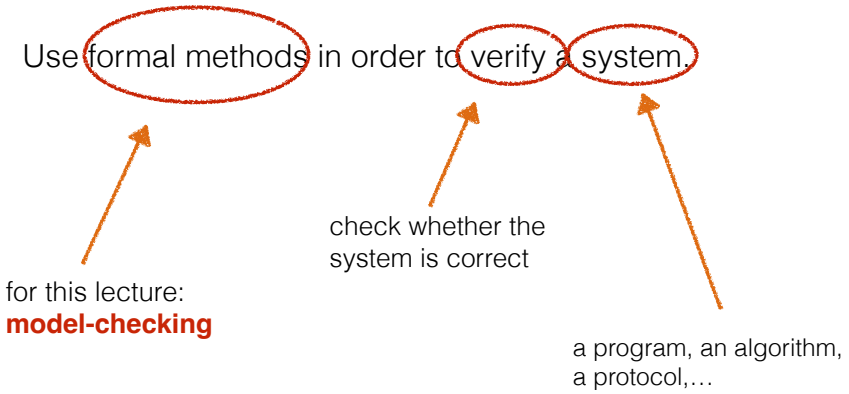
FORMAL VERIFICATION

Part 1

François Laroussinie
IRIF, Univ. Paris Cité

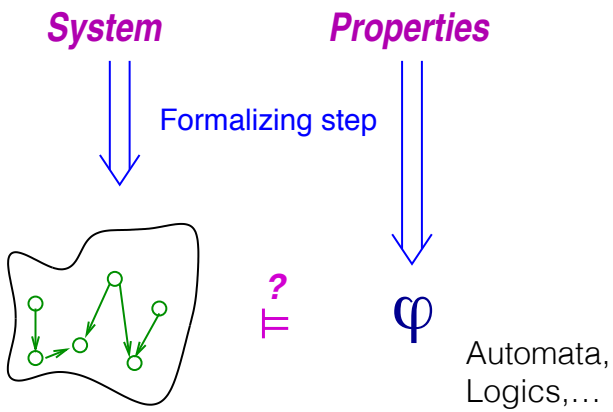
Plan

What is the formal verification ?



General framework:

Model-checking



For what types of « system » ?

For critical systems.
→ When a mistake can have serious consequences.

- Transport
- Energy
- Medicine
- Embedded systems
- Communication
- …

https://en.wikipedia.org/wiki/List_of_software_bugs

For what types of « system » ?

For reactive systems.
→ Systems that maintain an ongoing interaction with their environment

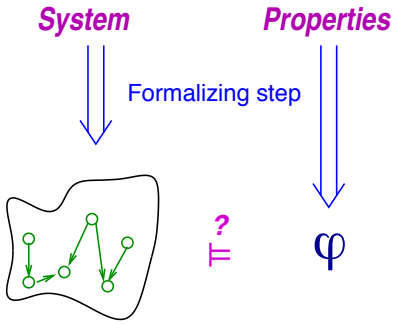
(very different from classical systems with one input and one output !)

Their correctness properties deal with the events along (possibly infinite) executions.

Ex: "If a problem occurs, an alarm will start".

or: along an execution, if a problem occurs at some point, then an alarm will be turned on afterwards.

Model-checking



4 steps:
- Build the model
- Write the formal specification
- Use a model-checker
- (Deduce something about the original system !)

All the steps are difficult (and connected) !

- Build the model ?
  Simplify (abstract) the system, keep the pertinent informations.
  Avoid the complexity blow-up.
- Write the formal specification
  Translate properties into some logics (for a model).
- Use a model-checker
- (Deduce something about the original system !)
  Adapt the model, start again etc.

Verification problems

Model-checking:
input: a model $\mathbf{S}$ and a formula φ
output: yes iff $\mathbf{S} \models \varphi$.

Satisfiability:
input: a formula φ
output: yes iff there exists a model satisfying φ.
(+ a model if the answer is yes…)

Controller synthesis:
input: a model $\mathbf{S}$ and a formula φ
output: a controller $\boldsymbol{C}$ s.t. $\mathbf{S} \times \boldsymbol{C} \models \varphi$.

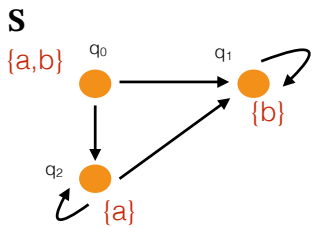
Build the model

Which formal model ?

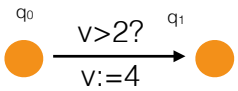
There are many possibilities…

- Labeled transition systems (or Kripke structures)
- Parallel compositions of LTS
- Automata (with variables, clocks, channels…)
- Petri nets
- Process algebra
- Probabilistic transition systems
- …

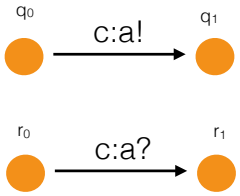
Labeled Transition System:



With variables:



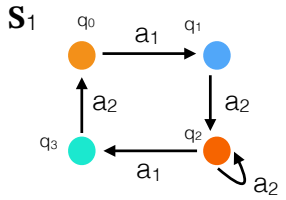
With channels:

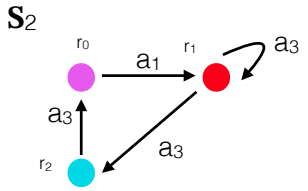


…

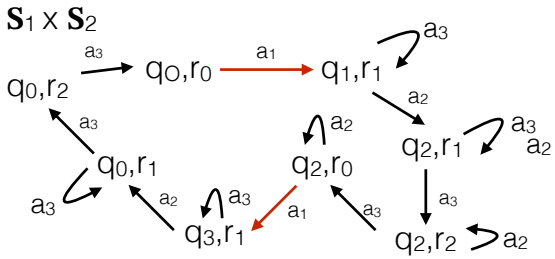
Here: finite labeled transition systems.

