UNIVERSITÉ DE LORRAINE

TELECOM nancy
Ingénieurs du numérique • Inspiring your digital future

MVSI Course
Modelling and Verification of Computer Systems

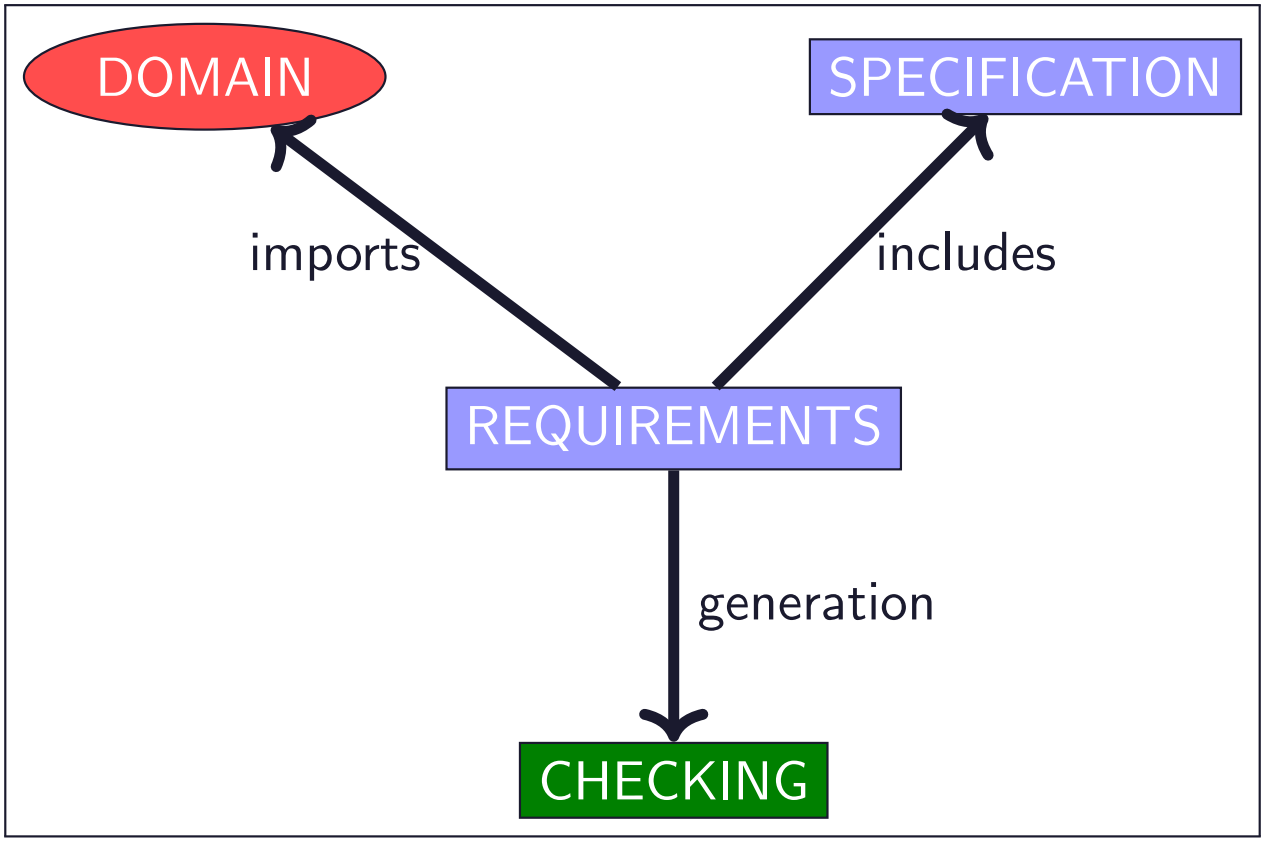
Lecture 1 – Modelling, Specification and Verification (I)

Dominique Méry
Telecom Nancy, Université de Lorraine
November 20, 2026

Plan

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
   - Expression of discrete transition systems
4. Transition system in action with TLA/TLA$^+$ (file **tla0.tla**)
   - Computing GCD
   - Simple Access Control
   - Module and Configuration in TLA$^+$

Tryptic Pattern



Part 1

Programming by Contract

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: $pre(x_0)$ and x_f satisfies post : $post(x_0, x_f)$
- $pre(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow post(x_0, x_f)$
- $\mathbb{D}$ is the RTE domain of X

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

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

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

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

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

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

$\longrightarrow$

Example of an annotation

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

$\longrightarrow \longrightarrow$

Example of an annotation

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 $x_f = 0$
$\ell_0 : \{ x = x_0 \wedge x_0 \in \mathbb{N}\}$
WHILE $0 < X$ **DO**
$\ell_1 : \{0 < x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
$X := X - 1;$
$\ell_2 : \{0 \leq x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
OD;
$\ell_3 : \{x = 0\}$

Part 2

Summary of the Tryptich

Tryptich $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

Tryptich $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

- $\mathcal{R}$: system requirements.

Tryptich $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

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

Tryptich $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

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

Tryptich $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

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

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

Tryptich $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

- $\mathcal{R}$: 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, ...

Part 3

Transition Systems

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.

Observation of a system

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:
  $R = s(np) \in 0..s(cap)-1 \wedge s'(np) = s(np)+1 \wedge s'(cap) = s(cap)$ 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 $\Sigma \stackrel{def}{=} \texttt{Var} \longrightarrow \mathrm{D}$ where `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 `x` and logical variable symbols v, and constant symbols c are used.

Interpretation of assertions

1. $[\![\mathtt{x}]\!](s) = s(\mathtt{x}) = x$: x is the value of the variable $\mathtt{x}$ in s.
2. $[\![\mathtt{x}]\!](s') = s'(\mathtt{x}) = x'$: x' is the value of the variable $\mathtt{x}$ in s'.
3. $[\![c]\!](s)$ is the value of c in s, in other words the value of the constant c in s.
4. $[\![\varphi(x) \wedge \psi(x)]\!](s) = [\![\varphi(x)]\!](s)$ and $[\![\psi(x)]\!](s)$ where *and* is the classical interpretation of symbol $\wedge$ according to the truth table.
5. $[\![\mathtt{x} = 6 \wedge y = \mathtt{x}{+}8]\!](s) \stackrel{def}{=} [\![\mathtt{x}]\!](s) = [\![6]\!](s)$ **and** $[\![y]\!](s) = [\![x]\!](s){+}[\![8]\!](s) = (x = 6$ **and** $y = x{+}8$ where y is a logical variable distinct of $\mathtt{x}$ and where $[\![\mathtt{x}]\!](s) = s(\mathtt{x}) = x$.

Notations

- $[\![\mathtt{x}]\!](s)$ is the value of $\mathtt{x}$ in s and its value will be distinguished by the font used: $\mathtt{x}$ is the `tt` font of LaTeX and x is the math font of LaTeX.
- Using the name of the variable $\mathtt{x}$ as its current value, i.e. x and $[\![\mathtt{x}]\!](s')$ is the value of $\mathtt{x}$ 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.

Notations

flexible variable

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

- x is the current value of $\mathtt{x}$ in other words the value at the observation time of $\mathtt{x}$.
- x' is the next value of $\mathtt{x}$ in other words the value at the next observation time of $\mathtt{x}$.
- x_0 is the initial value of $\mathtt{x}$ in other words the value at the initial observation time of $\mathtt{x}$.

