UNIVERSITÉ DE LORRAINE

TELECOM Nancy
Ingénieurs du numérique • Inspiring your digital future

Cours MALG & MOVEX

Modélisation, spécification et vérification (II)

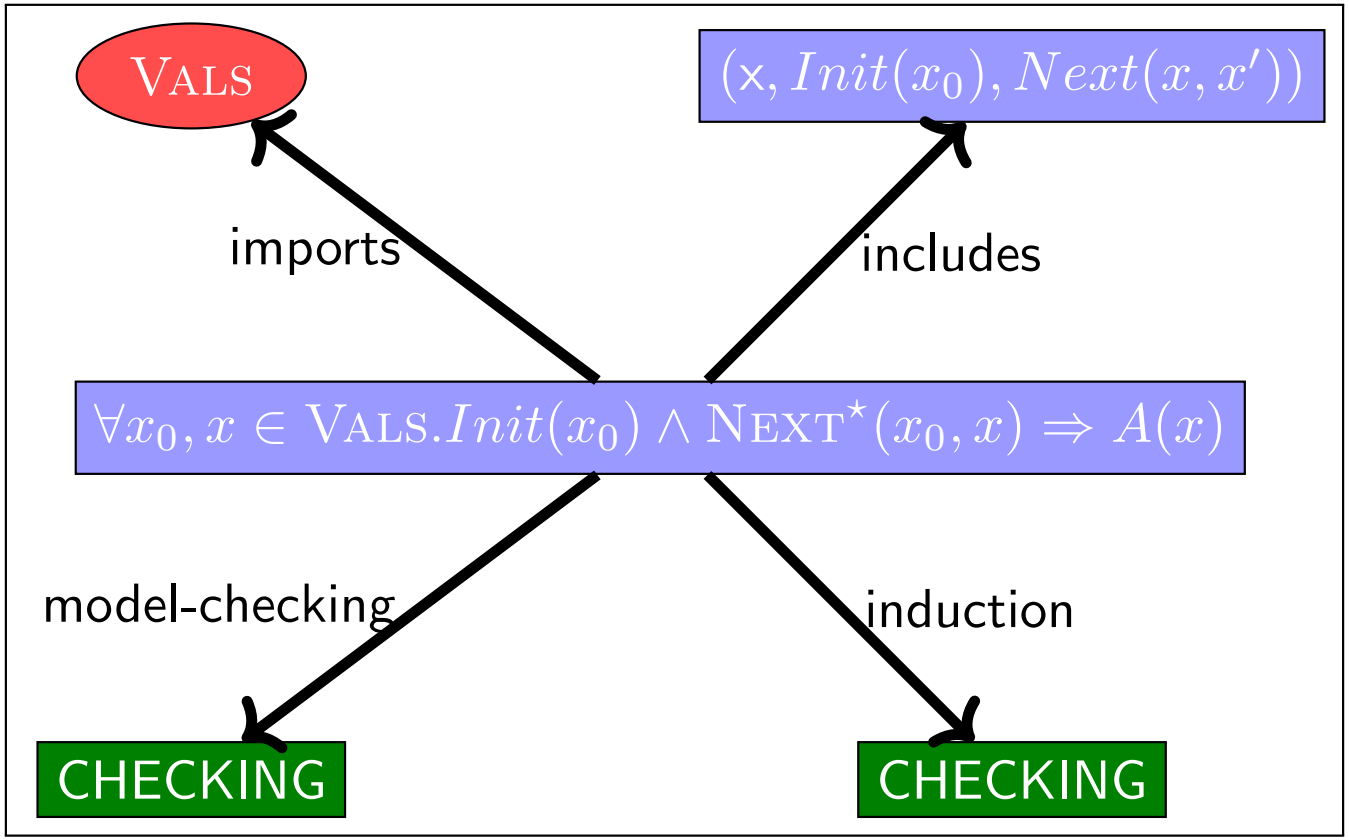
Dominique Méry
Telecom Nancy, Université de Lorraine

Année universitaire 2023-2024

Plan

Sommaire

Tryptic Pattern for Safety Properties



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.

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 $\Sigma \stackrel{def}{=} \mathtt{Var} \longrightarrow \mathrm{D}$ 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. $[\![\mathtt{x}]\!](s) = s(\mathtt{x}) = x$: 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. $[\![\varphi(x) \wedge \psi(x)]\!](s) = [\![\varphi(x)]\!](s)$ et $[\![\psi(x)]\!](s)$ 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.

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

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 $PEOPLE$.

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

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 ($s, s' \in \mathcal{V}ar(S) \longrightarrow \text{VALS}$). $s \xrightarrow[R]{} s'$ 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 $BA(e)(s, c, x, x')$.

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

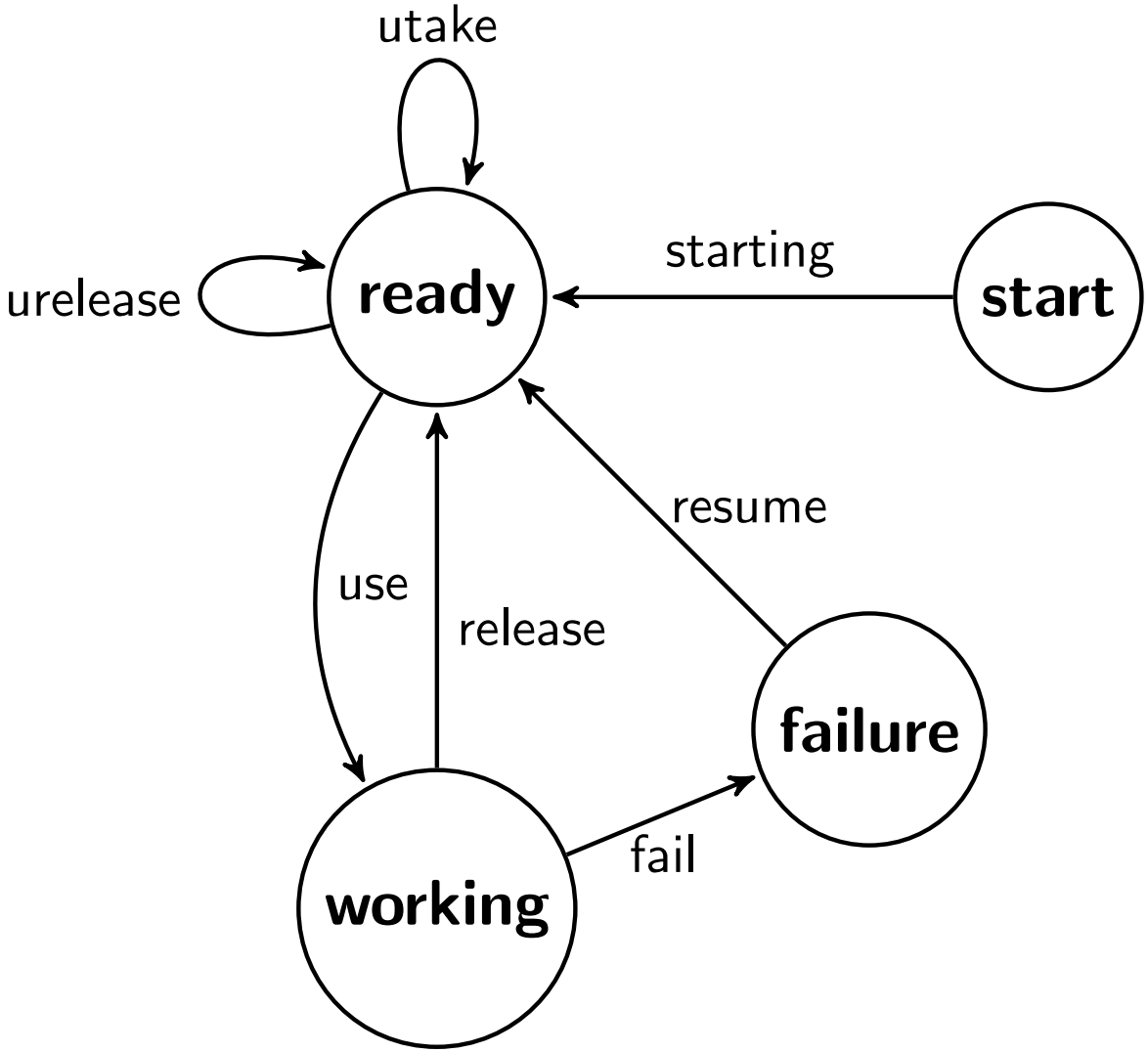
- $Next(x, x')$ or
  $Next(s, c, x, x') \stackrel{def}{=} BA(e_0)(s, c, x, x') \vee \ldots \vee BA(e_n)(s, c, x, x').$
- the transitive reflexive closure of the relation $Next^\star(s, c, x_0, x) \stackrel{def}{=}$
  $$\begin{cases} \vee\ x = x_0 \\ \vee\ Next(s, c, x_0, x) \\ \vee\ \exists\ xi \in \textsc{Vals}.Next^\star(s, c, x_0, xi) \wedge Next(s, c, xi, x) \end{cases}$$

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

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

Current Subsection Summary

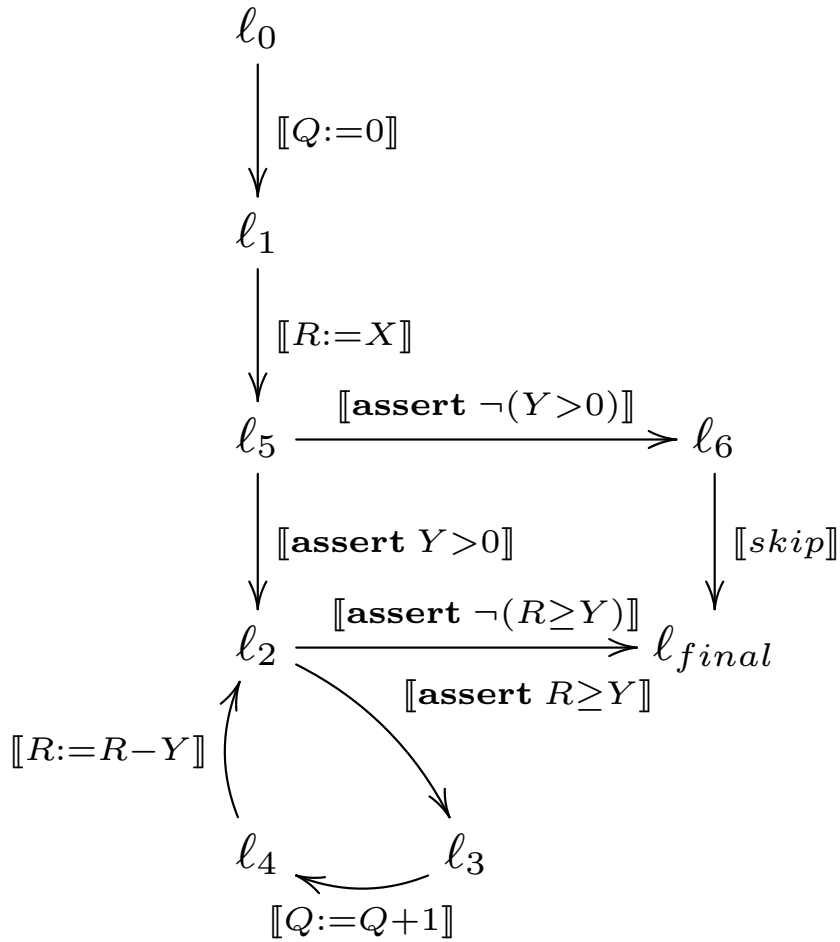
Automata



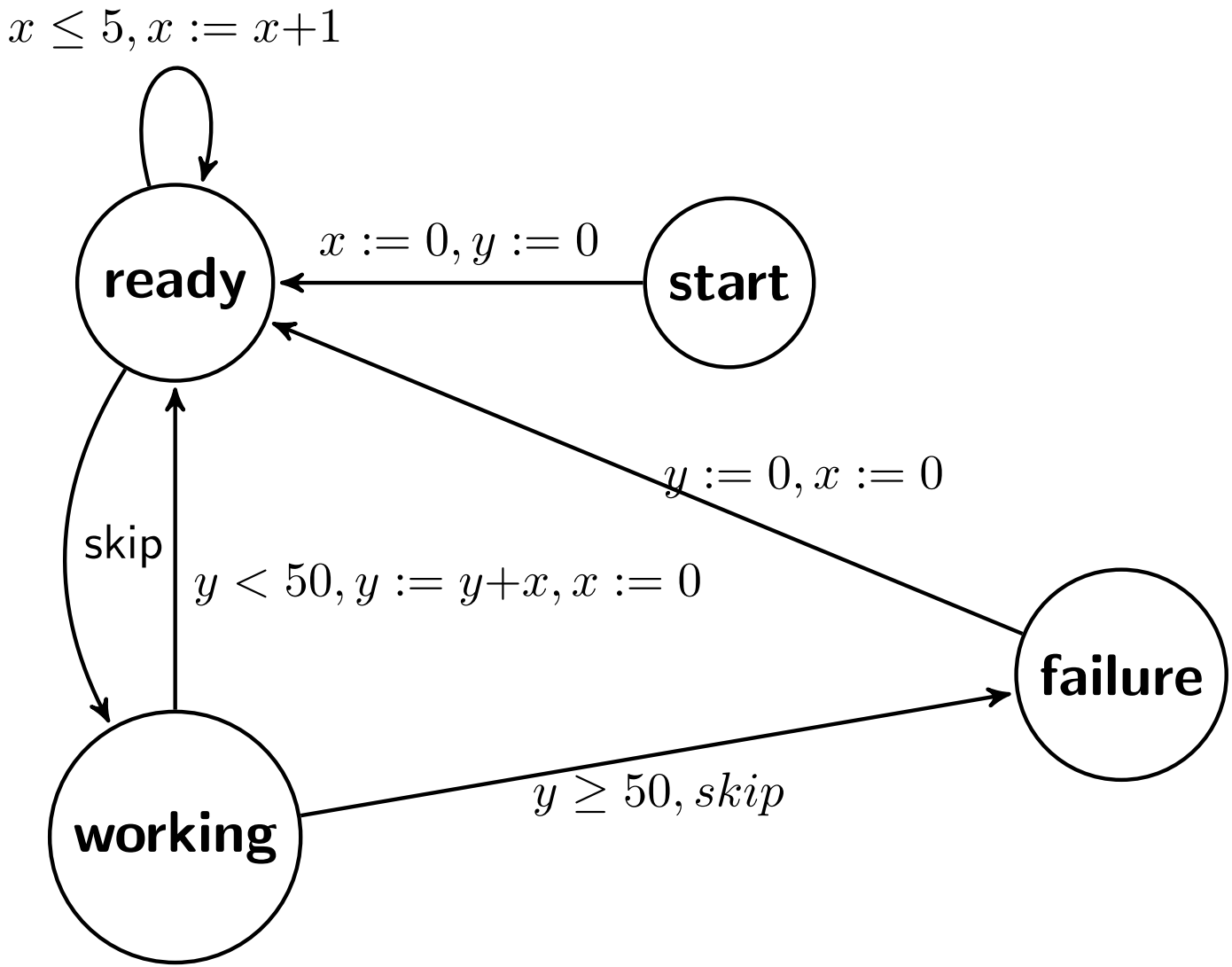
- there is an implicit control variable $pc \in \{\textbf{start}, \textbf{ready}, \textbf{working}, \textbf{failure}\}$ expressing the current visited state.

Programme as flowchart

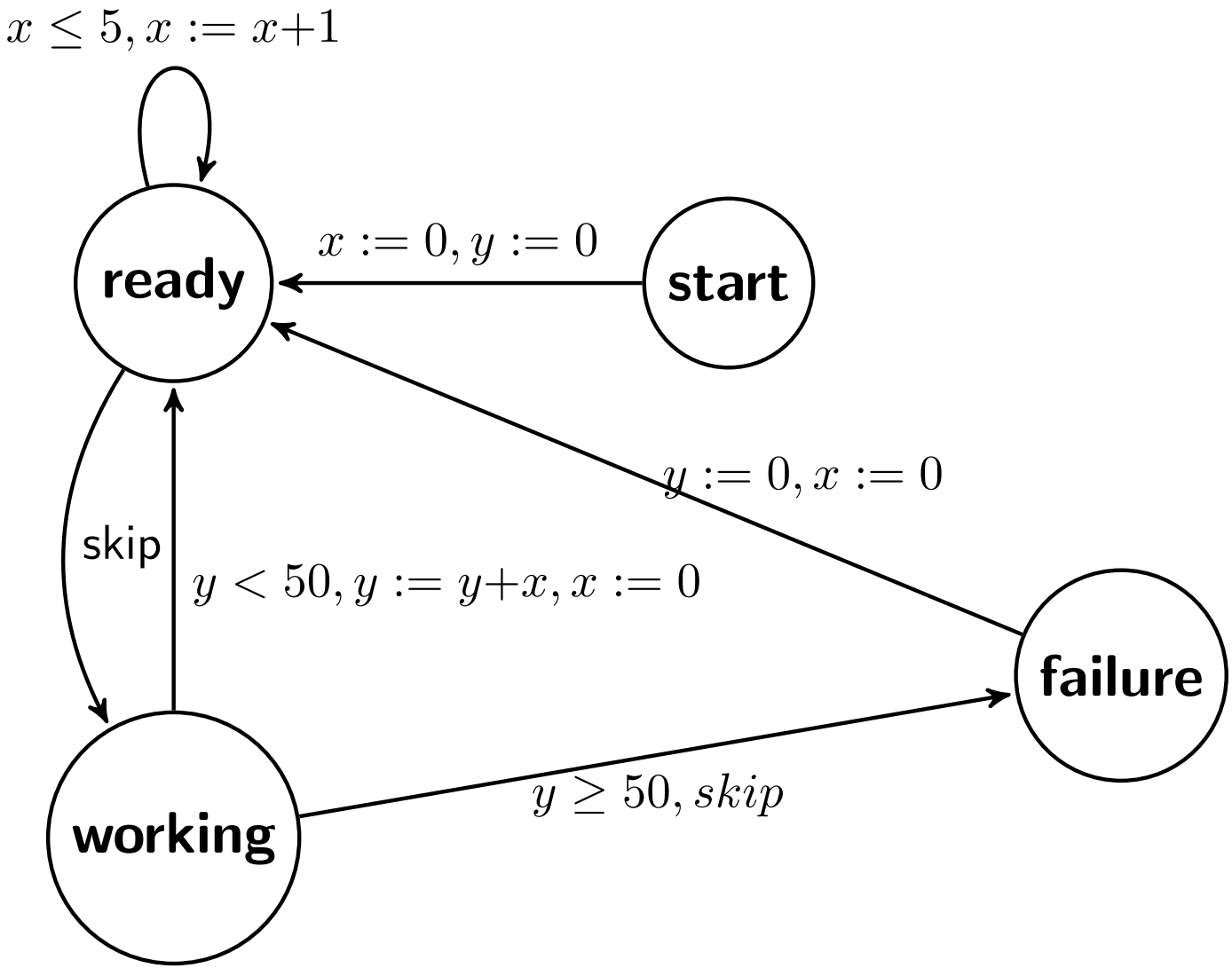
$\ell_0[Q := 0];$
$\ell_1[R := X];$
IF $\ell_5[Y > 0]$
  WHILE $\ell_2[R \geq Y]$
    $\ell_3[Q := Q{+}1];$
    $\ell_4[R := R{-}Y]$
  ENDWHILE
ELSE
  $\ell_6[skip]$
ENDIF



