Cours Modélisation et Vérification des Systèmes Informatiques

Formation par apprentissage

Dominique Méry
Telecom Nancy
Université de Lorraine

14 septembre 2021

Sommaire général

Sommaire

Sommaire

Esquisse des cours, TDs et TPs

- Découpage de l'unité : 10 cours, 5 TDs, 5 TPs
- Contenu :
  - Principes de modélisation des systèmes informatiques : systèmes de transition
  - Propriétés d'un système informatique : sûreté, vivacité, disponibilité, sécurité, dépendabilité
  - Modélisation de propriétés de systèmes
  - Vérification de propriétés de systèmes
- Outils
  - TLA/TLA$^+$ avec TLC : document à télécharger à `https://arxiv.org/pdf/1912.10633.pdf` et installation à `https://tla.msr-inria.inria.fr/tlaps/content/Home.html`
  - Event-B avec Rodin : `http://www.event-b.org`
  - Frama-c
- Contrôle des connaissances : deux écrits et un TP

Plan

Modélisation, spécification et vérification

- Modélisation relationnelle d'un programme
- Notations ensemblistes
- Propriétés de programme et pré/post-conditions
- Méthodes de preuve de programmes et principes d'induction
- Techniques de Model-Checking
- Propriétés de système comme sûreté, vivacité, équité, ...
- Outils : Plate-forme TLC, RODIN, PAT

Sommaire général

Sommaire

Sommaire

Detecting overflows of computations

Listing 1 – Function average

```
#include <stdio.h>
#include <limits.h>
int average(int a,int b)
{
  return((a+b)/2);
}

int main()
{
  int x,y;
  x=INT_MAX; y=INT_MAX;
  printf("Average  for %d and %d is %d\n",x,y,
         average(x,y));
  return 0;
}
```

Execution

Execution produces a result

```
Average  for 2147483647 and 2147483647 is -1
```

Execution

Execution produces a result

```
Average  for 2147483647 and 2147483647 is -1
```

Using frama-c produces a required annotation

```
int average(int a, int b)
{
  int __retres;
  /*@ assert rte: signed_overflow: -2147483648 <= a + b; */
  /*@ assert rte: signed_overflow: a + b <= 2147483647; */
  __retres = (a + b) / 2;
  return __retres;
}
```

Annotated Example 1

Listing 2 – Function average

```
#include <stdio.h>
#include <limits.h>
/*@  requires  0 <= a;
     requires a <= INT_MAX ;
     requires  0 <= b;
     requires b <= INT_MAX ;
     requires  0 <= a+b;
     requires a+b <= INT_MAX ;
     ensures \result <= INT_MAX;
*/
int average(int a,int b)
{
  return((a+b)/2);
}

int main()
{
  int x,y;
  x=INT_MAX; y=INT_MAX;
```

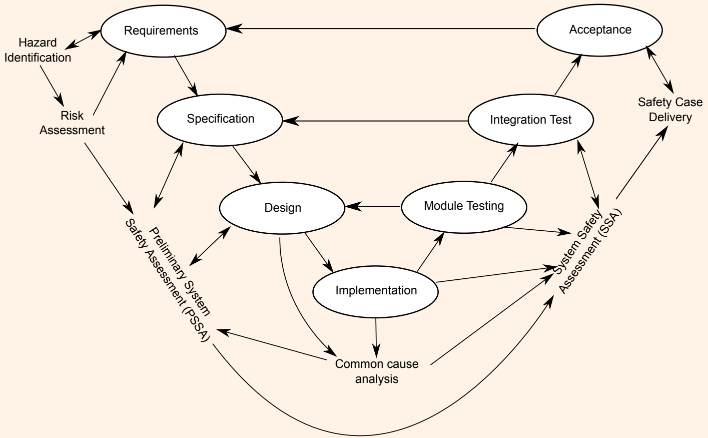
Sommaire général

Sommaire

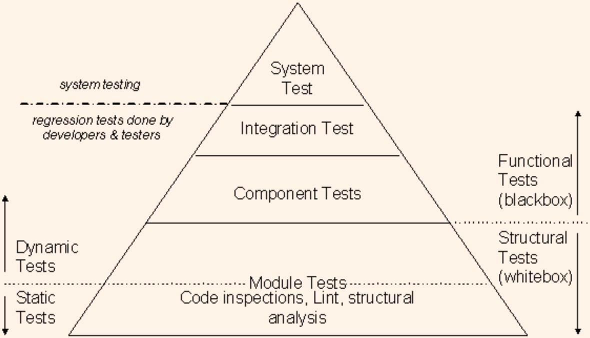
Sommaire

Approches traditionnelles

Spiral Model, Waterfall Model, V-Shaped Model, etc.



Taxonomie des tests



Identification des étapes

- Requirements

Identification des étapes

- Requirements
  - Document of the customer describing a system to develop
  - Functional properties : safety.
  - Non-functional properties : time issues, efficiency, ...
  - Safety and security of the system
  - Contract for the developer

Identification des étapes

- Requirements
  - Document of the customer describing a system to develop
  - Functional properties : safety.
  - Non-functional properties : time issues, efficiency, . . .
  - Safety and security of the system
  - Contract for the developer
- Modelling

Identification des étapes

- Requirements
  - Document of the customer describing a system to develop
  - Functional properties : safety.
  - Non-functional properties : time issues, efficiency, ...
  - Safety and security of the system
  - Contract for the developer
- Modelling
  - Specification : Defining pre/post specification
  - Design : Architecture of the software system
  - Implementation
    - Production of codes
    - Programming Languages
    - Environments
    - Libraries

Identification des étapes

- Requirements
  - Document of the customer describing a system to develop
  - Functional properties : safety.
  - Non-functional properties : time issues, efficiency, . . .
  - Safety and security of the system
  - Contract for the developer
- Modelling
  - Specification : Defining pre/post specification
  - Design : Architecture of the software system
  - Implementation
    - Production of codes
    - Programming Languages
    - Environments
    - Libraries
- Testing

Identification des étapes

- Requirements
  - Document of the customer describing a system to develop
  - Functional properties : safety.
  - Non-functional properties : time issues, efficiency, . . .
  - Safety and security of the system
  - Contract for the developer
- Modelling
  - Specification : Defining pre/post specification
  - Design : Architecture of the software system
  - Implementation
    - Production of codes
    - Programming Languages
    - Environments
    - Libraries
- Testing
  - Unit Testing
  - Integration Testiong
  - Conformance Testing

Challenges

Challenges

- Quality of Software System
- Safety and security of system
- System Engineering
- Composition of components
- Models for access controls

Objectifs

- Un système est un nom désignant une collection ou une classe d'entités de nature très diverse : système de gestion de stocks, système de gestion des données, système de contrôle d'accès à des ressources, système de guidage d'un missile, système de communication, système de production, ...

