Cours MALG & MOVEX

Modélisation, spécification et vérification (I)

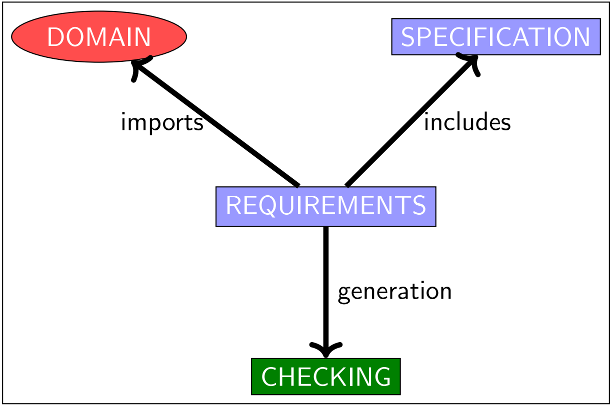
Dominique Méry
Telecom Nancy, Université de Lorraine
(23 mars 2025 at 10:10 P.M.)

Année universitaire 2024-2025

Plan

Sommaire

Tryptic Pattern



- Typing Properties using Typechecker (see for instance functional programming languages as ML, CAML, OCAML, ...)
- Invariance and safety (A *nothing bad will happen !*) properties for a program P :
  - Transformation of P into a relational model M simulating P
  - Expression of safety properties :
    $\forall s, s' \in \Sigma.(s \in Init_{\mathrm{S}} \wedge s \xrightarrow{\star} s') \Rightarrow (s' \in A)$.
  - Definition of the set of reachable states of P using M :
    $\textsc{reachable}(\mathrm{M}) = Init_{\mathrm{S}} \cup \longrightarrow [\textsc{reachable}(\mathrm{M})]$
  - Main property of $\textsc{reachable}(\mathrm{M})$: $\textsc{reachable}(\mathrm{M}) \subseteq A$
  - Characterization of $\textsc{reachable}(\mathrm{M})$:
    $\textsc{reachable}(\mathrm{M}) = \mathrm{FP}(\textsc{reachable}(\mathrm{M}))$

Problems

- Proving automatically $\text{REACHABLE}(M) \subseteq A$:

Problems

- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$: undecidable ... no program is able to prove it automatically !

Problems

- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$: undecidable ... no program is able to prove it automatically !
- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$:

Problems

- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$: undecidable ... no program is able to prove it automatically !
- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$: possible when restrictions over the set of states is possible (finite set of states)

Problems

- Proving automatically $\mathrm{REACHABLE}(\mathrm{M}) \subseteq A$: undecidable ... no program is able to prove it automatically !
- Proving automatically $\mathrm{REACHABLE}(\mathrm{M}) \subseteq A$: possible when restrictions over the set of states is possible (finite set of states)
- Proving automatically $\mathrm{REACHABLE}(\mathrm{M}) \subseteq A$:

Problems

- Proving automatically $\text{REACHABLE}(\mathrm{M}) \subseteq A$: undecidable ... no program is able to prove it automatically !
- Proving automatically $\text{REACHABLE}(\mathrm{M}) \subseteq A$: possible when restrictions over the set of states is possible (finite set of states)
- Proving automatically $\text{REACHABLE}(\mathrm{M}) \subseteq A$: possible for some classes of systems and with some tools.

Problems

- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$: undecidable ... no program is able to prove it automatically !
- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$: possible when restrictions over the set of states is possible (finite set of states)
- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$: possible for some classes of systems and with some tools.
- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$: changing the domain and solving in another domain as abstract interpretation if making possible

Problems

- Proving automatically $\text{REACHABLE}(\mathrm{M}) \subseteq A$: undecidable ... no program is able to prove it automatically !
- Proving automatically $\text{REACHABLE}(\mathrm{M}) \subseteq A$: possible when restrictions over the set of states is possible (finite set of states)
- Proving automatically $\text{REACHABLE}(\mathrm{M}) \subseteq A$: possible for some classes of systems and with some tools.
- Proving automatically $\text{REACHABLE}(\mathrm{M}) \subseteq A$: changing the domain and solving in another domain as abstract interpretation if making possible
- Proving automatically $\text{REACHABLE}(\mathrm{M}) \subseteq A$:

Problems

- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$: undecidable ... no program is able to prove it automatically !
- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$: possible when restrictions over the set of states is possible (finite set of states)
- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$: possible for some classes of systems and with some tools.
- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$: changing the domain and solving in another domain as abstract interpretation if making possible
- Proving automatically $\textsc{reachable}(\mathrm{M}) \subseteq A$: approximating semantics of programs

Decidability or Undecidability

- A problem $x \in P$ is generally stated by the function $\chi_{x \in P}$ where $\chi_{x \in P}(u) = 1$, if $P(u)$ is true and $\chi_{x \in P}(u) = 0$, if $P(u)$ is false :
  - Problem 1 : $x \in 0..n$ where $n \in \mathbb{N}$
  - Problem 1 : $w \in \mathcal{L}(G)$ where G is a grammar over the finite set of alphabet symbols Σ and $\mathcal{L}(G) \subseteq \Sigma^\star$.

Decidability or Undecidability

- A problem $x \in P$ is generally stated by the function $\chi_{x \in P}$ where $\chi_{x \in P}(u) = 1$, if $P(u)$ is true and $\chi_{x \in P}(u) = 0$, if $P(u)$ is false :
  - Problem 1 : $x \in 0..n$ where $n \in \mathbb{N}$
  - Problem 1 : $w \in \mathcal{L}(G)$ where G is a grammar over the finite set of alphabet symbols Σ and $\mathcal{L}(G) \subseteq \Sigma^\star$.
- A problem $x \in P$ is decidable, when the function $\chi_{x \in P}$ is computable or more precisely the function can be computed by a program