A small system as an automaton

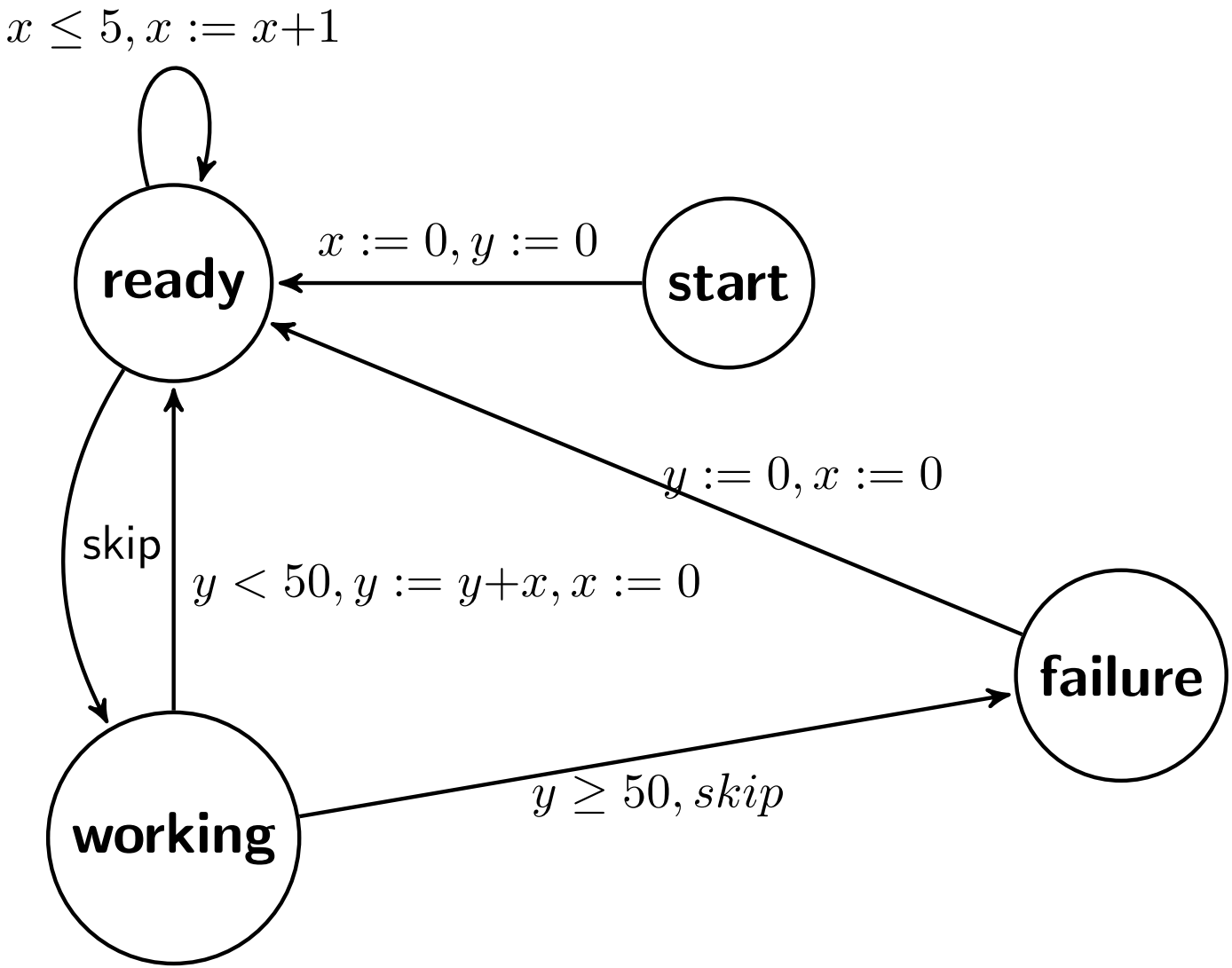


A small system as an automaton



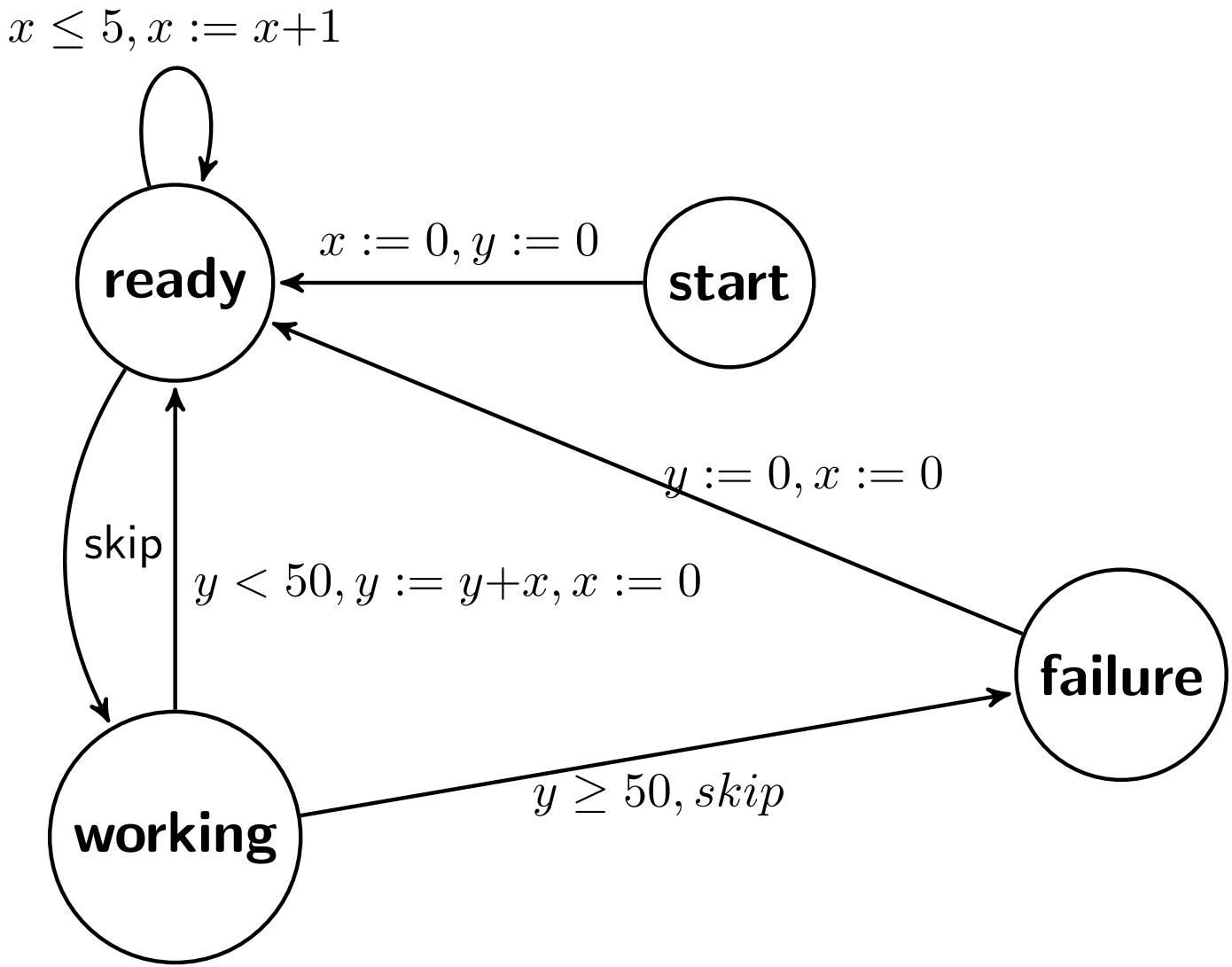
- safety1 : $0 \leq x \leq 5$

A small system as an automaton



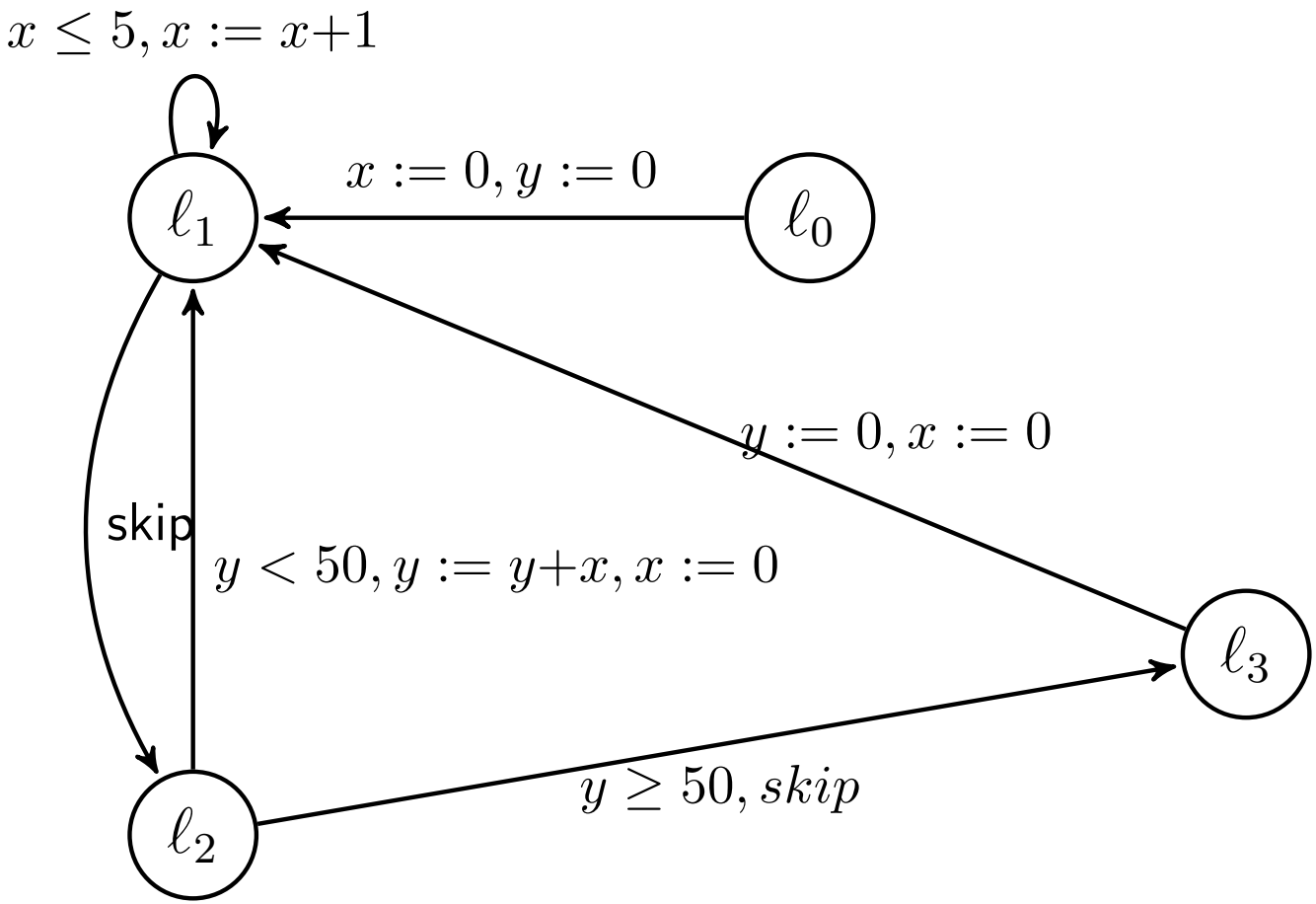
- safety1 : $0 \leq x \leq 5$ et ...

A small system as an automaton

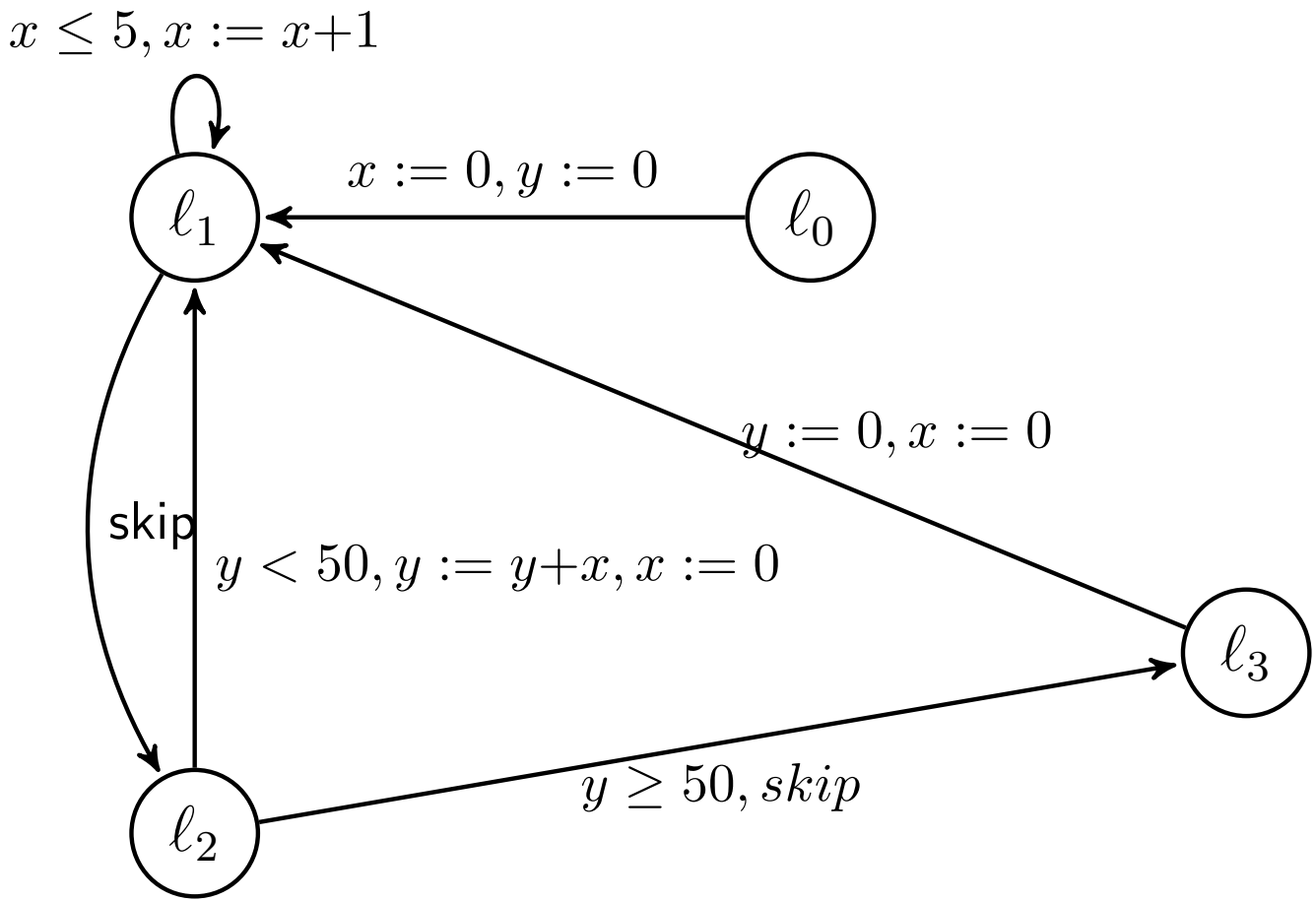


- safety1 : $0 \leq x \leq 5$ et ... safety2 : $0 \leq y \leq 56$

A small system as an automaton



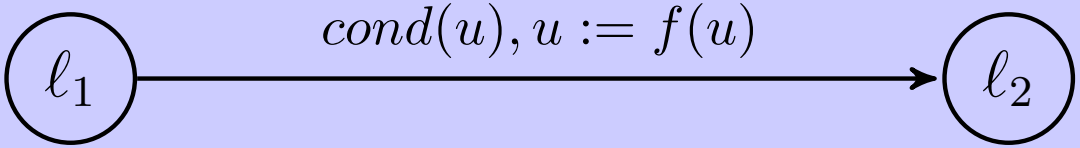
A small system as an automaton



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

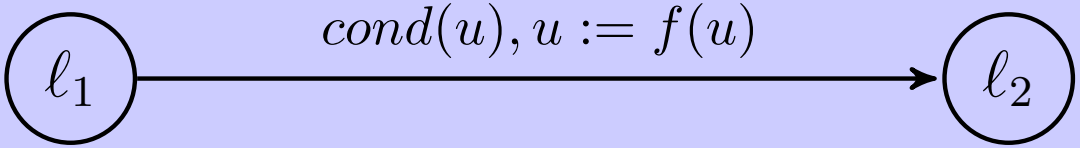
Forms of transitions

Transition between two control states

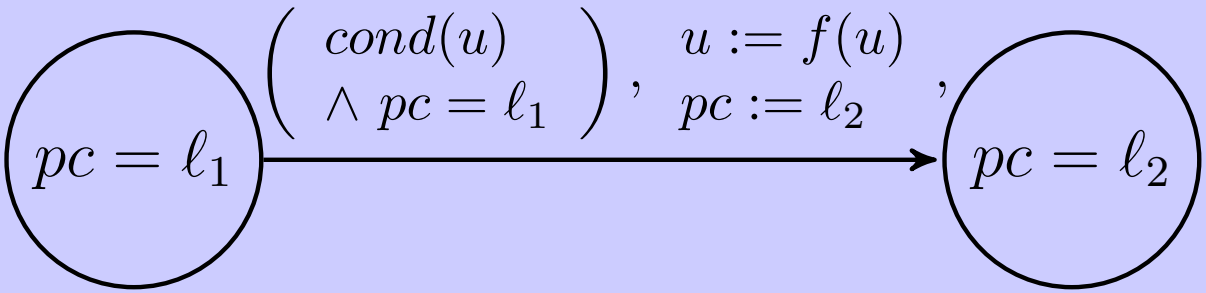


Forms of transitions

Transition between two control states

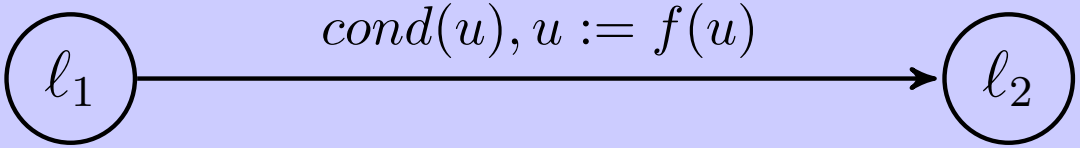


Transition between two control states

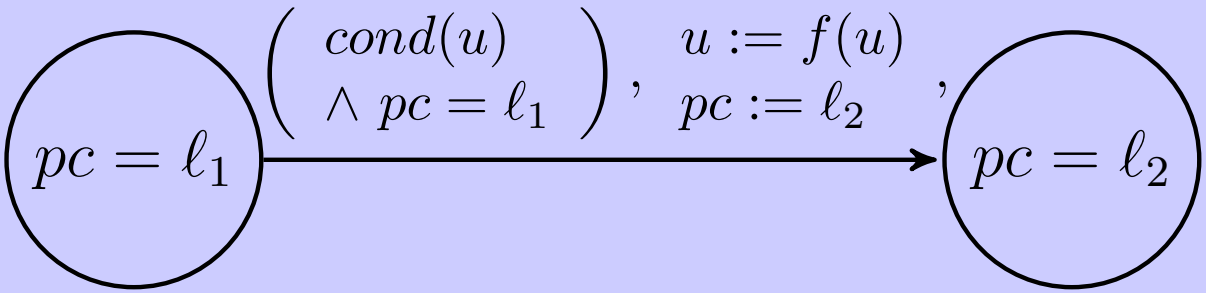


Forms of transitions

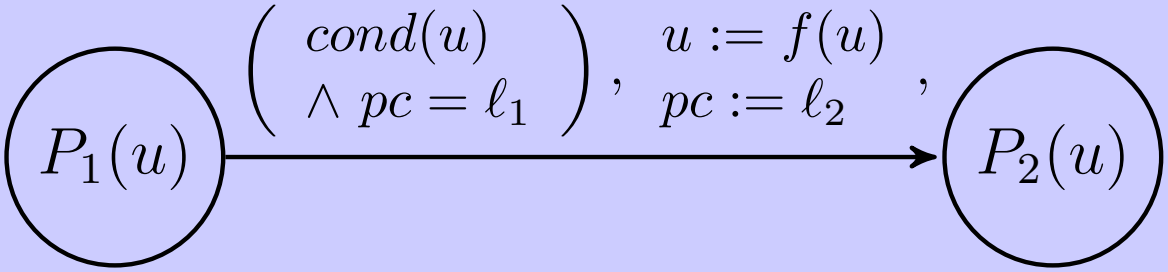
Transition between two control states



Transition between two control states



Transition between two predicates



Current Subsection Summary

Modèle relationnel d'un système

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

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

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.
- $\textsc{Vals}$ 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'.
- $\textsc{Init}(x)$ définit l'ensemble des valeurs initiales de X.
- la relation r_0 est la relation $Id[\textsc{Vals}]$, identité sur $\textsc{Vals}$.