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

State property of a system

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

- $P_1(x) \stackrel{def}{=} x \in 18..22$: x is a value between 18 and 22 and $\hat{P_1} = \{18, 19, 20, 21, 22\}$.
- $P_2(p) \stackrel{def}{=} p \subset PEOPLE \wedge card(p) \leq n$: 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 $s_1 \cup \ldots \cup s_p \subseteq \mathrm{D}$.

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*
- $ax2(suc \in \mathbb{N} \rightarrow \mathbb{N} \wedge (!i.i \in \mathbb{N} \Rightarrow suc(i) = i+1))$: *The function suc is the total function which associates any natural i with its successor. successor*
- $ax3(\forall A.A \subseteq \mathbb{N} \wedge 0 \in \mathbb{N} \wedge suc[A] \subseteq A \Rightarrow \mathbb{N} \rightarrow \subseteq A)$: 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 ($s, s' \in \mathcal{V}ar(S) \longrightarrow$ 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

Event-based model of a system

⊠ 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

$$(AX(s,c), x, \textsc{Vals}, Init(x), \{e_0, \ldots, e_n\})$$

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
$EM(s, c, x, \textsc{Vals}, Init(x)\{e_0, \ldots, e_n\}) =$
$(AX(s,c), x, \textsc{Vals}, Init(x), \{e_0, \ldots, e_n\})$.

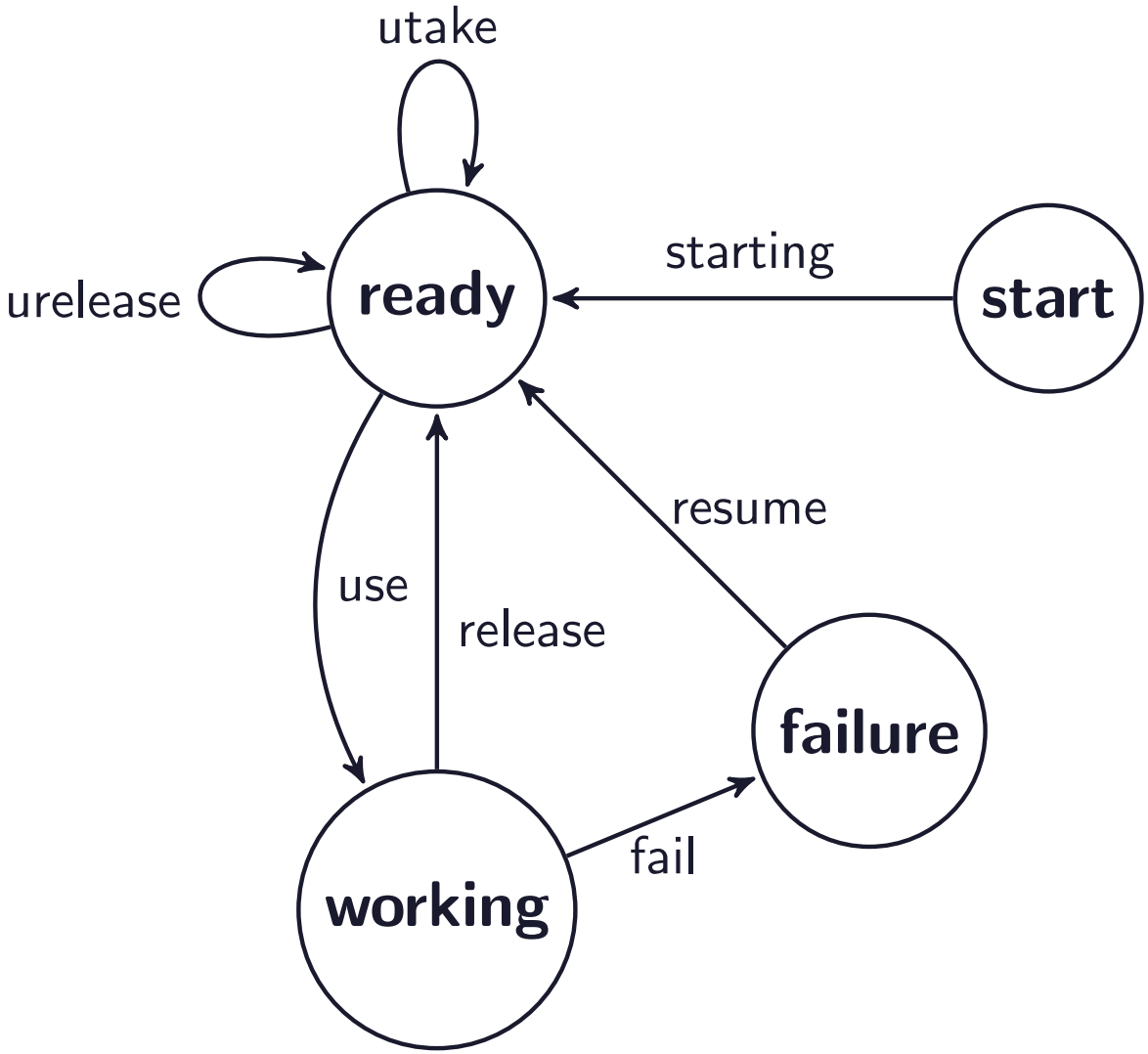
Safety property

- $Next(x, x')$ 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 $Next^\star(s, c, x_0, x) \stackrel{def}{=}$
  $$\begin{cases} \vee\ x = x_0 \\ \vee\ Next(s, c, x_0, x) \\ \vee\ \exists\ xi \in \textsc{Vals}.Next^\star(s, c, x_0, xi) \wedge Next(s, c, xi, x) \end{cases}$$