Parallel composition & synchronisation





S_1	S_2
a_1	a_1
a_2	-
-	a_3

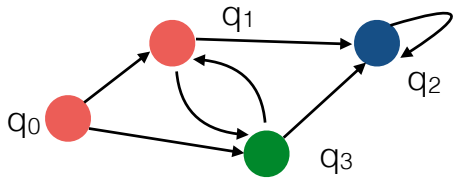


Kripke structures

AP: the set of atomic propositions.

Definition:
A Kripke structure **S** is a 4-tuple $\langle Q, q_0, R, \ell \rangle$:

- Q is a finite set of states
- $q_0 \in Q$ is the initial state
- $R \subseteq Q \times Q$ is the accessibility relation (often required to be total)
- $\ell : Q \to 2^{AP}$: gives the valuation of atomic propositions in each state.



Executions/paths from q_0:
$q_0(q_1q_3)^\omega$,
$q_0q_1q_3(q_2)^\omega$, …

Kripke structures

Definition:
A Kripke structure **S** is a 4-tuple $\langle Q, q_0, R, \ell \rangle$:

- Q is a finite set of states
- $q_0 \in Q$ is the initial state
- $R \subseteq Q \times Q$ is the accessibility relation (often required to be total)
- $\ell : Q \rightarrow 2^{AP}$: gives the valuation of atomic propositions in each state.

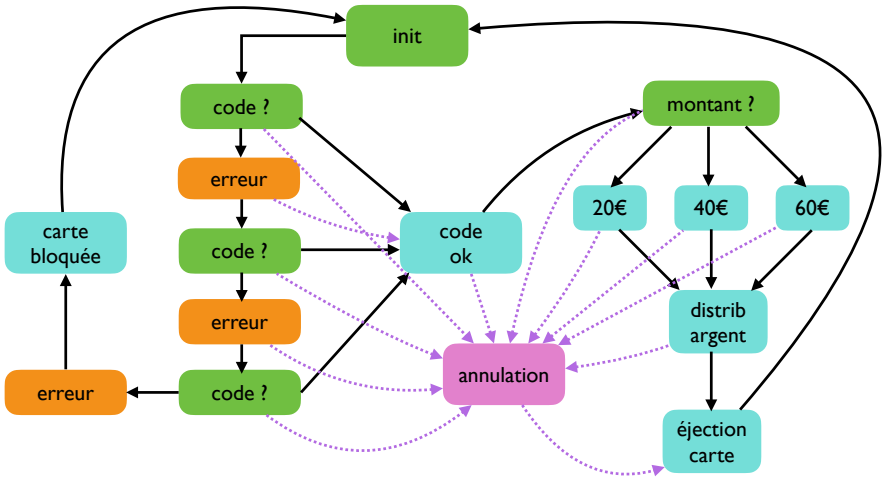
Executions: $\rho = s_0 s_1 s_2 \ldots \in Q^\omega$ such that $\forall i \geq 0, (s_i, s_{i+1}) \in R$

Notation:

$\rho(i) = s_i \qquad \rho_{\geq i} = s_i s_{i+1} \ldots \qquad \rho_{\leq i} = s_0 \ldots s_i$

Exec(q) = set of exec. issued from q.
Exec(**S**) = set of all exec. in **S**.

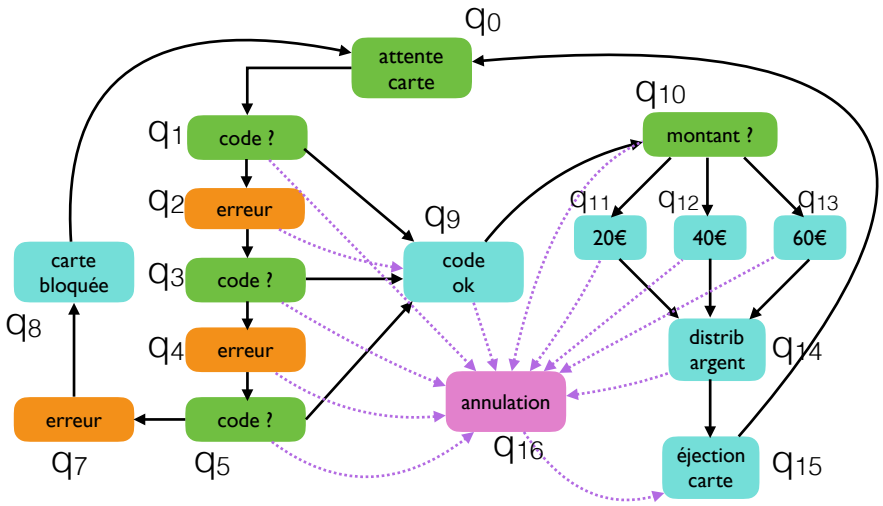
Modelling a cash dispenser



A Kripke structure

S = (Q,Act,→,q_0,AP,L)

AP={att. carte, code?, c. bloquée, code ok, …}



Example:
Mutual exclusion problem

Mutual exclusion problem

2 processes communicate with shared memory.
They aim at reaching a *critical* section (CS) and at any time at most one process can be in CS.

How to design a correct protocol ?

Correct ?

1. Mutual exclusion: Both processes van never be at their CS at the same time.
2. No starvation: If a process asks for the CS, it will reach it sometime in the future (if the execution is fair).