⊠ Definition

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

⊠ Definition(assertion)
Soit $(Th(s,c), \mathsf{X}, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système $\mathcal{S}$. Une propriété A est une propriété assertionnelle de sûreté pour le système $\mathcal{S}$, si

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

⊠ Definition(relation)
Soit $(Th(s,c), \mathsf{X}, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système $\mathcal{S}$. Une propriété R est une propriété relationnelle de sûreté pour le système $\mathcal{S}$, si

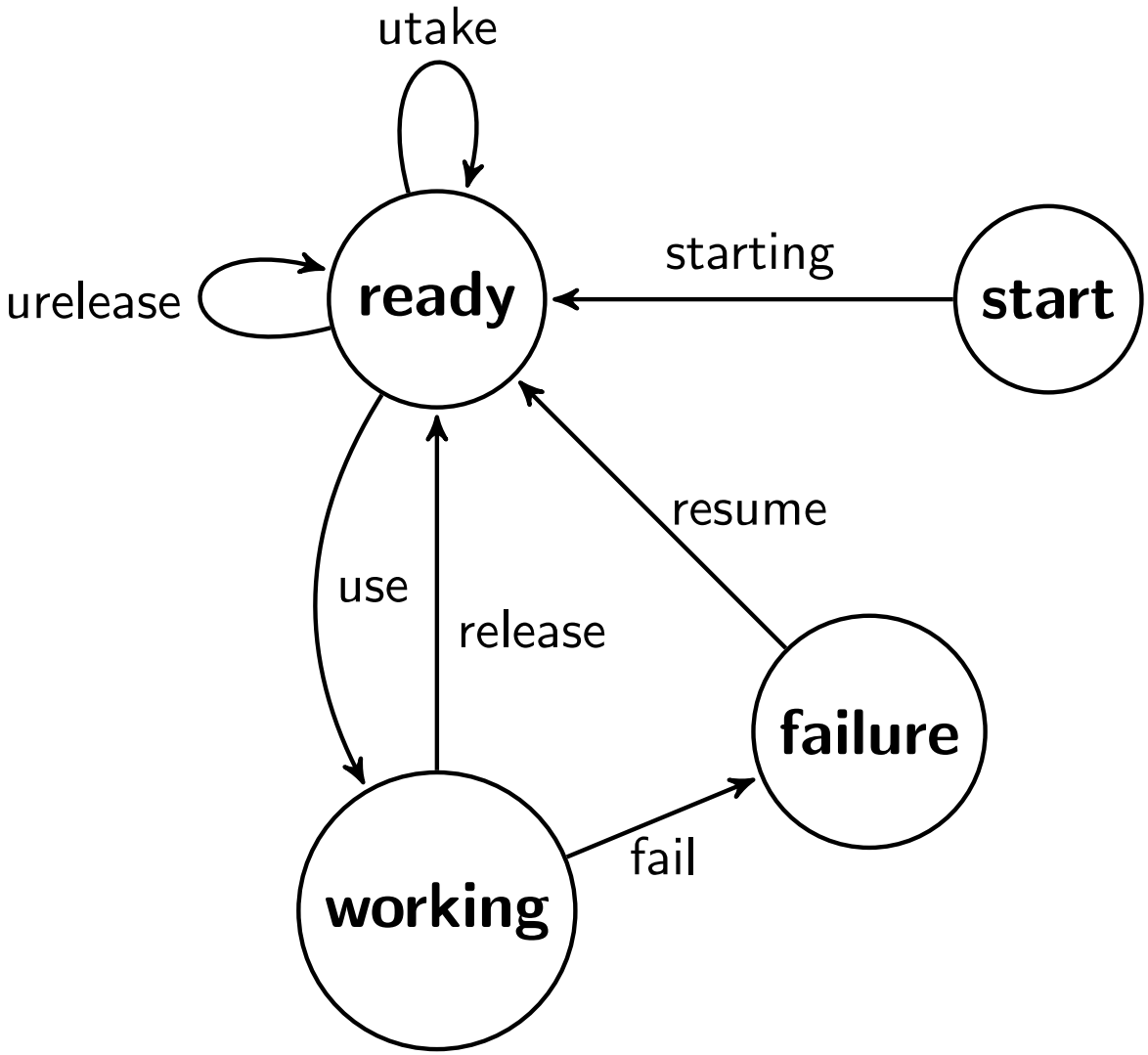
$$\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow R(x_0, x).$$

Assertion versus relation

- 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).

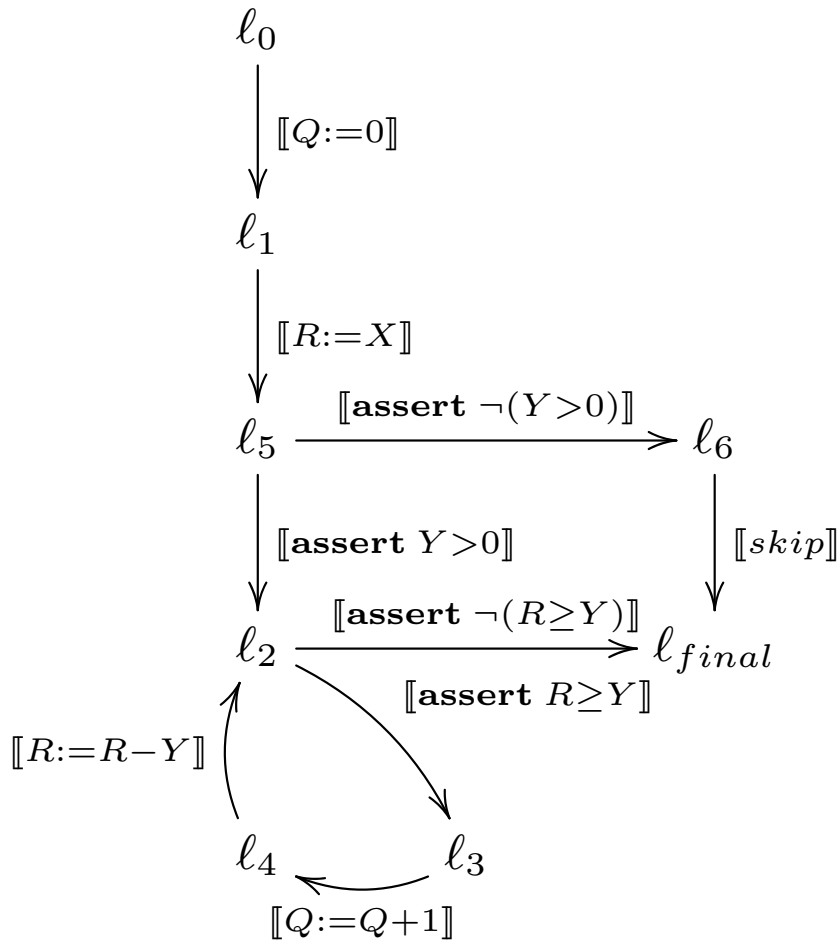
Current Subsection Summary

Automata


utake
urelease
ready
starting
start
use
release
resume
failure
working
fail

Programme as flowchart

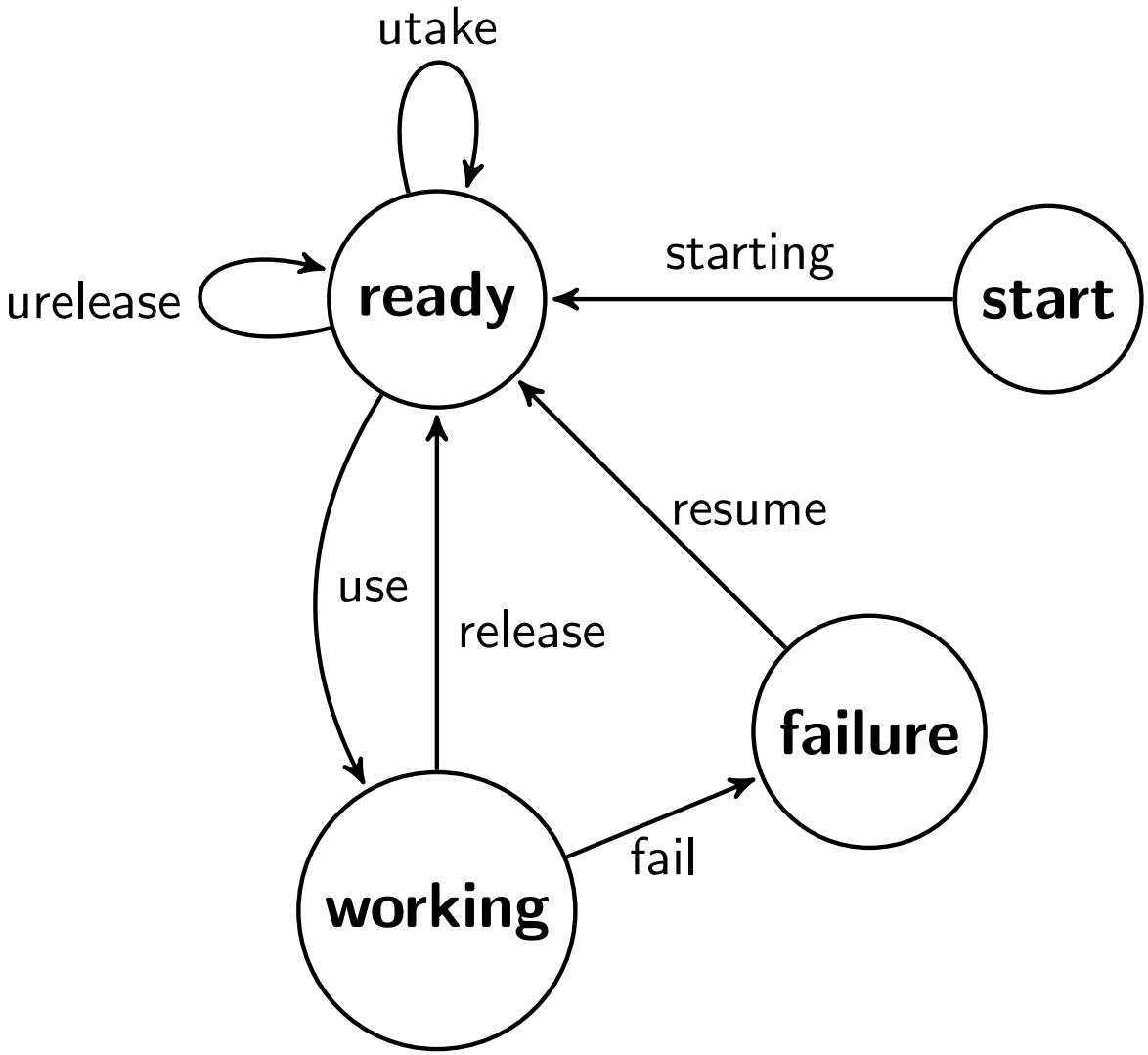
$\ell_0[Q := 0];$
$\ell_1[R := X];$
IF $\ell_5[Y > 0]$
  WHILE $\ell_2[R \geq Y]$
    $\ell_3[Q := Q{+}1];$
    $\ell_4[R := R{-}Y]$
  ENDWHILE
ELSE
  $\ell_6[skip]$
ENDIF



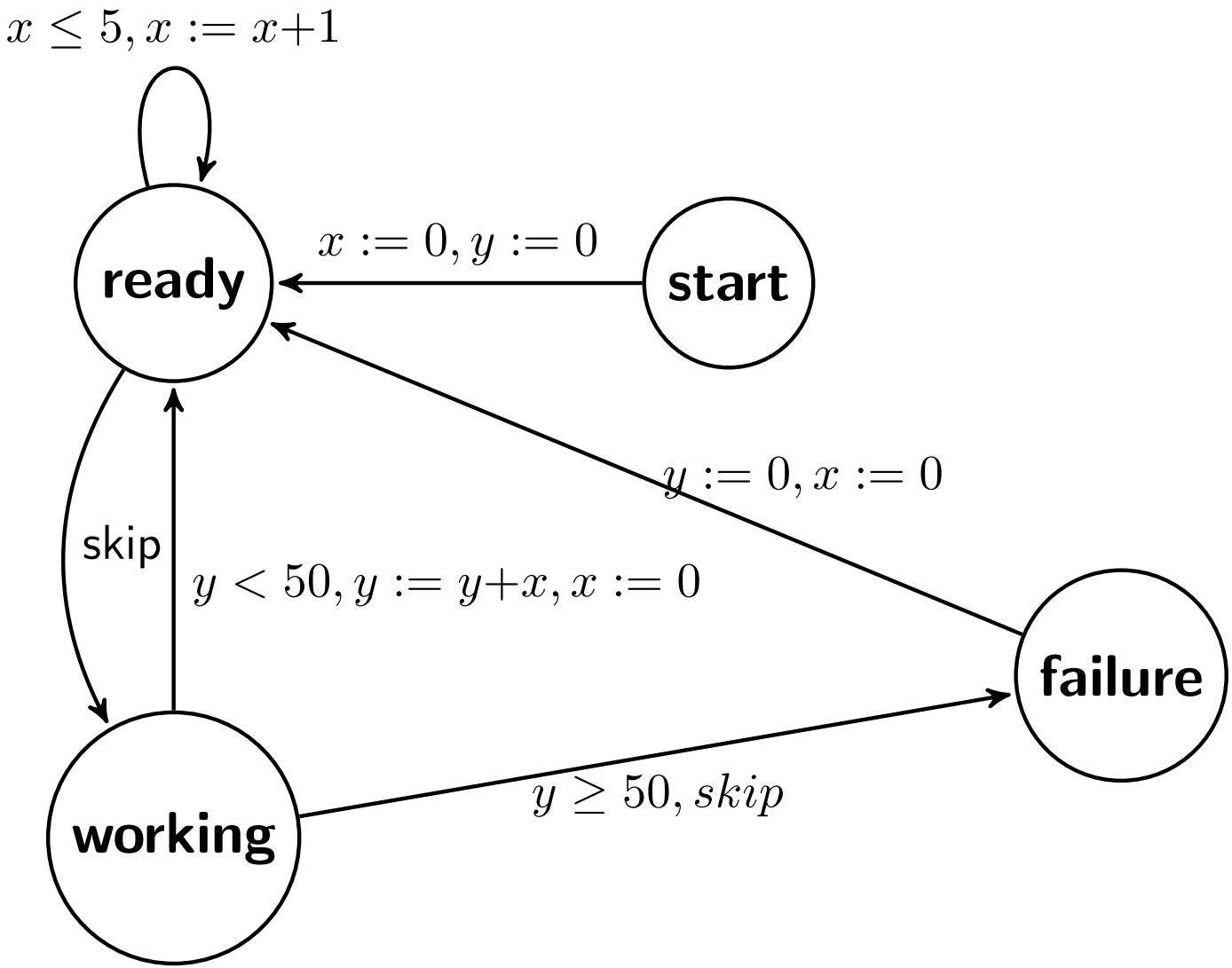
Observations

- Un automate a des états de contrôle : compteur ordinal d'un programme
- Un automate a des étiquettes : événements, actions, ...
- Un automate peut aussi avoir des variables explicites qui sont modifiées par des actions
- Un automate décrit des exécutions possibles qui sont des chemins suivant les informations de l'automate.