Decidability or Undecidability

- A problem $x \in P$ is generally stated by the function $\chi_{x \in P}$ where $\chi_{x \in P}(u) = 1$, if $P(u)$ is true and $\chi_{x \in P}(u) = 0$, if $P(u)$ is false :
  - Problem 1 : $x \in 0..n$ where $n \in \mathbb{N}$
  - Problem 1 : $w \in \mathcal{L}(G)$ where G is a grammar over the finite set of alphabet symbols Σ and $\mathcal{L}(G) \subseteq \Sigma^\star$.
- A problem $x \in P$ is decidable, when the function $\chi_{x \in P}$ is computable or more precisely the function can be computed by a program
- Problem of the correctness of a program :
  - Assume that $\mathcal{F}$ is the set of unary function over natural numbers : $\mathcal{F} = \mathbb{N} \nrightarrow \mathbb{N}$.
  - $\mathcal{C} \subseteq \mathcal{F}$: the set of computable (or programmable) functions is $\mathcal{C}$
  - $f \in \mathcal{C} = \{\Phi_0, \Phi_1, \ldots, \Phi_n, \ldots\}$: the set of computable functions is denumerable.
  - The problem $x \in dom(\Phi_y)$ is not decidable and it expresses the correctness of programs.

Quelques observations

Implicite versus explicite

- Ecrire $101 = 5$ peut avoir une signification

Quelques observations

Implicite versus explicite

- Ecrire *101* $=$ *5* peut avoir une signification
- Le code du nombre n est *101* à gauche du symbole $=$ et le code du nombre n est sa représentation en base 10 à droite.
- $n_{10} = 5$ et $n_2 = 101$
- Vérification : $base(2, 10, 101) = 1.2^2+0.2+1.2^0 = 5_{10}$

Example : description of static behaviour

- A train moving at absolute speed $spd1$
- A person walking in this train with relative speed $spd2$
  - One may compute the absolute speed of the person
- Modelling
  - Syntax. Classical expressions
    - Type $Speed = Float$
    - $spd1, spd2 : Speed$
    - $AbsoluteSpeed = spd1+spd2$
  - Semantics
    - If $spd1 = 25.6$ and $spd2 = 24.4$ then $AbsoluteSpeed = 50.0$
    - If $spd1 = "val"$ and $spd2 = 24.4$ then exception raised
  - Pragmatics
    - What if $spd1 is$ given in mph (miles per hour) and $spd2$ in km/s (kilometers per second) ?
    - What if $spd1$ is a relative speed ?

Intuition

- Un programme P *produit* des résultats à partir de données en accord avec une sémantique :
  - STATES est l'ensemble de tous les états de P : $STATES = \mathrm{X} \rightarrow \mathbb{Z}$ où X désigne les variables de P.
  - s_0 et s_f deux états de STATES : $\mathcal{D}(P)(s_0) = s_f$ signifie que P est exécuté à partir d'un état s_0 et produit un état s_f.
  - Pour un état s de P courant, on notera $s(X) = x$ pour distinguer la valeur de la variable X et sa valeur courante en s :

Intuition

- Un programme P *produit* des résultats à partir de données en accord avec une sémantique :
  - STATES est l'ensemble de tous les états de P : $STATES = \mathsf{X} \rightarrow \mathbb{Z}$ où X désigne les variables de P.
  - s_0 et s_f deux états de STATES : $\mathcal{D}(P)(s_0) = s_f$ signifie que P est exécuté à partir d'un état s_0 et produit un état s_f.
  - Pour un état s de P courant, on notera $s(X) = x$ pour distinguer la valeur de la variable X et sa valeur courante en s :
  $$s_0(X) = x_0,\ s_f(X) = x_f,\ s'(X) = x'$$

Intuition

- Un programme P *produit* des résultats à partir de données en accord avec une sémantique :
  - STATES est l'ensemble de tous les états de P : $STATES = \mathsf{X} \rightarrow \mathbb{Z}$ où X désigne les variables de P.
  - s_0 et s_f deux états de STATES : $\mathcal{D}(P)(s_0) = s_f$ signifie que P est exécuté à partir d'un état s_0 et produit un état s_f.
  - Pour un état s de P courant, on notera $s(X) = x$ pour distinguer la valeur de la variable X et sa valeur courante en s :
  $$s_0(X) = x_0,\ s_f(X) = x_f,\ s'(X) = x'$$
  - $\mathcal{D}(P)(s_0) = s_f$ définit la relation suivante sur l'ensemble des valeurs :
  $$x_0 \xrightarrow{\mathsf{P}} x_f$$

Intuition

- Un programme P *produit* des résultats à partir de données en accord avec une sémantique :
  - STATES est l'ensemble de tous les états de P : $STATES = \mathsf{X} \rightarrow \mathbb{Z}$ où X désigne les variables de P.
  - s_0 et s_f deux états de STATES : $\mathcal{D}(P)(s_0) = s_f$ signifie que P est exécuté à partir d'un état s_0 et produit un état s_f.
  - Pour un état s de P courant, on notera $s(X) = x$ pour distinguer la valeur de la variable X et sa valeur courante en s :
    $$s_0(X) = x_0,\ s_f(X) = x_f,\ s'(X) = x'$$
  - $\mathcal{D}(P)(s_0) = s_f$ définit la relation suivante sur l'ensemble des valeurs :
    $$x_0 \xrightarrow{\mathsf{P}} x_f$$
- 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 : $x_0 \xrightarrow{\mathsf{P}} x_f$
  - x_0 satisfait pre : $\mathsf{pre}(x_0)$
  - x_f satisfait post : $\mathsf{post}(x_0, x_f)$
  - $\mathsf{pre}(x_0) \wedge x_0 \xrightarrow{\mathsf{P}} x_f \Rightarrow \mathsf{post}(x_0, x_f)$

Vérification du contrat

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)

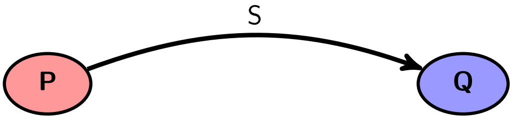
requires $pre(x_0)$
ensures $post(x_0, x_f)$
variables X

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

- $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- $P_f(x_0, x) \Rightarrow post(x_0, x)$
- some conditions for verification related to pairs $\ell \longrightarrow \ell'$