(…)

Mutual exclusion algo.

```
boolean D1:= False
boolean D2 := False
```

```
Processus P1:
loop forever:
 p1: non-crit. section
 p2: D1 := True
 p3: await (not D2)
 p4: critical section
 p5: D1 := False
```

```
Processus P2:
loop forever:
 p1: non-crit. section
 p2: D2 := True
 p3: await (not D1)
 p4: critical section
 p5: D2 := False
```

+ every action is atomic.

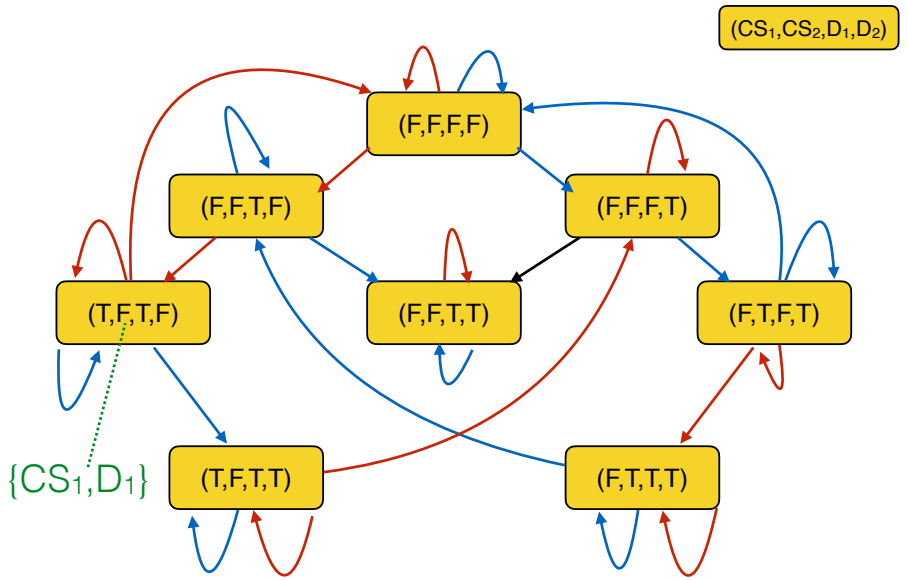
Mutual exclusion algo.

→ a KS with AP={D_1,D_2,SC_1,SC_2}

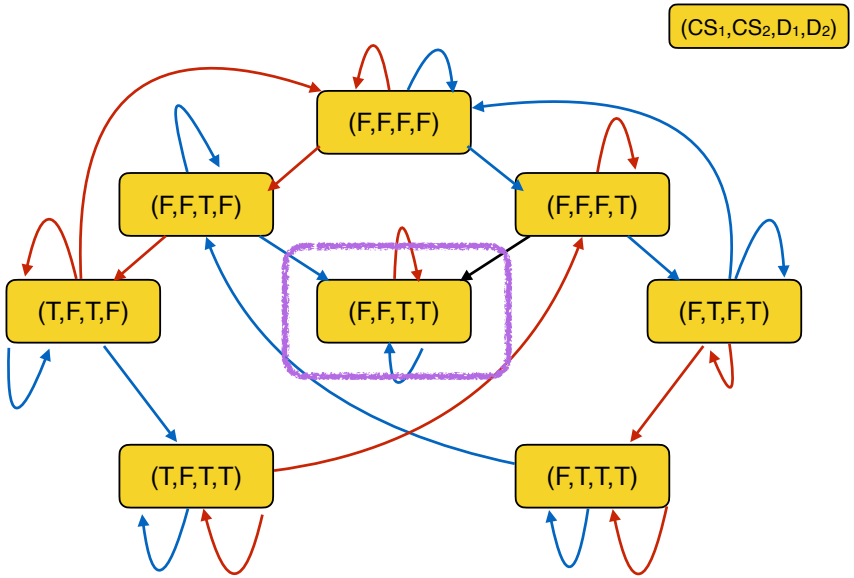
- D_1 labels states where the variable D1 equals T,
- D_2 labels states where the variable D2 equals T,
- CS_1 labels states where process 1 is in CS,
- CS_2 labels states where process 2 is in CS.

→ 4 atomic propositions.

Mutual exclusion algo.



Is-it correct ? No !



Temporal logics for the specification of reactive systems

Temporal Logics

Temporal logics extend Propositional Logic with « temporal modalities ».
Formulas re interpreted over models where a notion of time is defined.

There are a lot of temporal logics !!

- LTL « Linear-time Temporal Logic »
- CTL « Computation Tree Logic »

Temporal logics

TLs are a good formalism for specifying properties of reactive systems.

[Pnueli 1977]

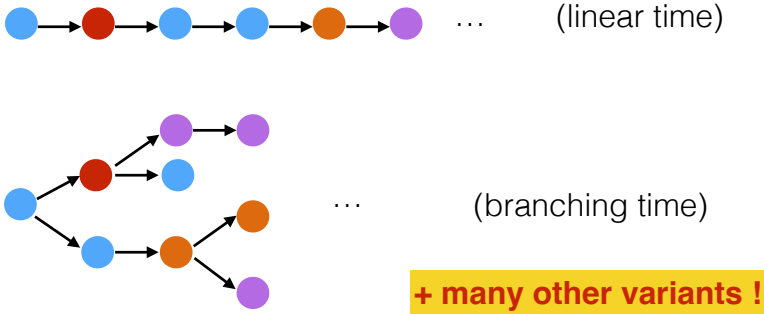
Allow to state properties about the *order of events along* an execution.

- natural semantics,
- good expressive power,
- succinctness,
- efficient decision procedures (and tools !),
- many extensions (for timed systems, probabilistic systems, games, data,…).

Temporal logics

Temporal logics = propositional logic
+
temporal modalities

+ a model of time !