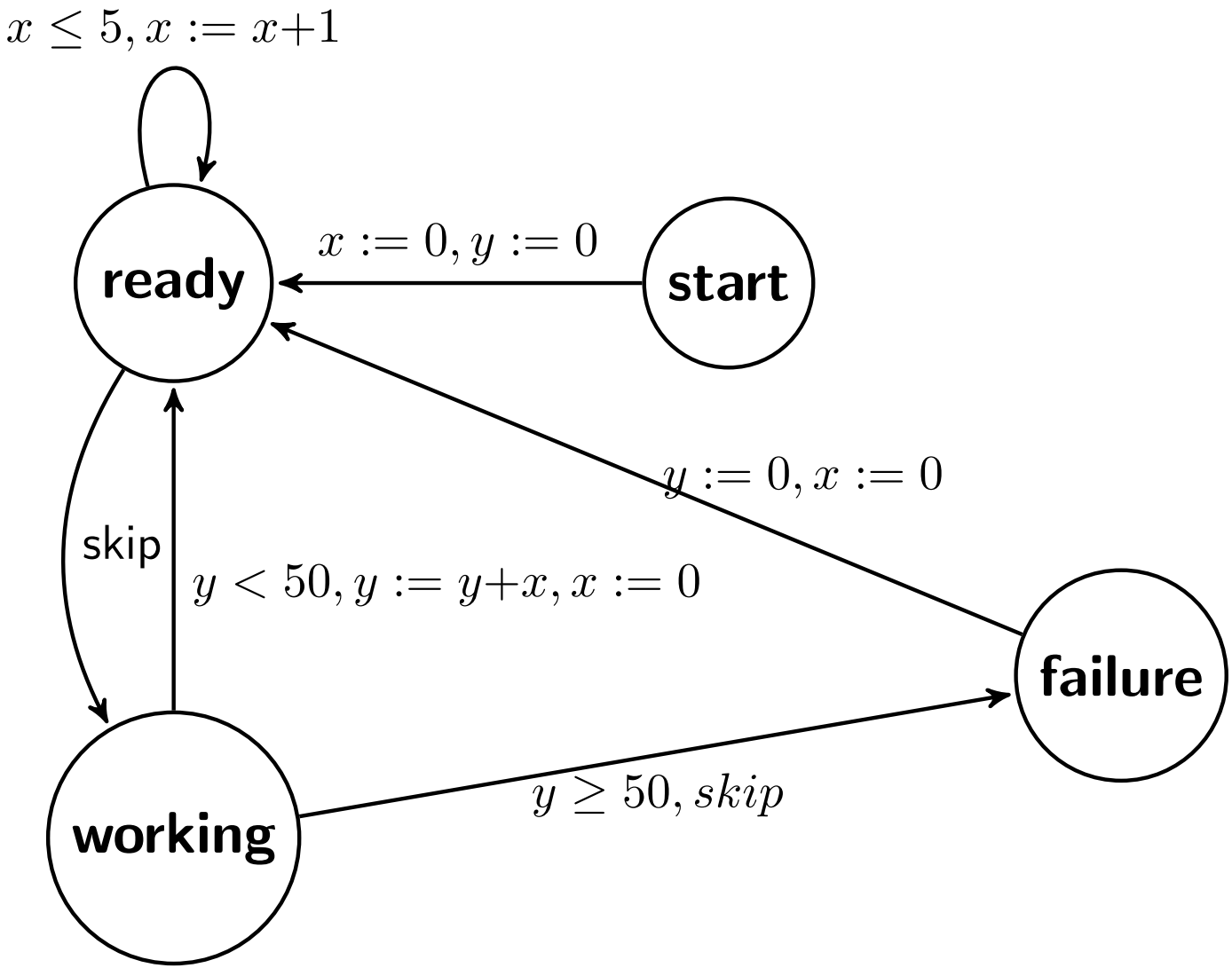
Un petit système en tant qu'automate



Un petit système en tant qu'automate

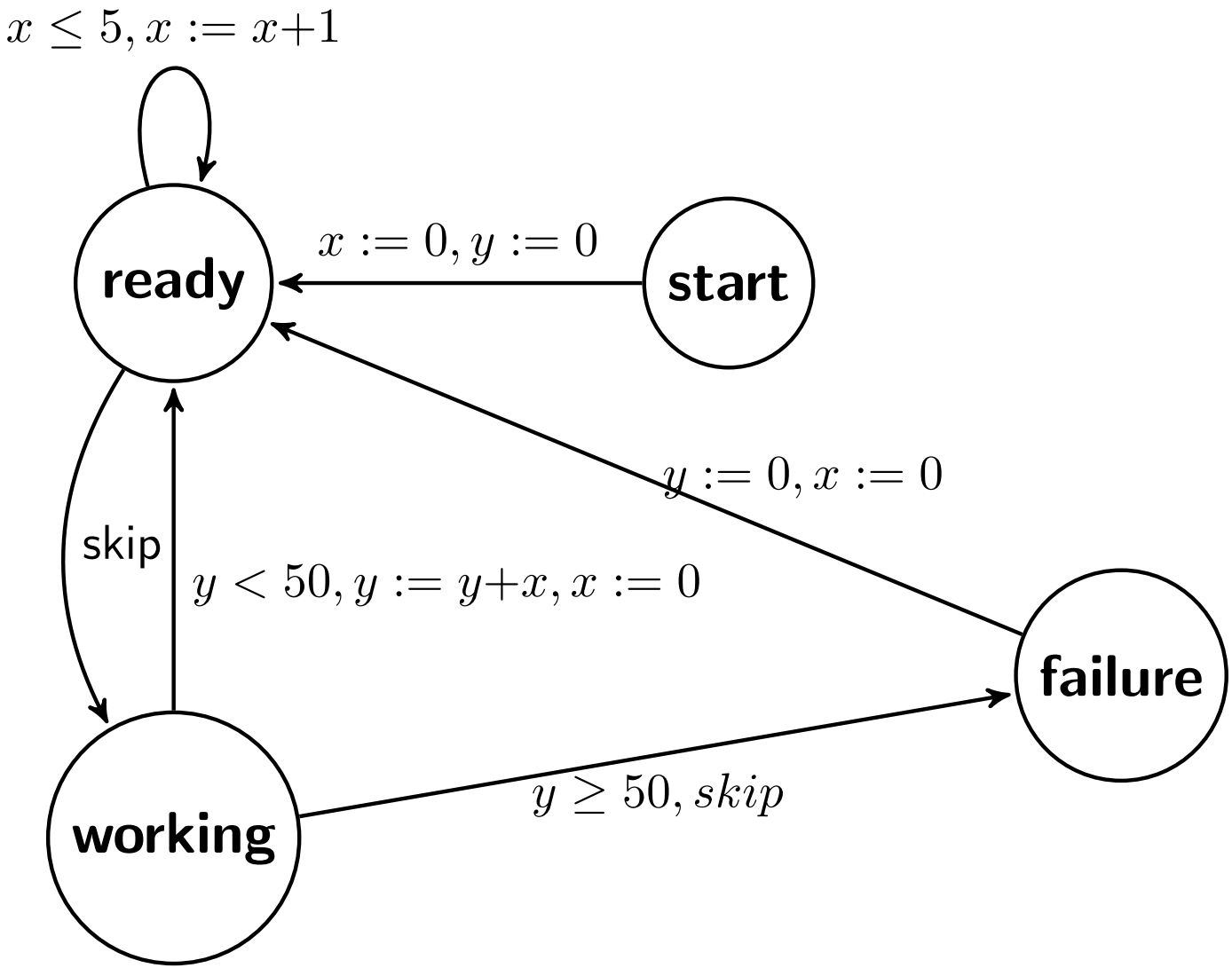


Un petit système en tant qu'automate



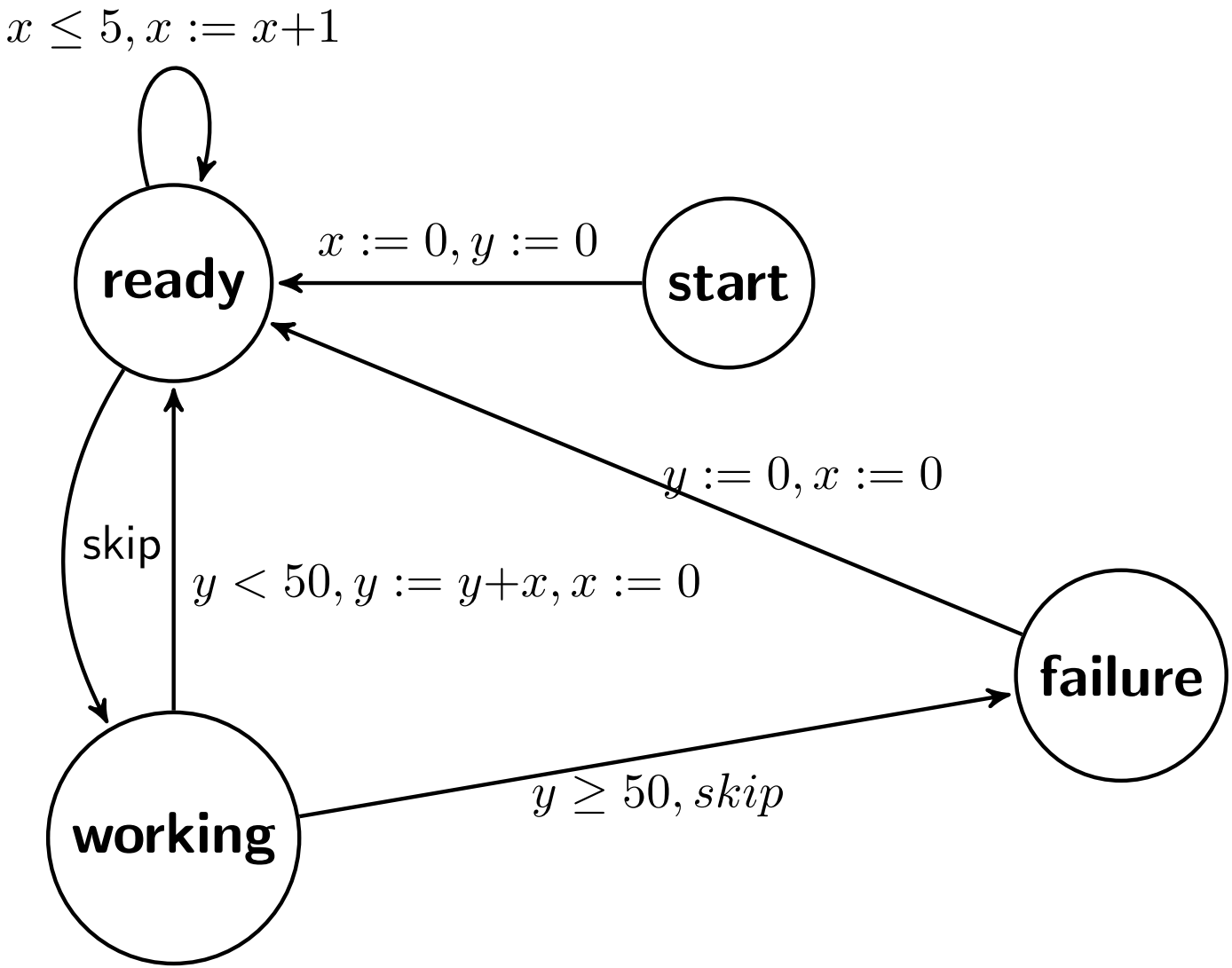
- safety1 : $0 \leq x \leq 5$

Un petit système en tant qu'automate



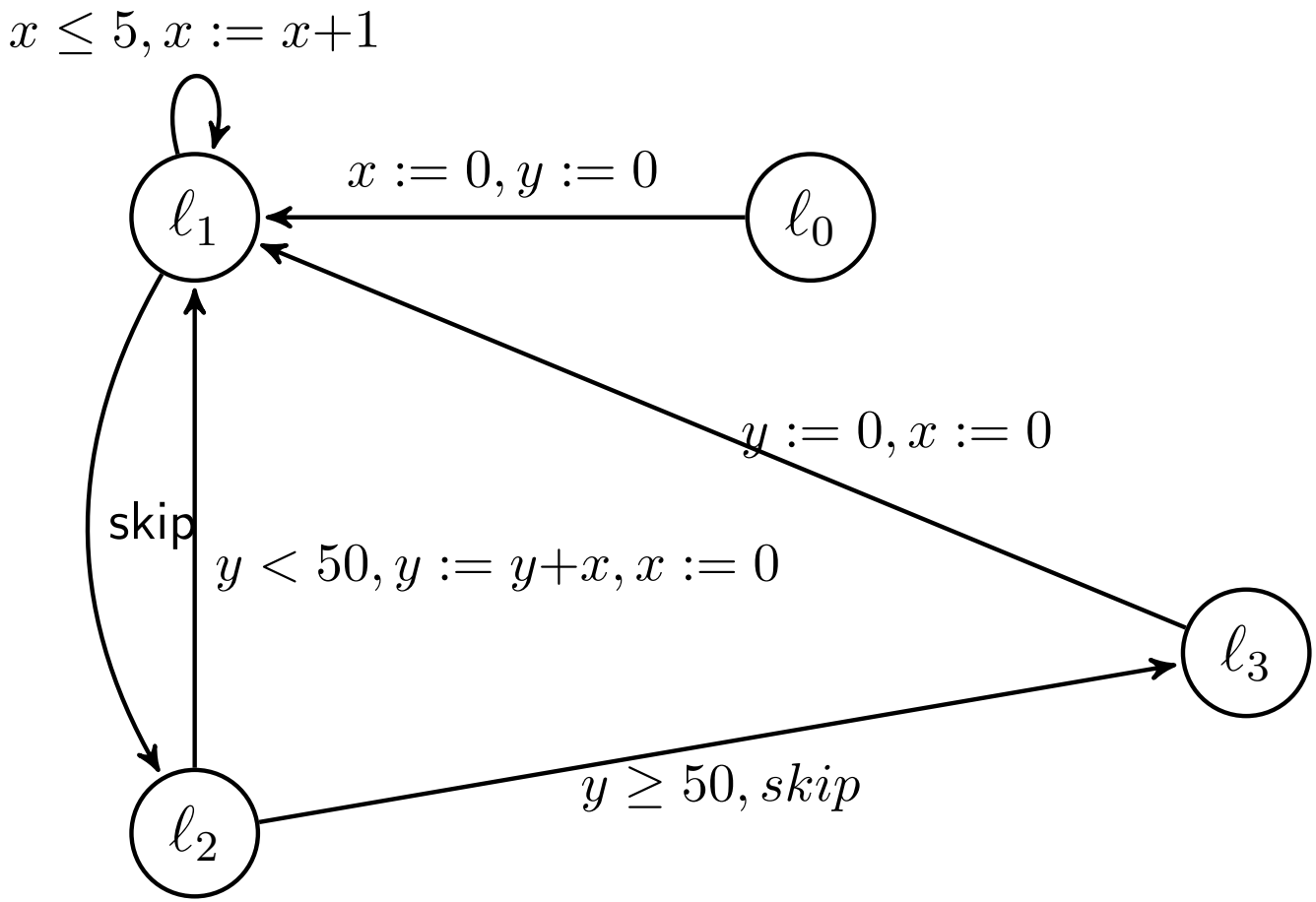
- safety1 : $0 \leq x \leq 5$ et ...

Un petit système en tant qu'automate

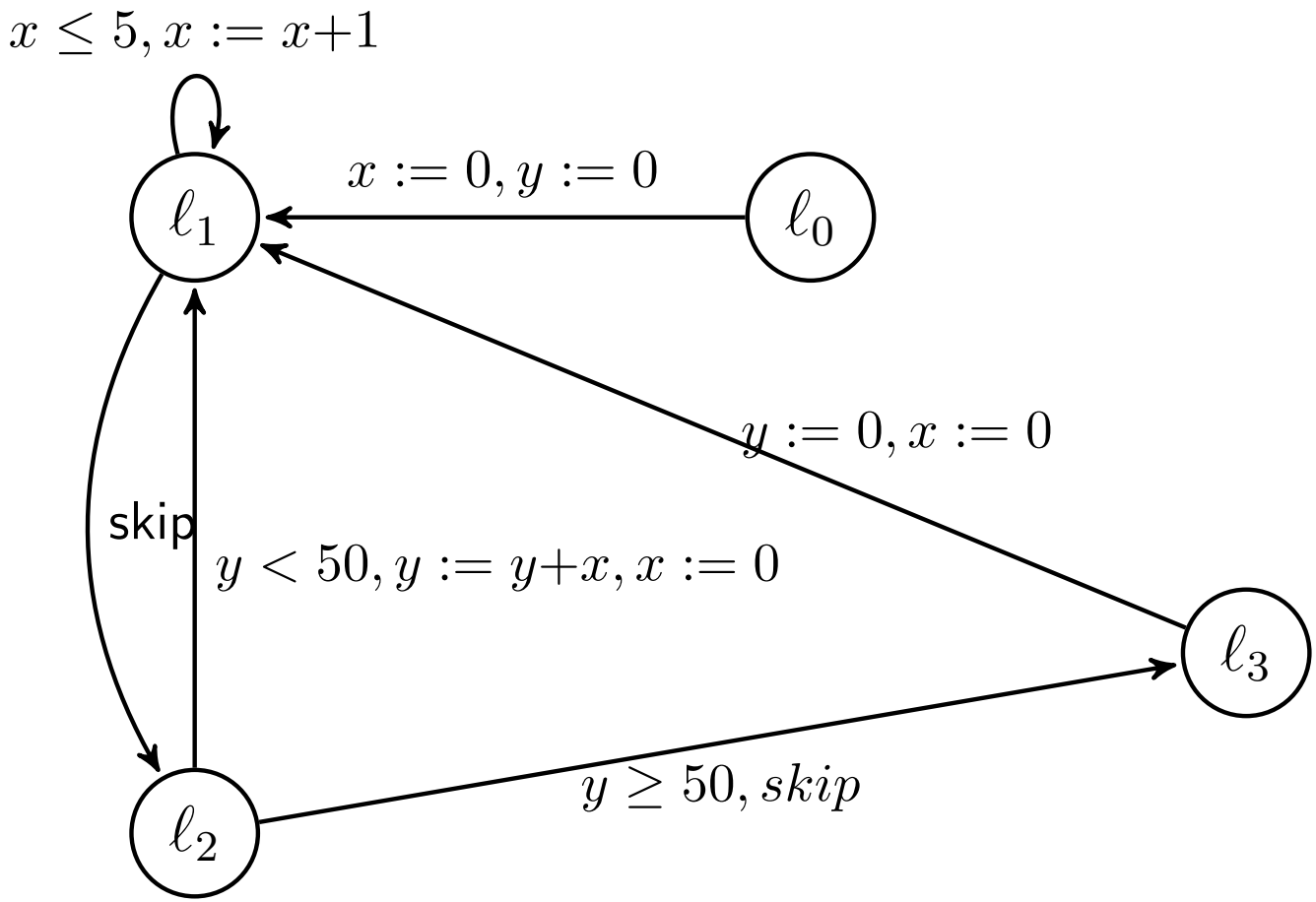


- safety1 : $0 \leq x \leq 5$ et ... safety2 : $0 \leq y \leq 56$

Un petit système en tant qu'automate



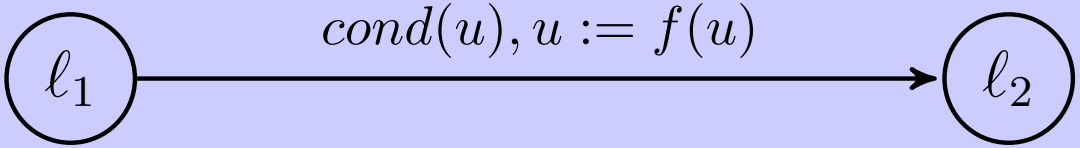
Un petit système en tant qu'automate



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

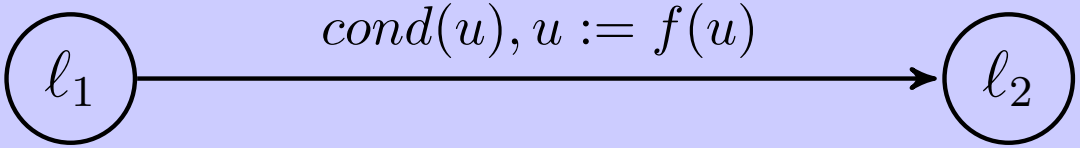
Quelques formes de transitions

Transition entre deux états de contrôle

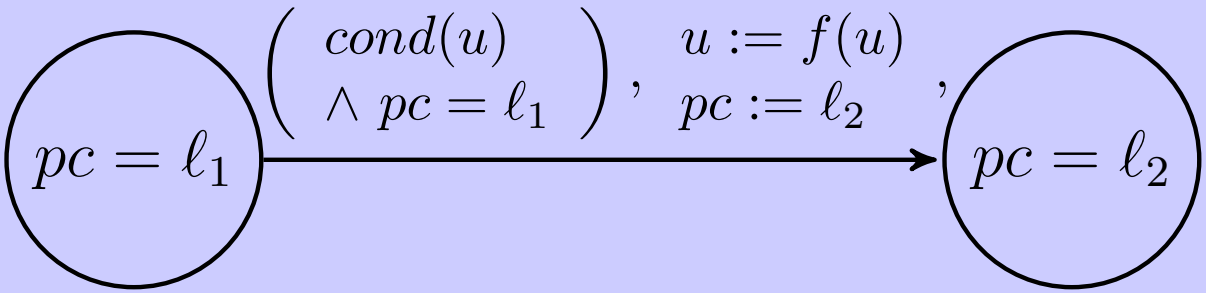


Quelques formes de transitions

Transition entre deux états de contrôle

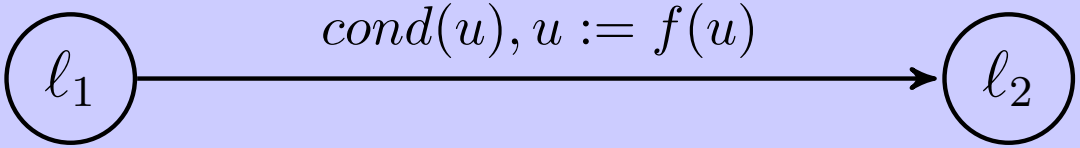


Transition entre deux états de contrôle

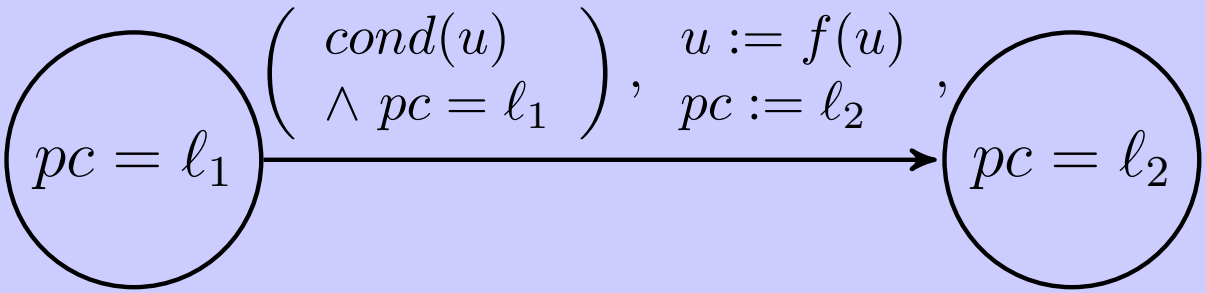


Quelques formes de transitions

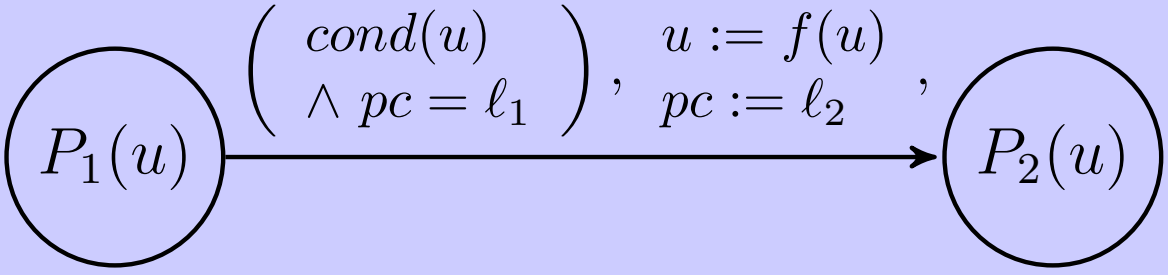
Transition entre deux états de contrôle



Transition entre deux états de contrôle



Transition entre deux prédicats



Current Summary

Current Subsection Summary

Computing GCD

MODULE *pgcd*

EXTENDS *Naturals*, *TLC*
CONSTANNOTATNTS a, b
VARIABLES x, y

$$\mathit{Init} \triangleq x = a \ \wedge \ y = b$$

$$a1 \triangleq x > y \ \wedge \ x' = x - y \ \wedge \ y' = y$$
$$a2 \triangleq x < y \ \wedge \ y' = y - x \ \wedge \ x' = x$$
$$\mathit{over} \triangleq x = y \ \wedge \ x' = x \ \wedge \ y' = y$$

$$\mathit{Next} \triangleq a_1 \ \vee \ a_2 \ \vee \ \mathit{over}$$

$$\mathit{test} \triangleq x \neq y$$

Calcul du pgcd

```
------------------ 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
===========================================
```

