Cours MALG & MOVEX

Modélisation, spécification et vérification (I)
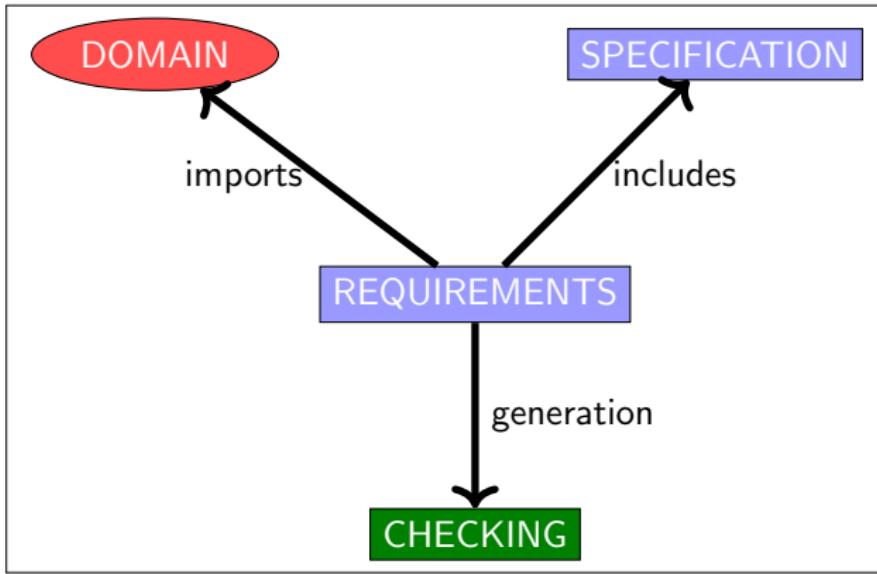
Dominique Méry

Telecom Nancy, Université de Lorraine

Année universitaire 2025-2026 (16 février 2026 12:51 A.M.)

Plan

Sommaire

Current Summary

1 Programming by Contract

2 Summary of the Tryptich

3 Transition Systems
  Overview of Transition Systems as Modelling Tool
  Expression of transition systems
  Main concepts of discrete transition system

4 Transition system in action with TLA/TLA[+]
 Example 1 ADD (cours0.tla)
  Example 2 Simple Access Control (cours2.tla)
  Transition system using TLA/TLA[+]

5 stop movex 2

6 The TLA[+] Toolbox

7 PlusCal an algorithmic notation inside TLA[+]

8 Conclusion

Method for verifying program properties

A program P *satisfies* a (pre,post) contract :

- ▶ P transforms a variable x from initial values x_0 and produces a final value x_f : $x_0 \xrightarrow{P} x_f$

- ▶ x_0 satisfies pre : $\mathsf{pre}(x_0)$ and x_f satisfies post : $\mathsf{post}(x_0, x_f)$

- ▶ $\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

- ▶ $\mathbb{D}$ est le domaine RTE de X

requires $pre(x_0)$
ensures $post(x_0, x_f)$
variables X

$\begin{bmatrix} \text{begin} \\ 0 : P_0(x_0, x) \\ \texttt{instruction}_0 \\ \dots \\ i : P_i(x_0, x) \\ \dots \\ \texttt{instruction}_{f-1} \\ f : P_f(x_0, x) \\ \text{end} \end{bmatrix}$

- ▶ $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- ▶ $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$
- ▶ For any pair of labels ℓ, ℓ' such that $\ell \longrightarrow \ell'$, one verifies that, pour any values $x, x' \in \mathrm{MEMORY}$

$$\left(\begin{pmatrix} pre(x_0) \wedge P_\ell(x_0, x)) \\ \wedge cond_{\ell,\ell'}(x) \wedge x' = f_{\ell,\ell'}(x) \\ \Rightarrow P_{\ell'}(x_0, x') \end{pmatrix} \right),$$

- ▶ For any pair of labels m, n such that $m \longrightarrow n$, one verifies that, $\forall x, x' \in \mathrm{MEMORY} : pre(x_0) \wedge P_m(x_0, x) \Rightarrow \mathbf{DOM}(m, n)(x)$

Example of an annotation

```
VARIABLES X
REQUIRES ...
ENSURES ...
WHILE 0 < X DO
  X := X−1;
 OD;
```

VARIABLES X
REQUIRES ...
ENSURES ...
WHILE $0 < X$ **DO**
  $X := X-1;$
 OD;

$\longrightarrow$

VARIABLES X
REQUIRES $\ldots$
ENSURES $\ldots$
WHILE $0 < X$ **DO**
$\quad X := X-1;$
OD;

$\longrightarrow \longrightarrow$

VARIABLES X
REQUIRES $\ldots$
ENSURES $\ldots$
WHILE $0 < X$ **DO**
  $X := X - 1;$
 OD;

$\longrightarrow \longrightarrow$

CONTRACT EX
VARIABLES $X(int)$
REQUIRES $x_0 \in \mathbb{N}$
ENSURES $x_f = 0$
  $\ell_0 : \{\ x = x_0 \land x_0 \in \mathbb{N}\}$
WHILE $0 < X$ **DO**
  $\ell_1 : \{0 < x \leq x_0 \land x_0 \in \mathbb{N}\}$
  $X := X - 1;$
  $\ell_2 : \{0 \leq x \leq x_0 \land x_0 \in \mathbb{N}\}$
 OD;
  $\ell_3 : \{x = 0\}$

Tryptic Pattern

Current Summary

1 Programming by Contract

2 Summary of the Tryptich

3 Transition Systems
Overview of Transition Systems as Modelling Tool
Expression of transition systems
Main concepts of discrete transition system

4 Transition system in action with TLA/TLA$^+$
Example 1 ADD (cours0.tla)
Example 2 Simple Access Control (cours2.tla)
Transition system using TLA/TLA$^+$

5 stop movex 2

6 The TLA$^+$ Toolbox

7 PlusCal an algorithmic notation inside TLA$^+$

8 Conclusion

▶ $\mathcal{R}$: system requirements.

- ▶ $\mathcal{R}$: system requirements.
- ▶ $\mathcal{D}$: problem domain.

- $\mathcal{R}$: system requirements.
- $\mathcal{D}$: problem domain.
- $\mathcal{S}$: system specification.

- ▶ $\mathcal{R}$: system requirements.
- ▶ $\mathcal{D}$: problem domain.
- ▶ $\mathcal{S}$: system specification.

$$\mathcal{D}, \mathcal{S} \quad \text{SATISFIES} \quad \mathcal{R}$$

- ▶ $\mathcal{R}$: system requirements.
- ▶ $\mathcal{D}$: problem domain.
- ▶ $\mathcal{S}$: system specification.

$$\mathcal{D}, \mathcal{S} \quad \text{SATISFIES} \quad \mathcal{R}$$

- ▶ $\mathcal{R}$: pre/post.
- ▶ $\mathcal{D}$: integers, reals, . . .
- ▶ $\mathcal{S}$: code, procedure, program, . . .