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

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

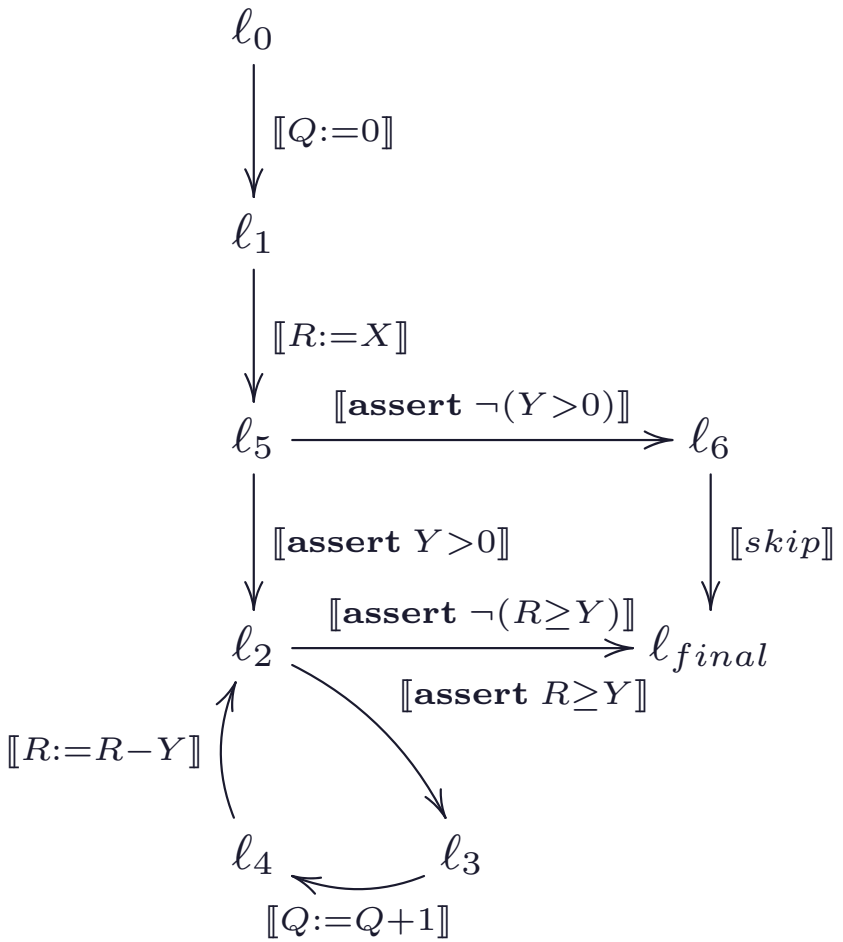
Automata



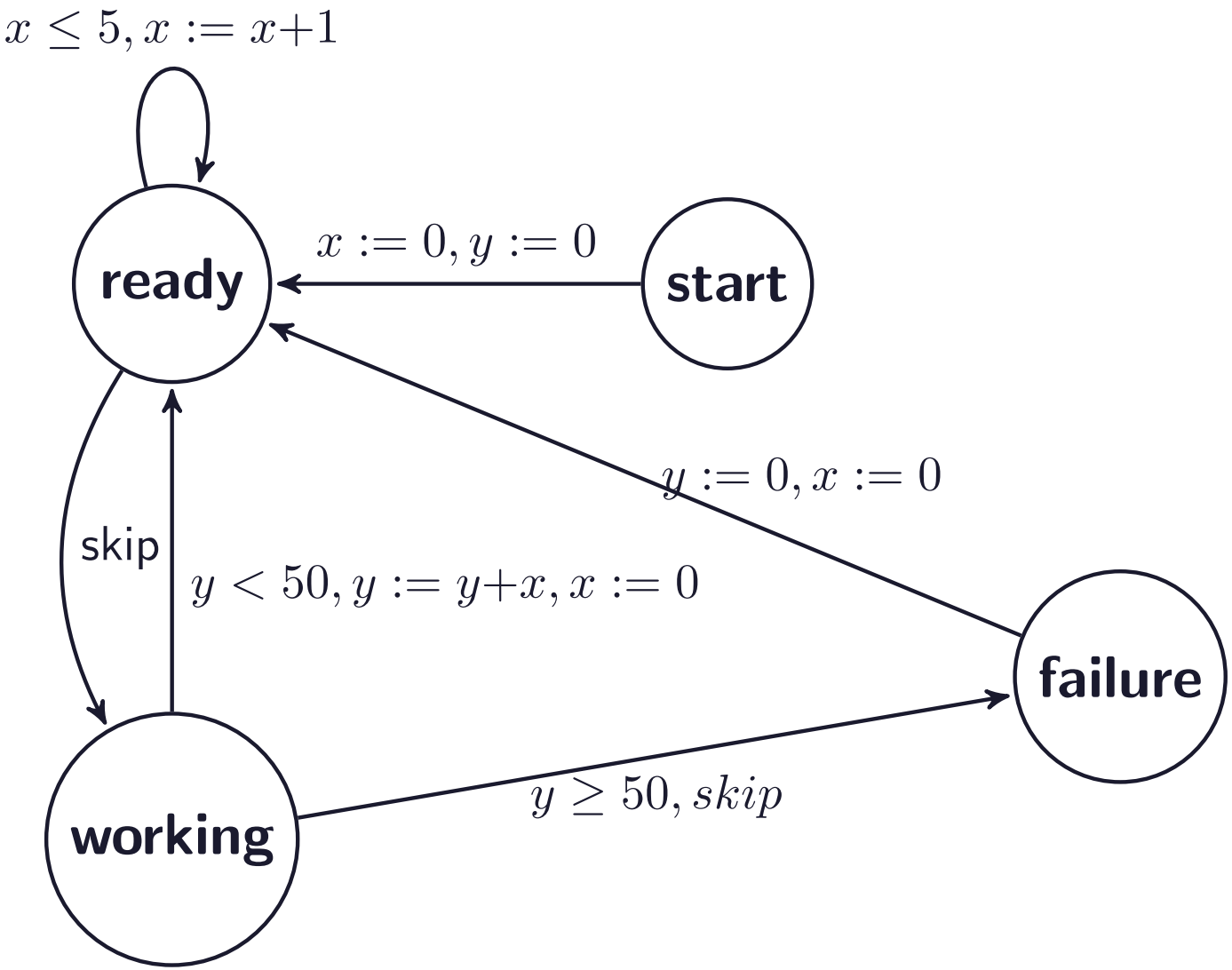
- there is an implicit control variable $pc \in \{\textbf{start}, \textbf{ready}, \textbf{working}, \textbf{failure}\}$ expressing the current visited state

Programme as flowchart

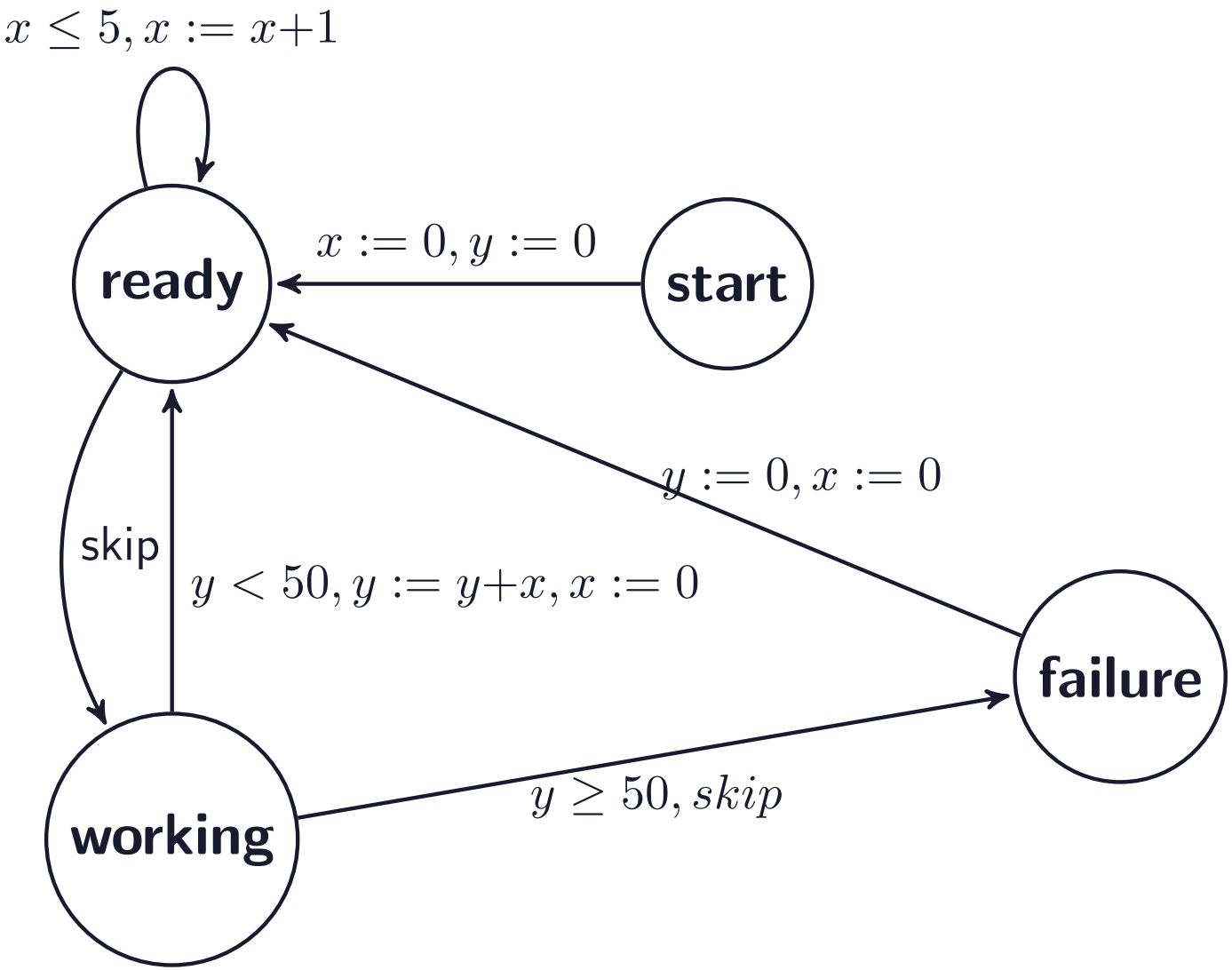
$\ell_0[Q := 0];$
$\ell_1[R := X];$
IF $\ell_5[Y > 0]$
  WHILE $\ell_2[R \geq Y]$
    $\ell_3[Q := Q{+}1];$
    $\ell_4[R := R{-}Y]$
  ENDWHILE
ELSE
  $\ell_6[skip]$
ENDIF