Objectifs

- Un système est un nom désignant une collection ou une classe d'entités de nature très diverse : système de gestion de stocks, système de gestion des données, système de contrôle d'accès à des ressources, système de guidage d'un missile, système de communication, système de production, ...
- Systèmes à logiciels prépondérants

Objectifs

- Un système est un nom désignant une collection ou une classe d'entités de nature très diverse : système de gestion de stocks, système de gestion des données, système de contrôle d'accès à des ressources, système de guidage d'un missile, système de communication, système de production, ...
- Systèmes à logiciels prépondérants
- Systèmes modélisant un calcul :
  - calcul d'une valeur y à partir d'une donnée x et vérifiant la relation suivante : $y = f(x)$
  - allocation de ressources limitées à un ensemble de processus clients par un processues serveurs
  - gestion du train d'atterissage d'un avion

Objectifs

- Un système est un nom désignant une collection ou une classe d'entités de nature très diverse : système de gestion de stocks, système de gestion des données, système de contrôle d'accès à des ressources, système de guidage d'un missile, système de communication, système de production, ...
- Systèmes à logiciels prépondérants
- Systèmes modélisant un calcul :
  - calcul d'une valeur y à partir d'une donnée x et vérifiant la relation suivante : $y = f(x)$
  - allocation de ressources limitées à un ensemble de processus clients par un processues serveurs
  - gestion du train d'atterissage d'un avion
- Problèmes :
  - Représenter un système
  - Esquisser un système
  - Décrire des propriétés du système
  - Analyser le système
  - Simuler le système
  - Vérifier le système
  - Valider le système
  - Modéliser le système

Etablir des relations

- Vérification : Le programme P répond à la spécification Φ :

Etablir des relations

- Vérification : Le programme P répond à la spécification Φ :

$$P \models \Phi \textbf{ ou } P \textbf{ satisfait ou respecte } \Phi$$

Etablir des relations

- Vérification : Le programme P répond à la spécification Φ :

$$P \models \Phi \text{ ou } P \text{ satisfait ou respecte } \Phi$$

a-t-on bien construit le système ?

Etablir des relations

- Vérification : Le programme P répond à la spécification Φ :

$$P \models \Phi \text{ ou } P \text{ satisfait ou respecte } \Phi$$

a-t-on bien construit le système ?

- Validation : Le programme P fait bien ce qu'on attend de lui :

Etablir des relations

- Vérification : Le programme P répond à la spécification Φ :

  $P \models \Phi$ **ou** P **satisfait ou respecte** Φ

  a-t-on bien construit le système ?

- Validation : Le programme P fait bien ce qu'on attend de lui :

  tests, simulation, animation

Etablir des relations

- Vérification : Le programme P répond à la spécification Φ :

  $P \models \Phi$ **ou** P **satisfait ou respecte** Φ

  a-t-on bien construit le système ?

- Validation : Le programme P fait bien ce qu'on attend de lui :

  tests, simulation, animation

  a-t-on construit le bon système ?

Etablir des relations

- Vérification : Le programme P répond à la spécification Φ :

  $P \models \Phi$ **ou** P **satisfait ou respecte** Φ

  a-t-on bien construit le système ?

- Validation : Le programme P fait bien ce qu'on attend de lui :

  tests, simulation, animation

  a-t-on construit le bon système ?

- Conception : Le programme P est conçu en relation avec une spécification Φ :

Etablir des relations

- Vérification : Le programme P répond à la spécification Φ :

  $P \models \Phi$ **ou** P **satisfait ou respecte** Φ

  a-t-on bien construit le système ?

- Validation : Le programme P fait bien ce qu'on attend de lui :

  tests, simulation, animation

  a-t-on construit le bon système ?

- Conception : Le programme P est conçu en relation avec une spécification Φ :

  correction par construction, raffinement

  La correction par construction vise à produire un programme à partir d'une spécification automatiquement

Sommaire général

Sommaire

Sommaire

AI or Abstract Interpretation

- Static Analysis *computes* approximations
- Abstract Interpretation (AI) provides a mathematical framework for relating *approximations*
- Properties of programs are generally non computable :
  - the halting problem is undecidable
  - Model checking is computing over finite structures
  - Proof assistant may be useful for proving partial correctness or total correctness by applying induction pricniles (see Event B)
  - AI provides another solution by trabfering results from a *concret* framework to an *abstract* structure

Static Analysis of Program Properties

- $\mathcal{CS}(P)$ is the concrete semantics of a program P : the set of reachable states of P.
- $\mathcal{AS}(P)$ is the approximation of $\mathcal{CS}(P)$: $\mathcal{CS}(P) \subseteq \mathcal{AS}(P)$.
- $\mathcal{CS}(P)$ is generally not computable and we will seek for *computable* approximation or abstract semantics $\mathcal{AS}(P)$.
- Problems : $\mathcal{AS}(P)$ may *loose* the expression of properties.