A program P *satisfies* a (pre,post) contract :

▶ P transforms a variable v from initial values v_0 and produces a final value v_f : $v_0 \xrightarrow{\text{P}} v_f$

▶ v_0 satisfies pre : $\text{pre}(v_0)$ and v_f satisfies post : $\text{post}(v_0, v_f)$

▶ $\text{pre}(v_0) \wedge v_0 \xrightarrow{\text{P}} v_f \Rightarrow \text{post}(v_0, v_f)$

▶ $\mathbb{D}$ est le domaine RTE de V

```
requires pre(v_0)
ensures post(v_0, v_f)
variables X
    begin
    0 : P_0(v_0, v)
    instruction_0
    ...
    i : P_i(v_0, v)
    ...
    instruction_{f-1}
    f : P_f(v_0, v)
    end
```

▶ $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$

▶ $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$

▶ For any pair of labels ℓ, ℓ' such that $\ell \longrightarrow \ell'$, one verifies that, pour any values $v, v' \in \text{MEMORY}$

$$\left(\begin{array}{c} \left(\begin{array}{c} pre(v_0) \wedge P_\ell(v_0, v)) \\ \wedge cond_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$

▶ For any pair of labels m, n such taht $m \longrightarrow n$, one verifies that, $\forall v, v' \in \text{MEMORY}$:
$$pre(v_0) \wedge P_m(v_0, v) \Rightarrow \textbf{DOM}(m, n)(v)$$

Example of an annotation

VARIABLES X
REQUIRES $\ldots$
ENSURES $\ldots$
WHILE $0 < X$ **DO**
  $X := X - 1;$
 OD;

Example of an annotation

VARIABLES X
REQUIRES ...
ENSURES ...
WHILE $0 < X$ **DO**
  $X := X-1;$
 OD;

$\longrightarrow$

VARIABLES X
REQUIRES $\ldots$
ENSURES $\ldots$
WHILE $0 < X$ **DO**
  $X := X - 1;$
 OD;

$\longrightarrow \longrightarrow$

VARIABLES X
REQUIRES ...
ENSURES ...
WHILE $0 < X$ **DO**
  $X := X-1;$
 OD;

$\longrightarrow \longrightarrow \longrightarrow$

Example of an annotation

```
VARIABLES X
REQUIRES ...
ENSURES ...
WHILE 0 < X DO
  X := X−1;
OD;
```

$\longrightarrow \longrightarrow \longrightarrow \longrightarrow$

Example of an annotation

VARIABLES X
REQUIRES $\ldots$
ENSURES $\ldots$
WHILE $0 < X$ **DO**
$\quad X := X-1;$
OD;

$\longrightarrow \longrightarrow \longrightarrow \longrightarrow$

CONTRACT EX
VARIABLES $X(int)$
REQUIRES $x_0 \in \mathbb{N}$
ENSURES $x_f = 0$
$\quad \ell_0 : \{\ x = x_0 \wedge x_0 \in \mathbb{N}\}$
WHILE $0 < X$ **DO**
$\quad \ell_1 : \{0 < x \le x_0 \wedge x_0 \in \mathbb{N}\}$
$\quad X := X-1;$
$\quad \ell_2 : \{0 \le x \le x_0 \wedge x_0 \in \mathbb{N}\}$
OD;
$\quad \ell_3 : \{x = 0\}$

Current Summary

Current Subsection Summary

Transition system

A transition system $\mathcal{ST}$ is given by a set of states Σ, a set of initial states $Init$ and a binary relation $\mathcal{R}$ on Σ.

▶ The set of terminal states $Term$ defines specific states, identifying particular states associated with a termination state and this set can be empty, in which case the transition system does not terminate.

event

A transformation is caused by an event that updates a temperature from a sensor, or a computer updating a computer variable, or an actuator sending a signal to a controlled entity.

An observation of a system S is based on the following points :

▶ a state $s \in \Sigma$ allows you to observe elements and reports on these elements, such as the number of people in the meeting room or the capacity of the room : $s(np)$ and $s(cap)$ are two positive integers.

▶ a relationship between two states s and s' observes a transformation of the state s into a state s' and we will note $s \xrightarrow{R} s'$ which expresses the observation of a relationship R :
$R = s(np) \in 0..s(cap)-1 \land s'(np) = s(np)+1 \land s'(cap) = s(cap)$ is an expression of R observing that one more person has entered the room.

▶ a trace $s_0 \xrightarrow{R_0} s_1 \xrightarrow{R_1} s_2 \xrightarrow{R_2} s_3 \xrightarrow{R} \ldots \xrightarrow{R_{i-1}} s_i \xrightarrow{R_i} \ldots$ is a trace generated by the different observations $R_0, \ldots R_p, \ldots$

Observation of a system

▶ observing changes of state that correspond either to physical or biological phenomena or to artefactual structures such as a program, a service or a platform.

▶ An observation generally leads to the identification of a few possible transformations of the observed state, and the closed-model hypothesis follows naturally.

▶ One consequence is that there are visible transformations and invisible transformations.

▶ These invisible transformations of the state are expressed by an identity relation called event skip (or stuttering [**?**].

▶ A modelling produces a closed model with a skip event modelling what is not visible in the observed state.

assertions

▶ a language of assertions $\mathcal{L}$ (or a language of formulae) is supposed to be given : $\mathcal{P}(\Sigma)$ (the set of parts of Σ)

▶ $\varphi(s)$ (or $s \in \hat{\varphi}$) means that φ is true in s.

▶ Properties of a system S which interest us are the state properties expressing that *nothing bad can happen*.

▶ Examples : *the number of people in the meeting room is always smaller than the maximum allowed by law* or *the computer variable storing the number of wheel revolutions is sufficient and no overflow will happen*.

▶ Safety properties : the partial correctness (PC) of an algorithm A with respect to its pre/post specifications (PC), the absence of errors at runtime (RTE) ...

▶ Properties are expressed in the language $\mathcal{L}$ whose elements are combined by logical connectors or by instantiations of variable values in the computer sense called flexible.

- hypothesis : a system S is modelled by a set of states Σ, and $\Sigma \stackrel{def}{=} \mathtt{Var} \longrightarrow D$ where $\mathtt{Var}$ is the variable (or list of variables) of the system S and D is the domain of possible values of variables.

- The interpretation of a formula P in a state $s \in \Sigma$ is denoted $[\![P]\!](s)$ or sometimes $s \in \hat{P}$.

- A distinction is made between flexible variable symbols $\mathtt{x}$ and logical variable symbols v, and constant symbols c are used.

Interpretation of assertions

1. $[\![x]\!](s) = s(x) = x$: x is the value of the variable x in s.

2. $[\![x]\!](s') = s'(x) = x'$: x' is the value of the variable x in s'.

3. $[\![c]\!](s)$ is the value of c in s, in other words the value of the constant c in s.

4. $[\![\varphi(x) \wedge \psi(x)]\!](s) = [\![\varphi(x)]\!](s)$ et $[\![\psi(x)]\!](s)$ where *and* is the classical interpretation of symbol $\wedge$ according to the truth table.

5. $[\![x = 6 \wedge y = x+8]\!](s) \overset{def}{=} [\![x]\!](s) = [\![6]\!](s)$ **and** $[\![y]\!](s) = [\![x]\!](s)+[\![8]\!](s) = (x = 6$ **and** $y = x+8$ where y is a logical variable distinct of x and where $[\![x]\!](s) = s(x) = x$.

- $[\![\mathrm{x}]\!](s)$ is the value of x in s and its value will be distinguished by the font used : x is the tt font of LaTeX and x is the math font of LaTeX.

- Using the name of the variable x as its current value, i.e. x and $[\![\mathrm{x}]\!](s')$ is the value of x in s' and will be noted x'.

- The transition relation as a relation linking the state of the variables in s and the state of the variables in s' using the prime notation as defined by L. Lamport for TLA.

- Types of variable depending on whether we are talking about the computer variable, its value or whether we are defining constants such as np, the number of processes, or π, which designates the constant π.

- a current observation refers to a current state for both endurant and perdurant information data in the sense of the Dines Bjørner.

flexible variable

A flexible variable x is a name related to a perdurant information according to a state of the (current observed) system :

- ▶ x is the current value of x in other words the value at the observation time of x.

- ▶ x' is the next value of x in other words the value at the next observation time of x.