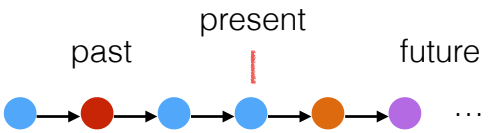


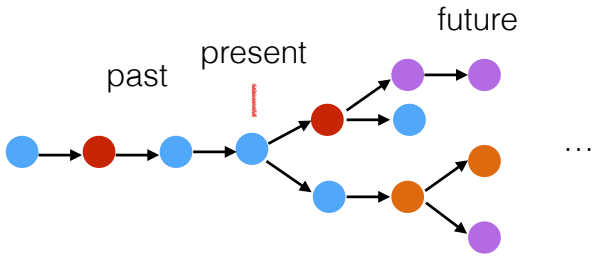
… (linear time)

… (branching time)

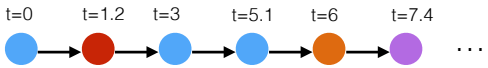
+ many other variants !!

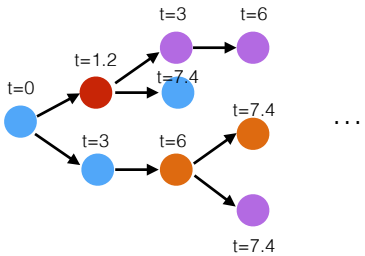
Models of time
with past





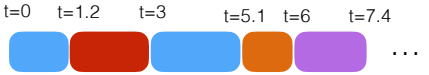
Models of time
with (discrete) real-time

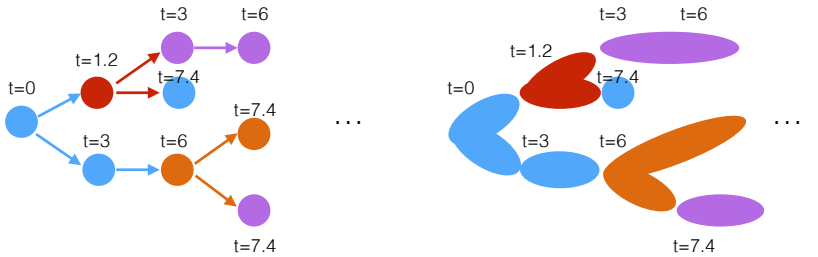




→ Quantitative notion of time

Models of time
with (dense) real-time





Linear Time

LTL « Linear-time Temporal Logic »

The behaviour of a system (ie a Kripke structure) is viewed as a set of its executions.

LTL (with past)

Syntax: $P \in AP$

$\phi,\psi ::= P \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \mathbf{X}\,\phi \mid \phi \mathbf{U} \psi \mid \mathbf{X^{-1}}\,\phi \mid \phi \mathbf{S} \psi$

$\mathbf{X}\,\phi$: Next (tomorrow) ϕ
$\phi \mathbf{U} \psi$: ϕ until ψ
$\mathbf{X^{-1}}\,\phi$: Previous (yesterday) ϕ
$\phi\ \mathbf{S^{-1}} \psi$: ϕ since ψ

→ LTL formulas are interpreted over a position along a linear model where every position is labeled with atomic propositions. For example:

- $\pi : \mathbf{N} \rightarrow 2^{AP}$
- $\pi \in (2^{AP})^{\omega}$
- $\rho \in Exec(q) + \ell$: a (labeled) execution in a KS.
- …

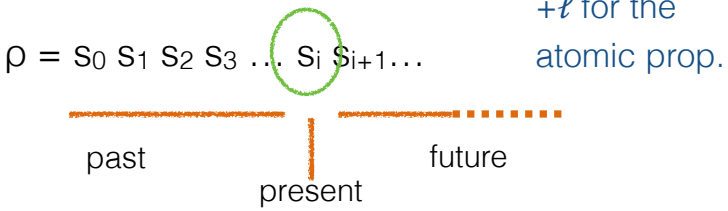
LTL (with past)

Syntax: $P \in AP$

$\phi,\psi ::= P \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \mathbf{X}\,\phi \mid \phi \mathbf{U} \psi \mid \mathbf{X^{-1}}\,\phi \mid \phi \mathbf{S} \psi$

→ LTL formulas are interpreted over a position along an execution (of some Kripke structure $\mathbf{S}$).

$+\ell$ for the atomic prop.



LTL (with past)

Syntax:

$\phi,\psi ::= P \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \mathbf{X}\,\phi \mid \phi \mathbf{U} \psi \mid \mathbf{X^{-1}}\,\phi \mid \phi \mathbf{S} \psi$

Semantics:

$\mathbf{S}=\langle Q,q_0,R,\ell \rangle$, $\rho \in Exec(\mathbf{S})$

$i \geq 0$

$\rho,i \models \phi$?

$\rho,i \models P$ iff $P \in L(\rho(i))$

$\rho,i \models \neg\phi$ iff $\rho,i \not\models \phi$

$\rho,i \models \phi \wedge \psi$ iff ($\rho,i \models \phi$ and $\rho,i \models \psi$)

$\rho,i \models \phi \vee \psi$ iff ($\rho,i \models \phi$ or $\rho,i \models \psi$)

LTL (with past)

Syntax:

$\phi,\psi ::= P \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \mathbf{X}\,\phi \mid \phi \mathbf{U} \psi \mid \mathbf{X^{-1}}\,\phi \mid \phi \mathbf{S} \psi$

Semantics:

$\mathbf{S}=\langle Q,q_0,R,\ell \rangle$, $\rho \in Exec(\mathbf{S})$ $i \geq 0$

$\rho,i \models \mathbf{X}\,\phi$ iff $\rho,i+1 \models \phi$

$\rho,i \models \phi \mathbf{U} \psi$ iff ($\exists\, j \geq i.$ $\rho,j \models \psi$ and

$(\forall\, i \leq k < j$ we have $\rho,k \models \phi$))

$\rho,i \models \mathbf{X^{-1}}\,\phi$ iff ($i>0$ and $\rho,i-1 \models \phi$)

$\rho,i \models \phi \mathbf{S} \psi$ iff ($\exists\, 0 \leq j \leq i.$ $\rho,j \models \psi$ and

$(\forall\, j < k \leq i$ we have $\rho,k \models \phi$))

LTL (with past)

$\mathbf{S} \models \phi$?