Static Analysis of Program Properties

- φ is a program property stating the possible bugs or arrors which we want to avoid.
- $\mathcal{CS}(P)$ is the concrete semantics of a program P : the set of reachable states of P.
- $\mathcal{AS}(P)$ is the approximation of $\mathcal{CS}(P)$: $\mathcal{CS}(P) \subseteq \mathcal{AS}(P)$.
- Case 1 : $\mathcal{CS}(P) \cap \varphi = \varnothing$ and $\mathcal{AS}(P) \cap \varphi = \varnothing$
- Case 2 : $\mathcal{CS}(P) \cap \varphi \neq \varnothing$ and $\mathcal{AS}(P) \cap \varphi \neq \varnothing$
- Case 3 : $\mathcal{CS}(P) \cap \varphi = \varnothing$ and $\mathcal{AS}(P) \cap \varphi \neq \varnothing$

Static Analysis of Program Properties

- Case 1 : $\mathcal{CS}(P) \cap \varphi = \varnothing$ and $\mathcal{AS}(P) \cap \varphi = \varnothing$:
  - P is safe with respect to φ and no error specified by φ is possible for P.
  - Checking is computable on the approximation
- Case 2 : $\mathcal{CS}(P) \cap \varphi \neq \varnothing$ and $\mathcal{AS}(P) \cap \varphi \neq \varnothing$:
  - An error is detected on the approximation and on the concrete semantics.
  - P is unsafe with respect to φ
  - and an error is detected by the analyser.
- Case 3 : $\mathcal{CS}(P) \cap \varphi = \varnothing$ and $\mathcal{AS}(P) \cap \varphi \neq \varnothing$:
  - P is safe with respect to φ
  - but an error is detected by the analyser
  - A false alarm is provided by the analyzer
  - Approximation is over-approximating P with respect to φ
  - The analysis should be refined

Example of analysis

Algorithm 1 Euclidean Division of Two Natural Numbers

variables : X,Y,Q,R

values : x, y, q, r

precondition : $x, y \in \mathbb{N}$

postcondition: $x, y, r, q \in \mathbb{N} \wedge x = q{\cdot}y{+}r \wedge r < y$

$Q = 0; R = X\,;$

while $R \geq Y$ **do**

$\quad Q = Q{+}1;$

$\quad R := R{-}Y\,;$

Example of analysis

Algorithm 2 Euclidean Division of Two Natural Numbers

$\ell_0 : \{\ldots\}$
$Q = 0$;
$\ell_1 : \{\ldots\}$
$R = X$;
$\ell_2 : \{\ldots\}$
while $R \geq Y$ **do**
  $\ell_3 : \{\ldots\}$
  $Q = Q+1$;
  $\ell_4 : \{\ldots\}$
  $R := R-Y$;
;
$\ell_5 : \{\ldots\}$

- Annotation of the algorithm
- Interpretation of reachable states over the set of possible values of data and variables : $x, y, q, r \in \mathbb{I}$
- $\mathcal{D} = POWERSET(\mathbb{I}^4)$

Example of analysis

Algorithm 3 Euclidean Division of Two Natural Numbers

$\ell_0 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{I}\times\mathbb{I}\}$
$Q = 0\,;$
$\ell_1 : \{\ldots\}$
$R = X\,;$
$\ell_2 : \{\ldots\}$
while $R \geq Y$ **do**
  $\ell_3 : \{\ldots\}$
  $Q = Q{+}1\,;$
  $\ell_4 : \{\ldots\}$
  $R := R{-}Y\,;$
;
$\ell_5 : \{\ldots\}$

$\cdots$

Example of analysis

Algorithm 4 Euclidean Division of Two Natural Numbers

$\ell_0 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{I}\times\mathbb{I}\}$
$Q = 0$;
$\ell_1 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
$R = X$;
$\ell_2 : \{\ldots\}$
while $R \geq Y$ **do**
  $\ell_3 : \{\ldots\}$
  $Q = Q+1$;
  $\ell_4 : \{\ldots\}$
  $R := R-Y$;
;
$\ell_5 : \{\ldots\}$

$\cdots$

Example of analysis

Algorithm 5 Euclidean Division of Two Natural Numbers

$\ell_0 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{I}\times\mathbb{I}\}$
$Q = 0$;
$\ell_1 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
$R = X$;
$\ell_2 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
while $R \geq Y$ **do**
  $\ell_3 : \{\ldots\}$
  $Q = Q+1$;
  $\ell_4 : \{\ldots\}$
  $R := R-Y$;
;
$\ell_5 : \{\ldots\}$

$\cdots$

Example of analysis

Algorithm 6 Euclidean Division of Two Natural Numbers

$\ell_0 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{I}\times\mathbb{I}\}$
$Q = 0\,;$
$\ell_1 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
$R = X\,;$
$\ell_2 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
while $R \geq Y$ **do**
  $\ell_3 : \{\{(x,y,q,r) | (x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I} \wedge r \geq y\}$
  $Q = Q{+}1\,;$
  $\ell_4 : \{\ldots\}$
  $R := R{-}Y\,;$
;
$\ell_5 : \{\ldots\}$

Example of analysis

Algorithm 7 Euclidean Division of Two Natural Numbers

$\ell_0 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{I}\times\mathbb{I}\}$
$Q = 0$;
$\ell_1 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
$R = X$;
$\ell_2 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
while $R \geq Y$ **do**
  $\ell_3 : \{\{(x, y, q, r) | (x, y, q, r) \in \mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I} \wedge r \geq y\}$
  $Q = Q+1$;
  $\ell_4 : \{\{(x, y, q, r) | (x, y, q, r) \in \mathbb{I}\times\mathbb{I}\times\{0, 1\}\times\mathbb{I} \wedge r \geq y\}$
  $R := R-Y$;
;
$\ell_5 : \{\ldots\}$

Example of analysis

Algorithm 8 Euclidean Division of Two Natural Numbers

$\ell_0 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{I}\times\mathbb{I}\}$
$Q = 0\,;$
$\ell_1 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
$R = X\,;$
$\ell_2 : \{\mathbb{I}\times\mathbb{I}\times\{0,1\}\times\mathbb{I}\}$
while $R \geq Y$ **do**
  $\ell_3 : \{\{(x,y,q,r)|(x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I} \wedge r \geq y\}$
  $Q = Q{+}1\,;$
  $\ell_4 : \{\{(x,y,q,r)|(x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\{0,1\}\times\mathbb{I} \wedge r \geq y\}$
  $R := R{-}Y\,;$
;
$\ell_5 : \{\ldots\}$

Example of analysis

Algorithm 9 Euclidean Division of Two Natural Numbers

$\ell_0 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{I}\times\mathbb{I}\}$
$Q = 0\,;$
$\ell_1 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
$R = X\,;$
$\ell_2 : \{\mathbb{I}\times\mathbb{I}\times\{0,1\}\times\mathbb{I}\}$
while $R \geq Y$ **do**
  $\ell_3 : \{\{(x,y,q,r) | (x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\{0,1\}\times\mathbb{I} \wedge r \geq y\}$
  $Q = Q{+}1\,;$
  $\ell_4 : \{\{(x,y,q,r) | (x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\{0,1\}\times\mathbb{I} \wedge r \geq y\}$
  $R := R{-}Y\,;$
;
$\ell_5 : \{\ldots\}$