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 simpke access control

Première tentative

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

Exemple de modélisation TLA$^+$

Seconde tentative

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

$next_2 \triangleq entrer_2 \lor sortir$

Module for a simpke 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
======================================================================
```

Transition system in TLA/TLA$^+$

Soit $(Th(s,c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système $\mathcal{S}$. Une propriété A est une propriété de sûreté pour le système $\mathcal{S}$, si
$\forall x_0, x \in \textsc{Vals}. Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow A(x)$.

Transition system in TLA/TLA$^+$

Soit $(Th(s,c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système $\mathcal{S}$. Une propriété A est une propriété de sûreté pour le système $\mathcal{S}$, si
$\forall x_0, x \in \textsc{Vals}. Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow A(x)$.

- x est une variable ou une liste de variable : `VARIABLES x`

Transition system in TLA/TLA$^+$

Soit $(Th(s,c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système $\mathcal{S}$. Une propriété A est une propriété de sûreté pour le système $\mathcal{S}$, si
$\forall x_0, x \in \textsc{Vals}. Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow A(x)$.

- x est une variable ou une liste de variable : `VARIABLES x`
- $Init(x)$ est une variable ou une liste de variable : `init == Init(x)`

Transition system in TLA/TLA$^+$

Soit $(Th(s,c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système $\mathcal{S}$. Une propriété A est une propriété de sûreté pour le système $\mathcal{S}$, si
$\forall x_0, x \in \textsc{Vals}. Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow A(x)$.

- x est une variable ou une liste de variable : `VARIABLES x`
- $Init(x)$ est une variable ou une liste de variable : `init == Init(x)`
- $\textsc{Next}^\star(x_0, x)$ est la définition de la relation définissant ce que fait le système : `Next == a1 \/ a2 \/ .... \/ an`

Transition system in TLA/TLA$^+$

Soit $(Th(s,c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système $\mathcal{S}$. Une propriété A est une propriété de sûreté pour le système $\mathcal{S}$, si
$\forall x_0, x \in \textsc{Vals}. Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow A(x)$.

- x est une variable ou une liste de variable : `VARIABLES x`
- $Init(x)$ est une variable ou une liste de variable : `init == Init(x)`
- $\textsc{Next}^\star(x_0, x)$ est la définition de la relation définissant ce que fait le système : `Next == a1 \/ a2 \/ .... \/ an`
- $A(x)$ est une expression logique définissant une propriétét de sûreté à vérifier sur toutes les configurations du modèle : `Safety == A(x)`

Current Subsection Summary

Logique TLA et langage TLA$^+$

- TLA (Temporal Logic of Actions) sert à exprimer des formules en logique temporelle : $\Box P$ ou *toujours P*
- TLA$^+$ est un langage permettant de déclarer des constantes, des variables et des définitions :
  - `<def> == <expression>` : une définition `<def>` est la donnée d'une expression `<expreession>` qui utilise des éléments définis avant ou dans des modules qui *étendent* ce module.
  - Une variable x est soit sous la forme x soit sous la forme x' : x' est la valeur de x après.
  - Un module a un nom et rassemble des définitions et il peut être une extension d'autres modules.
  - `[f EXCEPT![i]=e]` est la fonction f où seule lavaleur en i a changé et vaut .
- Une configuration doit être définie pour évaluer une spécification

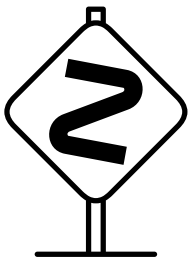
- Limitation des actions :

$$\begin{array}{l} \texttt{nom} \triangleq \\ \qquad \wedge\ cond(v, w) \\ \qquad \wedge\ v' = e(v, w) \\ \qquad \wedge\ w' = w \end{array}$$

- $e(v, w)$ doit être codable en Java.
- Modules standards : Naturals, Integers, TLC ...

Commentaires

- Téléchargez l'application le site de Microsoft pour votre ordinateur.
- Ecrivez des modèles et testez les !
- Limitations par les domaines des variables.



Permettre un raisonnement symbolique quel que soit l'ensemble des états

Current Summary

Annotation versus commentaire de programmes ou d'algorithmes

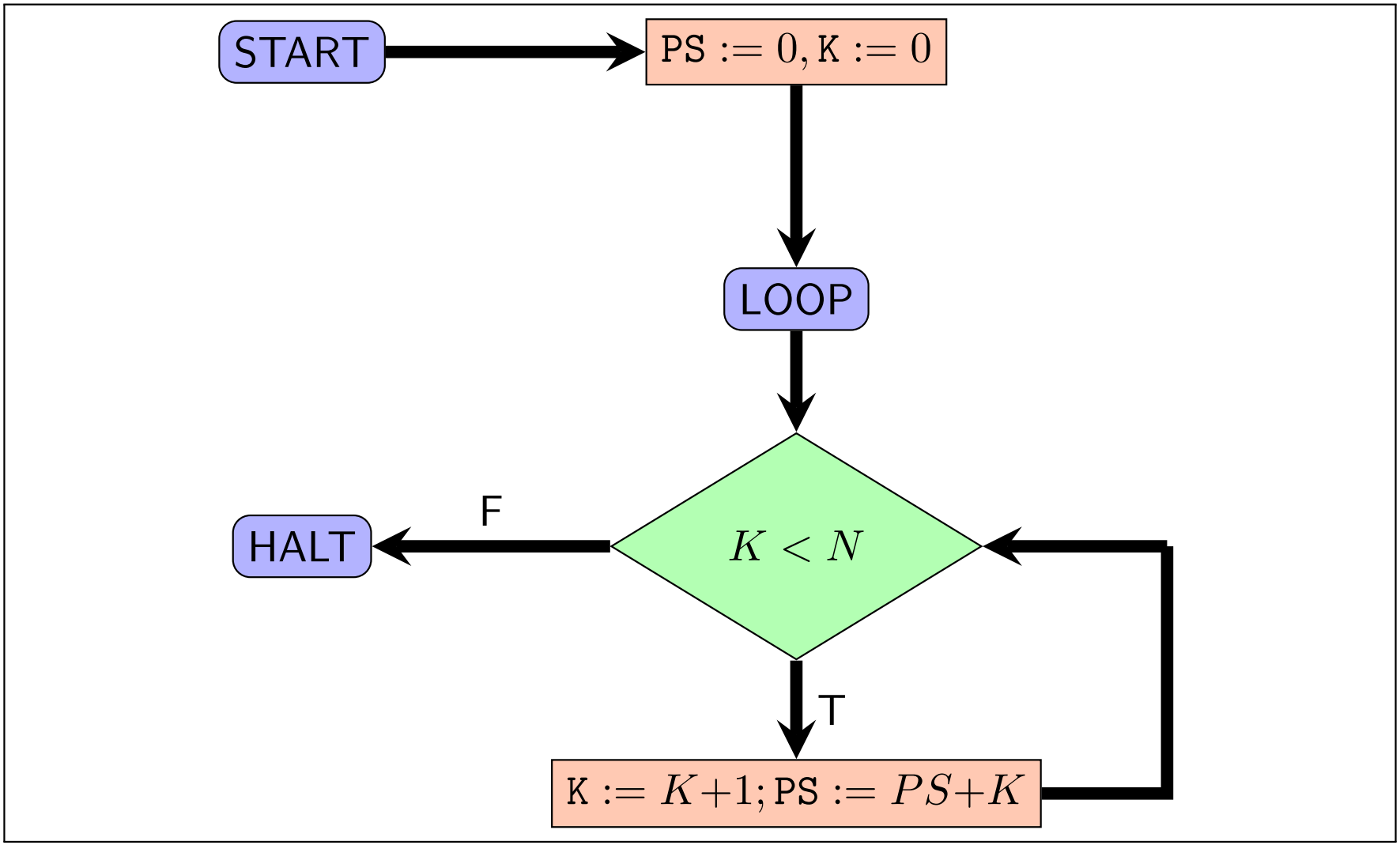
- Un programme ou un algorithme peuvent être annotés ou commentés.
- Un commentaire est une information pertinente destinée à être vue ou lue et qui a une importance relative dans l'esprit du concepteur.
- Un commentaire indique une information sur les données, sur les variables et donc sur l'état supposé du programme à l'exécution.
- Un commentaire est une annotation du texte du code qui nous permet de communiquer une information sémantique :
  - *à ce point, la variable k est plus petite sur n*
  - *l'indice e fait référence à une adresse licite de t et cette valeur est toujours positive*
  - *la somme des variables est positive*
- Les annotations peuvent être systématisées et obéir à une syntaxe spécifique définissant le langage d'annotations ;

```
/*@  assert l1:  z >= 3 && y == 3; */
z = z +y;
/*@  assert l2:  z >= 6 && y == 3; */
```

Calculer la somme des n premiers entiers (v0)

```
int fS(int n) {
  int ps = 0;
  int k = 0;
  while (k < n) {
    k = k + 1;
    ps = ps + k;
  };
  return ps;
}
```

Calculer la somme des n premiers entiers (flowchart)



Calculer la somme des n premiers entiers (v0)

```
// pre n>=0;
// post ps == n*(n+1) / 2;

int fS(int n) {
  int ps = 0;
  int k = 0;
  while (k < n) {
    k = k + 1;
    ps = ps + k;
  };
  return ps;
}


int main()
{

}
```

Calculer la somme des n premiers entiers (v0)

```
// pre n>=0;
// post ps == n*(n+1) / 2;

int fS(int n) {
  int ps = 0;
  int k = 0;
  while (k < n) {
    // ps = k*(k+1) / 2;
    k = k + 1;
    ps = ps + k;
  };
    // ps = n*(n+1) / 2;
  return ps;
}



int main()
{

}
```

Calculer la somme des n premiers entiers (v0)

```c
#include <stdio.h>

int fS(int n) {
  int ps = 0;
  int k = 0;
  while (k < n) {
    k = k + 1;
    ps = ps + k;

  };
  return ps;
}

int main()
{
  int    z = 3;
  printf("Value for z=%d is %d\n",z,fS(z));
  return 0;
}
```

Definition of S(n) and IS(n)

$\forall n \in \mathbb{N} : S(n) = \sum_{k=0}^{n} k$

$$\left[\begin{array}{l} IS(0) = 0 \\ n \geq 0, IS(n{+}1) = IS(n){+}(n{+}1) \end{array} \right.$$

Property for S(n) and IS(n)

$\forall n \in \mathbb{N} : S(n) = IS(n)$

- base 0 : $S(0) = 0 = IS(0)$
- induction $i{+}1$: $\forall j \in 0..i : S(j) = IS(j)$
  - $S(i{+}1) = \sum_{k=0}^{i+1} k$ (definition)
  - $S(i{+}1) = (\sum_{k=0}^{i} k){+}i{+}1$ (property of summation)
  - $S(i{+}1) = S(i){+}(i{+}1)$ (by definition of S(i))
  - $S(i{+}1) = IS(i){+}(i{+}1)$ (by inductive assumption on S(i) et IS(j))
  - $S(i{+}1) = IS(i){+}(i{+}1) = IS(i{+}1)$ (by defintion of IS(i+1))
- conclusion : $\forall i \in \mathbb{N} : S(i) = IS(i)$ (by induction principle)

Reformulation algorithmique du calcul de la somme des n premiers entiers

- $\forall n \in \mathbb{N} : S(n) = \sum_{k=0}^{n} k$
- $\left[\begin{array}{l} IS(0) = 0 \\ n \geq 0, IS(n{+}1) = IS(n){+}n \end{array} \right.$
- $\forall n \in \mathbb{N} : S(n) = IS(n)$
-
  - base 0 : $S(0) = 0$
  - induction $k{+}1$: $S(k{+}1) = S(k){+}i{+}1$
  - step $k{+}1$: $S(k{+}1) = S(k){+}k{+}1$
- $S(k) = ps$: current value of ps is S(k)
- $S(k{-}1) = ops$: current value of ops is $S(k{-}1)$
- step $k{+}1$: $ps = ops{+}k{+}1$

Calculer la somme des n premiers entiers (v1)

```c
#include <stdio.h>

int fS(int n) {
  int ps = 0;
  int k = 0;
  int ok=k, ops = 0;
  while (k < n) {
    ok=k;ops=ps;
    k = ok + 1;
    ps = ops + k;
  };
  return ps;
}

int main()
{
  int    z = 3;
  printf("Value for z=%d is %d\n",z,fS(z));
  return 0;
}
```

Calculer la somme des n premiers entiers (v2)

```
int fS(int n) {
  int ps = 0;
  int k = 0;
  int ok=k, ops = 0;
  while (k < n) {
 /*@ assert 0 <= k && k <= n
   && ps == S(k) && ops == S(ok);    */
     ok=k;ops=ps;
     k = ok + 1;
     ps = ops + k;
 /*@ assert 0 <= k && k <= n && ps == S(k)
   && ops == S(ok);    */
  };
  return ps;
}
```

Calculer la somme des n premiers entiers

```
/*@ axiomatic S {
  @ logic integer S(integer n);
  @ axiom S_0: S(0) == 0;
  @ axiom S_i: \forall integer i; i > 0 ==> S(i) == S(i-1)+i;
  @ } */