A small system as an automaton

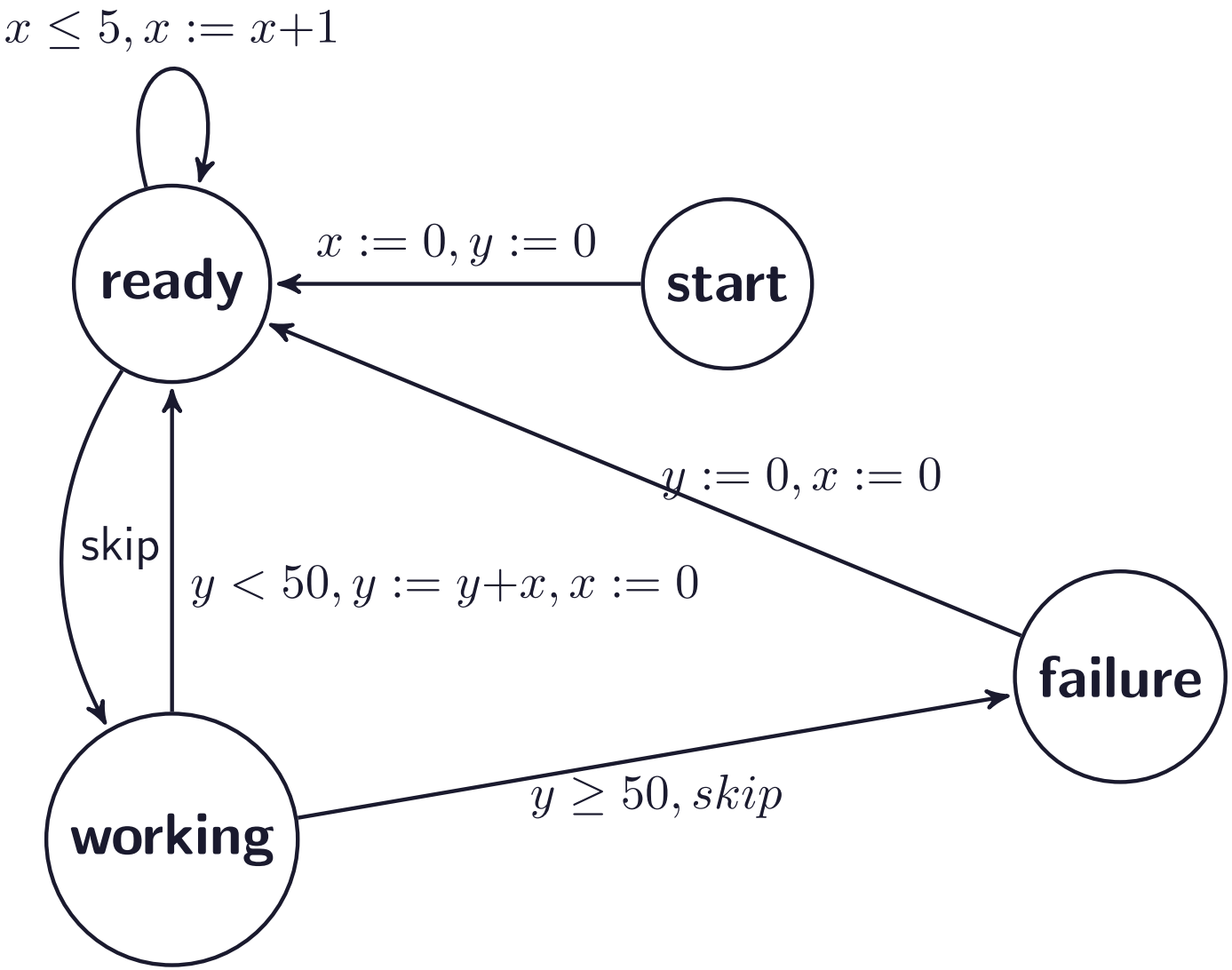


A small system as an automaton



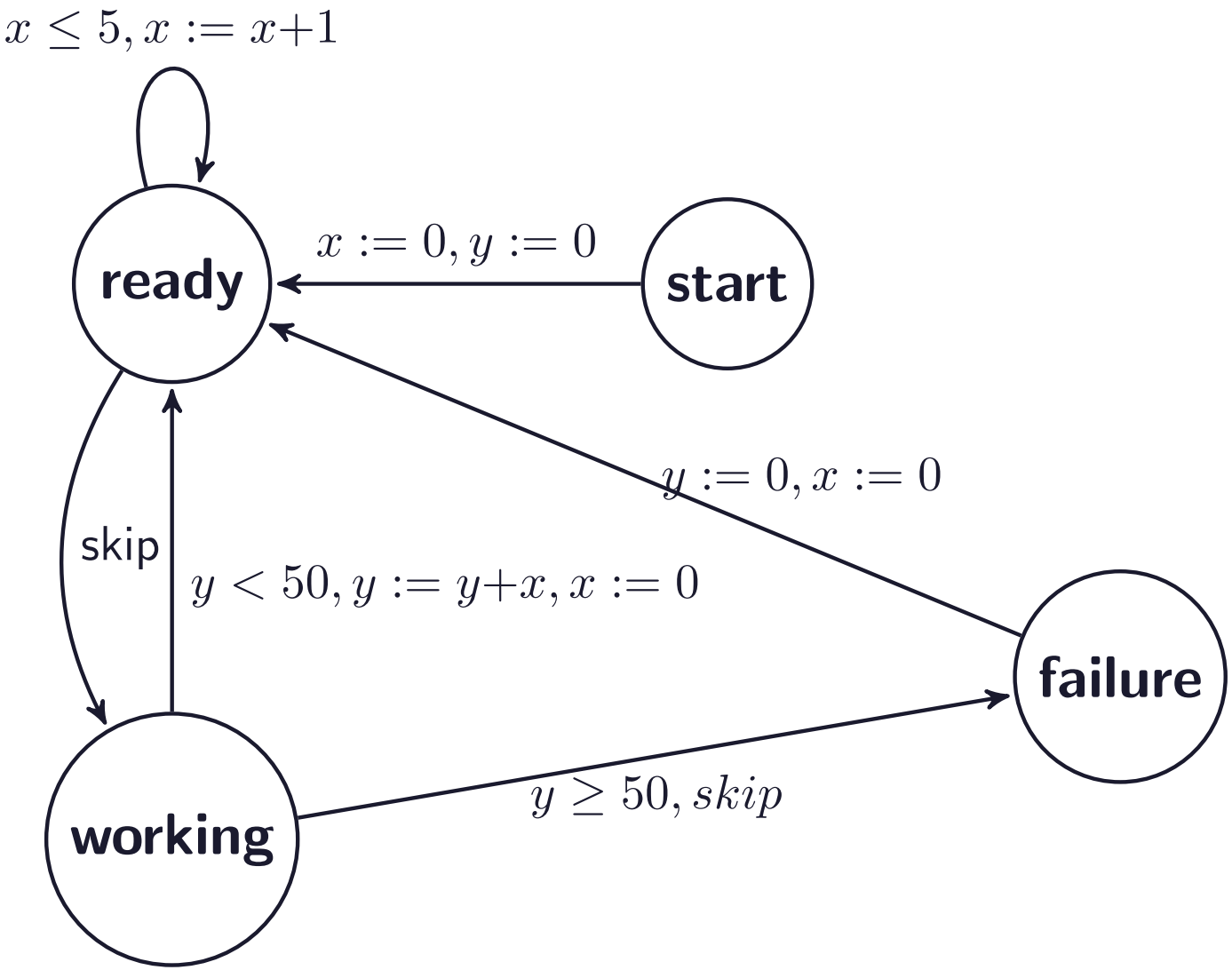