Example of analysis

Algorithm 10 Euclidean Division of Two Natural Numbers

$\ell_0 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{I}\times\mathbb{I}\}$
$Q = 0\,;$
$\ell_1 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
$R = X\,;$
$\ell_2 : \{\mathbb{I}\times\mathbb{I}\times\{0,1\}\times\mathbb{I}\}$
while $R \geq Y$ **do**
 $\ell_3 : \{\{(x,y,q,r)|(x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\{0,1\}\times\mathbb{I} \wedge r \geq y\}$
 $Q = Q{+}1\,;$
 $\ell_4 : \{\{(x,y,q,r)|(x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\{0,1,2\}\times\mathbb{I} \wedge r \geq y\}$
 $R := R{-}Y\,;$

;
$\ell_5 : \{\ldots\}$

Example of analysis

Algorithm 11 Euclidean Division of Two Natural Numbers

$\ell_0 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{I}\times\mathbb{I}\}$
$Q = 0\,;$
$\ell_1 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
$R = X\,;$
$\ell_2 : \{\mathbb{I}\times\mathbb{I}\times\{0,1,2\}\times\mathbb{I}\}$
while $R \geq Y$ **do**
  $\ell_3 : \{\{(x,y,q,r) | (x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\{0,1\}\times\mathbb{I} \wedge r \geq y\}$
  $Q = Q+1\,;$
  $\ell_4 : \{\{(x,y,q,r) | (x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\{0,1,2\}\times\mathbb{I} \wedge r \geq y\}$
  $R := R-Y\,;$
;
$\ell_5 : \{\ldots\}$

Example of analysis

Algorithm 12 Euclidean Division of Two Natural Numbers

$\ell_0 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{I}\times\mathbb{I}\}$
$Q = 0$;
$\ell_1 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
$R = X$;
$\ell_2 : \{\mathbb{I}\times\mathbb{I}\times\{0,1,2\}\times\mathbb{I}\}$
while $R \geq Y$ **do**
  $\ell_3 : \{\{(x,y,q,r)|(x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\{0,1\}\times\mathbb{I} \wedge r \geq y\}$
  $Q = Q+1$;
  $\ell_4 : \{\{(x,y,q,r)|(x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\{0,1,2\}\times\mathbb{I} \wedge r \geq y\}$
  $R := R-Y$;

;
$\ell_5 : \{\{(x,y,q,r)|(x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\{0,1,2\}\times\mathbb{I} \wedge r < y\}$

Example of analysis

Algorithm 13 Euclidean Division of Two Natural Numbers

$\ell_0 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{I}\times\mathbb{I}\}$
$Q = 0\,;$
$\ell_1 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
$R = X\,;$
$\ell_2 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{N}\times\mathbb{I}\}$
while $R \geq Y$ **do**
  $\ell_3 : \{\{(x,y,q,r)|(x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\mathbb{N}\times\mathbb{I} \wedge r \geq y\}$
  $Q = Q+1\,;$
  $\ell_4 : \{\{(x,y,q,r)|(x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\mathbb{N}\times\mathbb{I} \wedge r \geq y\}$
  $R := R-Y$

$\ell_5 : \{\{(x,y,q,r)|(x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times \mathbb{N}\times\mathbb{I} \wedge r < y\}$

Example of analysis : results of the analysis

- $\ell_0 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{I}\times\mathbb{I}\}$
- $\ell_1 : \{\mathbb{I}\times\mathbb{I}\times\{0\}\times\mathbb{I}\}$
- $\ell_2 : \{\mathbb{I}\times\mathbb{I}\times\mathbb{N}\times\mathbb{I}\}$
- $\ell_3 : \{(x,y,q,r) | (x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\mathbb{N}\times\mathbb{I} \wedge r \geq y\}$
- $\ell_4 : \{(x,y,q,r) | (x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\mathbb{N}\times\mathbb{I} \wedge r \geq y\}$
- $\ell_5 : \{(x,y,q,r) | (x,y,q,r) \in \mathbb{I}\times\mathbb{I}\times\mathbb{N}\times\mathbb{I} \wedge r < y\}$
- Applied Techniques : computing over subsets of $\mathbb{I}\times\mathbb{I}\times\mathbb{I}\times\mathbb{I}$ and calculations over these entities.

Example of analysis : preconditions over x and y

Algorithm 14 Euclidean Division of Two Natural Numbers

$\ell_0 : \{\mathbb{N}\times\mathbb{N}_0\times\mathbb{I}\times\mathbb{I}\}$
$Q = 0$;
$\ell_1 : \{\mathbb{N}\times\mathbb{N}_0\times\{0\}\times\mathbb{I}\}$
$R = X$;
$\ell_2 : \{\mathbb{N}\times\mathbb{N}_0\times\mathbb{N}\times\mathbb{N}\}$
while $R \geq Y$ **do**
    $\ell_3 : \{\{(x,y,q,r)|(x,y,q,r) \in \mathbb{N}\times\mathbb{N}_0\times\mathbb{N}\times\mathbb{N} \wedge r \geq y\}$
    $Q = Q+1$;
    $\ell_4 : \{\{(x,y,q,r)|(x,y,q,r) \in \mathbb{N}\times\mathbb{N}_0\times\mathbb{N}\times\mathbb{N} \wedge r \geq y\}$
    $R := R-Y$
$\ell_5 : \{\{(x,y,q,r)|(x,y,q,r) \in \mathbb{N}\times\mathbb{N}_0\times \mathbb{N}\times\mathbb{N} \wedge r < y\}$