- ▶ x_0 is the initial value of x in other words the value at the initial observation time of x.

A logical variable x is a name related to an endurant entity designated by this name.

state property of a system

Let be a system S whose flexible variables x are the elements of $\mathcal{V}ar(S)$. A property $P(x)$ of S is a logical expression involving ,freely the flexible variables x and whose interpretation is the set of values of the domain of $x : P(x)$ is true in x, if the value x satisfies $P(x)$.

For each property $P(x)$, we can associate a subset of D denoted $\hat{P}$ and, in this case, $P(x)$ is true in x. is equivalent to $x \in \hat{P}$.

Examples of property

▶ $P_1(x) \overset{def}{=} x \in 18..22 : x$ is a value between 18 and 22 and $\hat{P_1} = \{18, 19, 20, 21, 22\}$.

▶ $P_2(p) \overset{def}{=} p \subset PEOPLE \land card(p) \leq n : p$ is a set of persons and that set has at most n elements and $\hat{P_2} = \{p_1 \ldots p_n\}$. In this example, we use a logical variable n and a name for a constant $PEOPLE$.

Endurant information data

basic set of a system S

The list of symbols $s_1, s_2, \ldots, s_p$ corresponds to the list of basic set symbols in the D domain of S and $s_1 \cup \ldots \cup s_p \subseteq$ D.

constants of system S

The list of symbols $c_1, c_2, \ldots, c_q$ corresponds to the list of symbols for the constants of S.

Examples of constant and set

▶ $fred$ is a constant and is linked to the set $PEOPLE$ using the expression $fred \in PEOPLE$ which means that $fred$ is a person from $PEOPLE$.

▶ aut is a constant which is used to express the table of authorisations associated with the use of vehicles. the expression $aut \subseteq PEOPLE \times CARS$ where $CARS$ denotes a set of cars.

Axiom of system S

axiom of system S

An axiom ax(s,c) of S is a logical expression describing a constant or constants of S and can be defined as an expression depending on symbols of constants expressing a set-theoretical expression using symbols of sets and symbols of constants already defined.

Examples of axiom

► $ax1(fred \in PEOPLE)$: *fred is a person from the set PEOPLE*

► $ax2(suc \in \mathbb{N} \to \mathbb{N} \wedge (!i.i \in \mathbb{N} \Rightarrow suc(i) = i+1))$: *The function suc is the total function which associates any natural i with its successor. successor*

► $ax3(\forall A.A \subseteq \mathbb{N} \wedge 0 \in \mathbb{N} \wedge suc[A] \subseteq A \Rightarrow \mathbb{N}\to \subseteq A)$: This axiom states the induction property for natural numbers. It is an instantiation of the fixed-point theorem.

► $ax4(\forall x.x = 2 \Rightarrow x+2 = 1)$: This axiom poses an obvious problem of consistency and care should be taken not to use this kind of statement as axiom.

Axiomatics for S

. . . .

⊠ Definition(axiomatics for S)

The list of axioms of S is called the axiomatics of S and is denoted
$AX(S, s, c)$ where s denotes the basic sets and c denotes the constants
of S.

. . . .

. . . .

⊠ Definition(theorem for S)

A property $P(s, c)$ is a theorem for S, if $AX(S, s, c) \vdash P(s, c)$ is a valid
sequent.

Theorems for S are denoted by $TH(S, s, c)$.

. . . .

Let s, s' be two states of S ($s, s' \in \mathcal{V}ar(S) \longrightarrow \text{VALS}$). $s \xrightarrow[R]{} s'$ will be
written as a relation $\mathrm{R}(x, x')$ where x and x' designate values of x before
and after the observation of R.

. . . .

⊠ Definition(event)

Let $\mathcal{V}ar(S)$ be the set of flexible variables of S. Let s be the basis sets
and c the constants of S. An event e for S is a relational expression of the
form $R(s, c, x, x')$ denoted $BA(e)(s, c, x, x')$

. . . .

⊠ Definition(event-based model of a system)

Let $\mathcal{V}ar(S)$ be the set of flexible variables of S denoted x. Let s be the list of basis sets of the system S. Let c be the list of constants of the system S. Let D be a domain containing sets s. An event-based model for a system S is defined by

$$(AX(s,c), x, \text{VALS}, Init(x), \{e_0, \ldots, e_n\})$$

where

- ▶ $AX(s,c)$ is an axiomatic theory defining the sets, constants and static properties of these elements.

- ▶ $Init(x)$ defines the possible initial values of x.

- ▶ $\{e_0, \ldots, e_n\}$ is a finite set of events of S and e_0 is a particular event present in each event-based model defined by $BA(e_0)(x,x') = (x' = x)$.

The event-based model is denoted
$EM(s, c, x, \text{VALS}, Init(x)\{e_0, \ldots, e_n\}) = (AX(s,c), x, \text{VALS}, Init(x), \{e_0, \ldots, e_n\})$.
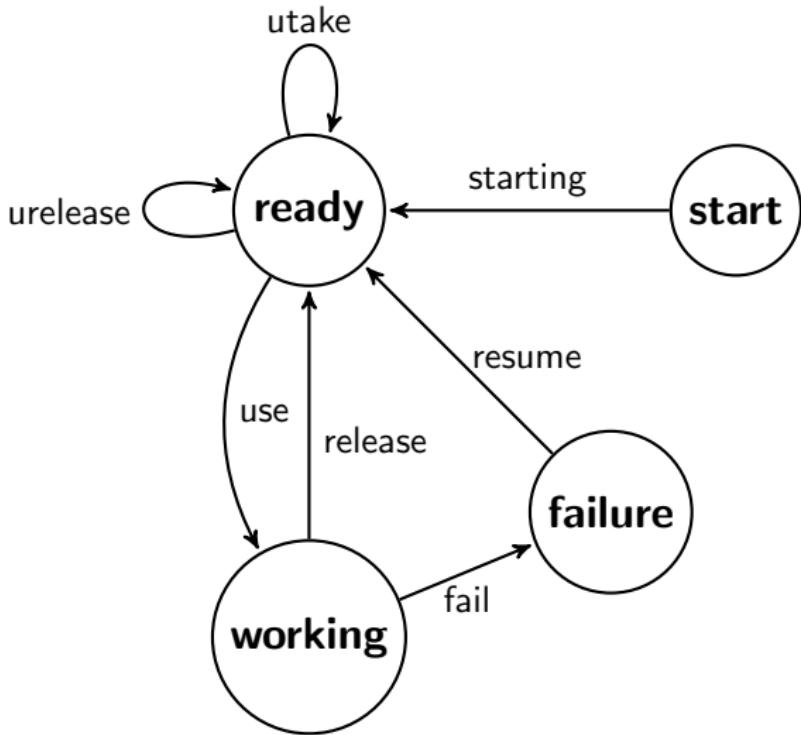
- $Next(x, x')$ or
  $Next(s, c, x, x') \stackrel{def}{=} BA(e_0)(s, c, x, x') \vee \ldots \vee BA(e_n)(s, c, x, x').$

- the transitive reflexive closure of the relation $Next^\star(s, c, x_0, x) \stackrel{def}{=}$
  $$\left\{ \begin{array}{l} \vee \ x = x_0 \\ \vee \ Next(s, c, x_0, x) \\ \vee \ \exists \ xi \in \text{VALS}.Next^\star(s, c, x_0, xi) \wedge Next(s, c, xi, x) \end{array} \right.$$

. . . .