/*@ requires n >= 0;
   assigns \nothing ;
   ensures \result == S(n);
*/
int fS(int n) {
  int ps = 0;
  int k = 0;
  int ok=k,ops=ps;
  /*@ loop invariant 0 <= k && k <= n && ps == S(k) && ops == S(ok) ;
    loop assigns ps, k,ops,ok;
   */
  while (k < n) {
/*@ assert l0: 0 <= k && k <= n && ps == S(k) && ops == S(ok);    */
    ops=ps;ok=k;
     k = ok + 1;
    ps = ops + k;
 /*@ assert l1:  0 <= k && k <= n && ps == S(k) && ops == S(ok);    */
  };
 /*@ assert  ps == S(n);    */
  return ps;
}
```

Observations sur le calcul

- Définition des fonctions mathématiques nécessaires pour exprimer le calcul de la somme des n premiers nombres entiers.
- Expression des résultats intermédiaires appelés *sommes partielles*
- Relation entre la preuve par induction et la forme du corps de l'itération.
- Induction et calcul sont liés.

Observations sur le calcul

- Définition des fonctions mathématiques nécessaires pour exprimer le calcul de la somme des n premiers nombres entiers.
- Expression des résultats intermédiaires appelés *sommes partielles*
- Relation entre la preuve par induction et la forme du corps de l'itération.
- Induction et calcul sont liés.

$$x_0 \xrightarrow{P} x \tag{1}$$

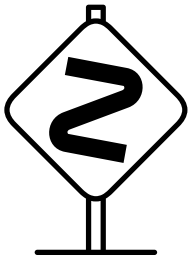
$$x_0 \xrightarrow{\star} x \tag{2}$$

$$x_0 \longrightarrow x_1 \longrightarrow \ldots \longrightarrow x \tag{3}$$

$$x_0 \longrightarrow x_1 \longrightarrow \ldots \longrightarrow x_n \xrightarrow{step} x \tag{4}$$

Current Summary

Quelques simplifications et notations



On convient des notations suivantes équivalentes : $x \in E$ est équivalent à $E(x)$ pour toute valeur $x \in$ Vals. Cette simplification permet de relier un ensemble $U \subseteq$ Vals à une assertion $U(x)$ en considérant que $U(x)$ et $x \in U$ désigne le même concept.

Les deux expressions suivantes sont équivalentes :

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

¡

- $\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow A(x)$
- $\forall x \in \textsc{Vals}.(\exists x_0.x_0 \in \textsc{Vals} \wedge Init(x_0) \wedge \textsc{Next}^\star(x_0, x)) \Rightarrow A(x).$
- $\textsc{reachable}(M) = \{u | u \in \textsc{Vals} \wedge (\exists x_0.x_0 \in \textsc{Vals} \wedge Init(x_0) \wedge \textsc{Next}^\star(x_0, u))\}$ est l'ensemble des états accessibles à partir des états initiaux et on doit montrer la propriété de sûreté $A(x)$ en montrant l'inclusion des ensembles (model-checking) :

$$\textsc{reachable}(M) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$$

Principe assertionnel d'induction

Soit $(Th(s,c), x, \textsc{Vals}, Init(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système S.
Une propriété $A(x)$ est une propriété de sûreté pour le système S, si et seulement s'il existe une propriété d'état $I(x)$, telle que :

$$\forall x, x' \in \textsc{Vals} \ : \ \begin{cases} (1)\ Init(x) \Rightarrow I(x) \\ (2)\ I(x) \Rightarrow A(x) \\ (3)\ I(x) \wedge \textsc{Next}(x, x') \Rightarrow I(x') \end{cases}$$

La propriété $I(x)$ est appelée un invariant inductif de S et est une propriété de sûreté particulière plus forte que les autres propriétés de sûreté.

Justification (correction)

Soit une propriété $I(x)$ telle que :

$$\forall x, x' \in \textsc{Vals} \ : \ \begin{cases} (1)\ Init(x) \Rightarrow I(x) \\ (2)\ I(x) \Rightarrow A(x) \\ (3)\ I(x) \wedge \textsc{Next}(x, x') \Rightarrow I(x') \end{cases}$$

Alors $A(x)$ est une propriété de sûreté pour pour le système S modélisé par M.

Soient x et $x' \in \textsc{Vals}$ tels que $INIT(x) \wedge \textsc{Next}^{\cdot}(x, x')$.

- On peut construire une suite telle que :
  $(x = x_0) \xrightarrow[\textsc{Next}]{} x_1 \xrightarrow[\textsc{Next}]{} x_2 \xrightarrow[\textsc{Next}]{} \ldots \xrightarrow[\textsc{Next}]{} (x_i = x')$.
- L'hypothèse (1) nous permet de déduire $I(x_0)$.
- L'hypothèse (3) nous permet de déduire $I(x_1)$, $I(x_2)$, $I(x_3)$, ..., $I(x_i)$. En utilisant l'hypothèse (2) pour x', nous en déduisons que x' satisfait A.

Justification (complétude)

$\forall x_0, x \cdot x, y \in \textsc{Vals} \wedge Init(x_0) \wedge x_0 \xrightarrow[\text{Next}]{\star} x \Rightarrow A(x)$

$\textsc{Prouvons que}$: il existe une propriété $I(x)$ telle que :

$$\forall x, x' \in \textsc{Vals} \ : \begin{cases} (1)\ Init(x) \Rightarrow I(x) \\ (2)\ I(x) \Rightarrow A(x) \\ (3)\ I(x) \wedge \textsc{Next}(x, x') \Rightarrow I(x') \end{cases}$$

- Nous considérons la propriété suivante :
  $I(x) \ \widehat{=}\ \exists x_0 \in \textsc{Vals} \cdot Init(x_0) \wedge x_0 \xrightarrow[\text{Next}]{\star} x.$
- $I(x)$ exprime que la valeur x est accessible à partir d'une valeur initiale x_0.
- Les trois propriétés sont simples à vérifier pour $I(x)$. $I(x)$ est appelé le plus fort invariant de l'algorithme $\mathcal{A}$.

Assertion versus relation

- 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).
- Deux types de principes sont proposés : assertionnel et relationnel.
- Nous utilisons l'expression de propriété de sûreté, alors que généralement il s'agit d'une propriété d'invariance ($\Box$ propriété) et d'invariant au lieu d'invariant inductif.

⊠ Definition(assertion)
Soit $(Th(s,c), \mathsf{X}, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système $\mathcal{S}$. Une propriété A est une propriété assertionnelle de sûreté pour le système $\mathcal{S}$, si

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

⊠ Definition(relation)
Soit $(Th(s,c), \mathsf{X}, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système $\mathcal{S}$. Une propriété R est une propriété relationnelle de sûreté pour le système $\mathcal{S}$, si

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

Complétude et correction

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

si, et seulement si,

il existe $I \in \mathcal{P}(\textsc{Vals})$

$$\forall x_0, x, x' \in \textsc{Vals} \ : \ \begin{cases} (1)\ Init(x_0) \Rightarrow I(x_0) \\ (2)\ I(x) \Rightarrow A(x) \\ (3)\ I(x) \wedge \textsc{Next}(x, x') \Rightarrow I(x') \end{cases}$$

si, et seulement si,

$$\exists i \in \mathcal{P}(\textsc{Vals}). \left[\begin{array}{ll} (1) & Init \subseteq i \\ (2) & i \subseteq A \\ (3) & \forall x, x' \in \textsc{Vals}. i(x) \wedge \textsc{Next}(x, x') \Rightarrow i(x') \end{array} \right.$$

Complétude et correction

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

si, et seulement si,

il existe $R \in \mathcal{P}(\textsc{Vals} \times \textsc{Vals})$

$$\forall x_0, x, x' \in \textsc{Vals} \ : \begin{cases} (1)\ Init(x_0) \Rightarrow R(x_0, x_0) \\ (2)\ R(x_0, x) \Rightarrow A(x_0, x) \\ (3)\ R(x_0, x) \wedge \textsc{Next}(x, x') \Rightarrow R(x_0, x') \end{cases}$$

si, et seulement si,

$\exists R \in \mathcal{P}(\textsc{Vals} \times \textsc{Vals}).$

$$\begin{bmatrix} (1)\ Init \times Init \subseteq R \\ (2)\ R \subseteq A \\ (3)\ \forall x, x' \in \textsc{Vals}. R(x_9, x) \wedge \textsc{Next}(x, x') \Rightarrow R(x_0, x') \end{bmatrix}$$

Conséquences dur les méthodes pratiques

- La propriété invariante I est définie par
  $\mathsf{I}(x) \stackrel{def}{=} \exists x_0 \in \textsc{Vals}.Init(x_0) \wedge \textsc{Next}^{\star}(x_0, x)$
- La propriété invariante R est définie par
  $\mathsf{R}(x_0, x) \stackrel{def}{=} Init(x_0) \wedge \textsc{Next}^{\star}(x_0, x)$

⊠ Definition(assertion)
Soit $(Th(s,c), \mathsf{X}, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système $\mathcal{S}$. Une propriété A est une propriété assertionnelle de sûreté pour le système $\mathcal{S}$, si

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

⊠ Definition(relation)
Soit $(Th(s,c), \mathsf{X}, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel M d'un système $\mathcal{S}$. Une propriété R est une propriété relationnelle de sûreté pour le système $\mathcal{S}$, si

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

Relation etre les deux types de propriétés assertionnelle et relationnelle

- $\forall x_0, x \in \textsc{Vals}. Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow R(x_0, x)$ (R) est une propriété relationnelle de sûreté.
- Soit $y = (x_0, x)$, $y_0 = (x_0, x_0)$, et
  $\textsc{NextR}(y, y') \stackrel{def}{=} \textsc{Next}(x, x') \wedge y = (x_0, x) \wedge y' = (x_0, x')$
- (R) est réécrit comme suit :
  $$\forall y_0, y \in \textsc{Vals}\times\textsc{Vals}. Init(x_0) \wedge y_0 = (x_0, x_0) \wedge \textsc{NextR}^\star(y_0, y) \Rightarrow R(y) \text{ (R)}$$
- Par la propriété de correction et de complétude
- il existe une propriété d'état $IR(y)$, telle que :
  $$\forall y_0, y \in \textsc{Vals}\times\textsc{Vals}. \begin{cases} (1)\ Init(x_0) \wedge y_0 = (x_0, x_0) \Rightarrow IR(y) \\ (2)\ IR(y) \Rightarrow R(y) \\ (3)\ IR(y) \wedge \textsc{NextR}(y, y') \Rightarrow IR(y') \end{cases}$$
- il existe une propriété relationnelle $IR(x_0, x)$, telle que :
  $$\forall x_0, x \in \textsc{Vals}. \begin{cases} (1)\ Init(x_0) \wedge \Rightarrow IR(x_0, x) \\ (2)\ IR(x_0, x) \Rightarrow R(x_0, x) \\ (3)\ IR(x_0, x) \wedge \textsc{Next}(x, x') \Rightarrow IR(x_0, x') \end{cases}$$

On obtient donc deux types de principes d'induction selon les propriétés de sûreté.

Complétude et correction

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

si, et seulement si,

il existe $I \in \mathcal{P}(\textsc{Vals})$

$$\forall x_0, x, x' \in \textsc{Vals} \ : \ \begin{cases} (1)\ Init(x_0) \Rightarrow I(x_0) \\ (2)\ I(x) \Rightarrow A(x) \\ (3)\ I(x) \wedge \textsc{Next}(x, x') \Rightarrow I(x') \end{cases}$$

- L'absence d'erreurs à l'exécution est caractérisée comme une propriété assertionnelle, puisqu'elle porte sur le fait qu'un état est sans erreurs à l'exécution si ls calculs sont définis en cet état. principe

Complétude et correction

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

si, et seulement si,

il existe $R \in \mathcal{P}(\textsc{Vals} \times \textsc{Vals})$

$$\forall x_0, x, x' \in \textsc{Vals} \ : \begin{cases} (1)\ Init(x_0) \Rightarrow R(x_0, x_0) \\ (2)\ R(x_0, x) \Rightarrow A(x_0, x) \\ (3)\ R(x_0, x) \wedge \textsc{Next}(x, x') \Rightarrow R(x_0, x') \end{cases}$$

- La correction partielle est caractérisée comme une relation entre l'état initial et l'état courant.

Sommaire sur les deux principes d'induction

Principe assertionnel de sûreté ou d'invariance

$$\exists I(x) \in \mathcal{P}(\textsc{Vals}).\ \left[\begin{array}{l} \forall x, x' \in \textsc{Vals}. \\ (1)\quad Init(x) \Rightarrow I(x) \\ (2)\quad I(x) \Rightarrow A(x) \\ (3)\quad I(x) \wedge \textsc{Next}(x, x') \Rightarrow I(x') \end{array}\right.$$

Principe relationnel de sûreté ou d'invariance

$$\exists IR(x_0, x) \in \mathcal{P}(\textsc{Vals} \times \textsc{Vals}).$$
$$\left[\begin{array}{l} \forall x_0, x, x' \in \textsc{Vals}. \\ (1)\quad Init(x_0) \Rightarrow IR(x_0, x_0) \\ (2)\quad IR(x_0, x) \Rightarrow R(x_0, x) \\ (3)\quad IR(x_0, x) \wedge \textsc{Next}(x, x') \Rightarrow IR(x_0, x') \end{array}\right.$$

Current Summary

Vérification du contrat : ce qui est la technique

Un programme P *remplit* un contrat (pre,post) :

- P transforme une variable x à partir d'une valeur initiale x_0 et produisant une valeur finale x_f : $x_0 \xrightarrow{P} x_f$
- x_0 satisfait pre : pre(x_0) and x_f satisfait post : post(x_0, x_f)
- pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)

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

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

- $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$
- conditions de vérification pour toutes les paires $\ell \longrightarrow \ell'$

- On considère un langage de programmation classique noté PROGRAMS
- et nous supposons que ce langage de programmation dispose de l'affectation, de la conditionnelle, de l'itération bornée, de l'itération non-bornée, de variables simples ou structurées comme les tableaux et de la définition de constantes.
- On se donne un programme P de PROGRAMS ; ce programme comprend
  - des variables notées globalement v,
  - des constantes notées globalement pc,
  - des types associés aux variables notés globalement VALS et identifiés à un ensemble de valeurs possibles des variables,
  - des instructions suivant un ordre défini par la syntaxe du langage de programmation.

Hypothèses

- on définit un ensemble de points de contrôle $\textsc{Locations}$
- pour chaque programme ou algorithme P. $\textsc{Locations}$ est un ensemble fini de valeurs et une variable cachée notée pc parcourt cet ensemble selon l'enchaînement.
- l'espace des valeurs possibles $\textsc{Vals}$ est un produit cartésien de la forme $\textsc{Locations} \times \textsc{Memory}$
- les variables x du système se décomposent en deux entités indépendantes $x = (pc, v)$ avec comme conditions $pc \in \textsc{Locations}$ et $v \in \textsc{Memory}$.

$$x = (pc, v) \wedge pc \in \textsc{Locations} \wedge v \in \textsc{Memory} \tag{5}$$

On considère un programme P annoté ; on se donne un modèle relationnel $\mathcal{MP} = (Th(s, c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ où

- $Th(s, c)$ est une théorie définissant les ensembles, les constantes et les propriétés statiques de ce programme
- x est une liste de variables flexibles et x comprend une partie contrôle et une partie mémoire.
- $\textsc{Locations} \times \textsc{Memory}$ 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' et conformes à la relation de succession $\longrightarrow$ entre les points de contrôle.
- $\textsc{Init}(x)$ définit l'ensemble des valeurs initiales de (pc_0, v) et $x = (pc_0, v)$ avec $pre(v)$ qui caractérise les valeurs initiales de v au point initial.

Hypothèses syr le calcul

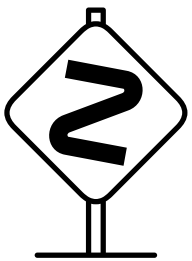
On suppose qu'il existe un graphe sur l'ensemble des valeurs de contrôle définissant la relation de flux et nous notons cette structure $(\textsc{Locations}, \longrightarrow)$.

⊠ Definition

$\ell_1 \longrightarrow \ell_2 \stackrel{def}{=} pc = \ell_1 \wedge pc' = \ell_2$

⊠ Definition(Annotation d'un point de contrôle)

Soit une structure $(\textsc{Locations}, \longrightarrow)$ et une étiquette $\ell \in \textsc{Locations}$. Une annotation d'un point de contrôle ℓ est un prédicat $P_\ell(v)$ (version assertionnelle) ou $P_\ell(v_0, v)$ (version relationnelle).



$P_\ell(v_0, v)$ exprime une relation entre la valeur initiale de V notée v_0 et v la valur courante de V au point ℓ et donc $P_\ell(v_0, v) \Rightarrow pre(v_0)$ précise que v_0 est une valeur initiale.

Relation entre annotation et invariant

- Les étiquettes ℓ appartiennent à $\textsc{Locations}$: $\ell \in \textsc{Locations}$.
- Les variables v appartiennent à $\textsc{Memory}$= $v \in \textsc{Memory}$.
- $pre(v_0)$ spécifie les valeurs initiales de v.
- Chaque fois que le contrôle est en ℓ, v satisfait $P_\ell(v)$: $pc = \ell \Rightarrow P_\ell(v_0, v)$.
- A tout état (ℓ, v) du programme, la propriété suivante est vraie mais doit être prouvée :

$$J(\ell_0, v_0, pc, v) \stackrel{def}{=} \left[\begin{array}{l} \wedge\ pc \in \textsc{Locations} \\ \wedge\ v \in \textsc{Memory} \\ \ldots \\ \wedge\ pc = \ell \Rightarrow P_\ell(v_0, v) \\ \ldots \end{array}\right.$$