Example of analysis produced by Interproc

Annotated program after forward analysis

```
Annotated program after forward analysis
var Q : int, R : int, X : int, Y : int;
begin
  /* (L2 C5) top */
  Q = 0; /* (L3 C4) [|Q=0|] */
  R = Y; /* (L4 C4) [|Q=0|] */
  if Y > 0 then
     /* (L5 C15) [|Q>=0; Y-1>=0|] */
     while R >= Y do
       /* (L6 C20) [|Q>=0; R-1>=0; Y-1>=0|] */
       Q = Q + 1; /* (L7 C13)
                     [|Q-1>=0; R-1>=0; Y-1>=0|] */
       R = R - Y; /* (L8 C15)
                     [|Q-1>=0; Y-1>=0|] */
     done; /* (L9 C7) [|Q>=0; Y-1>=0|] */
  else
    /* (L10 C4) [|Q=0; -Y>=0|] */
    skip; /* (L11 C7) [|Q=0; -Y>=0|] */
  endif; /* (L12 C6) [|Q>=0|] */
end
```

Example of analysis produced by Interproc

Annotated program after forward analysis

```
Annotated program after forward analysis
var Q : int, R : int, X : int, Y : int;
begin
  /* (L2 C5) top */
  Q = 0; /* (L3 C4) [|Q>=0; -Q+11>=0|] */
  while Q <= 10 do
    /* (L4 C19) [|Q>=0; -Q+10>=0|] */
    Q = Q + 1; /* (L5 C13)
                  [|Q-1>=0; -Q+11>=0|] */
  done; /* (L6 C5) [|Q-11=0|] */
end
```

Problèmes

Problèmes

- Représenter un système

Problèmes

- Représenter un système
- Esquisser un système

Problèmes

- Représenter un système
- Esquisser un système
- Décrire des propriétés du système

Problèmes

- Représenter un système
- Esquisser un système
- Décrire des propriétés du système
- Analyser le système

Problèmes

- Représenter un système
- Esquisser un système
- Décrire des propriétés du système
- Analyser le système
- Simuler le système

Problèmes

- Représenter un système
- Esquisser un système
- Décrire des propriétés du système
- Analyser le système
- Simuler le système
- Vérifier le système

Problèmes

- Représenter un système
- Esquisser un système
- Décrire des propriétés du système
- Analyser le système
- Simuler le système
- Vérifier le système
- Valider le système

Problèmes

- Représenter un système
- Esquisser un système
- Décrire des propriétés du système
- Analyser le système
- Simuler le système
- Vérifier le système
- Valider le système
- Modéliser le système

Problèmes

- Représenter un système
- Esquisser un système
- Décrire des propriétés du système
- Analyser le système
- Simuler le système
- Vérifier le système
- Valider le système
- Modéliser le système
- Modéliser le domaine ou les domaines du problème

Problèmes

- Représenter un système
- Esquisser un système
- Décrire des propriétés du système
- Analyser le système
- Simuler le système
- Vérifier le système
- Valider le système
- Modéliser le système
- Modéliser le domaine ou les domaines du problème
- Modéliser l'environnement :

Problèmes

- Représenter un système
- Esquisser un système
- Décrire des propriétés du système
- Analyser le système
- Simuler le système
- Vérifier le système
- Valider le système
- Modéliser le système
- Modéliser le domaine ou les domaines du problème
- Modéliser l'environnement : physique,

Problèmes

- Représenter un système
- Esquisser un système
- Décrire des propriétés du système
- Analyser le système
- Simuler le système
- Vérifier le système
- Valider le système
- Modéliser le système
- Modéliser le domaine ou les domaines du problème
- Modéliser l'environnement : physique, biologie,

Problèmes

- Représenter un système
- Esquisser un système
- Décrire des propriétés du système
- Analyser le système
- Simuler le système
- Vérifier le système
- Valider le système
- Modéliser le système
- Modéliser le domaine ou les domaines du problème
- Modéliser l'environnement : physique, biologie, épistémologie,

Problèmes

- Représenter un système
- Esquisser un système
- Décrire des propriétés du système
- Analyser le système
- Simuler le système
- Vérifier le système
- Valider le système
- Modéliser le système
- Modéliser le domaine ou les domaines du problème
- Modéliser l'environnement : physique, biologie, épistémologie, éthique,

Problèmes

- Représenter un système
- Esquisser un système
- Décrire des propriétés du système
- Analyser le système
- Simuler le système
- Vérifier le système
- Valider le système
- Modéliser le système
- Modéliser le domaine ou les domaines du problème
- Modéliser l'environnement : physique, biologie, épistémologie, éthique,santé,...

Sommaire général

Sommaire

Sommaire

Sommaire général

Sommaire

Sommaire

Spécification d'un système

PROCEDURE PROC(X; **VAR** Y)
PRECONDITION P(X)
POSTCONDITION Q(X,Y)

- P(X) : spécification de ce que doivent satisfaire les données
- Q(X,Y) : spécification de la relation entre les données et les résultats attendus
- Calcul modélisé par $R(x,y)$: $\forall x,y.P(x) \wedge R(x,y) \Rightarrow Q(x,y)$:
  - $z=(x,y,t)$: l'état courant est défini par une liste de variables contenant les valeurs des données (P(x)), les valeurs des résultats (Q(x,y)) et les valeurs des variables temporaires ou locals appelées t·
  - au point *init* : $P(x_0) \wedge x=x_0 \wedge y \in D_2 \wedge t \in D_3 \wedge z=z_0$ noté aussi $S(z_0)$
  - la relation entre l'état initial est l'état courant est alors $S(x_0,y_0,t_0) \xrightarrow{R(x,y)} S(x_f,y_f,t_f)$ où f désigne le dernier état de ce calcul
  - $P(x_0) \wedge (S(x_0,y_0,t_0) \xrightarrow{R(x,y)} S(x_f,y_f,t_f)) \Rightarrow Q(x_0,y_0,t_0,x_f,y_f,t_f)$

Spécification d'un système

```
PROCEDURE PROC(X; VAR Y)
PRECONDITION P(X)
POSTCONDITION Q(X,Y)
BEGIN
code calculant R(x, y)
END
```

- Calcul de $R(x, y)$: donner une signification ou un sens à un code C :
  - *Sémantique opérationnelle* : le code ou programme est vu comme un ensemble de relations élémentaires permettant de calculer par combinaison de ces relations élémentaires et par fermeture du graphe de ces relations.
  - *Sémantique dénotationnelle* : le code ou le programme est lié à une fonction calculant les valeurs du calcul du programme
  - *Sémantique axiomatique* : le code ou le programme est cacatérisé par des axiomes et des règles d'inférence Logique de Hoare
- $\mathcal{M}(C)(x) = y$ si, et seulement si, $R(x, y)$
- $x \xrightarrow{C} y$ si, et seulement si, $R(x, y)$
- $\{P\}C\{Q\}$ si, et seulement si, $R(x, y)$.