$\mathbf{S} \models \phi$ if and only if $\rho,0 \models \phi$ $\forall\, \rho \in Exec(q_0)$

Semantics of LTL

$\rho,i \models P$ iff $P \in L(\rho(i))$
$\rho,i \models \neg\phi$ iff $\rho,i \not\models \phi$
$\rho,i \models \phi \wedge \psi$ iff ($\rho,i \models \phi$ and $\rho,i \models \psi$)
$\rho,i \models \phi \vee \psi$ iff ($\rho,i \models \phi$ or $\rho,i \models \psi$)

$\rho,i \models \mathbf{X}\,\phi$ iff $\rho,i+1 \models \phi$
$\rho,i \models \phi \mathbf{U} \psi$ iff ($\exists\, j \geq i.\ \rho,j \models \psi$ and ($\forall\, i \leq k < j$ we have $\rho,k \models \phi$))
$\rho,i \models \mathbf{X^{-1}}\,\phi$ iff ($i>0$ and $\rho,i-1 \models \phi$)
$\rho,i \models \phi \mathbf{S} \psi$ iff ($\exists\, 0 \leq j \leq i.\ \rho,j \models \psi$ and ($\forall\, j < k \leq i$ we have $\rho,k \models \phi$))

With LTL, a Kripke structure is viewed as a set infinite words over 2^{AP} :

$\rho = q_0 q_1 q_2 q_3 q_4 \ldots \in Q^\omega + \ell$
$= \ell(q_0)\, \ell(q_1)\, \ell(q_2)\, \ell(q_2) \ldots \in (2^{AP})^\omega$

New operators

Boolean connectives: $\Rightarrow$, $\Longleftrightarrow$, $\top$, $\bot$, …

Temporal modalities:

$\mathbf{F}\,\psi = \top\, \mathbf{U}\, \psi$ « sometime in the future »
$\mathbf{F^{-1}}\,\psi = \top\, \mathbf{S}\, \psi$ « sometime in the past »

And
$\mathbf{G}\,\psi = \neg\, \mathbf{F}\, \neg\, \psi$ « Always in the future »
$\mathbf{G^{-1}}\,\psi = \neg \mathbf{F^{-1}}\, \neg\psi$ « Always in the past »

New operators

Weak until :

$\rho,i \models \psi\, \mathbf{W}\, \phi$ iff ($\forall\, k \geq i$, $\rho,k \models \psi$ ou $\exists\, j \geq i.$ ($\rho,j \models \phi$ et $\forall\, i \leq k < j$ on a $\rho,k \models \psi$)

$\rho,i \models \psi\, \mathbf{W}\, \phi \iff \rho,i \models \mathbf{G}\,\psi \vee \psi\, \mathbf{U}\, \phi \qquad (\forall\, \rho, \forall\, i)$

Release

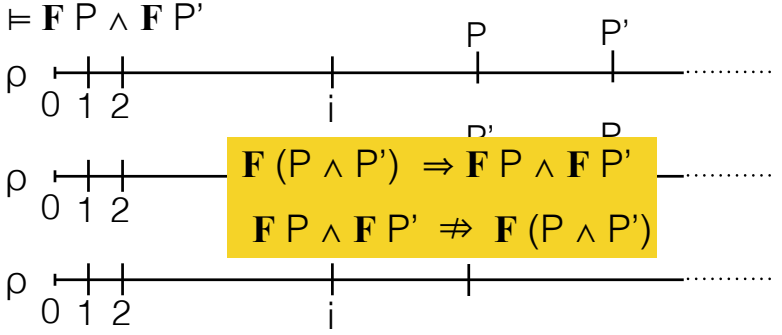
$\rho,i \models \psi\, \mathbf{R}\, \phi$ = iff ($\forall\, k \geq i$, ($\rho,k \models \phi$ ou $\exists\, i \leq j < k$ $\rho,j \models \psi$))

$\rho,i \models \psi\, \mathbf{R}\, \phi \iff \rho,i \models \phi\ \mathbf{W}\ (\psi \wedge \phi) \qquad (\forall\, \rho, \forall\, i)$

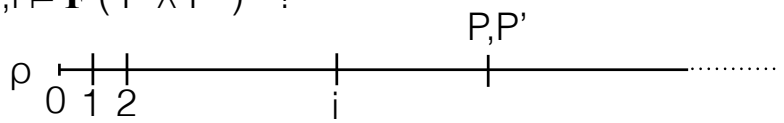
Examples

Compare $\mathbf{F}\, P \wedge \mathbf{F}\, P'$ and $\mathbf{F}\, (P \wedge P')$

$\rho,i \models \mathbf{F}\, P \wedge \mathbf{F}\, P'$

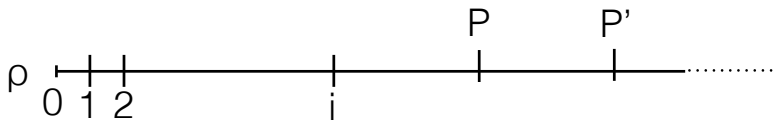


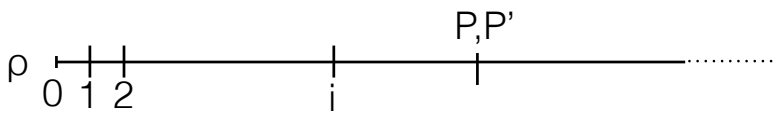
$\rho,i \models \mathbf{F}\, (\, P \wedge P'\,)$?