- safety1: $0 \leq x \leq 5$

A small system as an automaton



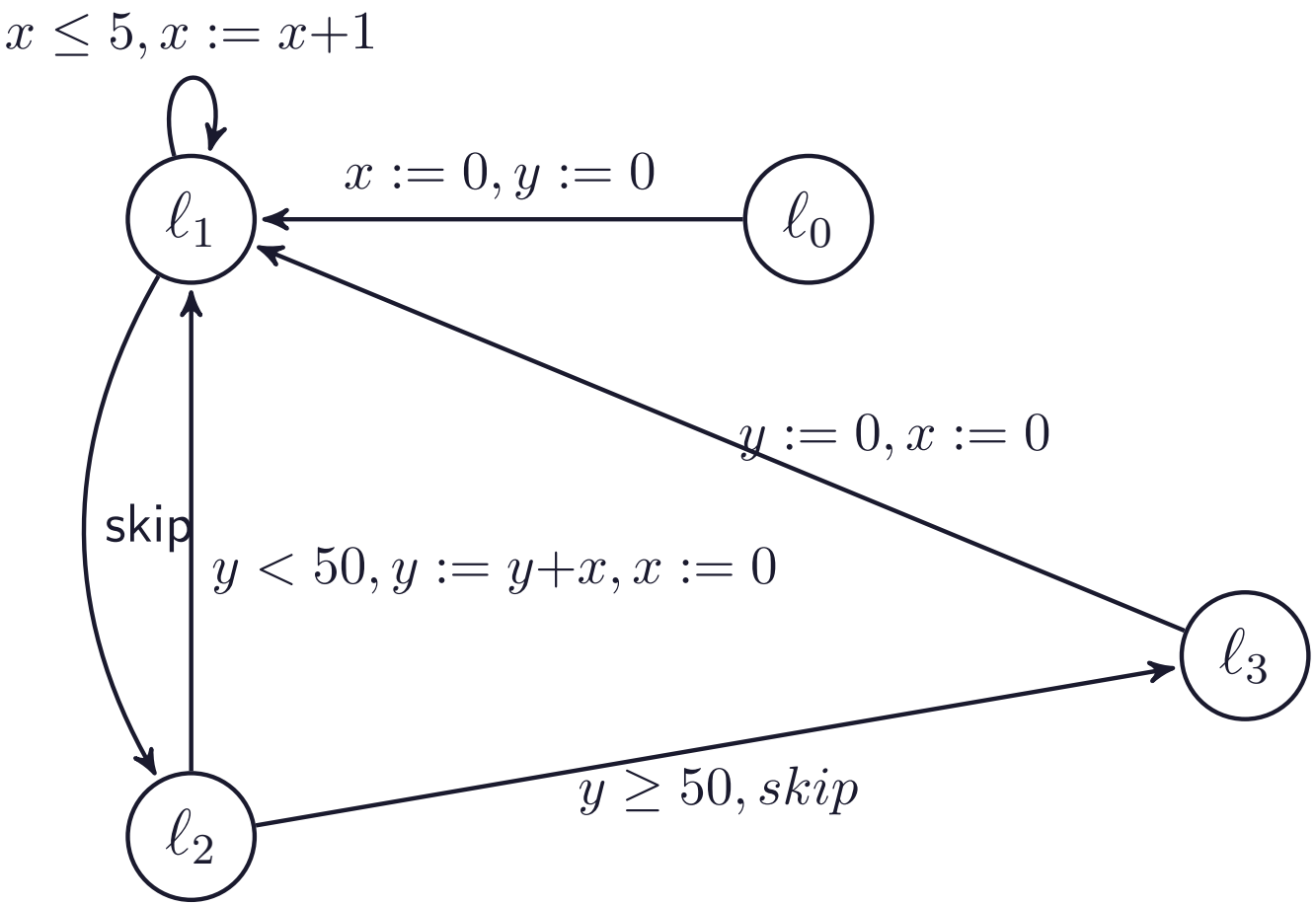
- safety1: $0 \leq x \leq 5$ and ...