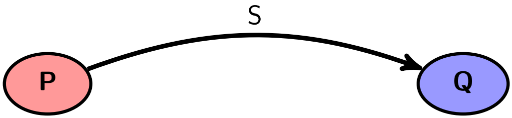
Asserted Program $\{P\}$ S $\{Q\}$

$$\{P\}\ S\ \{Q\}$$



Asserted Program $\{P\}$ S $\{Q\}$

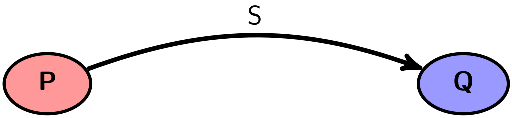
$$\{P\}\ S\ \{Q\}$$



- $\{P\}$ S $\{Q\}$: *asserted program*

Asserted Program {P} S {Q}

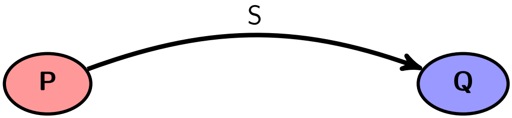
{P} S {Q}



- {P} S {Q} : *asserted program*
- P $\Rightarrow WP(S)(\mathsf{Q})$: *logical formula*

Asserted Program {P} S {Q}

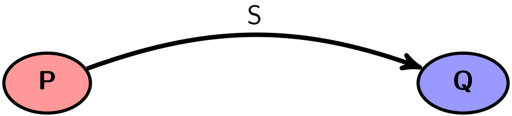
{P} S {Q}



- {P} S {Q} : *asserted program*
- P $\Rightarrow WP(S)$(Q) : *logical formula*
- C(S) $\vdash$ P $\Rightarrow WP(S)$(Q) : *logical formula*

Asserted Program {P} S {Q}

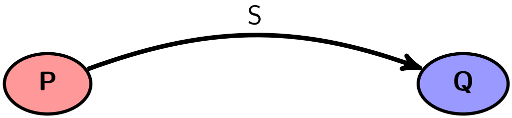
{P} S {Q}



- {P} S {Q} : *asserted program*
- P $\Rightarrow WP(S)$(Q) : *logical formula*
- C(S) $\vdash$ P $\Rightarrow WP(S)$(Q) : *logical formula*
- $SP(S)$(P) $\Rightarrow$ Q : *logical formula*

Asserted Program {P} S {Q}

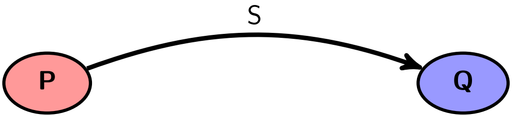
{P} S {Q}



- {P} S {Q} : *asserted program*
- P $\Rightarrow WP(S)$(Q) : *logical formula*
- C$(S) \vdash$ P $\Rightarrow WP(S)$(Q) : *logical formula*
- $SP(S)$(P) $\Rightarrow$ Q : *logical formula*
- C$(S) \vdash SP(S)$(P) $\Rightarrow$ Q : *logical formula*

Asserted Program {P} S {Q}

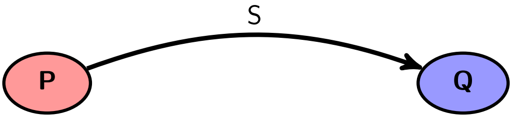
{P} S {Q}



- {P} S {Q} : *asserted program*
- P $\Rightarrow WP(S)$(Q) : *logical formula*
- C(S) $\vdash$ P $\Rightarrow WP(S)$(Q) : *logical formula*
- $SP(S)$(P) $\Rightarrow$ Q : *logical formula*
- C(S) $\vdash SP(S)$(P) $\Rightarrow$ Q : *logical formula*

Asserted Program {P} S {Q}

{P} S {Q}



- {P} S {Q} : *asserted program*
- P $\Rightarrow WP(S)$(Q) : *logical formula*
- C(S) $\vdash$ P $\Rightarrow WP(S)$(Q) : *logical formula*
- $SP(S)$(P) $\Rightarrow$ Q : *logical formula*
- C(S) $\vdash SP(S)$(P) $\Rightarrow$ Q : *logical formula*

Predicate Transformer

$WP(S)$(Q) is the Weakest-Precondition of S for Q and is a predicate transformer but $WP(S)(.)$ is not a computable function over the set of predicates.

Method for verifying program properties

correctness and Run Time Errors

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

- P transforms a variable x from initial values x_0 and produces a final value x_f : $x_0 \xrightarrow{P} x_f$
- x_0 satisfies pre : pre(x_0) and x_f satisfies post : post(x_0, x_f)
- pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)
- $\mathbb{D}$ est le domaine RTE de X

requires $pre(x_0)$ ensures $post(x_0, x_f)$ variables X begin $0 : P_0(x_0, x)$ `instruction`$_0$ $\ldots$ $i : P_i(x_0, x)$ $\ldots$ `instruction`$_{f-1}$ $f : P_f(x_0, x)$ end

- $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$
- For any pair of labelsℓ, ℓ' such that $\ell \longrightarrow \ell'$, one verifies that, pour any values $x, x' \in \textsc{Memory}$
$$\left(\begin{array}{l} \left(\begin{array}{l} pre(x_0) \wedge P_\ell(x_0, x)) \\ \wedge cond_{\ell,\ell'}(x) \wedge x' = f_{\ell,\ell'}(x) \end{array} \right) \\ \Rightarrow P_{\ell'}(x_0, x') \end{array} \right),$$
- For any pair of labels m, n such taht $m \longrightarrow n$, one verifies that, $\forall x, x' \in \textsc{Memory} : pre(x_0) \wedge P_m(x_0, x) \Rightarrow$ **DOM**$(m, n)(x)$

Example of an annotation

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

Example of an annotation

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

$\longrightarrow$

Example of an annotation

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

$\longrightarrow \longrightarrow$

Example of an annotation

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

$\longrightarrow \longrightarrow$

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

Summary

Summary

Modelling, specification and verification

Modelling, specification and verification

- Set-theoretical notations using the Eevent-B modelling language
- Relational modelling of a program or an algorithm
- Program properties as safety, invariance, pre and post conditions
- Design by contract
- Method for proving invariance properties of programs and induction principles as Floyd's method, Hoar logics,
- Techniques for Model-Checking
- Tools : the toolset RODIN, the toolset TLAPS, the toolset PAT, the toolset Eiffel Studio, the toolset Frama-c, . . .