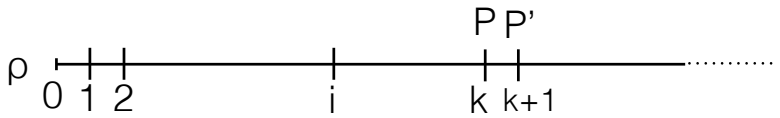
Examples

$\rho,i \models \mathbf{F}\,(P \wedge \mathbf{F}\,P')$?





$\rho,i \models \mathbf{F}\,(P \wedge \mathbf{X}\,P')$?



Examples

$\rho,i \models \neg\,\mathbf{X}\,\phi$

$\Leftrightarrow\ \rho,i \models \mathbf{X}\,\neg\,\phi$ (because executions are infinite)

$\rho,i \models \neg\,\mathbf{X^{-1}}\,\phi$

$\Leftrightarrow\ \rho,i \models (\mathbf{X}\,\neg\,\phi) \vee (\neg\,\mathbf{X^{-1}}\,\top)$ (because past is finite)

Examples

G (problem ⇒ **F** alarm)

G (alarm ⇒ $\mathbf{F^{-1}}$ problem)

G (request ⇒ **F** service)

G (¬ bug)

Examples

G F init

F G ok

GF request ⇒ **GF** service

GF (a ∧ b) entails **GF** a ∧ **GF** b

GF a ∧ **GF** b does not entail **GF** (a ∧ b)

G a ⇒ **FG** a ⇒ **GF** a ⇒ **F** a

Mutual exclusion

1. Mutual exclusion: Both processes can never be at their CS at the same time.

$\neg\, \mathbf{F}\, (CS_1 \wedge CS_2)$ $\qquad$ $\mathbf{G}\, (\neg\, CS_1 \vee \neg\, CS_2)$

2. No starvation: If a process asks for the CS, it will reach it sometime in the future (if the execution is fair).

?

$\mathbf{G}\, (D_1 \Rightarrow \mathbf{F}\ CS_1) \wedge \mathbf{G}\, (D_2 \Rightarrow \mathbf{F}\ CS_2)$

Specifying reactive systems

Safety properties:
"bad things do not happen".
Ex: There is at most one process in the Critical Section.

Liveness properties:
"good things do happen (eventually)".
Ex: Every request for the CS is eventually granted.

Fairness properties:
→ Verification of fair executions.
Ex: Every process should be executed infinitely often.

Specification of a lift

H. Barringer ("Up and down, the temporal way", 1985)

Up and Down The Temporal Way

H. BARRINGER*
Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL

A formal specification of a multiple-lift system is constructed. The example illustrates and justifies one of many possible system specification styles based on temporal techniques.

Received September 1985

1. INTRODUCTION

Over the last decade there has been widespread research directed at obtaining techniques for the analysis, specification and development of concurrent systems. Several of these lines of research have led to the belief that temporal logic is a useful tool for reasoning about such systems.[1–4] The use of temporal logic enables, in particular, analysis of both safety and liveness properties in a single uniform logical framework (see Ref. 5 for extensive examples). More recently, techniques have been developed for achieving compositional temporal proof systems.[6, 7] Compositionality is an essential requirement for the hierarchical development of implementations from formal specifications. Without compositionality, the check on consistency of a development step would be delayed until all interactions between the developed components are known, essentially, at the implementation level; clearly, it could be rather costly if such a consistency check then showed that the system did not achieve the overall specification. In general, compositionality can be achieved by realising that a specification of any component must include assumptions about the behaviour of the environment in which the component will reside. In the temporal framework, this requires that one can distinguish actions made by a component from those made by its environment; in Refs 6 and 7 the coarse technique of labelling actions is used for just that purpose. Although it is sometimes possible

5 presents the multiple-lift system. Final comments are made in Section 6. For convenience, an appendix contains the semantics of the logic used in this article.

2. INFORMAL REQUIREMENTS

The following is a description of the lift-control system problem as set by Neil Davis.

A Lift-Control System

An n-lift system is to be installed in a building with m floors. The lifts and the control mechanism are supplied by a manufacturer. The internal mechanisms of these are assumed (given) in this problem.

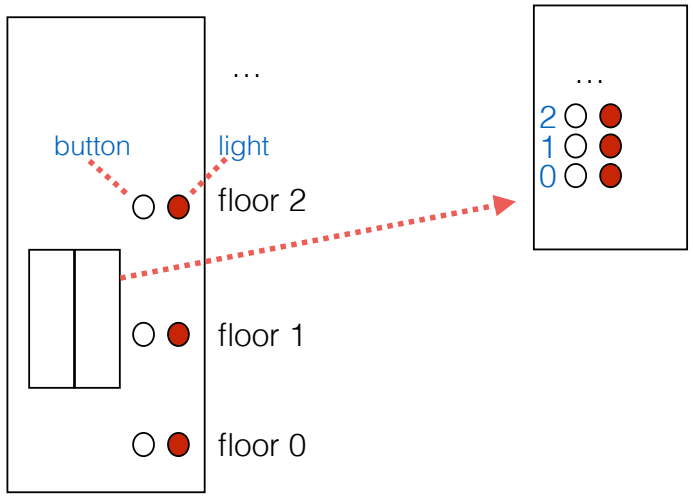
Design the logic to move lifts between floors in the building according to the following rules.

(1) Each lift has a set of buttons, one button for each floor. These illuminate when pressed and cause the lift to visit the corresponding floor. The illumination is cancelled when the corresponding floor is visited (i.e. stopped at) by the lift.