A small system as an automaton

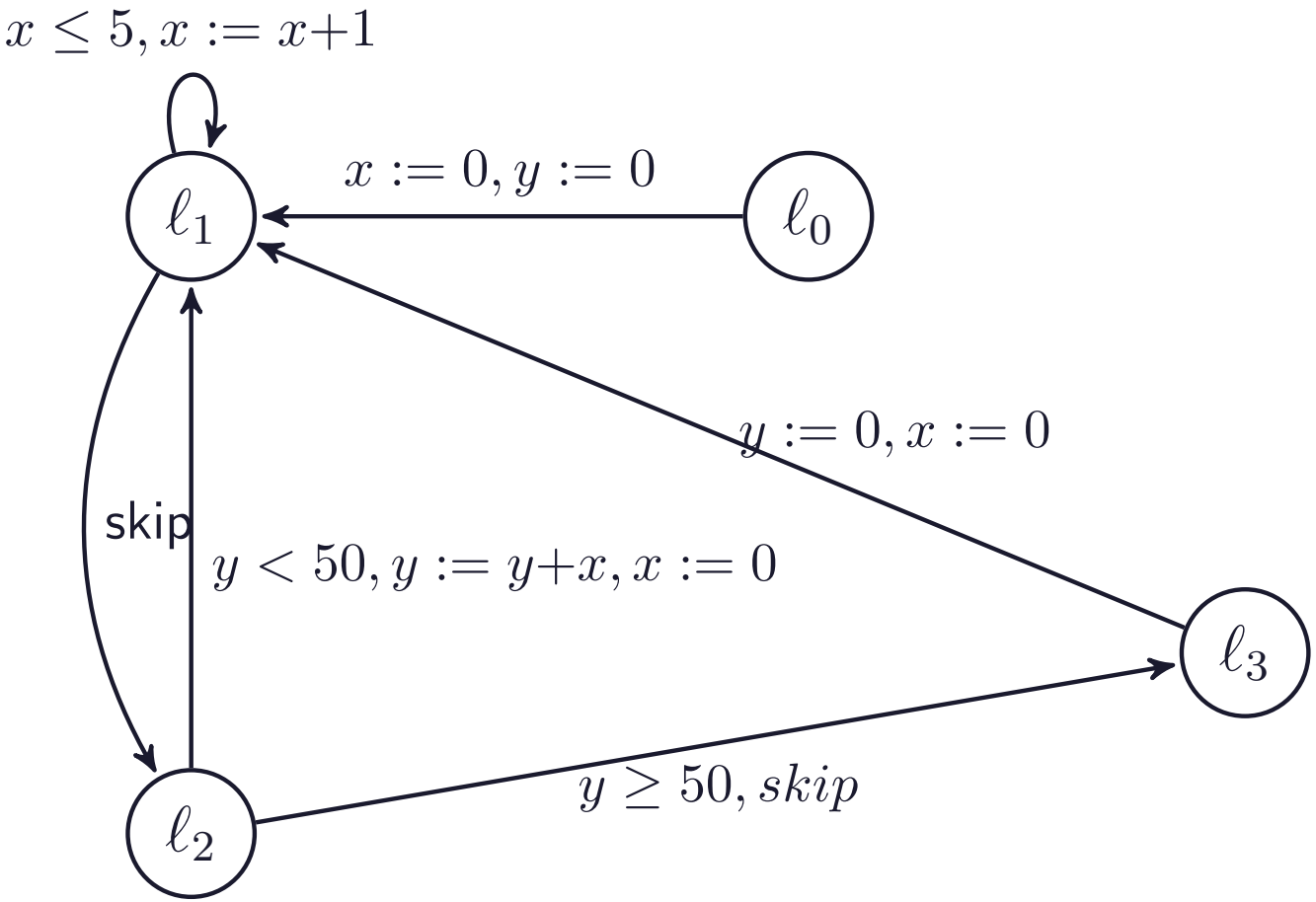


- safety1: $0 \leq x \leq 5$ and . . . safety2: $0 \leq y \leq 56$

A small system as an automaton



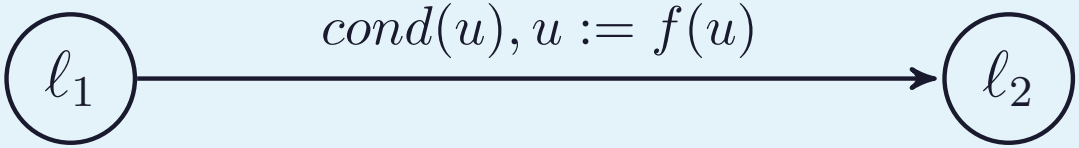
A small system as an automaton



- safety1: $0 \leq x \leq 5$ and safety2: $0 \leq y \leq 56$
- $skip = x := x, y := y$
- skip $= TRUE, x := x, y := y = TRUE, skip$

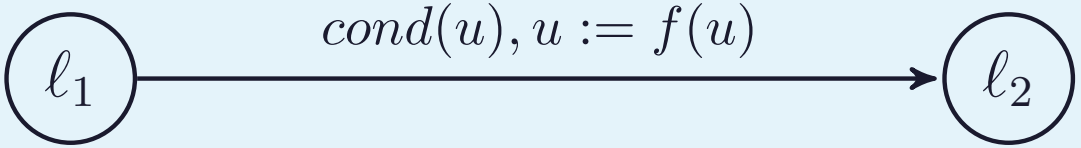
Forms of transitions

Transition between two control states

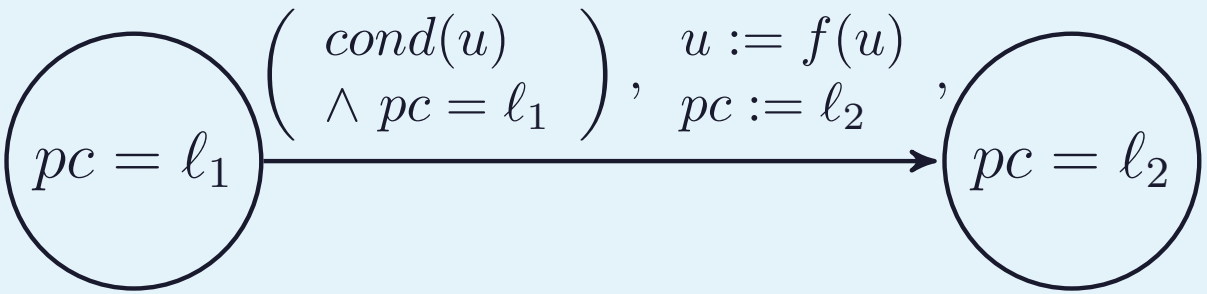


Forms of transitions

Transition between two control states

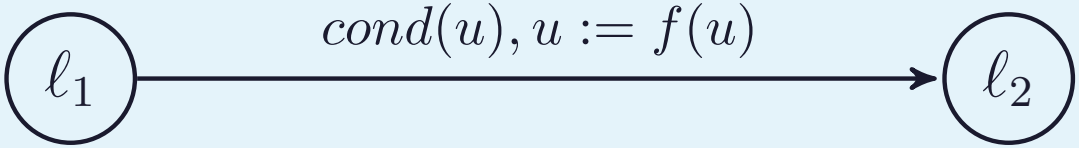


Transition between two control states

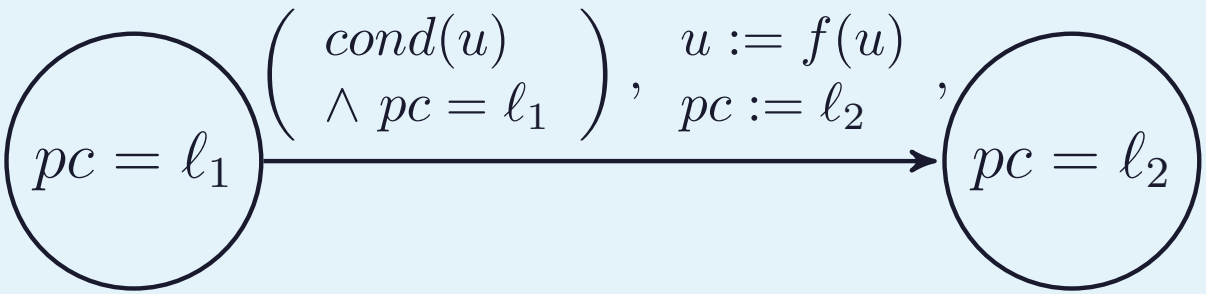


Forms of transitions

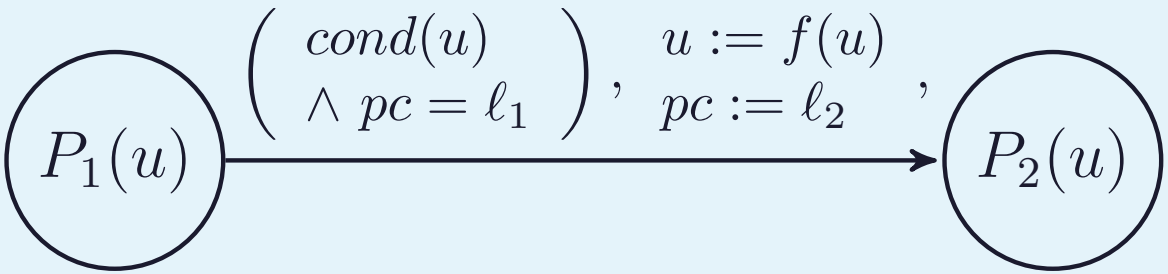
Transition between two control states



Transition between two control states



Transition between two predicates



Relational model of a system

A relational model $\mathcal{MS}$ for a system $\mathcal{S}$ is a structure

$$(Th(s,c), \mathsf{X}, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$$

where

- $Th(s,c)$ is a theory defining the sets, the constants and the static properties of these elements.
- X is a list of flexible variables.
- $\textsc{Vals}$ is a set of possible values for X.
- $\{r_0, \ldots, r_n\}$ is a finite set of relations linking the values before x and the values after x'.
- $\textsc{Init}(x)$ defines the set of initial values of X.
- the relation r_0 is the relation $Id[\textsc{Vals}]$, identity on $\textsc{Vals}$.