Programme annoté

```
#include<stdio.h>
int main()
{
int a=1,b=2;  // D\'eclarer les variables
int somme = 0;          // D\'eclarer somme
// l1: a=1 & b=2
somme = a + b;  // Calculer la somme
// l2: somme = a + b & a = 1 & b = 2
 printf("La valeur de la somme  de %d+%d  est: %d\n",a,b,somme);
// l3: somme = 3
return(0);
}
```

Programme annoté

$$
\begin{array}{l}
//\ell_1 : \{P_{\ell_1}(x)\} \\
\quad \textbf{FOR } i := 1 \ \textbf{ TO } \ n \textbf{ DO} \\
\qquad \ell_2 : \{P_{\ell_2}(i, x)\} \\
\qquad\quad S(x); \\
\qquad \ell_3 : \{P_{\ell_3}(i, x)\} \\
\quad \textbf{ENDFOR} \\
\ell_4 : \{P_{\ell_4}(x)\}
\end{array}
$$

Programme annoté

$$
\begin{array}{l}
//\ell_1 : \{P_{\ell_1}(x)\} \\
\quad \textbf{FOR } i := 1 \ \ \textbf{TO} \ \ n \ \textbf{DO} \\
\qquad \ell_2 : \{P_{\ell_2}(i,x)\} \\
\qquad\quad S(x); \\
\qquad \ell_3 : \{P_{\ell_3}(i,x)\} \\
\quad \textbf{ENDFOR} \\
\ell_4 : \{P_{\ell_4}(x)\}
\end{array}
$$

Oui mais il faut ajouter des conditions de vérification. . .

Programme annoté

$$
\begin{array}{l}
//\ell_1 : \{P_{\ell_1}(x)\} \\
\quad \textbf{FOR } i := 1 \ \textbf{ TO } \ n \textbf{ DO} \\
\qquad \ell_2 : \{P_{\ell_2}(i,x)\} \\
\qquad\quad S(x); \\
\qquad \ell_3 : \{P_{\ell_3}(i,x)\} \\
\quad \textbf{ENDFOR} \\
\ell_4 : \{P_{\ell_4}(x)\}
\end{array}
$$

Oui mais il faut ajouter des conditions de vérification...

$$
\begin{array}{l}
(1)\ c = \ell_1 \wedge P_{\ell_1}(x) \wedge 1 \leq n \wedge c' = \ell_2 \wedge i' = 1 \wedge x' = x \Rightarrow c' = \ell_2 \wedge P_{\ell_2}(i',x' \\
(2)\ c = \ell_1 \wedge P_{\ell_1}(x) \wedge \neg(1 \leq n) \wedge c' = \ell_4 \wedge x' = x \Rightarrow c' = \ell_4 \wedge P_{\ell_4}(x') \\
(3)\ c = \ell_3 \wedge P_{\ell_3}(x,i) \wedge i{+}1 \leq n \wedge c' = \ell_2 \wedge i' = i{+}1 \wedge x' = x \Rightarrow c' = \ell_2 \wedge P_{\ell} \\
(4)\ c = \ell_2 \wedge P_{\ell_3}(x,i) \wedge \neg(i{+}1 \leq n) \wedge c' = \ell_4 \wedge x' = x \wedge i' = i{+}1 \Rightarrow c' = \ell_4
\end{array}
$$

Sommaire général

Sommaire

Sommaire

Modèle relationnel d'un système

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

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

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.
- $\textrm{VALS}$ 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'.
- $\textrm{INIT}(x)$ définit l'ensemble des valeurs initiales de x.
- la relation r_0 est la relation $Id[\textrm{VALS}]$, identité sur $\textrm{VALS}$.

Definition

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

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

pour une variable x, nous définissons les valeurs suivantes :

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

Propriétés de sûreté et d'invariance dans un modèle relationnel

Definition

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

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

Exemple de modélisation TLA$^+$

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*

Exemple de modélisation TLA$^+$

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

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

Exemple de modélisation TLA$^+$

Troisième tentative

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

$next_3 \triangleq entrer_2 \lor sortir2$

$safety_1 \triangleq np \leq 19$

$question_1 \triangleq np \neq 6$

```
------------------------------- 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
----------------------------------------------------------------------------
(* 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
============================================================================
```

Traduction de la définition

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

- x est une variable ou une liste de variable : `VARIABLES x`
- $Init(x)$ est une variable ou une liste de variable : `init == Init(x)`
- $\text{NEXT}^\star(x_0, x)$ 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)`

Propriété universelle d'un système discret

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

$\{u | u \in \text{VALS} \wedge (\exists x_0.x_0 \in \text{VALS} \wedge Init(x_0) \wedge \text{NEXT}^\star(x_0, u))\}$ est l'ensemble des états accessibles à partir des états initiaux que nous noterons $\text{REACHABLE}(M)$.

Les expressions suivantes sont équivalentes :

- $\forall x_0, x \in \text{VALS}.Init(x_0) \wedge \text{NEXT}^\star(x_0, x) \Rightarrow A(x)$
- $\forall x \in \text{VALS}.(\exists x_0.x_0 \in \text{VALS} \wedge Init(x_0) \wedge \text{NEXT}^\star(x_0, x)) \Rightarrow A(x)$
- $\text{REACHABLE}(M) \subseteq \{x \in \text{VALS} | A(x)\}$

Exploration exhaustive des états accessibles

Vérification de l'inclusion

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

Méthode de vérification

- Génération de l'ensemble $\textsc{reachable}(M)$ à l'aide de la condition initiale *Init* et de la relation de calcul *Next*
- Test de la condition *A(x)* pour chaque nouvelle valeur x engendrée.

Observation

- Le calcul de $\textsc{reachable}(M)$ est celui de la composante connexe d'un graphe et ce calcul peut être très complexe et nécessiter des moyens de calcul puissants.
- $\textsc{reachable}(M)$ peut être infini et la vérification revient à parcourir ou engendrer un ensemble infini pendant un temps infini !...
- $\textsc{reachable}(M) \cap \{x \in \textsc{Vals} | \neg A(x)\} = \varnothing$ est une autre façon de poser la question de l'inclusion et l'exploration peut produire une valeur ne satisfaisint pas $A(x)$: la méthode produit un contre-exemple.