(2) Each floor has two buttons (except ground and top), one to request an up-lift and one to request a down-lift. These buttons illuminate when pressed. The buttons are cancelled when a lift visits the floor and is either travelling in the desired direction, or visiting the floor with no requests outstanding.

In the latter case, if both floor-request buttons are illuminated, only one should be cancelled. The algorithm used to decide which to service should minimise the waiting time for both requests.

Specification of a lift



Specification of a lift

Hypothesis:

- A floor door is open or closed.
- A button is pressed or depressed.
- An indicator light is on or off.
- The cabin is present at floor i, or it is absent.

Specification of a lift

P1. Safe doors:
A floor door is never opened if the cabin is not present at the given floor.

P2. Indicator lights:
The indicator lights correctly reflect the current requests.

P3. Services:
All requests are eventually satisfied.

P4. Smart service:
The cabin only services the requested floors and does not move when there is no request.

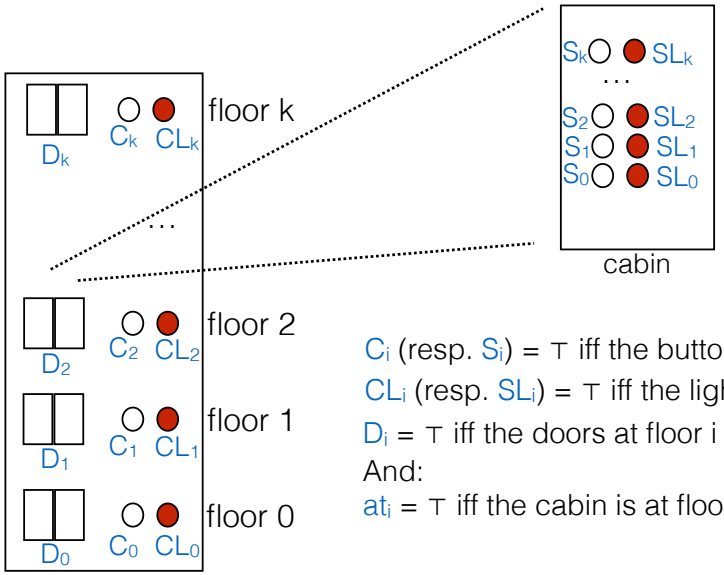
Specification of a lift

P5. Diligent service:
The cabin does not pass by a floor for which it has a request without servicing it.

P6. Direct movements:
The cabin always moves directly from previous to next serviced floor.

P7. Priorities:
The cabin services in priority requests that do not imply a change of direction (upward or downward).

Specification of a lift: the atomic prop.



C_i (resp. S_i) = $\top$ iff the button is pressed
CL_i (resp. SL_i) = $\top$ iff the light is on
D_i = $\top$ iff the doors at floor i are opened
And:
at_i = $\top$ iff the cabin is at floor i

Specification of a lift

P1. Safe doors:
A floor door is never opened if the cabin is not present at the given floor.

$$\bigwedge_{i=1..k} \mathbf{G}\,(\,D_i \Rightarrow at_i)$$

Specification of a lift

P2. Indicator lights:
The indicator lights correctly reflect the current requests.

$\bigwedge_i \mathbf{G}\,(\,C_i \Rightarrow (LC_i \vee servicing_i\,))$: turned on when necessary

with: $servicing_i = at_i \wedge D_i$

$\bigwedge_i \mathbf{G}\,(\,LC_i \Rightarrow (\,LC_i\ \mathbf{W}\ servicing_i\,)\,)$: stay lit when necessary

$\bigwedge_i \mathbf{G}\,(\,servicing_i \Rightarrow (\neg\, LC_i \wedge \neg\, SL_i))$: turned off when necessary

$\bigwedge_i \mathbf{G}\,(\,(\neg\, LC_i\,) \Rightarrow (\,(\neg LC_i)\ \mathbf{W}\ C_i)\,)$:only turned on when necessary

or $\bigwedge_i \mathbf{G}\,(\,LC_i \Rightarrow (\,LC_i\ \mathbf{S}\ C_i)\,)$

(and the same for S_i and SL_i)

Specification of a lift

P2. Indicator lights:
The indicator lights correctly reflect the current requests.

An *alternative* is:

$$\bigwedge_i \mathbf{G}\,(\,LC_i \Longleftrightarrow (\,(\neg servicing_i\,)\ \mathbf{S}\ (C_i \wedge \neg\, servicing_i))\,)$$

(and the same for S_i and SL_i)

Specification of a lift

P3. Services:
All requests are eventually satisfied.

$$\bigwedge_i \mathbf{G}\,(\,request_i \Rightarrow \mathbf{F}\ servicing_i\,))$$

with: $request_i = C_i \vee S_i$

P4. Smart service:
The cabin only services the requested floors and does not move when there is no request.

$$\bigwedge_i \mathbf{G}\,(\ servicing_i \Rightarrow [\ servicing_i\ \mathbf{S}\ (CL_i \vee SL_i)\ ]\ )$$

$$\bigwedge_i \mathbf{G}\,(\ at_i \Rightarrow (\ at_i\ \mathbf{W}\ (\ \bigvee_{j\neq i} (CL_j \vee SL_j)\ )\ )\ )$$

Specification of a lift

P5. Diligent service:
The cabin does not pass by a floor for which it has a request without servicing it.

$$\bigwedge_i \mathbf{G}\,([(\ LC_i \vee LS_i) \wedge at_i\ ] \Rightarrow (at_i\ \mathbf{U}\ servicing_i\ ))$$

Specification of a lift

P6. Direct movements:
The cabin always moves directly from previous to next serviced floor.