- $J(\ell_0, v_0, pc, v)$ est un invariant construit à partir des annotations produites mais il faut montrer que cet invariant permet de vérifier les trois conditions du principe d'induction.

ℓ_0 désigne l'étiquette marquant le début de l'algorithrme et ℓ_f est la fin du programme. On pourra utiliser simplement 0 et f.

Vérification à faire

$$x = (pc, v) \text{ et } J(\ell_0, v_0, pc, v) \stackrel{def}{=} \left[\begin{array}{l} \wedge\ pc \in \textsc{Locations} \\ \wedge\ v \in \textsc{Memory} \\ \ldots \\ \wedge\ pc = \ell \Rightarrow P_\ell(v_0, v) \\ \ldots \end{array} \right.$$

Soit $(Th(s,c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel pour ce programme. Une propriété $\mathrm{A}(x_0, x)$ est une propriété de sûreté pour P, si $\forall x_0, x \in \textsc{Locations} \times \textsc{Memory}. Init(x_0) \wedge x_0 \xrightarrow[\textsc{Next}]{} x \Rightarrow A(x)$.

On sait que cette propriété implique qu'il existe une propriété d'état $\mathrm{I}(x_0, x)$ telle que les trois propriétés sont vérifiées mais on applique cette vérification pour J :

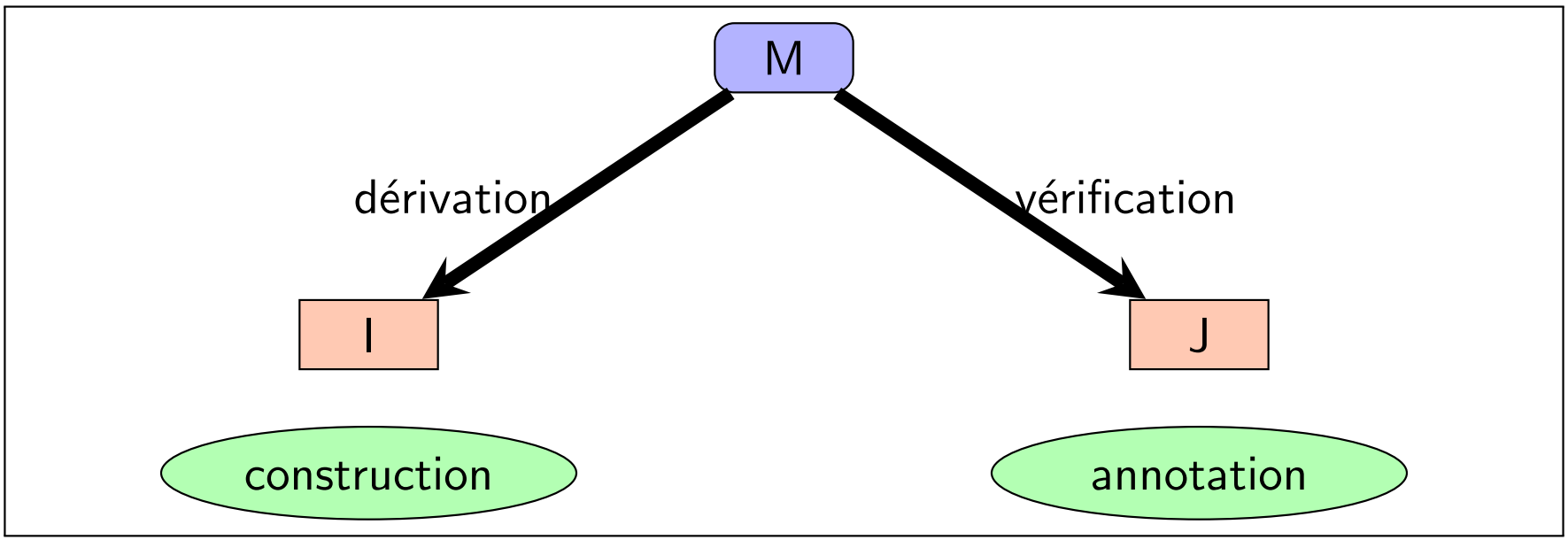
$\forall x_0, x, x' \in \textsc{Locations} \times \textsc{Memory}$:

$$\left\{ \begin{array}{ll} (1) & \textsc{Init}(x_0) \Rightarrow \mathrm{J}(x_0, x_0) \\ (2) & \mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x) \\ (3) & \forall i \in \{0, \ldots, n\} : \mathrm{J}(x_0, x) \wedge x\ r_i\ x' \Rightarrow \mathrm{J}(x_0, x') \end{array} \right.$$

$\forall i \in \{0, \ldots, n\} : \mathbf{J}(x_0, x) \wedge x\ r_i\ x' \Rightarrow \mathbf{J}(x_0, x')$ **est équivalent à** $\mathbf{J}(x_0, x) \wedge (\exists i \in \{0, \ldots, n\} : x\ r_i\ x') \Rightarrow \mathbf{J}(x_0, x')$

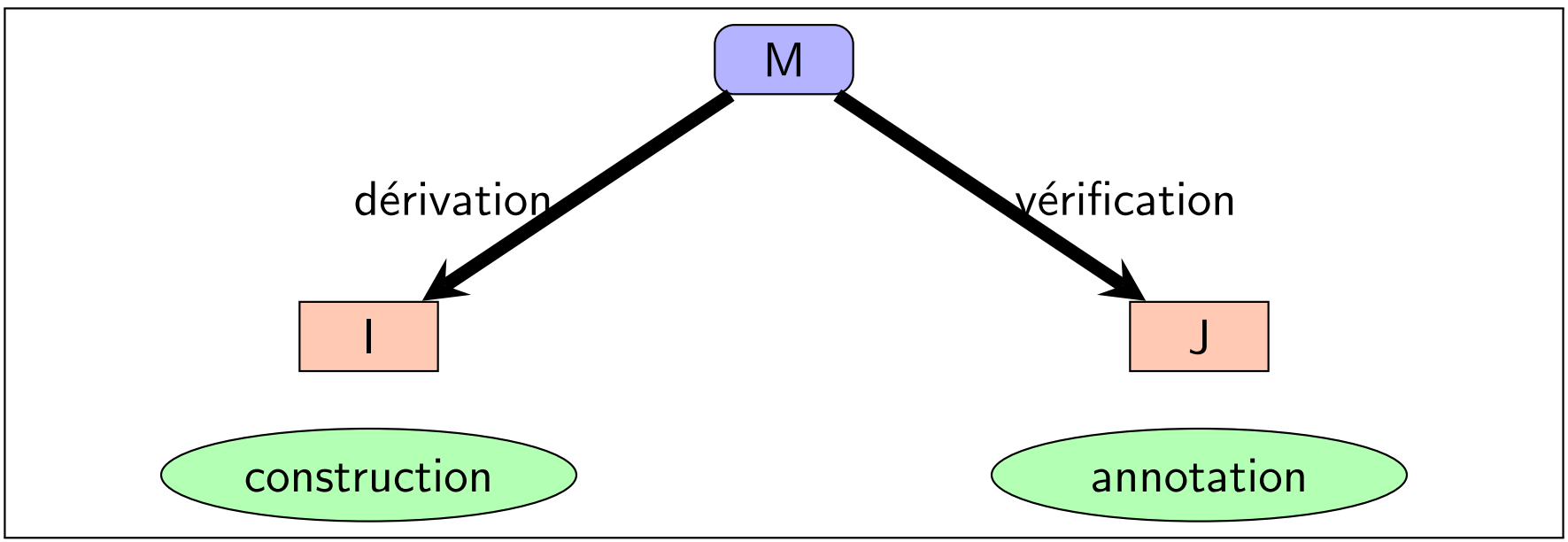
Vérification à faire

- Application de la correction du principe relationnel d'induction : si on vérifie les trois propriétés, alors A est une propriété de sûreté pour le modèle en question (vérification).
- Si on veut montrer que A est une propriété de sûreté, alors on doit utiliser l'invariant pour construire des annotations pour le modèle (dérivation).



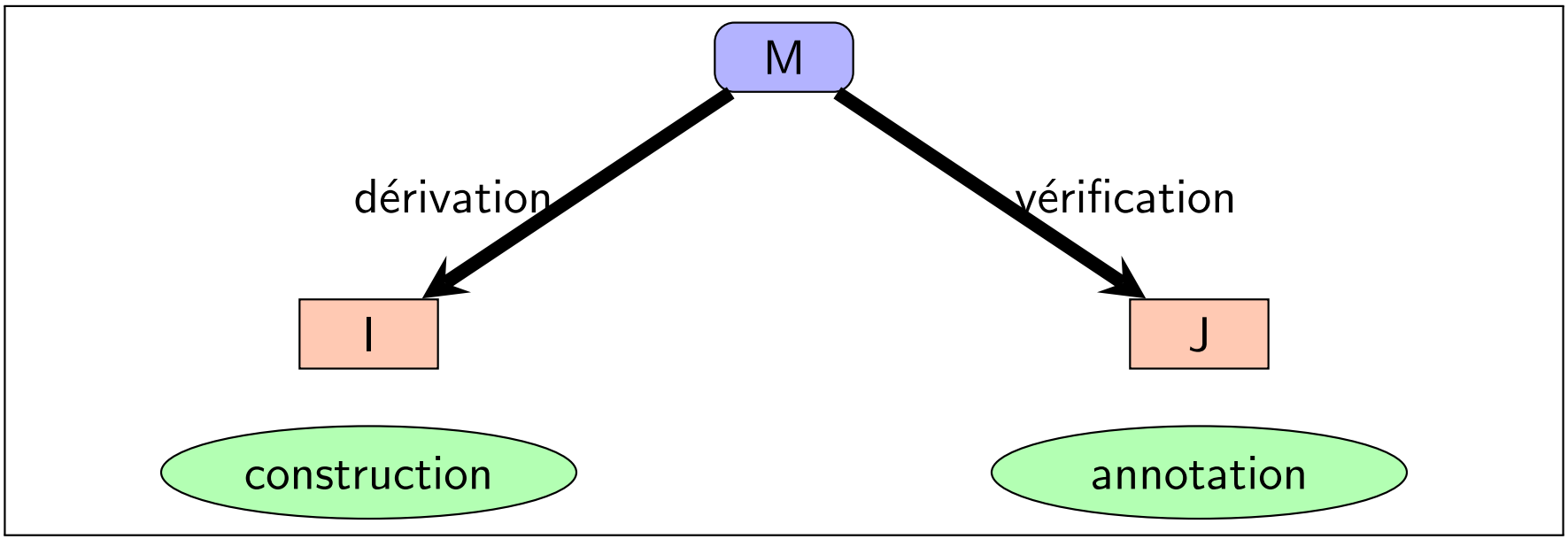
Utilisation du principe relationnel d'induction (RI)

- $\textsc{Vals} = \textsc{Locations} \times \textsc{Memory}$
- $J(pc_0, v_0, pc, v) \stackrel{def}{=} \exists x_0, x \in \textsc{Vals}. I(x_0, x) \wedge x = (pc, v) \wedge x_0 = (pc_0, v_0)$ (deduction)
- $I(x_0, x) \stackrel{def}{=} \exists pc_0, pc \in \textsc{Locations}, v_0, v \in \textsc{Memory}. J(pc_0, v_0, pc, v) \wedge x = (pc, v) \wedge x_0 = (pc_0, v_0)$ (induction)



Utilisation du principe assertionnel d'induction (AI)

- $\textsc{Vals} = \textsc{Locations} \times \textsc{Memory}$
- $J(pc, v) \stackrel{def}{=} \exists x \in \textsc{Vals}. I(x) \wedge x = (pc, v)$ (deduction)
- $I(x) \stackrel{def}{=} \exists pc \in \textsc{Locations}, v \in \textsc{Memory}. J(pc, v) \wedge x = (pc, v)$ (induction)



Adaptation aux programmes

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

Relations et définitions

$x = (\ell, v)$, $x_0 = (\ell_0, v_0)$, $I(x)$, $IR(x_0, x)$ et les annotations $P_\ell(v)$, $RP_\ell(v_0, v)$ sont liées ainsi :

- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge I(x))$
- $IR(x_0, x) \stackrel{def}{=} \exists \ell, v, v_0(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge RP_\ell(v_0, v))$
- $RP_\ell(v_0, v) \stackrel{def}{=} \exists x, x_0.(x, x_0 \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge IR(x_0, x))$

La transformation est fondée la relation de transition définie pour chaque couple d'étiquettes de contrôle qui se suivent est exprimée très simplement par la forme relationnelle suivante :

$$x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$$

Transition d'une étiquette vers une autre étiquette

- La transition de ℓ à ℓ' est possible, quand la condition $cond_{\ell,\ell'}(v)$ est vraie pour V et quand le contrôle est en ℓ ($pc = \ell$).
- Quand la transition est observée, les variables V sont transformées comme suit $v' = f_{\ell,\ell'}(v)$.
- La définition de la transition n'exprime aucune hypothèse liée à une stratégie d'exécution comme l'équité par exemple.
- $cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v)$ est une expression où les expressions $cond_{\ell,\ell'}(v)$ et $= f_{\ell,\ell'}(v)$ posent des questions de définition :
  - $DOM(\ell,\ell')(v) \stackrel{def}{=} DEF(cond_{\ell,\ell'}(v))(v) \wedge DEF(f_{\ell,\ell'}(v))$
  - $DEF(E(X))(x)$,signifie que l'expression $E(X)$ est définie pour x la valeur courante de X.
- Certaines transitions peuvent conduire à des catastrophes :
  - $DEF(X{+}1)(x) \stackrel{def}{=} x{+}1 \in D$ où D est le domaine de codage de X par exemple $D = -2^{31}|..2^{31}-1$ pour un codage sur 32 bits.
  - $DEF(T(I{+}1) < V)(t,x,v) \stackrel{def}{=} i{+}1 \in dom(t) \wedge v \in D \wedge t(i{+}1) \in D$

Relations pour des instructions de programmes

$$\ell : P_\ell(v_0, v)$$
$$V := f_{\ell,\ell'}(V)$$
$$\ell' : P_{\ell'}(v_0, v)$$

$$\ell_1 : P_{\ell_1}(v_0, v)$$
WHILE $B(V)$ **DO**
$$\ell_2 : P_{\ell_2}(v_0, v)$$
$$\ldots$$
$$\ell_3 : P_{\ell_3}(v_0, v)$$
END
$$\ell_4 : P_{\ell_4}(v_0, v)$$

Traduction

- $(pc = \ell \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $cond_{\ell,\ell'}(v) \stackrel{def}{=} TRUE$

Traduction

$$\begin{cases} pc = \ell_1 \wedge b(v) \wedge v' = v \wedge pc' = \ell_2 \\ pc = \ell_1 \wedge \neg b(v) \wedge v' = v \wedge pc' = \ell_4 \\ pc = \ell_3 \wedge b(v) \wedge v' = v \wedge pc' = \ell_2 \\ pc = \ell_3 \wedge \neg b(v) \wedge v' = v \wedge pc' = \ell_4 \end{cases}$$

Vérification du contrat : ce qui sera la technique

Un programme P *remplit* un contrat (pre,post) :

- P transforme une variable v à partir d'une valeur initiale v_0 et produisant une valeur finale v_f : $v_0 \xrightarrow{P} v_f$
- v_0 satisfait pre : pre(v_0) and v_f satisfait post : post(v_0, v_f)
- pre$(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow$ post(v_0, v_f)

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables V
begin
$0 : P_0(v_0, v)$
$\texttt{instruction}_0$
$\ldots$
$i : P_i(v_0, v)$
$\ldots$
$\texttt{instruction}_{f-1}$
$f : P_f(v_0, v)$
end

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- conditions sur les transitions ℓ, ℓ' à définir à partir des principes d'induction.

Vérification du contrat : un exemple simple

variables U, V
requires $u_0, v_0 \in \mathbb{N}$
ensures $u_f + v_f = u_0 + v_0$

begin
$0 : u = u_0 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N}$
$U := U + 2$
$1 : u = u_0 + 2 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N}$
$V := V - 2$
$2 : u = u_0 + 2 \wedge v = v_0 - 2 \wedge u_0, v_0 \in \mathbb{N}$
end

Vérification à faire

$$x = (pc, v) et J(\ell_0, v_0, pc, v) \stackrel{def}{=} \left[\begin{array}{l} \wedge\ pc \in \textsc{Locations} \\ \wedge\ v \in \textsc{Memory} \\ \ldots \\ \wedge\ pc = \ell \Rightarrow P_\ell(v_0, v) \\ \ldots \end{array} \right.$$

Soit $(Th(s,c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel pour ce programme. Une propriété $\mathrm{A}(x)$ est une propriété de sûreté pour P, si $\forall x_0, x \in \textsc{Locations} \times \textsc{Memory}. Init(x_0) \wedge x_0 \xrightarrow[\textsc{Next}]{} x \Rightarrow A(x)$. On sait que cette propriété implique qu'il existe une propriété d'état $\mathrm{I}(x)$ telle que les trois propriétés sont vérifiées mais on applique cette vérification pour J :

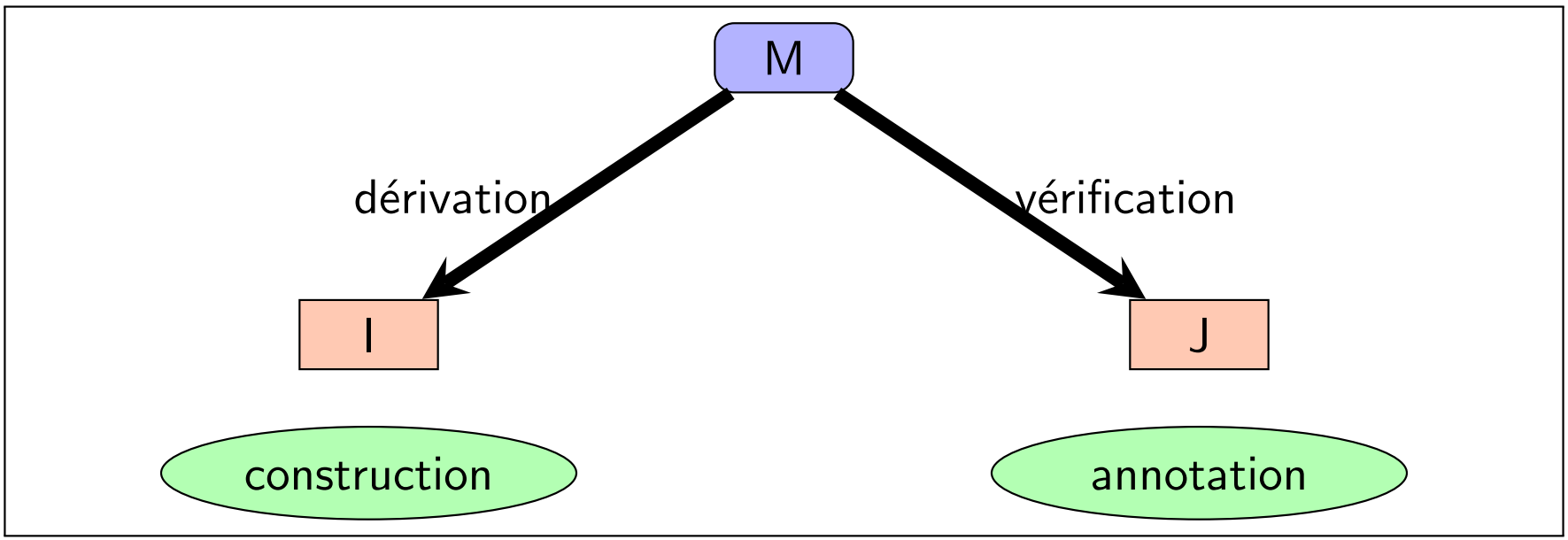
$\forall x_0, x, x' \in \textsc{Locations} \times \textsc{Memory}$:

$$\left\{ \begin{array}{ll} (1) & \textsc{Init}(x_0) \Rightarrow \mathrm{J}(x_0, x_0) \\ (2) & \mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x) \\ (3) & \forall i \in \{0, \ldots, n\} : \mathrm{J}(x_0, x) \wedge x\ r_i\ x' \Rightarrow \mathrm{J}(x_0, x') \end{array} \right.$$

- $\forall i \in \{0, \ldots, n\} : \mathrm{J}(x) \wedge x\ r_i\ x' \Rightarrow \mathrm{J}(x')$ est équivalent à $\mathrm{J}(x) \wedge (\exists i \in \{0, \ldots, n\} : x\ r_i\ x') \Rightarrow \mathrm{J}(x')$

Vérification à faire

- Application de la correction du principe relationnel d'induction : si on vérifie les trois propriétés, alors A est une propriété de sûreté pour le modèle en question (vérification).
- Si on veut montrer que A est une propriété de sûreté, alors on doit utiliser l'invariant pour construire des annotations pour le modèle (dérivation).



Vérification à faire : retour sur l'exemple

- $x = (pc, u, v)$
- $J(0, u_0, v_0, pc, u, v) \stackrel{def}{=}$
$$\left[\begin{array}{l} \wedge\ pc \in \{0,1,2\} \\ \wedge\ u, v \in \mathbb{Z} \\ \wedge\ pc = 0 \Rightarrow u = u_0 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \\ \wedge\ pc = 1 \Rightarrow u = u_0{+}2 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \\ \wedge\ pc = 2 \Rightarrow u = u_0{+}2 \wedge v = v_0{-}2 \wedge u_0, v_0 \in \mathbb{N} \end{array}\right.$$
- $A(0, u_0, v_0, pc, u, v) \stackrel{def}{=} (pc = 2 \Rightarrow u{+}v = u_0{+}v_0{-}2 \wedge u_0, v_0 \in \mathbb{N})$

$\forall pc, u, v, pc', u', v' \in \{0,1,2\}\times\mathbb{Z}$:
$$\left\{\begin{array}{ll} (1) & \textsc{Init}(0, u_0, v_0) \Rightarrow \mathrm{J}(0, u_0, v_0, 0, u_0, v_0) \\ (2) & \mathrm{J}(0, u_0, v_0, pc, u, v) \Rightarrow \mathrm{A}(0, u_0, v_0, pc, u, v) \\ (3) & \forall i \in \{0, \ldots, n\} : \mathrm{J}(0, u_0, v_0, pc, u, v) \wedge x\ r_i\ pc', u', v' \Rightarrow \mathrm{J}(0, u_0, v_0, pc', \end{array}\right.$$

Vérification sur l'exemple

1. $\textsc{Init}(0, u_0, v_0,) \Rightarrow \mathrm{J}(0, u_0, v_0, 0, u_0, v_0)$:
   $pc = 0 \wedge u = u_0 \wedge v = v_0 \wedge u_0, v_0 \in \mathbb{N} \Rightarrow \mathrm{J}(0, u_0, v_0, 0, u_0, v_0)$:
2. $\mathrm{J}(0, u_0, v_0, pc, u, v) \Rightarrow \mathrm{A}(0, u_0, v_0, pc, u, v)$
   $\mathrm{J}(pc, u, v) \Rightarrow (pc = 2 \Rightarrow u{+}v = u_0{+}v_0{-}2 \wedge u_0, v_0 \in \mathbb{N})$
3. $\forall i \in \{0, \ldots, n\} : \mathrm{J}(0, u_0, v_0, pc, u, v) \wedge x\ r_i\ pc', u', v' \Rightarrow$
   $\mathrm{J}(0, u_0, v_0, pc', u', v')$

$$\left[\begin{array}{l} r01(pc, u, v, pc', u', v') \stackrel{def}{=} pc = 0 \wedge u' = u{+}2 \wedge pc' = 1 \wedge v' = v \\ r12(pc, u, v, pc', u', v') \stackrel{def}{=} pc = 1 \wedge v' = v{-}2 \wedge pc' = 2 \wedge u' = u \end{array}\right.$$

- $J(0, u_0, v_0, pc, u, v) \wedge r01(pc, u, v, pc', u', v') \Rightarrow J(pc', u', v')$
- $J(0, u_0, v_0, pc, u, v) \wedge r12(pc, u, v, pc', u', v') \Rightarrow J(0, u_0, v_0, pc', u', v')$

Quelques règles de calcul logique

lification 1 $[A \wedge (A \Rightarrow B)] \longrightarrow [A \wedge B]$

lification 2 $[A \wedge (B = C) \wedge D \Rightarrow E \wedge (B = F) \wedge G] \longrightarrow [A \wedge (B = C) \wedge D \Rightarrow E \wedge (C = F) \wedge G]$

lification 3 $[A \wedge (B = C) \wedge D \Rightarrow E \wedge (F = F) \wedge G] \longrightarrow [A \wedge (B = C) \wedge D \Rightarrow E \wedge TRUE \wedge G]$

lification 4 $[A \Rightarrow B \wedge TRUE \wedge C] \longrightarrow [A \Rightarrow B \wedge C]$

▶

lification 5 $[A \wedge (B = C \Rightarrow U) \wedge (B = D \wedge B = C \Rightarrow V) wedge C \neq D \wedge E] \longrightarrow [A \wedge B = C \wedge U \wedge C \neq D \wedge E]$

$J(pc,u,v) \wedge r01(pc,u,v,pc',u',v') \Rightarrow J(pc',u',v')$ **(1)**

$$\left(\begin{array}{l} \wedge\ pc \in \{0,1,2\} \\ \wedge\ u,v \in \mathbb{Z} \\ \wedge\ pc=0 \Rightarrow u=u_0 \wedge v=v_0 \wedge u_0,v_0 \in \mathbb{N} \\ \wedge\ pc=1 \Rightarrow u=u_0{+}2 \wedge v=v_0 \wedge u_0,v_0 \in \mathbb{N} \\ \wedge\ pc=2 \Rightarrow u=u_0{+}2 \wedge v=v_0{-}2 \wedge u_0,v_0 \in \mathbb{N} \end{array}\right) \wedge \left(\begin{array}{l} \wedge pc=0 \\ \wedge u'=u{+}2 \\ \wedge pc'=1 \\ \wedge v'=v \end{array}\right)$$

$$\Rightarrow$$

$$\left(\begin{array}{l} \wedge\ pc' \in \{0,1,2\} \\ \wedge\ u',v' \in \mathbb{Z} \\ \wedge\ pc'=0 \Rightarrow u'=u_0 \wedge v'=v_0 \wedge u_0,v_0 \in \mathbb{N} \\ \wedge\ pc'=1 \Rightarrow u'=u_0{+}2 \wedge v'=v_0 \wedge u_0,v_0 \in \mathbb{N} \\ \wedge\ pc'=2 \Rightarrow u'=u_0{+}2 \wedge v'=v_0{-}2 \wedge u_0,v_0 \in \mathbb{N} \end{array}\right)$$

Utilisation de TLA Toolbox pour vérifier ces éléments : cours1.tla

Initialisation

Le modèle relationnel $M(P)$ pour le programme P annoté est donc défini comme suit :
$M(P) \stackrel{def}{=}$
$(Th(s,c),(pc,v),\text{LOCATIONS}\times\text{MEMORY},Init(\ell,v),\{r_{\ell,\ell'}|\ell,\ell' \in \text{LOCATIONS} \wedge \ell \longrightarrow \ell'\})$.
La définition de $Init(x)$ est dépendante de la précondition de P :

$$Init(x) \stackrel{def}{=} .x = (\ell_0, v) \wedge \textbf{pre}(\text{P})(v).$$

Conditions initiales

Les deux propriétés suivantes sont équivalentes :

- $\forall x_0 \in \text{VALS} : \textit{Init}(x_0) \Rightarrow \text{J}(x_0, x_0)$
- $\forall v \in \text{MEMORY}.\textbf{pre}(\text{P})(v) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$

Pas d'induction

- Les relations r_i correspondent aux transitions satisfaisant $\ell \longrightarrow \ell'$ et on associe à chaque r_i la relation $r_{\ell,\ell'}$
  - $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $J(x_0, x) \stackrel{def}{=} \exists v_0, \ell, v.(\ell \in \textsc{Locations} \wedge v_0, v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v_0, v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x_0, x.(x_0, x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$

Pas d'induction

Les deux propriétés suivantes sont équivalentes :

- $\forall i \in \{0, \ldots, n\} : \mathrm{J}(x_0, x) \wedge x\ r_i\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $\forall \ell, \ell' \in \textsc{Locations} : \ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell' \wedge P_\ell(v_0, v')$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell'$ (Tautologie)

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell'$ (Tautologie)
- $pc = \ell \wedge P_\ell(v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \Rightarrow P_{\ell'}(v_0, v')$

Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell'$ (Tautologie)
- $pc = \ell \wedge P_\ell(v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ P_{\ell'}(v_0, v')$
- $P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

Conclusion

- $J(x_0, x) \stackrel{def}{=} \exists \ell, v, v_0.(\ell \in \text{LOCATIONS} \wedge v, v_0 \in \text{MEMORY} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x, x_0 \in \text{VALS} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge J(x_0), x)$
- $\text{J}(x_0, x) \Rightarrow \text{A}(x_0, x)$
- $\exists \ell, v, v_0.(\ell \in \text{LOCATIONS} \wedge v, v_0 \in \text{MEMORY} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v)) \Rightarrow \text{A}(x_0, x)$
- $\forall \ell, v, v_0.(\ell \in \text{LOCATIONS} \wedge v, v_0 \in \text{MEMORY} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v)) \Rightarrow \text{A}(x_0, x)$
- $\forall \ell \in \text{LOCATIONS}, v, v_0 \in \text{MEMORY}.P_\ell(v_0, v) \Rightarrow \text{A}(\ell_0, v_0, \ell, v)$