Summary

Summary

Logics

Logics

- Propositional Formulae and first order formulae
- Models
- Sequents Calculus
- Proofs and deduction
- Resolution
- Tools : the toolset Rodin

Summary

Fixed-Point Theory

Fixed-Point Theory

- Complete Partiallly Ordered Sets (CPO) and Complete Lattices
- Fixed-Point Theorems : Kleene, Tarski, . . .
- Abstract Interpretation
- Galois Lattices
- Static analysis of Programs by abstract interpretation.
- Semantics of programs

Summary

Summary

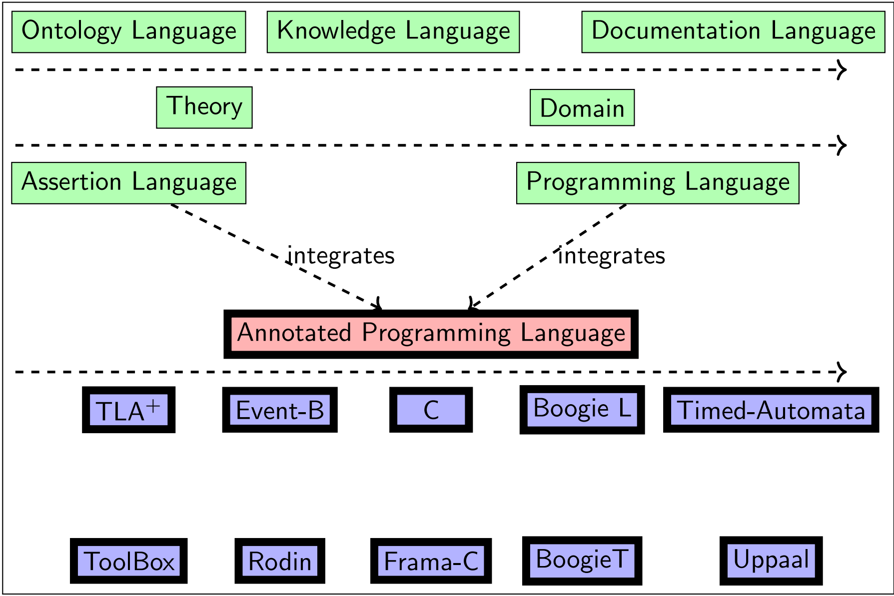
Computability, Decidability, complexity, Undecidabiliy

Summary

Computability, Decidability, complexity, Undecidabiliy

- Models of computing : Turing Machines, Partially Recursive Functions, URM, ...
- Church's Thesis
- Decidability
- Undecidability
- Complexity

Summry of concepts



Summary

Summary

Tools

Summary

Tools

- The TLA$^+$ ToolBox
- The RODIN platform
- Frama-C
- Boogie and the Visual Studio Suite
- UPPAAL

Informations et méthodologie du cours

- Documents sur Arche MODÈLES ET ALGORITHMES avec le mot de passe *mery2023*
- Alternance des cours et des TDs avec des séances sur machines.
- Intervention de Rosemary Monahan de NUI Maynooth en cours d'année pour un cours et un TD dupliqué pour IL.
- Deux groupes de TD
- Machine virtuelle et machines telecom avec les logiciels installés.

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

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

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

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

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

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

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

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

$$\mathcal{D}, \mathcal{S} \quad \textsc{satisfies} \quad \mathcal{R}$$

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

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

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

Method for verifying program properties correctness and Run Time Errors

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

- P transforms a variable v from initial values v_0 and produces a final value v_f : $v_0 \xrightarrow{\text{P}} v_f$
- v_0 satisfies pre : pre(v_0) and v_f satisfies post : post(v_0, v_f)
- pre$(v_0) \wedge v_0 \xrightarrow{\text{P}} v_f \Rightarrow$ post(v_0, v_f)
- $\mathbb{D}$ est le domaine RTE de V

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

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

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- For any pair of labelsℓ, ℓ' such that $\ell \longrightarrow \ell'$, one verifies that, pour any values $v, v' \in \textsc{Memory}$
$$\left(\begin{array}{l} \left(\begin{array}{l} pre(v_0) \wedge P_\ell(v_0, v)) \\ \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \end{array} \right) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$
- For any pair of labels m, n such taht $m \longrightarrow n$, one verifies that, $\forall v, v' \in \textsc{Memory}$: $pre(v_0) \wedge P_m(v_0, v) \Rightarrow$ **DOM**$(m, n)(v)$

Example of an annotation

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

Example of an annotation

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

$\longrightarrow$

Example of an annotation

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

$\longrightarrow \longrightarrow$

Example of an annotation

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

$\longrightarrow \longrightarrow \longrightarrow$

Example of an annotation

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

$\longrightarrow \longrightarrow \longrightarrow \longrightarrow$

Example of an annotation

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

$\longrightarrow \longrightarrow \longrightarrow \longrightarrow$

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

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, ... R_p, ...

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.

- 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

- $[\![\mathrm{x}]\!](s)$ is the value of `x` in s and its value will be distinguished by the font used : `x` is the `tt` font of LaTeX and x is the math font of LaTeX.
- Using the name of the variable `x` as its current value, i.e. x and $[\![\mathrm{x}]\!](s')$ is the value of `x` in s' and will be noted x'.
- The transition relation as a relation linking the state of the variables in s and the state of the variables in s' using the prime notation as defined by L. Lamport for TLA.
- Types of variable depending on whether we are talking about the computer variable, its value or whether we are defining constants such as np, the number of processes, or π, which designates the constant π.
- a current observation refers to a current state for both endurant and perdurant information data in the sense of the Dines Bjørner.

Notations

flexible variable

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

- x is the current value of x in other words the value at the observation time of x.
- x' is the next value of x in other words the value at the next observation time of x.
- x_0 is the initial value of x in other words the value at the initial observation time of x.