$$\bigwedge_{i<j} \mathbf{G}\,(\ From\_i\_to\_j\ \Rightarrow (at_i \vee betw\_floors)\ \mathbf{U}\ (at_{i+1} \wedge\ (at_{i+1} \vee betw\_floors)\ \mathbf{U}\ (at_{i+2}\ \ldots (at_{j-1} \wedge (at_{j-1} \vee betw\_floors)\ \mathbf{U}\ at_j))))$$

$\neg at_0 \wedge \ldots \wedge \neg at_n$

$servicing_i \wedge [\ (servicing_i \vee \neg\ service)\ \mathbf{U}\ servicing_j\ ]$

$service = servicing_0 \vee servicing_1 \vee \ldots \vee servicing_k$

Specification of a lift

P7. Priorities:
The cabin services in priority requests that do not imply a change of direction (upward or downward).

$$Up = \bigvee_{i=1..k} [\ (at_i \vee betw\_floors\ )\ \mathbf{S}\ at_{i-1}\ \wedge\ (at_i \vee betw\_floors)\ \mathbf{U}\ at_i\ ]$$

$$Down = \bigvee_{i=0..k-1} [\ (at_i \vee betw\_floors\ )\ \mathbf{S}\ at_{i+1}\ \wedge\ (at_i \vee betw\_floors)\ \mathbf{U}\ at_i\ ]$$

$$\mathbf{G} \bigwedge_{i=0..k-1} [(servicing_i \wedge Down \wedge \bigvee_{j<i} (CL_j \vee SL_j)\ ) \Rightarrow \bigvee_{n<i} From\_i\_to\_n]$$

$$\mathbf{G} \bigwedge_{i=1..k} [(servicing_i \wedge Up \wedge \bigvee_{j>i} (CL_j \vee SL_j)\ ) \Rightarrow \bigvee_{n>i} From\_i\_to\_n]$$

Peterson's algorithm

Peterson's algorithm

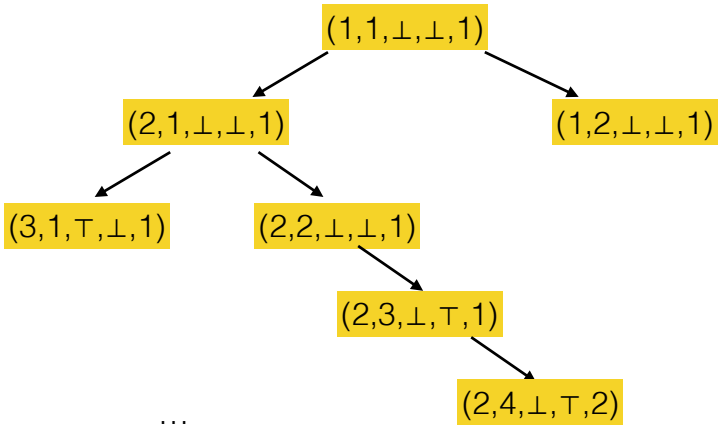
```
boolean D1:= False
boolean D2 := False
integer  turn:= _
```

```
Processus P1:
loop forever:
 p1: Non-crit. section
 p2: D1 := True
 p3: turn:=2
 p4: await (¬D2 v turn=1)
 p5: critical section
 p6: D1 := False
```

```
Processus P2:
loop forever:
 q1: Non-crit. section
 q2: D2 := True
 q3: turn:=1
 q4: await (¬D1v turn=2)
 q5: critical section
 q6: D2 := False
```

Peterson's algorithm

state : (p,q,D1,D2,turn)



6x6x2x2x2 = 288 states (OK, simplifications are possible…)

Mutual exclusion

1. Mutual exclusion: Both processes can never be at their CS at the same time.

$\neg\, \mathbf{F}\, (CS_1 \wedge CS_2)$ $\quad$ $\mathbf{G}\, (\neg\, CS_1 \vee \neg\, CS_2)$ ✓

2. No starvation: If a process asks for the CS, it will reach it sometime in the future (if the execution is fair).

$\mathbf{G}\, (D_1 \Rightarrow \mathbf{F}\ CS_1) \wedge \mathbf{G}\, (D_2 \Rightarrow \mathbf{F}\ CS_2)$ ✗

Mutual exclusion

2. No starvation: If a process asks for the CS, it will reach it sometime in the future (if the execution is fair).

How can we express the fairness ?

How can we characterise the fair executions ?

288x2 states !!

Add a boolean « p_a » in states to specify which process has performed the last action ($\perp$ for P_1, $\top$ for P_2).

$$\mathbf{G}\,(\,D_1 \Rightarrow (\,(\mathbf{GF}\,\neg p_a) \Rightarrow \mathbf{F}\;\; CS_1)$$ ✓

Peterson's algorithm V2

Is it okay to modify the model in order to express a property ?

Peterson's algorithm with Prism

https://www.prismmodelchecker.org/

```
global turn : [1 .. 2];
```

```
module P1
        D1 : bool init false;
        p : [0..3] init 0;
    [] (p=0) -> true;
    [] (p=0) -> (p'=1) & (D1'=true);
    [] (p=1) -> (p'=2) & (turn'=2);
    [] (p=2) & ((turn=1) | (D2=false)) -> (p'=3);
    [] (p=2) & ((turn=2) & (D2=true)) -> (p'=2);
    [] (p=3) -> (p'=0) & (D1'=false);
endmodule
```

```
module P2
        D2 : bool init false;
        q : [0..3] init 0;
    [] (q=0) -> true;
    [] (q=0) -> (q'=1) & (D2'=true);
    [] (q=1) -> (q'=2) & (turn'=1);
    [] (q=2) & ((turn=2) | (D1=false)) -> (q'=3);
    [] (q=2) & ((turn=1) & (D1=true)) -> (q'=2);
    [] (q=3) -> (q'=0) & (D2'=false);
endmodule
```

We use a « high-level » language !