⊠ Definition(safety property)

A property $P(x)$ is a safety property for the system S, if

$$\forall x_0, x \in \text{VALS}.Init(x_0) \wedge Next^\star(s, c, x_0, x) \Rightarrow P(x).$$
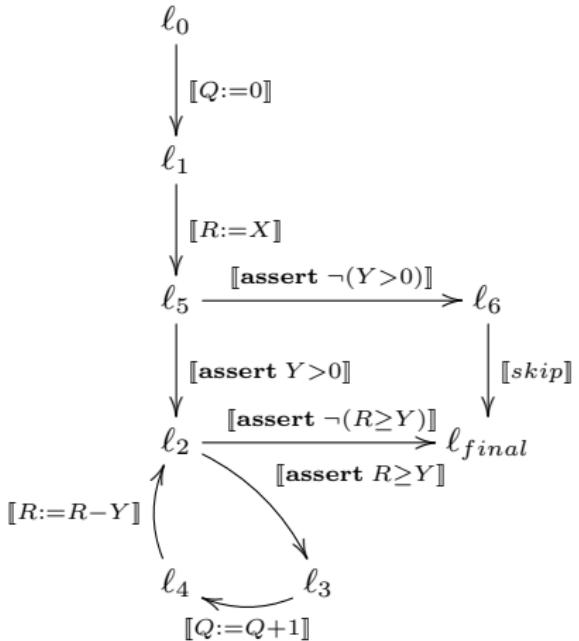
. . . .

Current Subsection Summary

Automata

▶ there is an implicit control variable
$pc \in \{\textbf{start}, \textbf{ready}, \textbf{working}, \textbf{failure}\}$ expressing the current visited state.
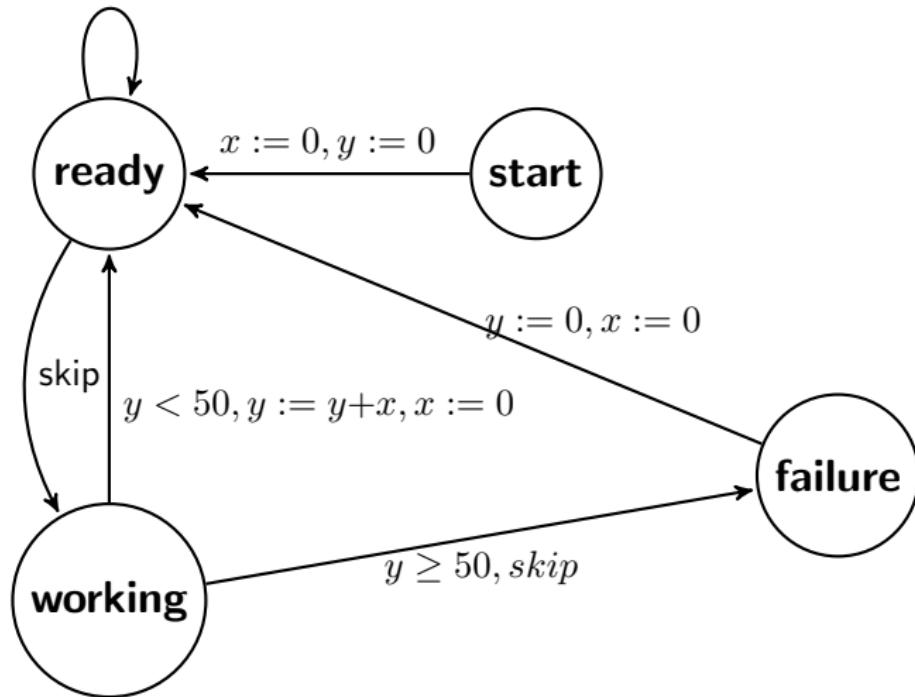
$\ell_0[Q := 0];$
$\ell_1[R := X];$
IF $\ell_5[Y > 0]$
    WHILE $\ell_2[R \geq Y]$
      $\ell_3[Q := Q+1];$
      $\ell_4[R := R-Y]$
    ENDWHILE
ELSE
  $\ell_6[skip]$
ENDIF
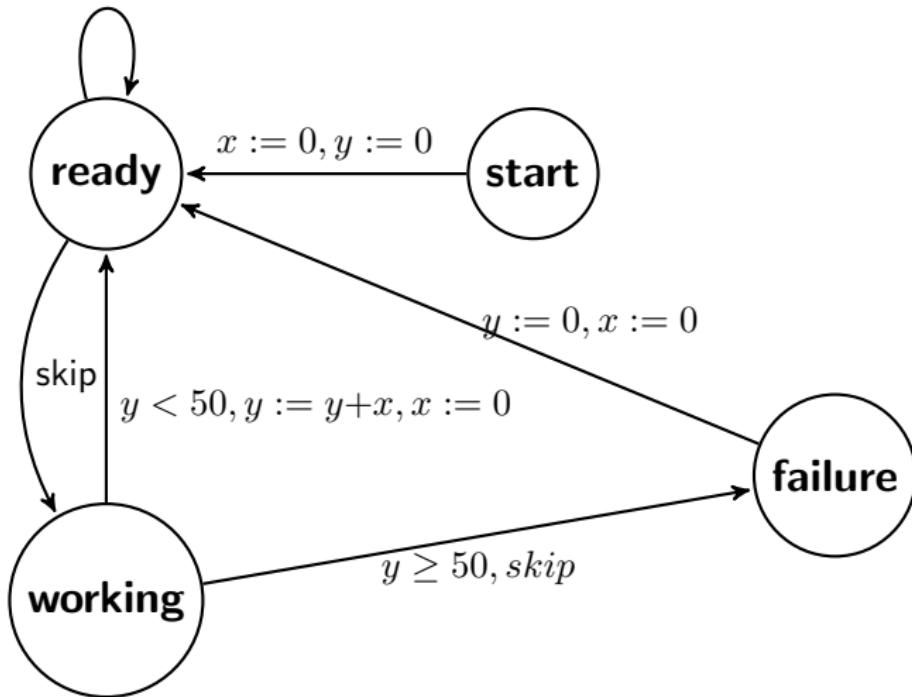
$x \leq 5, x := x+1$