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

state property of a system

Let be a system S whose flexible variables x are the elements of $\mathcal{V}ar(S)$. A property $P(x)$ of S is a logical expression involving ,freely the flexible variables x and whose interpretation is the set of values of the domain of x : $P(x)$ is true in x, if the value x satisfies $P(x)$.
For each property $P(x)$, we can associate a subset of D denoted $\hat{P}$ and, in this case, $P(x)$ is true in x. is equivalent to $x \in \hat{P}$.

Examples of property

- $P_1(x) \stackrel{def}{=} x \in 18..22$: x is a value between 18 and 22 and $\hat{P_1} = \{18, 19, 20, 21, 22\}$.
- $P_2(p) \stackrel{def}{=} p \subset PEOPLE \wedge card(p) \leq n$: p is a set of persons and that set has at most n elements and $\hat{P_2} = \{p_1 \ldots p_n\}$. In this example, we use a logical variable n and a name for a constant $PEOPLE$.

Endurant information data

basic set of a system S

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

constants of system S

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

Examples of constant and set

- $fred$ is a constant and is linked to the set $PEOPLE$ using the expression $fred \in PEOPLE$ which means that $fred$ is a person from $PEOPLE$.
- aut is a constant which is used to express the table of authorisations associated with the use of vehicles. the expression $aut \subseteq PEOPLE \times CARS$ where $CARS$ denotes a set of cars.

axiom of system S

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 ($s, s' \in \mathcal{V}ar(S) \longrightarrow \textsc{Vals}$). $s \underset{R}{\longrightarrow} s'$ will be written as a relation $\mathrm{R}(x,x')$ where x and x' designate values of $\mathtt{x}$ before and after the observation of R.

. . . .

⊠ Definition(event)

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

. . . .

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

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

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 $Next(s, c, x, x') \stackrel{def}{=} BA(e_0)(s, c, x, x') \lor \ldots \lor BA(e_n)(s, c, x, x')$.
- the transitive reflexive closure of the relation $Next^\star(s, c, x_0, x) \stackrel{def}{=}$
$$\left\{\begin{array}{l} \lor\ x = x_0 \\ \lor\ Next(s, c, x_0, x) \\ \lor\ \exists\ xi \in \textsc{Vals}.Next^\star(s, c, x_0, xi) \land Next(s, c, xi, x) \end{array}\right.$$

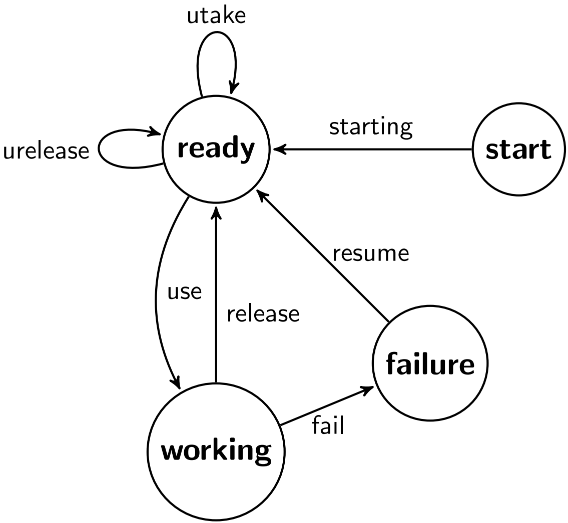
. . . .

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

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

. . . .

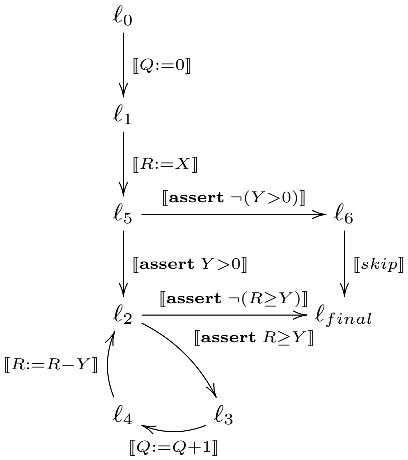
Automata



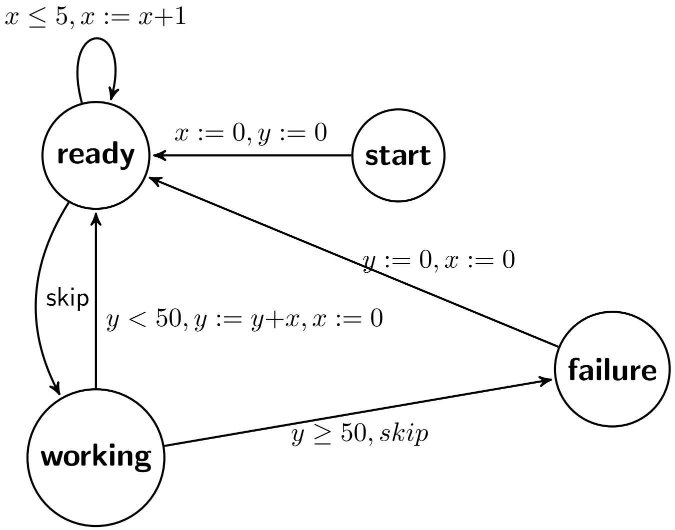
- there is an implicit control variable $pc \in \{\textbf{start}, \textbf{ready}, \textbf{working}, \textbf{failure}\}$ expressing the current visited state.