Méthode de vérification par exploration exhaustive des états accessibles

Deux cas possibles

- Cas 1 : $\textsc{reachable}(M) \subseteq \{x \in \textsc{Vals} | A(x)\}$ ou $\textsc{reachable}(M) \cap \{x \in \textsc{Vals} | \neg A(x)\} = \varnothing$: la propriété $A(x)$ est une propriété de sûreté du système modélisé par *Init* et *Next* pour la variable x.
- Cas 2 : $v \in \textsc{reachable}(M) \cap \{x \in \textsc{Vals} | \neg A(x)\}$: v est un contre-exemple et la propriété $A(x)$ n'est pas vérifiée par le modèle construit.

Sommaire général

Sommaire

Sommaire

Méthode de vérification par exploration exhaustive des états accessibles

- Correction partielle : quand le calcul atteint un état de contrôle valant halt, la variable x satisfait $post(x)$ et la propriété à évaluer est $c = halt \implies post(x)$
- Absence d'erreurs à l'exécution : quel que soit l'état de la variable x au cours du calcul sa valeur est comprise entre min et max et la propriété à évaluer est $x \in min..max$.
- Accessibilité d'un état du système modélisé : la valeur de x peut satisfaire le prédicat $P(x)$ au cours du calcul et la propriété à tester est $\neg P(x)$.

Sommaire général

Sommaire

Sommaire

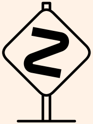
Méthode de preuves de propriétés d'invariance

$\textsc{reachable}(M) = \{x \in \textsc{Vals} | \exists x_0 . x_0 \in \textsc{Vals} \wedge \mathit{Init}(x_0) \wedge \textsc{Next}^\star(x_0, x)\}$

- $\textsc{reachable}(M) \subseteq \{x \in \textsc{Vals} | A(x)\}$
- ou de manière équivalente
- $(\exists x_0 . x_0 \in \textsc{Vals} \wedge \mathit{Init}(x_0) \wedge \textsc{Next}^\star(x_0, x)) \implies A(x)$

Objectifs

- La vérification de l'inclusion n'est pas toujours possible en temps fini.
- Explorer les mécanismes de raisonnement déductif pour montrer l'inclusion.



Introduire un principe d'induction sur les transitions d'états

Propriété universelle d'un système discret

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

$\{u | u \in \textsc{Vals} \wedge (\exists x_0.x_0 \in \textsc{Vals} \wedge Init(x_0) \wedge \textsc{Next}^\star(x_0, u))\}$ est l'ensemble des états accessibles à partir des états initiaux que nous noterons $\textsc{reachable}(M)$.

Les deux expressions suivantes sont équivalentes :

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

Principe 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 :

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

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 (condition suffisante)

Soit une propriété $I(x)$ telle que :

$$\forall x, x' \in \textrm{VALS} \ : \left\{ \begin{array}{l} (1)\ Init(x) \Rightarrow I(x) \\ (2)\ I(x) \Rightarrow A(x) \\ (3)\ I(x) \wedge \textrm{NEXT}(x,x') \Rightarrow I(x') \end{array} \right.$$

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

Soient x_0 et $x \in \textrm{VALS}$ tels que $INIT(x_0) \wedge \textrm{NEXT}^{\cdot}(x_0, x)$.

- On peut construire une suite telle que : $x_0 \underset{\textrm{NEXT}}{\longrightarrow} x_1 \underset{\textrm{NEXT}}{\longrightarrow} x_2 \underset{\textrm{NEXT}}{\longrightarrow} \ldots \underset{\textrm{NEXT}}{\longrightarrow} (x_i = x)$.
- 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 (condition nécessaire)

$\forall x_0, x \cdot x_0, x \in \textsc{Vals} \wedge Init(x_0) \wedge \textsc{Next}^{\star}(x_0, x) \Rightarrow A(x)$
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 :
  $I(x) \ \widehat{=} \ \exists x_0 \in \textsc{Vals} \cdot Init(x_0) \wedge \textsc{Next}^{\star}(x_0, x)$.
- $I(x)$ 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 $\mathcal{A}$.

Une équivalence utile

Les deux énoncés suivants sont équivalents :
(I) Il existe une propriété d'état $\mathrm{I}(x)$ telle que :

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

(II) Il existe une propriété d'état $\mathrm{I}(x)$ telle que :

$$\forall x, x' \in \textsc{Vals} : \begin{cases} (1) & \textsc{Init}(x) \Rightarrow \mathrm{I}(x) \\ (2) & \mathrm{I}(x) \Rightarrow \mathrm{A}(x) \\ (3) & \forall i \in \{0, \ldots, n\} : \mathrm{I}(x) \wedge x \ r_i \ x' \Rightarrow \mathrm{I}(x') \end{cases}$$

La preuve est immédiate en appliquant la règle suivante :
$\forall i \in \{0, \ldots, n\} : A \wedge x \ r_i \ x' \Rightarrow B \equiv (A \wedge (\exists i \in \{0, \ldots, n\} : x \ r_i \ x')) \Rightarrow B$
et la définition de $\textsc{Next}(x, x')$.
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.

Definition

invariant d'un système S Une propriété $I(x)$ est un invariant inductif d'un système S défini par un modèle M, si

$$\forall x, x' \in \textsc{Vals} : \begin{cases} (1) & \textsc{Init}(x) \Rightarrow \mathrm{I}(x) \\ (2) & \forall i \in \{0, \ldots, n\} : \mathrm{I}(x) \wedge x \ r_i \ x' \Rightarrow \mathrm{I}(x') \end{cases}$$

Les deux énoncés suivants sont équivalents :

- $\forall x, x' \in \textsc{Vals} : \mathrm{I}(x) \wedge x \ r_i \ x' \Rightarrow \mathrm{I}(x')$ (Hoare)
- $\forall x' \in \textsc{Vals} : (\exists x \in \textsc{Vals} : \mathrm{I}(x) \wedge x \ r_i \ x') \Rightarrow \mathrm{I}(x')$ (Floyd)

La preuve est immédiate en appliquant la règle suivante :
$\forall u.P(u) \Rightarrow Q \equiv (\exists u.P(u)) \Rightarrow Q$

Conception d'une méthode de preuves de propriétés d'invariance et de sûreté pour un langage de programmation

- On considère un langage de programmation classique noté $\textsc{Programs}$
- 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 $\textsc{Programs}$; ce programme comprend
  - des variables notées globalement v,
  - des constantes notées globalement c,
  - des types associés aux variables notés globalement $\textsc{Vals}$ et identifiés à un ensemble de valeurs possibles des variables,
  - des instructions suivant un ordre défini par la syntaxe du langage de programmation.