ready

$x := 0, y := 0$

start

skip

$y < 50, y := y+x, x := 0$

$y := 0, x := 0$

working

failure

$y \geq 50, skip$

▶ safety1 : $0 \leq x \leq 5$

$x \leq 5, x := x+1$

ready

$x := 0, y := 0$

start

skip

$y := 0, x := 0$

$y < 50, y := y+x, x := 0$

failure

working

$y \geq 50, skip$
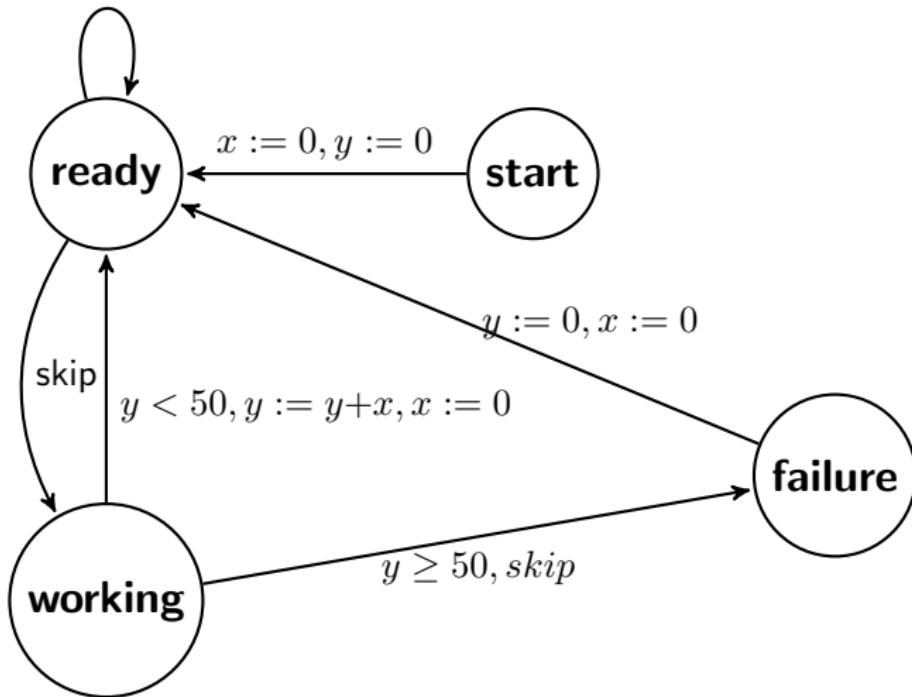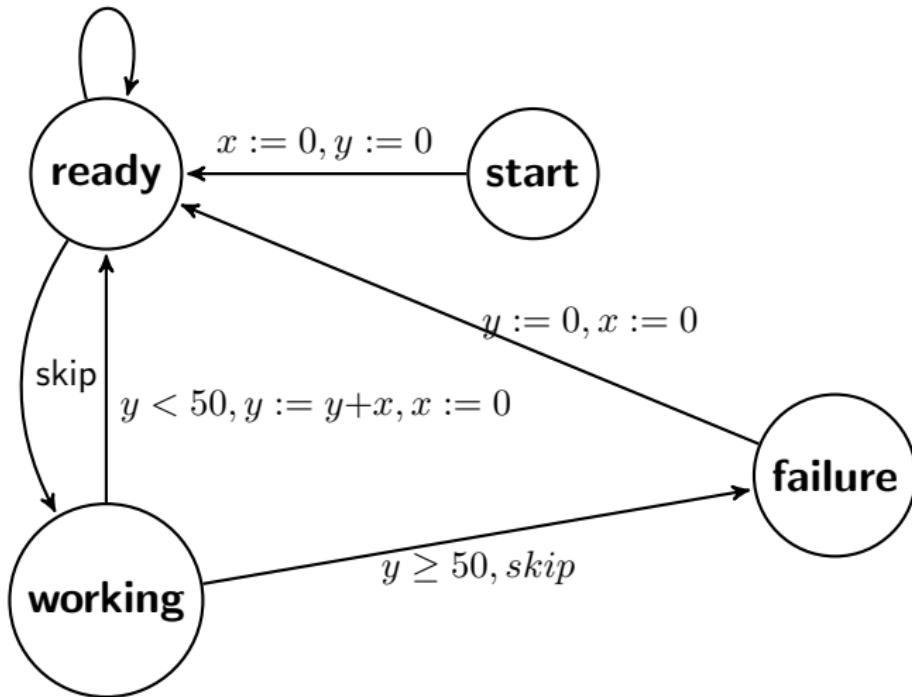
▶ safety1 : $0 \leq x \leq 5$ et ...

A small system as an automaton

$x \leq 5, x := x+1$

$x := 0, y := 0$

ready   **start**

$y := 0, x := 0$

skip

$y < 50, y := y+x, x := 0$

failure

working

$y \geq 50, skip$

▶ safety1 : $0 \leq x \leq 5$ et ... safety2 : $0 \leq y \leq 56$

A small system as an automaton
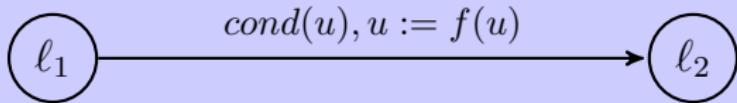
A small system as an automaton
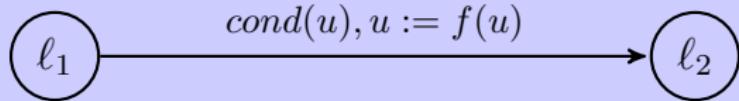
▶ safety1 : $0 \leq x \leq 5$ et safety2 : $0 \leq y \leq 56$

▶ $skip = x := x, y := y$

▶ skip $= TRUE, x := x, y := y = TRUE, skip$

Transition between two control states

$$\ell_1 \xrightarrow{\quad cond(u), u := f(u) \quad} \ell_2$$

Transition between two control states

$$cond(u), u := f(u)$$

$\ell_1 \longrightarrow \ell_2$
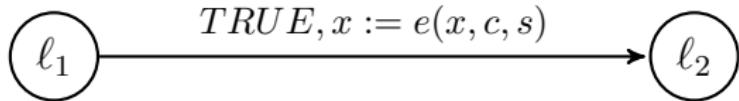
► assignment x :=e(x,c,s)

Transition between two control states

$$cond(u), u := f(u)$$
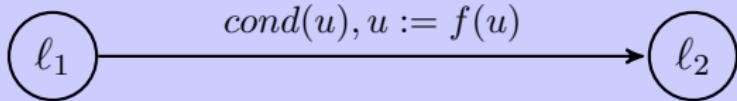
with ℓ_1 and ℓ_2 as control states.

▶ assignment x :=e(x,c,s)
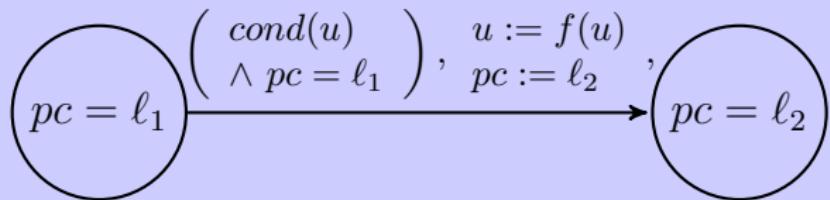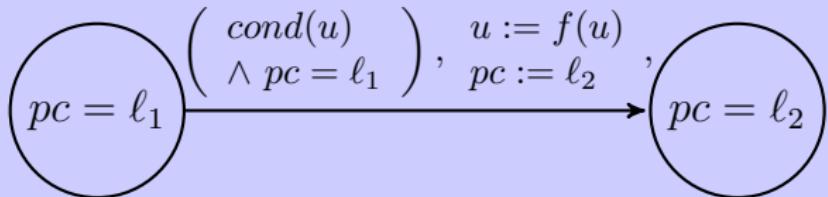
$$TRUE, x := e(x, c, s)$$

with ℓ_1 and ℓ_2 as control states.

Transition between two control states

$$\ell_1 \xrightarrow{\quad cond(u), u := f(u) \quad} \ell_2$$

▶ assignment x :=e(x,c,s)

$$\ell_1 \xrightarrow{\quad TRUE, x := e(x, c, s) \quad} \ell_2$$

ℓ_1 :
x :=e(x,c,s);
ℓ_2 :

Transition between two control states

$$cond(u), u := f(u)$$

with $\ell_1 \longrightarrow \ell_2$

▶ assignment x :=e(x,c,s)

$$TRUE, x := e(x, c, s)$$

$\ell_1 \longrightarrow \ell_2$

$\ell_1 :$
x :=e(x,c,s);
$\ell_2 :$

▶ if statement if(B(x),S1,S2)

Transition between two control states

$$\ell_1 \xrightarrow{\quad cond(u), u := f(u) \quad} \ell_2$$

▶ assignment x :=e(x,c,s)

$$\ell_1 \xrightarrow{\quad TRUE, x := e(x, c, s) \quad} \ell_2$$

ℓ_1 :
x :=e(x,c,s);
ℓ_2 :