Programme as flowchart

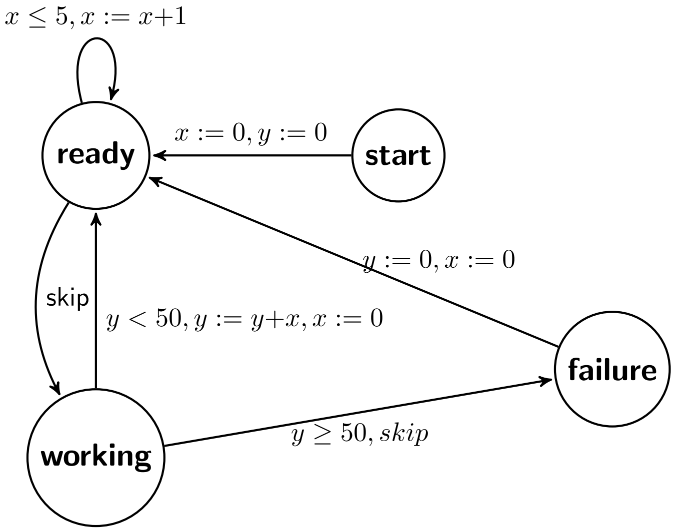
$\ell_0[Q := 0];$
$\ell_1[R := X];$
IF $\ell_5[Y > 0]$
    WHILE $\ell_2[R \geq Y]$
        $\ell_3[Q := Q+1];$
        $\ell_4[R := R-Y]$
    ENDWHILE
ELSE
    $\ell_6[skip]$
ENDIF



A small system as an automaton

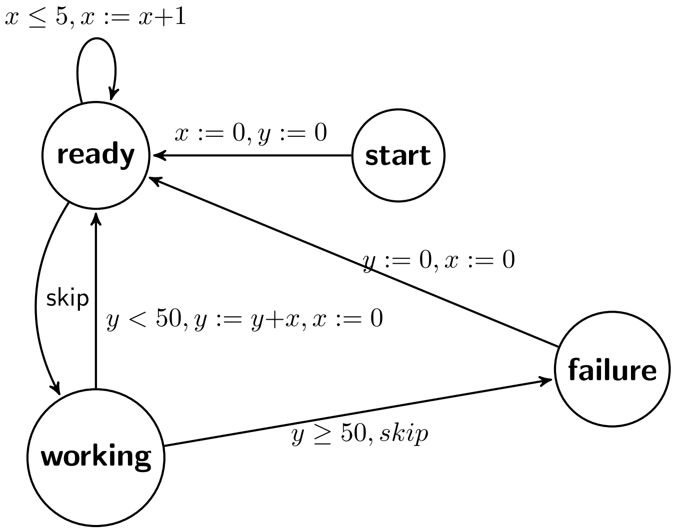


A small system as an automaton



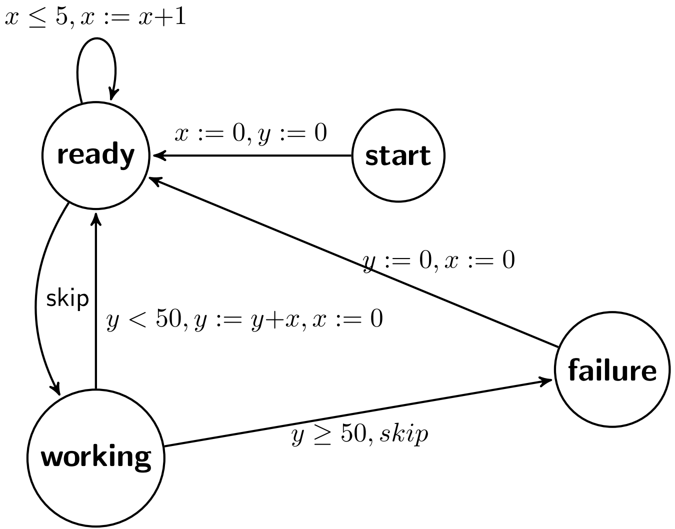
- safety1 : $0 \leq x \leq 5$

A small system as an automaton



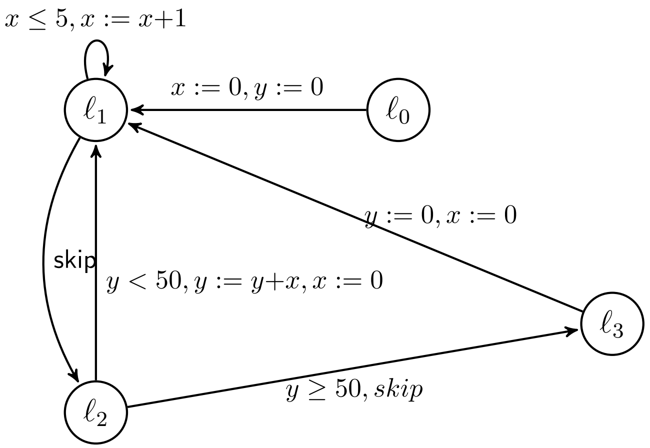
- safety1 : $0 \leq x \leq 5$ et ...

A small system as an automaton

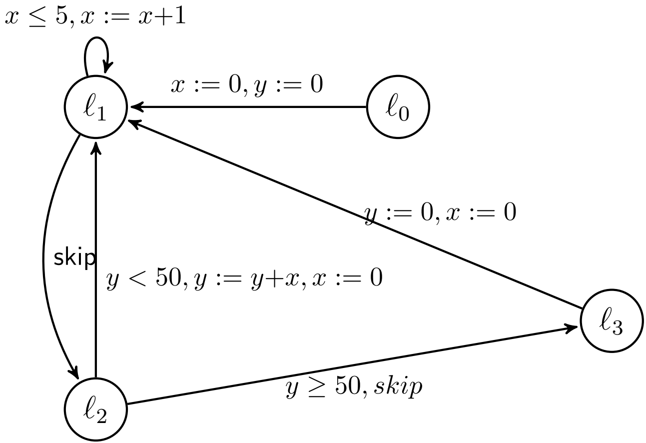


- safety1 : $0 \leq x \leq 5$ et ... safety2 : $0 \leq y \leq 56$

A small system as an automaton



A small system as an automaton



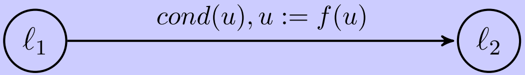
- safety1 : $0 \leq x \leq 5$ et safety2 : $0 \leq y \leq 56$
- $skip = x := x, y := y$
- skip $= TRUE, x := x, y := y = TRUE, skip$

Transition between two control states

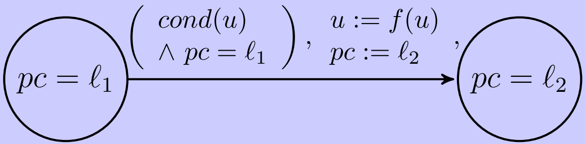
$$\ell_1 \xrightarrow{cond(u),\, u := f(u)} \ell_2$$

Forms of transitions

Transition between two control states



Transition between two control states



Forms of transitions

Transition between two control states

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

Transition between two control states

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

Transition between two predicates

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

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.