Conclusion

Les deux propriétés suivantes sont équivalentes :

- $\text{J}(x_0, x) \Rightarrow \text{A}(x_0, x)$
- $\forall \ell \in \text{LOCATIONS}, v, v_0 \in \text{MEMORY}.P_\ell(v_0, v) \Rightarrow \text{A}(\ell_0, v_0, \ell, v)$

Traduction des conditions de vérifications

Les conditions de vérification suivantes sont équivalentes :

- $\forall x_0, x, x' \in \text{LOCATIONS} \times \text{MEMORY}$:

$$\begin{cases} (1) & \text{INIT}(x_0) \Rightarrow \text{J}(x_0, x_0) \\ (2) & \text{J}(x_0, x) \Rightarrow \text{A}(x_0, x) \\ (3) & \forall i \in \{0, \ldots, n\} : \text{J}(x_0, x) \wedge x \; r_i \; x' \Rightarrow \text{J}(x_0, x') \end{cases}$$

- $\forall v_0, v, v' \in \text{MEMORY}$:

$$\begin{cases} (1) & \textbf{pre}(\text{P})(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v) \\ (2) & \forall \ell \in \text{LOCATIONS}. P_\ell(v_0, v) \Rightarrow \text{A}(\ell_0, v_0, \ell, v) \\ (3) & \forall \ell, \ell' \in \text{LOCATIONS} : \\ & \ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v') \end{cases}$$

- Le programme est annoté.
- Les annotations définissent un invariant à vérifier selon les conditions de vérification.
- $A(\ell, v)$ est l'énoncé de la propriété de sûreté à vérifier.

Méthode relationnelle de correction de propriétés de sûreté

Soit $\mathrm{A}(\ell_0, v_0, \ell, v)$ une propriété d'un programme P. Soit une famille d'annotations famille de propriétés $\{P_\ell(v_0, v) : \ell \in \textsc{Locations}\}$ pour ce programme. Si les conditions suivantes sont vérifiées :
alors $\mathrm{A}(\ell_0, v_0, \ell, v)$ est une propriété de sûreté pour le programme P.

⊠ DefinitionCondition de vérification
L'expression $P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$ où ℓ, ℓ' sont deux étiquettes liées par la relation $\longrightarrow$, est appelée une condition de vérification.

Floyd and Hoare

- $\forall v_0, v, v' \in \text{MEMORY}.\forall \ell, \ell' \in \text{LOCATIONS}.\ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$ est équivalent à $\forall \ell, \ell' \in \text{LOCATIONS}.\ell \longrightarrow \ell' \Rightarrow \forall v' \in \text{MEMORY}.P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell,\ell'}(v))$
- $\forall v_0, v, v' \in \text{MEMORY}.\forall \ell, \ell' \in \text{LOCATIONS}.\ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$ est équivalent à $\forall \ell, \ell' \in \text{LOCATIONS}.\ell \longrightarrow \ell' \Rightarrow \forall v' \in \text{MEMORY}.\ (\exists v \in \text{MEMORY}.P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v)) \Rightarrow P_{\ell'}(v_0, v')$

Condition de vérification pour l'affectation

Nous pouvons resumer les deux formes possibles de l'affectation suivante :

- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

$\ell : P_\ell(v_0, v)$
$V := f_{\ell,\ell'}(V)$
$\ell' : P_{\ell'}(v_0, v)$

Condition de vérification pour l'affectation

Nous pouvons resumer les deux formes possibles de l'affectation suivante :

- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

$\ell : P_\ell(v_0, v)$
$V := f_{\ell,\ell'}(V)$
$\ell' : P_{\ell'}(v_0, v)$

Condition de vérification pour l'affectation

Nous pouvons resumer les deux formes possibles de l'affectation suivante :

- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

$\ell : P_\ell(v_0, v)$
$V := f_{\ell,\ell'}(V)$
$\ell' : P_{\ell'}(v_0, v)$

Condition de vérification pour l'affectation

Nous pouvons resumer les deux formes possibles de l'affectation suivante :

$\ell : P_\ell(v_0, v)$
$V := f_{\ell,\ell'}(V)$
$\ell' : P_{\ell'}(v_0, v)$

- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v \in \text{MEMORY}. P_\ell(v_0, v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell,\ell'}(v))$ (l'axiomatique de Hoare).

Condition de vérification pour l'affectation

Nous pouvons resumer les deux formes possibles de l'affectation suivante :

$$\ell : P_\ell(v_0, v)$$
$$V := f_{\ell,\ell'}(V)$$
$$\ell' : P_{\ell'}(v_0, v)$$

- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v \in \text{MEMORY}. P_\ell(v_0, v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell,\ell'}(v))$ (l'axiomatique de Hoare).
- $\forall v \in \text{MEMORY}. (\exists v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v)) \Rightarrow P_{\ell'}(v_0, v')$ correspond à la règle d'affectation de Floyd.

Conditions de vérification pour l'itération

$\ell_1 : P_{\ell_1}(v_0, v)$
WHILE $B(v)$ **DO**
  $\ell_2 : P_{\ell_2}(v_0, v)$
  $\ldots$
  $\ell_3 : P_{\ell_3}(v_0, v)$
END
$\ell_4 : P_{\ell_4}(v_0, v)$

Pour la structure d'itération, les conditions de vérification sont les suivantes :

- $P_{\ell_1}(v_0, v) \wedge B(v) \Rightarrow P_{\ell_2}(v_0, v)$
- $P_{\ell_1}(v_0, v) \wedge \neg B(v) \Rightarrow P_{\ell_4}(v_0, v)$
- $P_{\ell_3}(v_0, v) \wedge B(v) \Rightarrow P_{\ell_2}(v_0, v)$
- $P_{\ell_3}(v_0, v) \wedge \neg B(v) \Rightarrow P_{\ell_4}(v_0, v)$

Conditions de vérification pour la conditionnelle

$\ell_1 : P_{\ell_1}(v_0, v)$
IF $B(v)$ **THEN**
  $\ell_2 : P_{\ell_2}(v_0, v)$
  $\ldots$
  $\ell_3 : P_{\ell_3}(v_0, v)$
ELSE
  $m_2 : P_{\ell_2}(v_0, v)$
  $\ldots$
  $m_3 : P_{\ell_3}(v_0, v)$
FI
$\ell_4 : P_{\ell_4}(v_0, v)$

Pour la structure de conditionnelle, les conditions suivantes :

- $P_{\ell_1}(v_0, v) \wedge B(v) \Rightarrow P_{\ell_2}(v_0, v)$
- $P_{\ell_3}(v_0, v) \Rightarrow P_{\ell_4}(v_0, v)$
- $P_{\ell_1}(v_0, v) \wedge \neg B(v) \Rightarrow P_{m_2}(v_0, v)$
- $P_{m_3}(v_0, v) \Rightarrow P_{\ell_4}(v_0, v)$

Soit v une variable d'état de P. **pre**$(\mathrm{P})(v)$ est la précondition de P pour v ; elle caractérise les valeurs initiales de v. **post**$(\mathrm{P})(v_0, v)$ est la postcondition de P pour v ; elle caractérise les valeurs finales de v en relation avec la valeur initiale v_0

Exemple

1. **pre**$(\mathrm{P})(x, y, z) = x, y, z \in \mathbb{N}$ et **post**$(\mathrm{P})(x_0, y_0, z_0, x, y, z) = z = x_0 \cdot y_0$
2. **pre**$(\mathrm{Q})(x, y, z) = x, y, z \in \mathbb{N}$ et
   post$(\mathrm{Q})(x_0, y_0, z_0, x, y, z) = z = x_0 + y_0$

$$\forall \underline{x}, \underline{y}, \underline{r}, \underline{q}, \overline{x}, \overline{y}, \overline{r}, \overline{q}.$$
$$\textbf{pre}(\mathrm{P})(\underline{x}, \underline{y}, \underline{r}, \underline{q}) \wedge (\underline{x}, \underline{y}, \underline{r}, \underline{q}) \xrightarrow{\mathrm{P}} (\overline{x}, \overline{y}, \overline{r}, \overline{q})$$
$$\Rightarrow \textbf{post}(\mathrm{P})(\underline{x}, \underline{y}, \underline{r}, \underline{q}, \overline{x}, \overline{y}, \overline{r}, \overline{q})$$

Correction partielle d'un programme

La correction partielle vise à établir qu'un programme P est partiellement correct par rapport à sa précondition et à sa postcondition.

- la spécification des données de P **pre**$(\mathrm{P})(\mathrm{v}_0)$
- la spécification des résultats de P **post**$(\mathrm{P})(\mathrm{v}_0,\mathrm{v})$
- une famille d'annotations de propriétés $\{P_\ell(v_0, v) : \ell \in \textsc{Locations}\}$ pour ce programme.
- une propriété de sûreté définissant la correction partielle $pc = \ell_f \Rightarrow$ **post**$(\mathrm{P})(v_0, v_f)$ où ℓ_f est l'étiquette marquant la fin du programme P

⊠ Definition

Le programme P est partiellement correct par rapport à **pre**$(\mathrm{P})(\mathrm{v}_0)$ et **post**$(\mathrm{P})(\mathrm{v}_0,\mathrm{v})$, si la propriété $pc = \ell_f \Rightarrow$ **post**$(\mathrm{P})(v_0, v)$ est une propriété de sûreté pour ce programme.

Correction partielle d'un programme

Si les conditions suivantes sont vérifiées :

- $\forall v_0, v \in \text{MEMORY} : \textbf{pre}(\text{P})(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$
- $\forall v_0, v \in \text{MEMORY} : P_{\ell_f}(v_0, v) \Rightarrow \textbf{post}(\text{P})(v_0, v)$
- $\forall \ell, \ell' \in \text{LOCATIONS} : \ell \longrightarrow \ell' : \forall v_0, v, v' \in \text{MEMORY}. (P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$,

alors le programme P est partiellement correct par rapport à **pre**(P)(v_0) et **post**(P)(v_0,v).

- La correction partielle indique que si le programme termine normalement, alors la postcondition est vérifiée par les variables courantes.
- La sémantique du contrat est donc assez simple à donner :

Reformulation du calcul

- $\mathsf{pre}(v_0) \wedge v_0 \xrightarrow{\mathsf{P}} v_f \Rightarrow \mathsf{post}(v_0, v_f)$

(expression de la correction partielle)

- $pc_0 = \ell_0 \wedge \mathsf{pre}(v_0) \wedge (pc_0, v_0) \overset{\textsc{Next}^\star}{\longrightarrow} (pc, v) \wedge pc = \ell_f \Rightarrow \mathsf{post}(v_0, v_f)$

(big-step semantics et small-step semantics equivalence)

- $pc_0 = \ell_0 \wedge \mathsf{pre}(v_0) \wedge (pc_0, v_0) \overset{\textsc{Next}^\star}{\longrightarrow} (pc, v) \Rightarrow (pc = \ell_f \Rightarrow \mathsf{post}(v_0, v_f))$

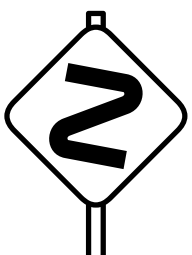
(implication and conjunction property)

- $Init(x_0) \wedge x_0 \overset{\textsc{Next}^\star}{\longrightarrow} x \Rightarrow \mathsf{PC}(x_0, x)$

$(Init(x_0) \overset{def}{=} pc_0 = \ell_0 \wedge \mathsf{pre}(v_0)$

$x_0 \overset{def}{=} (\ell_0, v_0)$ and $x \overset{def}{=} (pc, v)$

$\mathsf{PC}(x_0 x) \overset{def}{=} x_0 = (\ell_0, v_0) \wedge x = (pc, v) \Rightarrow (pc = \ell_f \Rightarrow \mathsf{post}(v_0, v_f))$



Partial correctness is a safety property and the relational method for safety properties is applied.

Vérification du contrat

Un programme P *remplit* un contrat (pre,post) :

- P transforme une variable v à partir d'une valeur initiale v_0 et produisant une valeur finale v_f : $v_0 \xrightarrow{P} v_f$
- v_0 satisfait pre : pre(v_0) and v_f satisfait post : post(v_0, v_f)
- pre$(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow$ post(v_0, v_f)

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables V

begin
$0 : P_0(v_0, v)$
$\texttt{instruction}_0$
...
$i : P_i(v_0, v)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(v_0, v)$
end

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- Pour toute paire d'étiquettes ℓ, ℓ' telle que $\ell \longrightarrow \ell'$, on vérifie que, pour toutes valeurs $v, v' \in \textsc{Memory}$

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

An Early Program Proof by Alan Turing

Turing, A. M. 1949. "Checking a Large Routine." In Report of a Conference on High Speed Automatic Calculating Machines, Univ. Math. Lab., Cambridge,pp. 67-69.

- Turing se pose une question fondamentale de la correction des routines ou programmes en 1949.
- Il s'agit sans doute (Jones !) de la méthode d'annotation et d'induction sur les programmes qui sera finalisée par Floyd en 1967.

Méthode de Floyd

- Au point 0, $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- Annotations : au point i, l'assertion $P_i(x_0, x)$ est vraie.
- Au point final f, $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$

Absence d'erreurs à l'exécution

- La transition à exécuter est celle allant de ℓ à ℓ' et caractérisée par la condition ou garde $cond_{\ell,\ell'}(v)$ sur v et une transformation de la variable v, $v' = f_{\ell,\ell'}(v)$.
- Une condition d'absence d'erreur est définie par $\mathbf{DOM}(\ell,\ell')(v)$ pour la transition considérée. $\mathbf{DOM}(\ell,\ell')(v)$ signifie que la transition $\ell \longrightarrow \ell'$ est possible et ne conduit pas à une erreur.
- Une erreur est un débordement arithmétique, une référence à un élément de tableau qui 'existe pas, une référence à un pointeur nul, ...

exemple

1. La transition correspond à une affectation de la forme $x := x+y$ ou $y := x+y$:
   $\mathbf{DOM}(x+y)(x,y) \stackrel{def}{=} \mathbf{DOM}(x)(x,y) \wedge \mathbf{DOM}(y)(x,y) \wedge x+y \in int$
2. La transition correspond à une affectation de la forme $x := x+1$ ou $y := x+1$:
   $\mathbf{DOM}(x+1)(x,y) \stackrel{def}{=} \mathbf{DOM}(x)(x,y) \wedge x+2 \in int$

Définition RTE

L'absence d'erreurs à l'exécution vise à établir qu'un programme P ne va pas produire des erreurs durant son exécution par rapport à sa précondition et à sa postcondition.

- la spécification des données de P **pre**$(\mathrm{P})(\mathrm{v})$
- la spécification des résultats de P **post**$(\mathrm{P})(\mathrm{v}_0,\mathrm{v})$
- une famille d'annotations de propriétés $\{P_\ell(v) : \ell \in \textsc{Locations}\}$ pour ce programme.
- une propriété de sûreté définissant l'absence d'erreurs à l'exécution :

$$\bigwedge_{\ell \in \textsc{Locations} - \{output\}, n \in \textsc{Locations}, \ell \longrightarrow n} (\mathbf{DOM}(\ell, n)(v))$$

☒ Definition

Le programme P ne produira pas d'erreurs à l'exécution par rapport à **pre**$(\mathrm{P})(\mathrm{v})$ et **post**$(\mathrm{P})(\mathrm{v}_0,\mathrm{v})$, si la propriété $\bigwedge_{\ell \in \textsc{Locations} - \{output\}, n \in \textsc{Locations}, \ell \longrightarrow n} (\mathbf{DOM}(\ell, n)(v))$ est une propriété de sûreté pour ce programme.

Principe d'induction pour RTE

RTE = Run Time Error

Si les conditions suivantes sont vérifiées :

- $\forall v_0, v \in \text{MEMORY} : \textbf{pre}(\text{P})(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$
- $\forall m \in \text{LOCATIONS} - \{\ell_f\}, n \in \text{LOCATIONS}, \forall v_0, v, v' \in \text{MEMORY} :$ $m \longrightarrow n : \textbf{pre}(\text{P})(v_0) \wedge P_m(v_0, v) \Rightarrow \textbf{DOM}(m, n)(v)$
- $\forall \ell, \ell' \in \text{LOCATIONS} : \ell \longrightarrow \ell' : \forall v_0, v, v' \in$ $\text{MEMORY}. (P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')),$

alors le programme P ne produira pas d'erreurs à l'exécution par rapport à **pre**(P)(v_0) et **post**(P)(v_0,v).

- On doit d'abord vérifier la correction partielle puis renforcer les assertions de la correction partielle par des conditions de domaine.
- On peut donc en déduire un contrat qui intègre aussi la vérification de l'absence d'erreurs à l'exécution.

Méthode relationnelle de vérification pour RTE

Un programme P *remplit* un contrat (pre,post) :

- P transforme une variable v à partir d'une valeur initiale v_0 et produisant une valeur finale v_f : $v_0 \xrightarrow{P} v_f$
- v_0 satisfait pre : pre(v_0) and v_f satisfait post : post(v_0, v_f)
- pre$(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow$ post(v_0, v_f)
- $\mathbb{D}$ est le domaine RTE de V

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables V
begin
$0 : P_0(v_0, v)$
$\texttt{instruction}_0$
...
$i : P_i(v_0, v)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(v_0, v)$
end

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(x_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- Pour toute paire d'étiquettes ℓ, ℓ' telle que $\ell \longrightarrow \ell'$, on vérifie que, pour toutes valeurs $v, v' \in \textsc{Memory}$

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

- $\forall m \in \textsc{Locations} - \{\ell_f\}, n \in \textsc{Locations}, \forall v_0, v, v' \in \textsc{Memory}$: $m \longrightarrow n$: $pre(v_0) \wedge P_m(v_0, v) \Rightarrow \mathbf{DOM}(m, n)(v)$