▶ if statement if(B(x),S1,S2)

$$\ell_0 \xrightarrow{\quad B(x), skip \quad} \ell_1$$

Transition between two control states

$$\ell_1 \xrightarrow{\quad cond(u), u := f(u) \quad} \ell_2$$

▶ assignment x :=e(x,c,s)

$$\ell_1 \xrightarrow{\quad TRUE, x := e(x, c, s) \quad} \ell_2$$

$\ell_1 :$
x :=e(x,c,s);
$\ell_2 :$

▶ if statement if(B(x),S1,S2)

$$\ell_0 \xrightarrow{\quad B(x), skip \quad} \ell_1$$

$$\ell_0 \xrightarrow{\quad not\ B(x), skip \quad} \ell_2$$

Transition between two control states

$$\ell_1 \xrightarrow{cond(u), u := f(u)} \ell_2$$

▶ assignment x :=e(x,c,s)

$$\ell_1 \xrightarrow{TRUE, x := e(x, c, s)} \ell_2$$

ℓ_1 :
x :=e(x,c,s);
ℓ_2 :

▶ if statement if(B(x),S1,S2)

$$\ell_0 \xrightarrow{B(x), skip} \ell_1$$

$$\ell_0 \xrightarrow{not\ B(x), skip} \ell_2$$

ℓ_0 :
if $B(x)$ then
    ℓ_1 :
    S1
else
    ℓ_2 :
    S2
fi

Others forms of transitions

Transition between two control states

$$pc = \ell_1 \quad \xrightarrow{\left(\begin{array}{c} cond(u) \\ \wedge \; pc = \ell_1 \end{array} \right), \quad \begin{array}{c} u := f(u) \\ pc := \ell_2 \end{array},} \quad pc = \ell_2$$

Others forms of transitions

Transition between two control states

$$pc = \ell_1 \quad \xrightarrow{\left(\begin{array}{c} cond(u) \\ \wedge\ pc = \ell_1 \end{array}\right),\ \begin{array}{c} u := f(u) \\ pc := \ell_2 \end{array},} \quad pc = \ell_2$$

Transition between two predicates

$$P_1(u) \quad \xrightarrow{\left(\begin{array}{c} cond(u) \\ \wedge\ pc = \ell_1 \end{array}\right),\ \begin{array}{c} u := f(u) \\ pc := \ell_2 \end{array},} \quad P_2(u)$$

Current Subsection Summary

Un modèle relationnel $\mathcal{MS}$ pour un système $\mathcal{S}$ est une structure

$$(Th(s,c), \mathrm{X}, \mathrm{VALS}, \mathrm{INIT}(x), \{r_0, \ldots, r_n\})$$

où

▶ $Th(s,c)$ est une théorie définissant les ensembles, les constantes et les propriétés statiques de ces éléments.

▶ X est une liste de variables flexibles.

▶ VALS est un ensemble de valeurs possibles pour X.

▶ $\{r_0, \ldots, r_n\}$ est un ensemble fini de relations reliant les valeurs avant x et les valeurs après x'.

▶ $\mathrm{INIT}(x)$ définit l'ensemble des valeurs initiales de X.

▶ la relation r_0 est la relation $Id[\mathrm{VALS}]$, identité sur VALS.

....

⊠ Definition

Soit $(Th(s,c), X, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel d'un système $\mathcal{S}$. La relation $\textsc{Next}$ associée à ce modèle est définie par la disjonction des relations r_i :

$\textsc{Next} \stackrel{def}{=} r_0 \vee \ldots \vee r_n$

....

pour une variable x, nous définissons les valeurs suivantes :

- ▶ x est la valeur courante de la variable X.
- ▶ x' est la valeur suivante de la variable X.
- ▶ x_0 ou $\underline{x}$ sont la valeur initiale de la variable X.
- ▶ $\overline{x}$ ou x_f est la valeur finale de la variable X, quand cette notion a du sens.

Propriétés de sûreté et d'invariance dans un modèle relationnel

. . . .

⊠ Definition(assertion)

Soit $(Th(s,c), \mathrm{X}, \mathrm{VALS}, \mathrm{INIT}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système $\mathcal{S}$. Une propriété A est une propriété assertionnelle de sûreté pour le système $\mathcal{S}$, si

$$\forall x_0, x \in \mathrm{VALS}.Init(x_0) \wedge \mathrm{NEXT}^\star(x_0, x) \Rightarrow A(x).$$

. . . .
. . . .

⊠ Definition(relation)

Soit $(Th(s,c), \mathrm{X}, \mathrm{VALS}, \mathrm{INIT}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système $\mathcal{S}$. Une propriété R est une propriété relationnelle de sûreté pour le système $\mathcal{S}$, si

$$\forall x_0, x \in \mathrm{VALS}.Init(x_0) \wedge \mathrm{NEXT}^\star(x_0, x) \Rightarrow R(x_0, x).$$

. . . .

▶ P. et R. Cousot développent une étude complète des propriétés d'invariance et de sûreté en mettant en évidence correspondances entre les différentes méthodes ou systèmes proposées par Turing, Floyd, Hoare, Wegbreit, Manna . . . et reformulent les principes d'induction utilisés pour définir ces méthodes de preuve (voir les deux cubes des 16 principes).

Current Summary

Current Subsection Summary

Listing 1 – cours0.tla

```
———————————— MODULE cours0 ————————————
EXTENDS Integers , Naturals , TLC
CONSTANTS x0 , y0
VARIABLES x , y
————————————————————————————————
ASSUME x0 \in Nat /\ y0 \in Nat
Init == x=x0 /\ y=y0
————————————————————————————————
(* actions *)
finger ==
    /\ y # 0
    /\ x '=x+1
----/\ -y '=y—1
over == y=0 /\ x '=x -/\ -y '=y
————————————————————————————————
Next == finger \/ over
————————————————————————————————
(* Safety properties to be verified *)
TypeOK ==
        /\ x \in Int
        /\ y \in Int
Inv ==
        /\ x \geq x0 /\ x <= x0+y0
        /\ 0 <= y /\ y <=y0
        /\ x+y = x0+y0
P1 == [] Inv
P2 == [] (0 <= y /\ y <= y0)
Q1 == y # 0
========================================
```

Listing 2 – cours0.tla

```
———————————— MODULE cours0 ————————————
EXTENDS Integers , Naturals , TLC
CONSTANTS x0 , y0
VARIABLES  x , y
————————————————————————————————
ASSUME x0 \in Nat /\ y0 \in Nat
Init == x=x0 /\ y=y0
————————————————————————————————
(* actions *)
finger ==
    /\ y # 0
    /\ x '=x+1
----/\ - y '=y-1
over == y=0 /\ x '=x-/\-y '=y
————————————————————————————————
Next == finger \/ over
————————————————————————————————
(* Safety properties to be verified *)
TypeOK ==
     /\ x \in Int
     /\ y \in Int
Inv   ==
     /\ x \geq x0 /\ x <= x0+y0
     /\ 0 <= y /\ y  <=y0
     /\ x+y = x0+y0
P1 == [] Inv
P2 == [] (0 <= y /\ y <= y0)
Q1 == y # 0
================================================
```

The balls y are moved towards the balls x, as long as there are any left.