. . . .

$\boxtimes$ 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

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

. . . .

. . . .

$\boxtimes$ 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

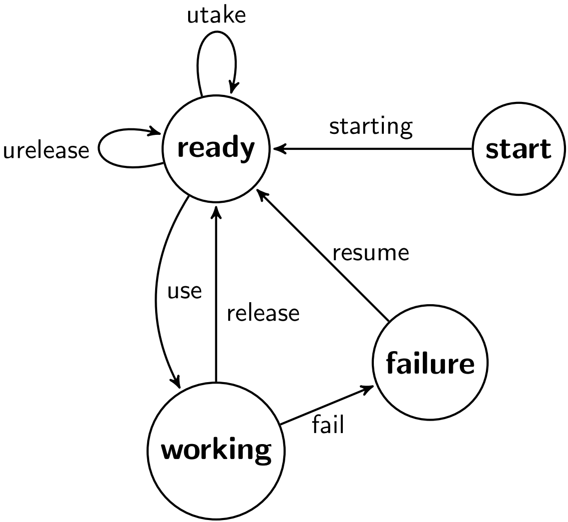
$$\forall x_0, x \in \textsc{Vals}.Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow R(x_0, x).$$

. . . .

Assertion versus relation

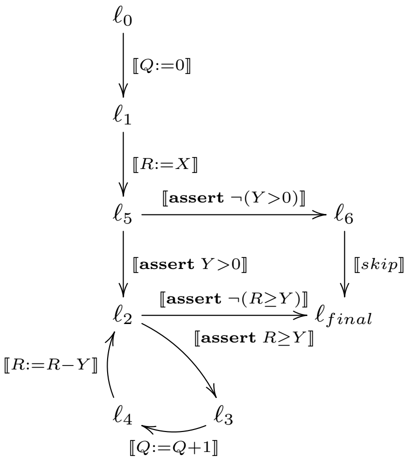
- P. et R. Cousot développent une étude complète des propriétés d'invariance et de sûreté en mettant en évidence correspondances entre les différentes méthodes ou systèmes proposées par Turing, Floyd, Hoare, Wegbreit, Manna ... et reformulent les principes d'induction utilisés pour définir ces méthodes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

Automata


utake
urelease
ready
starting
start
resume
use
release
failure
working
fail

Programme as flowchart

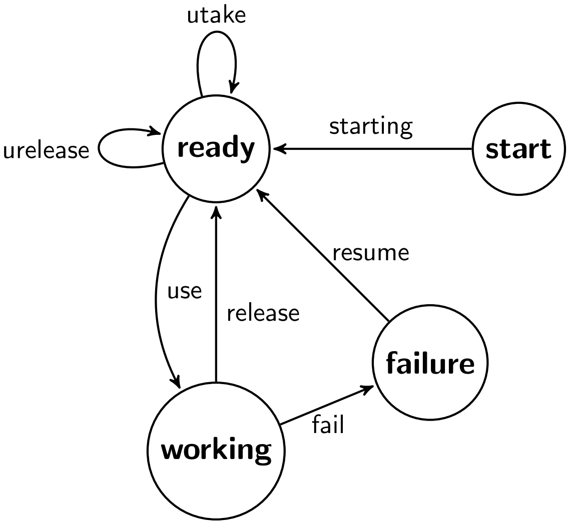
$\ell_0[Q := 0];$
$\ell_1[R := X];$
IF $\ell_5[Y > 0]$
    WHILE $\ell_2[R \geq Y]$
        $\ell_3[Q := Q{+}1];$
        $\ell_4[R := R{-}Y]$
    ENDWHILE
ELSE
    $\ell_6[skip]$
ENDIF



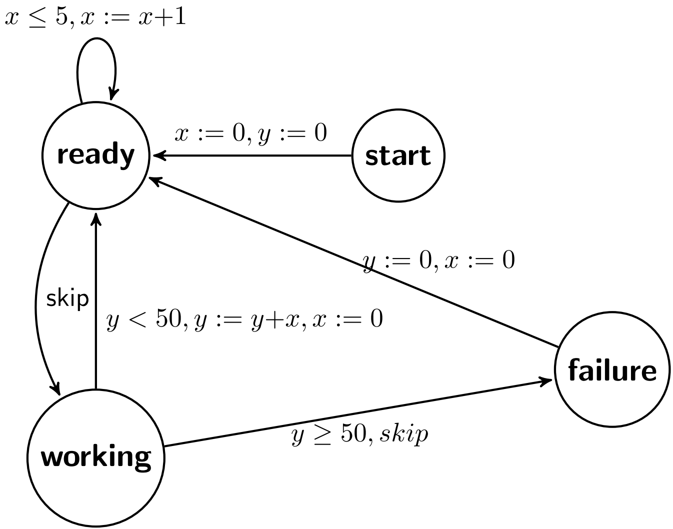
Observations

- 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.