⊠ Definition

Let $(Th(s,c), \mathrm{X}, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ be a relational model of a system $\mathcal{S}$. The relation $\textsc{Next}$ associated with this model is defined by the disjunction of the relations r_i:

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

for a variable x, we define the following values:

- x is the current value of the variable X.
- x' is the next value of the variable X.
- x_0 or $\underline{x}$ are the initial value of the variable X.
- $\overline{x}$ or x_f is the final value of the variable X, when this notion makes sense.

Safety and invariance properties in a relational model

⊠ Definition(assertion)
Let $(Th(s,c), \mathrm{X}, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ be a relational model M of a system $\mathcal{S}$. A property A is an assertional safety property for the system $\mathcal{S}$, if

$$\forall x_0, x \in \textsc{Vals}. Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow A(x).$$

⊠ Definition(relation)
Let $(Th(s,c), \mathrm{X}, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ be a relational model M of a system $\mathcal{S}$. A property R is a relational safety property for the system $\mathcal{S}$, if

$$\forall x_0, x \in \textsc{Vals}. Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow R(x_0, x).$$

Part 4

Transition system in action with TLA/TLA$^+$ (file tla0.tla)

Computing GCD

MODULE *gcd26*

EXTENDS *Naturals*, *TLC*

CONSTANTS a, b

VARIABLES x, y

$Init \triangleq x = a \land y = b$

actions

$a1 \triangleq x > y \quad \land x' = x - y \land y' = y$

$a2 \triangleq x < y \quad \land y' = y - x \land x' = x$

$over \triangleq x = y \land x' = x \land y' = y$

$Next \triangleq a1 \lor a2 \lor over$

Propriétés de sûreté à vérifier

$test \triangleq x \neq y$

Computing the function GCD

```
------------------- MODULE pgcd ----------------
EXTENDS Naturals,TLC
CONSTANTS a,b
VARIABLES  x,y
------------------------------------
Init == x=a /\ y=b
------------------------------------
a1 == x > y /\ x'=x-y /\ y'=y
a2 == x < y /\ y'=y-x /\ x'=x
over == x=y /\ x'=x /\ y'=y
------------------------------------
Next == a1 \/ a2 \/ over
------------------------------------
test == x # y
============================================
```

First step

- The system monitors the number of people present in the room and uses a variable np: $0 \leq np$.
- One requirement is that the maximum number of people is max: $np \leq max$.

First step

- The system monitors the number of people present in the room and uses a variable np: $0 \leq np$.
- One requirement is that the maximum number of people is max: $np \leq max$.

Listing 2: Module TLA+

```
1  ------------------------------ MODULE access_control ------------
2  (* using basic modules as Naturals.tla (operations over natural
       numbers) *)
3  (* TLC.tla (print) *)
4  ----------------------------------------------------------------
5  EXTENDS Naturals,TLC
6  ----------------------------------------------------------------
7  CONSTANTS max
8  ----------------------------------------------------------------
9  VARIABLES  np
10 ----------------------------------------------------------------
11 (* Assumptions *)
12 ASSUME max < 50000
13 -------------------------
14 ...
15 ================================================================
```

Second step

- Two actions `entrer` observing when somebody is getting in and `sortir` observing when somebody is getting out.
- `Next` is the disjunction of the two actions.
- Initially np is set to 0
- `test` is expressing safety requirements.

Second step

- Two actions `entrer` observing when somebody is getting in and `sortir` observing when somebody is getting out.
- `Next` is the disjunction of the two actions.
- Initially np is set to 0
- `test` is expressing safety requirements.

Listing 4: Module TLA+

```
1  ------------------------------ MODULE access_control -----------
2  ...
3  -------------------------
4  (*  tentative 1 *)
5  entrer == np < 50 /\ np'=np +1
6  sortir ==  np > 0 /\ np'=np-1
7  Next ==
8       \/ entrer
9       \/ sortir
10 Init == np=0
11 test1 == np # 12
12 test == 0 <= np /\ np <= max
13 ------------------------------------------------------------------------
14 .....
15 ========================================================================
```

Second step

- Two actions `entrer` observing when somebody is getting in and `sortir` observing when somebody is getting out.
- `Next` is the disjunction of the two actions.
- Initially np is set to 0
- `test` is expressing safety requirements.

Listing 5: Module TLA+

```
1  ------------------------------ MODULE access_control -----------
2  ...
3  -------------------------
4  (*  tentative 1 *)
5  entrer == np < 50 /\ np'=np +1
6  sortir ==  np > 0 /\ np'=np-1
7  Next ==
8       \/ entrer
9       \/ sortir
10 Init == np=0
11 test1 == np # 12
12 test == 0 <= np /\ np <= max
13 ------------------------------------------------------------------------
14 .....
15 ========================================================================
```



Third step

- Controling the access by subsituting 50 by max in `entrer`.
- `next2` is the disjunction of the two actions `entre2` and `sortir`

Third step

- Controling the access by subsituting 50 by max in `entrer`.
- `next2` is the disjunction of the two actions `entre2` and `sortir`

Listing 7: Module TLA+

```
1  ------------------------------- MODULE access_control ------------
2  ...
3  ------------------------------------------------------------------------

4  (* tentative 2 *)
5  entrer2 == np<max /\ np'=np+1
6  next2 ==  entrer2  \/  sortir
7  ------------------------------------------------------------------------

8  .....
9  ========================================================================
```

Third step

- Controling the access by subsituting 50 by max in `entrer`.
- `next2` is the disjunction of the two actions `entre2` and `sortir`

Listing 8: Module TLA+

```
1  ------------------------------ MODULE access_control -----------
2  ...
3  ---------------------------------------------------------------------------

4  (* tentative 2 *)
5  entrer2 == np<max /\ np'=np+1
6  next2 ==  entrer2  \/  sortir
7  ---------------------------------------------------------------------------

8  .....
9  ===========================================================
```



Success!

Final module for access control

Listing 9: Module TLA+

```
1  ------------------------------- MODULE access_control
       ---------------------------------
2  EXTENDS Naturals,TLC
3  ------------------------------------------------------------------------------

4  CONSTANTS max
5  ------------------------------------------------------------------------------

6  VARIABLES  np
7  ------------------------------------------------------------------------------

8  (* Assumptions *)
9  ASSUME max < 50000
10 -------------------------
11 (*  tentative 1 *)
12 entrer == np < 50 /\ np'=np +1
13 sortir ==  np > 0 /\ np'=np-1
14 Next ==
15      \/ entrer
16      \/ sortir
17 Init == np=0
18 test1 == np # 12
19 test == 0 <= np /\ np <= max
20 ------------------------------------------------------------------------------

21 (* tentative 2 *)
```

Transition system in TLA/TLA$^+$

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
$\forall x_0, x \in \textsc{Vals}. Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow A(x)$.