Current Subsection Summary

Module for a simpke access control

MODULE $ex1$

modules de base importables

EXTENDS $Naturals$, TLC

un système contrôle l'accès à une salle dont la capacité est de 19 personnes ; écrire un modèle de ce système en vérifiant la propriété de sûreté

VARIABLES np

Module for a simple access control

Première tentative

$entrer \triangleq np' = np + 1$
$sortir \triangleq np' = np - 1$
$next \triangleq entrer \lor sortir$
$init \triangleq np = 0$

Seconde tentative

$entrer_2 \triangleq np < 19 \wedge np' = np+1$

$next_2 \triangleq entrer_2 \vee sortir$

Module for a simple access control

Troisième tentative

$sortir_2 \triangleq np > 0 \ \wedge \ np' = np - 1$

$next_3 \triangleq entrer_2 \ \vee \ sortir2$

$safety_1 \triangleq np \leq 19$

$question_1 \triangleq np \neq 6$

Module for a simpke access control

```
----------------------------- MODULE ex1-----------------------------
(* modules de base importables *)
EXTENDS Naturals,TLC
----------------------------------------------------------------------
(* un syst\'eme contr\^ole l'acc\'es \'a une salle  dont la capacit\'e  est de 19 personnes
VARIABLES  np
----------------------------------------------------------------------
(* Premi\'ere tentative *)
entrer == np '=np +1
sortir == np'=np-1
next == entrer \/ sortir
init == np=0\fora
----------------------------------------------------------------------
(* Seconde tentative *)
entrer2 == np<19 /\ np'=np+1
next2 ==  entrer2  \/  sortir
----------------------------------------------------------------------
(* Troisi\'eme tentative *)
sortir2 == np>0 /\ np'=np-1
next3 ==  entrer2  \/  sortir2
------------------
safety1 == np \leq 19
question1 == np # 6
======================================================================
```

Current Subsection Summary

Let $(Th(s,c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ be a relational model M of a system $\mathcal{S}$. A property A is a safety property for the system $\mathcal{S}$ if $\forall x_0, x \in \text{VALS}.Init(x_0) \wedge \text{NEXT}^\star(x_0, x) \Rightarrow A(x)$.

Let $(Th(s,c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ be a relational model M of a system $\mathcal{S}$. A property A is a safety property for the system $\mathcal{S}$ if $\forall x_0, x \in \text{VALS}.Init(x_0) \wedge \text{NEXT}^\star(x_0, x) \Rightarrow A(x)$.

▶ x is a variable or a list of variables : `VARIABLES x`

Let $(Th(s,c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ be a relational model M of a system $\mathcal{S}$. A property A is a safety property for the system $\mathcal{S}$ if
$\forall x_0, x \in \text{VALS}.Init(x_0) \land \text{NEXT}^{\star}(x_0, x) \Rightarrow A(x)$.

- ▶ x is a variable or a list of variables : `VARIABLES x`
- ▶ $Init(x)$ is a variable or a list of variables : `init == Init(x)`

Let $(Th(s,c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ be a relational model M of a system $\mathcal{S}$. A property A is a safety property for the system $\mathcal{S}$ if $\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow A(x)$.

- x is a variable or a list of variables : `VARIABLES x`

- $Init(x)$ is a variable or a list of variables : `init == Init(x)`

- $\textsc{Next}^\star(x_0, x)$ is the definition of the relation defining what the system does : `Next == a1 \/ a2 \/ .... \/ an`

Let $(Th(s,c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ be a relational model M of a system $\mathcal{S}$. A property A is a safety property for the system $\mathcal{S}$ if $\forall x_0, x \in \text{VALS}.Init(x_0) \wedge \text{NEXT}^{\star}(x_0, x) \Rightarrow A(x)$.

- ▶ x is a variable or a list of variables : `VARIABLES x`
- ▶ $Init(x)$ is a variable or a list of variables : `init == Init(x)`
- ▶ $\text{NEXT}^{\star}(x_0, x)$ is the definition of the relation defining what the system does : `Next == a1 \/ a2 \/ .... \/ an`
- ▶ $A(x)$ is a logical expression defining a safety property to be verified on all configurations of the model : `Safety == A(x)`

▶ TLA (Temporal Logic of Actions) is used to express formulas in temporal logic : $\Box P$ or *always P*

▶ TLA$^+$ is a language that allows the declaration of constants, variables and definitions :

- <def> == <expression> : a definition <def> is the specification of an expression <expression> that uses elements defined before or in modules that *éxtend* this module.
- A variable x is either in the form x or in the form x' : x' is the value of x after.
- A module has a name and collects definitions, and it can be an extension of other modules.
- [f EXCEPT![i]=e] is the function f where only the value of i has changed and is equal to e.

▶ A configuration must be defined in order to evaluate a specification.

Limitation of actions form

condition,action

An action is a relationship between the variables before and the variables after, but TLC requires actions that can be calculated or evaluated according to Java.

The variables are denoted by x and an action is of the following form :

$$
\begin{aligned}
\texttt{nom} \triangleq& \\
& \wedge\ cond(x) \\
& \wedge\ x' = e(x, s, c)
\end{aligned}
$$

▶ $e(x, s, c)$ must be codable in Java and denotes an expression over sets s and constnat c.

▶ Standard modules : Naturals, Integers, TLC, etc.

Comments

▶ Download the application from the Microsoft website for your computer.

▶ Write models and test them !

▶ Limitations by variable domains.

▶ Each change or not chnage of a variable should be expressed explicitly :

- $x' = x$
- unchanged $<< vars >>$ is defining the expression $vars' = vars$

Enable symbolic reasoning regardless of the set of states

Current Summary

Current Summary

1. Programming by Contract

2. Summary of the Tryptich

3. Transition Systems
   - Overview of Transition Systems as Modelling Tool
   - Expression of transition systems
   - Main concepts of discrete transition system

4. Transition system in action with TLA/TLA$^+$
   - Example 1 ADD (cours0.tla)
   - Example 2 Simple Access Control (cours2.tla)
   - Transition system using TLA/TLA$^+$

5. stop movex 2

6. The TLA$^+$ Toolbox

7. PlusCal an algorithmic notation inside TLA$^+$

8. Conclusion

The TLA$^+$ Toolbox

https://github.com/tlaplus/

The TLA$^+$ tools require Java 11+ to run and the tla2tools.jar file contains multiple TLA$^+$ tools.
You add tla2tools.jar to your CLASSPATH environment variable.
Running java -jar tla2tools.jar is aliased to java -cp tla2tools.jar tlc2.TLC.

- ▶ `java -cp tla2tools.jar tla2sany.SANY -help` : The TLA$^+$ parser.
- ▶ `java -cp tla2tools.jar tlc2.TLC -help` : The TLA$^+$ model checker
- ▶ `java -cp tla2tools.jar pcal.trans -help` : The PlusCal-to-TLA$^+$ translator
- ▶ `java -cp tla2tools.jar tla2tex.TLA -help` : The TLA$^+$-to-LaTeX translator

Defining constants and properties ton check

Listing 3 – cours0.tla

```
----------------------- MODULE cours0 -----------------------
EXTENDS Integers , Naturals , TLC
CONSTANTS x0 , y0
VARIABLES  x , y
---------------------------------------------------------
ASSUME x0 \in Nat /\ y0 \in Nat
Init == x=x0 /\ y=y0
---------------------------------------------------------
(* actions *)
finger ==
      /\ y # 0
      /\ x '=x+1
----/\ -y '=y—1
over == y=0 /\ x '=x -/\ -y '=y
---------------------------------------------------------
Next == finger \/ over
---------------------------------------------------------
(* Safety properties to be verified *)
TypeOK ==
        /\ x \in Int
        /\ y \in Int
Inv   ==
        /\ x \geq x0 /\ x <= x0+y0
        /\  0 <= y /\ y  <=y0
        /\ x+y = x0+y0
P1 == [] Inv
P2 == [] (0 <= y /\ y <= y0 )
Q1 == y # 0

========================================================
```

Defining constants and properties ton check

Listing 4 – cours0.tla

```
———————————— MODULE cours0 ————————————
EXTENDS Integers , Naturals , TLC
CONSTANTS x0 , y0
VARIABLES  x , y
————————————————————————————————————————
ASSUME x0 \in Nat /\ y0 \in Nat
Init == x=x0 /\ y=y0
————————————————————————————————————————
(* actions *)
finger ==
    /\ y # 0
    /\ x '=x+1
- - - -/\ - y '=y—1
over == y=0 /\ x '=x - /\ - y '=y
————————————————————————————————————————
Next == finger \/ over
————————————————————————————————————————
(* Safety properties to be verified *)
TypeOK ==
      /\ x \in Int
      /\ y \in Int
Inv  ==
      /\ x \geq x0 /\ x <= x0+y0
      /\ 0 <= y /\ y <=y0
      /\ x+y = x0+y0
P1 == [] Inv
P2 == [] (0 <= y /\ y <= y0)
Q1 == y # 0
======================================================
```

The balls y are moved towards the balls x, as long as there are any left.