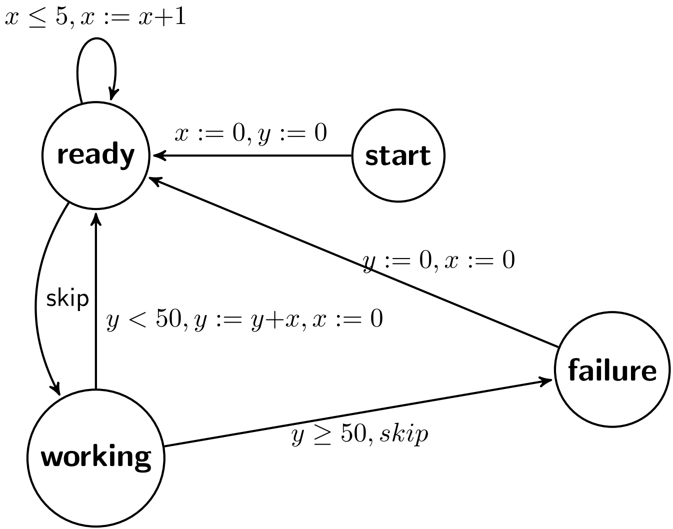
Un petit système en tant qu'automate



Un petit système en tant qu'automate

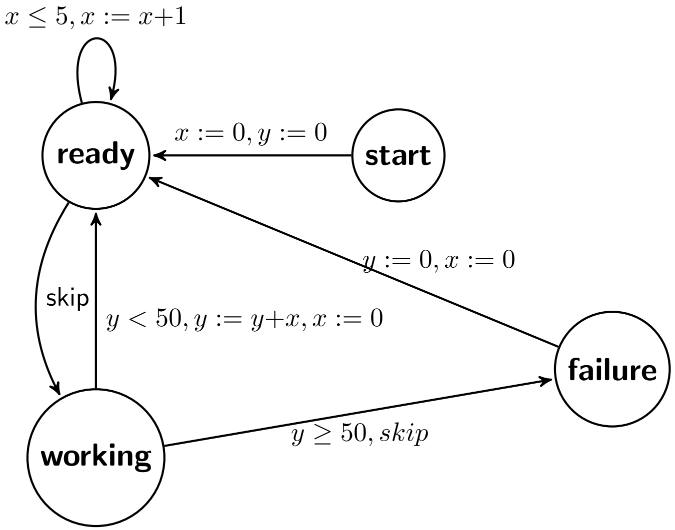


Un petit système en tant qu'automate



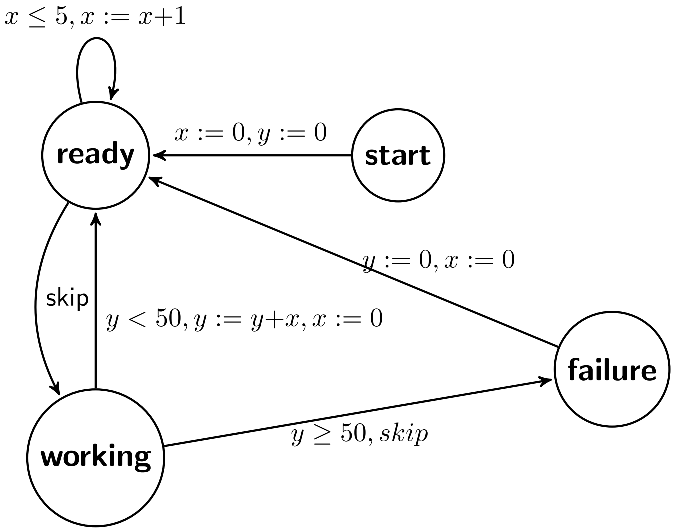
- safety1 : $0 \leq x \leq 5$

Un petit système en tant qu'automate



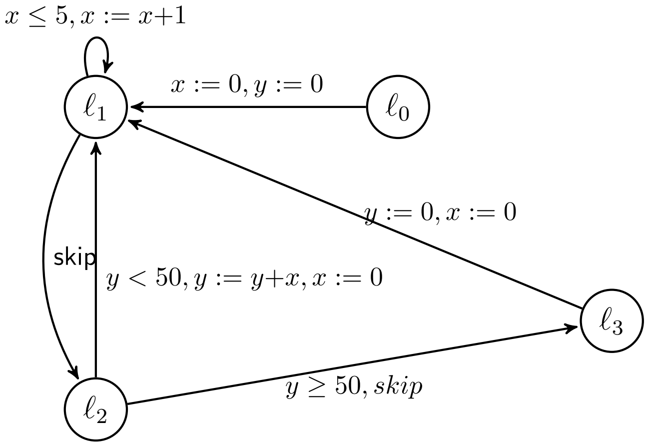
- safety1 : $0 \leq x \leq 5$ et ...

Un petit système en tant qu'automate

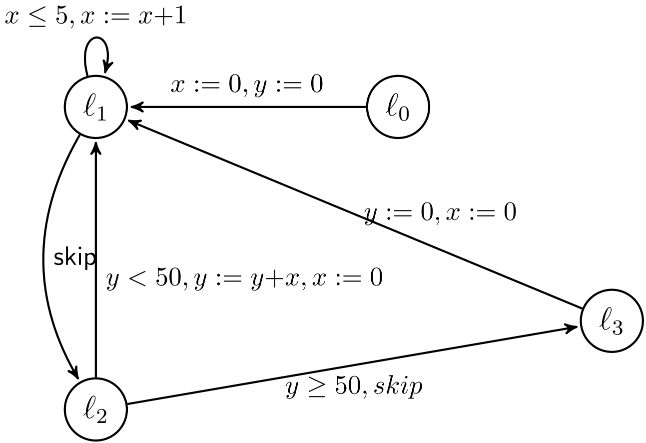


- safety1 : $0 \leq x \leq 5$ et ... safety2 : $0 \leq y \leq 56$

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

Transition entre deux états de contrôle

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

Quelques formes de transitions

Transition entre deux états de contrôle

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

Transition entre deux états de contrôle

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

Quelques formes de transitions

Transition entre deux états de contrôle

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

Transition entre deux états de contrôle

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

Transition entre deux prédicats

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

Computing GCD

MODULE *pgcd*

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

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

$$a1 \triangleq x > y \land x' = x - y \land y' = y$$
$$a2 \triangleq x < y \land y' = y - x \land x' = x$$
$$over \triangleq x = y \land x' = x \land y' = y$$

$$Next \triangleq a_1 \lor a_2 \lor over$$

$$test \triangleq x \neq y$$

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

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

$entrer \triangleq np' = np + 1$

$sortir \triangleq np' = np - 1$

$next \triangleq entrer \lor sortir$

$init \triangleq np = 0$

Exemple de modélisation TLA$^+$

Seconde tentative

$$entrer_2 \triangleq np < 19 \wedge np' = np+1$$
$$next_2 \triangleq entrer_2 \vee sortir$$

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

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

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)`

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`

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)`

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 :

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

- $e(v,w)$ 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