Transition system in TLA/TLA$^+$

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`

Transition system in TLA/TLA$^+$

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)`

Transition system in TLA/TLA$^+$

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`

Transition system in TLA/TLA$^+$

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`
- $A(x)$ is a logical expression defining a safety property to be verified for all configurations of the model: `Safety == A(x)`

The TLA logic and the TLA$^+$ language

- TLA (Temporal Logic of Actions) is used to express formulas in temporal logic: $\Box P$ or *always P, ...*
- TLA$^+$ is a language for declaring constants, variables and definitions:
  - `<def> == <expression>` : a definition `<def>` is the assignment of an expression `<expression>` that uses elements defined before or within modules that *extend* this module.
  - A variable x is either in the form x or in the form x': x' is the value of x afterwards.
  - A module has a name and contains definitions; it may 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 .
- A configuration must be defined in order to evaluate a specification

The TLA logic and the TLA$^+$ language

Temporal formulas in TLA$^+$: syntax

State formulas

A state formula is a property relating to the current state: $P(x_1, \ldots, x_n)$.

Action formulas

An action formula links two successive states: $A(x_1, \ldots, x_n, x'_1, \ldots, x'_n)$ where x' denotes the value of x in the next state and x is a flexible variable.

Temporal formulas

Let us assume two formulas F and G of TLA$^+$:

$\Box F$	:	always F
$\Diamond F$	:	eventually F
$F \Rightarrow G$	:	if F, then G
$F \rightsquigarrow G$	:	F leads to G

Semantics for temporal formulas

behavior

A behaviour is an infinite sequence of states: $\sigma = \sigma_0 \to \sigma_1 \to \sigma_2 \to \cdots$

$[\Box F$ — Always

$\sigma \models \Box F$ means that $\forall i \in \mathbb{N}$, $\sigma^{+i} \models F$

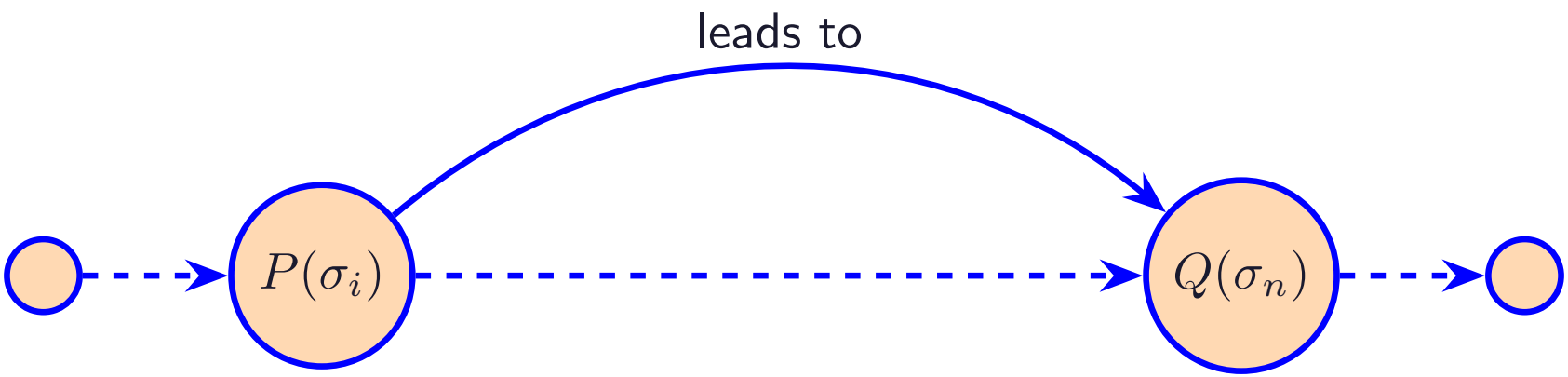
$[\Diamond F$ — Eventually

$\sigma \models \Diamond F$ means that $\exists i \in \mathbb{N}$, $\sigma^{+i} \models F$

Liveness properties

Intuitive meaning

P leads to Q means that for each state satisfying P there exists on the trace a state satifying Q.



Formal meaning

$\sigma \models P \rightsquigarrow Q$: $\forall i \geq 0 : \sigma^{+i} \models P \Rightarrow \exists j \geq i : \sigma^{+j} \models Q$

‘‘‘

The TLA logic and the TLA$^+$ language

- Restrictions on actions:

$$\texttt{nom} \triangleq \\ \quad \wedge\ cond(v, w) \\ \quad \wedge\ v' = e(v, w) \\ \quad \wedge\ w' = w$$

- $e(v, w)$ must be implementable in Java.
- Standard modules: Naturals, Integers, TLC...

General form for a module in TLA$^+$

- name for a module.
- EXTENDS used modules
- CONSTANTS parameters
- VARIABLES state variables
- definitions sets, functions, relations, operator as expression `def == expression`.
- Init initial state
- Next transitions
- Spec tempral specification
- THEOREM properties to check

General form for the configuration file .cfg

Listing 10: Configuration TLC

```
1  SPECIFICATION NomDeLaSpecification
2
3  CONSTANT
4      NomConstante = Valeur
5
6  INIT NomInitialisation
7
8  NEXT NomTransition
9
10 INVARIANT
11      NomInvariant1
12       NomInvariant2
13
14 PROPERTY NomPropriete1
15 PROPERTY NomPropriete2
16
17 CONSTRAINT NomContrainte
18
19 CHECK_DEADLOCK
```

Main directives for a configuration file

- SPECIFICATION Spec Specifies the behavioural action to be verified
- INIT Init Defines the initial state
- NEXT Next Defines the transitions
- CONSTANT C = value Assigns a value to a constant
- INVARIANT P Verifies that P is always true
- PROPERTY P Verifies a temporal property
- CONSTRAINT C Limits the state space being explored
- CHECK_DEADLOCK Requests deadlock detection

Command Line

- The tla2tools.jar file contains multiple TLA$^+$ tools: adding tla2tools.jar to your CLASSPATH
- EXPORT CLASSPATH=tla2tools.jar
- java tla2sany.SANY -help: The TLA$^+$ parser
- java tlc2.TLC -help: The TLA$^+$ model checker
- java tlc2.REPL: Enter the TLA$^+$ REPL
- java pcal.trans -help: The PlusCal-to-TLA$^+$ translator
- java tla2tex.TLA -help: The TLA$^+$-to-LaTeX translator
- java tla2sany.xml.XMLExporter -help: Export TLA$^+$ parse tree as XML
- Running java -jar tla2tools.jar is aliased to run tlc2.TLC.

Commentaires

- Download the application from the Microsoft site for your computer.
- Write models and test them!
- Limitations due to the domains of the variables.



Allow symbolic reasoning whatever the set of states

Steps for partial correctness PC

1. Identify the input data and the results produced: domains, types, conditions.
2. Normalise the code in the form of a contract defining the precondition and the postcondition: formalisation of the code in the form of a function, modified variables, unmodified variables, the *requires* assertion, and the *ensures* relation.
3. Annotation of the code of the procedure: control points, assertions for each point.
4. Generating proof obligations for partial correctness: PO(PC).
5. Checking proof obligations for partial correctness.