Defining models for TLC model checking

Listing 5 – cours0.cfg

```
\* SPECIFICATION
\* Uncomment the previous line and provide the specification name if it's declared
\* in the specification file. Comment INIT / NEXT parameters if you use SPECIFICATION.

CONSTANTS
    greeting = "Hello"
    x0 = 5
    y0 = 9

INIT Init
NEXT Next

PROPERTY
\* Uncomment the previous line and add property names
P1
P2

INVARIANT
\* Uncomment the previous line and add invariant names
TypeOK
Inv
```

Defining models for TLC model checking

Listing 6 – cours0.cfg

```
\* SPECIFICATION
\* Uncomment the previous line and provide the specification name if it's declared
\*-in-the-specification-file.-Comment-INIT-/-NEXT-parameters-if-you-use-SPECIFICATION.

CONSTANTS
----greeting-=-"Hello"
----x0-=-5
----y0-=-9

INIT-Init
NEXT-Next

PROPERTY
\*-Uncomment-the-previous-line-and-add-property-names
P1
P2

INVARIANT
\*-Uncomment-the-previous-line-and-add-invariant-names
TypeOK
Inv
```

- ▶ Setting constants
- ▶ Adding temporal properties as $[]P$ or $\Box P$.
- ▶ Adding invariant properties as Inv.

Current Summary

- Definition of a simple algorithmic language.
- Specific comment in between (* and *)
  `--algorithm name { definitions }`
- Generation of a TLA$^+$ specification with introduction of a new variable pc modelling control.
- The ToolBox tool has a translation feature.

Listing 7 – courspluscal0.tla

```
————————————— MODULE courspluscal0 —————————————
EXTENDS Naturals , Integers , TLC
CONSTANTS x0 , y0 , z0
—————————————————————————————
(* precondition *)
ASSUME x0 = y0 + 3*z0
—————————————————————————————
(*
——algorithm ex {
  variables x=x0 ,
           y = y0 ,
           z=z0 ;

{
l0 : assert x = y + 3*z /\ y=y0 /\ z=z0 ;
   x := y+3*z;
l1 : assert x = y0+3*z0 /\ y=y0 /\ z=z0 ;
}
}
*)
\* BEGIN TRANSLATION (chksum(pcal) = "cd0610de" /\ chksum(tla) = "2c474478")
VARIABLES x , y , z , pc

vars == << x , y , z , pc >>

Init == (* Global variables *)
       /\ x = x0
       /\ y = y0
       /\ z = z0
       /\ pc = "l0"

l0 == /\ pc = "l0"
     /\ Assert(x = y + 3*z /\ y=y0 /\ z=z0 ,
               "Failure - of - assertion - at - line - 15 , - column - 5.")
     /\ x ' -= - y+3*z
------/\ -pc ' = "l1"
```

Exemple

Listing 8 – courspluscal0.tla

```
——————————— MODULE courspluscal0 ———————————
EXTENDS Naturals, Integers, TLC
CONSTANTS x0, y0, z0
————————————————————————————————————
(* precondition *)
ASSUME x0 = y0 + 3*z0
————————————————————————————————————
(*
——algorithm ex {
  variables x=x0,
            y = y0,
            z=z0;

{
l0: assert x = y + 3*z /\ y=y0 /\ z=z0 ;
    x := y+3*z;
l1: assert x = y0+3*z0 /\ y=y0 /\ z=z0 ;
}
}
*)
\* BEGIN TRANSLATION (chksum(pcal) = "cd0610de" /\ chksum(tla) = "2c474478")
VARIABLES x, y, z, pc

vars == << x, y, z, pc >>

Init == (* Global variables *)
        /\ x = x0
        /\ y = y0
        /\ z = z0
        /\ pc = "l0"

l0 == /\ pc = "l0"
      /\ Assert(x = y + 3*z /\ y=y0 /\ z=z0,
                "Failure-of-assertion-at-line-15,-column-5.")
      /\ x' = y+3*z
------/\ pc' = "l1"
```

```
—algorithm Exemple
variables x = 0;

begin
   x := x + 1;
end algorithm
```

Variables and assignments

- ▶ Declaration in the `variables` section
- ▶ Assignment with `:=`
- ▶ Implicit types (integers, sets, sequences)

Boucles et conditions

```
while (x < 10) do
  if (x % 2 = 0) then
    x := x + 1;
  else
    x := x + 2;
  end if;
end while;
```

Advantages of PlusCal

- ▶ Better readability than TLA^+ expressions.
- ▶ Automatic verification made possible by model checking
- ▶ Suitable for distributed systems and our lectures on distributed algorithms.
- ▶ Based on solid mathematical foundations

Limitations

- ▶ Not an execution language
- ▶ TLA$^+$ learning curve
- ▶ Specific tools required

Macros and procedures in PlusCal

▶ Macros and procedures allow you to make the code simpler

▶ Improving the readability and the modularity

▶ Points

- **Macro** : reusable code fragment in the algorithm
- **Procedure** : function that can be called with arguments

▶ Syntax similar to an imperative language

Macros et Procedures

```
macro Name(var1, ...)
begin
 \* something to write
end macro;
```

```
procedure Name(arg1, ...)
variables var1 = ... \* not
\in, only =
begin
  Label:
  \* something
  return;
end procedure;
```

Example 1 : simple macro

```
——algorithm ExMacro
variables x = 0;

macro incrX()
begin
    x := x + 1;
end macro;

begin
    l0 : assert (x = 0);
    l1 : incrX();
    l2 : incrX();
    l3 : assert (x = 2);

end algorithm;
```

- ▶ File pluscalcours2.tla
- ▶ The macro incrX increments the variable x
- ▶ Called twice, x is equal to 2 at the end

Example2 : Procedure with parameter

```
--algorithm ExProcedure
variables x  = 1;

define
  Op(p) == 2*p
end define;

procedure test(v)
variables w = 1
begin
  l8: v:=v+w;
  l9: return;
end procedure;

begin
  l0: assert (x = 1);
  call test(x);
  nl: call test(x);
  l3: assert (x = 1);
   x := Op(x);
  l4: assert (x = 2);
  end algorithm;
```

► File pluscalcours1.tla

Summation

```
                        MODULE pluscalcours3
EXTENDS Integers , TLC

CONSTANTS x0   (* x0 is the input *), intmin , intmax

(* notations *)
typeInt (u) == u \in Int  (* u is an integer *)
DD(X) == intmin \leq X /\ X \leq intmax
sum(u) == (u*(u+1)) \div 2
pre(X) == X \geq 0

(* Checking calling conditions *)
ASSUME pre(x0)


(*
----algorithm sum {
variables ps=0,k=0,r,x=x0;
{
inloop: while (k < x) {
     k := k + 1;
     ps := ps + k+1;
   };
 outloop: r := ps ;
}
```

Current Summary

Comments

- The TLA/TLA$^+$ toolbox provides a framework for analysing algorithms, programs and computational protocols.
- The method is restricted to finite models of transition systems.
- The general methodology requires to use an induction-based process.