- 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 ℓ 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 $v \in \textsc{Memory}$.

$$x = (pc, v) \wedge pc \in \textsc{Locations} \wedge v \in \textsc{Memory} \tag{1}$$

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 $x = (pc_0, v)$.

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

Flôt du contrôle

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

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 $P_\ell(v)$.

Propriété de sûreté

Soit $(Th(s,c), x, \text{VALS}, \text{INIT}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel pour ce programme. Une propriété $\mathrm{A}(x)$ est une propriété de sûreté pour P, si $\forall y, x \in \text{LOCATIONS} \times \text{MEMORY}. Init(y) \wedge \text{NEXT}^{\star}(y,x) \Rightarrow A(x)$. On sait que cette propriété implique qu'il existe une propriété d'état $\mathrm{I}(x)$ telle que :

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

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

Relation entre un Invariant et des annotations

Il existe une famille de propriétés $\{P_\ell(v) : \ell \in \textsc{Locations}\}$ satisfaisant l'équivalence suivante : $\mathrm{I}(x) \equiv \bigvee_{\ell \in \textsc{Locations}} \left(\bigvee_{v \in \textsc{Memory}} x = (\ell, v) \wedge P_\ell(v) \right)$.

Il existe une famille de propriétés $\{P_\ell(v) : \ell \in \textsc{Locations}\}$ satisfaisant les propriétés suivantes

- $\mathrm{J}(pc, v) \stackrel{def}{=} \left(\begin{array}{l} \bigwedge_{\ell \in \textsc{Locations}} (pc = \ell \Rightarrow P_\ell(v)) \\ \wedge\ pc \in \textsc{Locations} \\ \wedge\ v \in \textsc{Memory} \end{array} \right.$.
- $I(x) \equiv \exists pc \in LOC, v \in \textsc{Memory}.x = (pc, v) \wedge \mathrm{J}(pc, v)$
- $\mathrm{J}(pc, v) \equiv \exists x \in \textsc{Vals}.x = (pc, v) \wedge I(x)$

Sous la relation $x = (\ell, v)$, nous avons donc les relations suivantes entre $I(x)$ et les annotations $P_\ell(v)$:

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

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 :
$cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v)$
la transition de ℓ à ℓ' est possible quand la condition $cond_{\ell,\ell'}(v)$ est vraie pour v et quand elle a lieu, les variables v sont transformées comme suit $v' = f_{\ell,\ell'}(v)$. On a donc la relation suivante $r_{\ell,\ell'}$ de transition :

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

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

$$Init(\ell,v) \stackrel{def}{=} \ell \in \textsc{InputLocations} \wedge \mathbf{pre}(\mathrm{P})(v).$$

Conditions de vérification

Conditions initiales

Les deux propriétés suivantes sont équivalentes :

- $\forall x \in \textsc{Vals} : \textsc{Init}(x) \Rightarrow \mathrm{I}(x)$
- $\forall \ell \in \textsc{InputLocations}, v \in \textsc{Memory}.\mathbf{pre}(\mathrm{P})(v) \Rightarrow P_\ell(v)$

Pas d'induction

Les deux propriétés suivantes sont équivalentes :

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

Conclusion

Les deux propriétés suivantes sont équivalentes :

- $\mathrm{I}(x) \Rightarrow \mathrm{A}(x)$ ou $\forall \ell \in \textsc{Locations}.P_\ell(v) \Rightarrow \mathrm{A}(\ell, v)$

Les conditions de vérification suivantes sont équivalentes :

- $\forall x, x' \in \textsc{Locations} \times \textsc{Memory}$:
$$\left\{\begin{array}{ll} (1) & \textsc{Init}(x) \Rightarrow \mathrm{I}(x) \\ (2) & \mathrm{I}(x) \Rightarrow \mathrm{A}(x) \\ (3) & \forall i \in \{0, \ldots, n\} : \mathrm{I}(x) \wedge x \ r_i \ x' \Rightarrow \mathrm{I}(x') \end{array}\right.$$
- $\forall v, v' \in \textsc{Memory}$:
$$\left\{\begin{array}{ll} (1) & \forall \ell \in \textsc{InputLocations}.\mathbf{pre}(\mathrm{P})(v) \Rightarrow P_\ell(v) \\ (2) & \forall \ell \in \textsc{Locations}.P_\ell(v) \Rightarrow \mathrm{A}(\ell, v) \\ (3) & \forall \ell, \ell' \in \textsc{Locations} : \\ \multicolumn{2}{l}{\ell \longrightarrow \ell' \Rightarrow P_\ell(v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v')} \end{array}\right.$$

Conditions de vérification

On en déduit une méthode de preuve de correction de propriétés de sûreté générale.

Méthode de correction de propriétés de sûreté

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

$$\left\{\begin{array}{l} (1)\;\; \forall \ell \in \textsc{InputLocations}.\textsc{Precondition}(v) \Rightarrow P_\ell(v) \\ (2)\;\; \forall \ell \in \textsc{Locations}.P_\ell(v) \Rightarrow \mathrm{A}(\ell, v) \\ (3)\;\; \forall \ell, \ell' \in \textsc{Locations} : \\ \ell \longrightarrow \ell' \Rightarrow P_\ell(v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v') \end{array}\right.,$$

alors $\mathrm{A}(\ell, v)$ est une propriété de sûreté pour le programme P.

Condition de vérification

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

Floyd and Hoare

- $\forall v, v' \in \textsc{Memory}.\forall \ell, \ell' \in \textsc{Locations}.\ell \longrightarrow \ell' \Rightarrow P_\ell(v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v')$ est équivalent à $\forall \ell, \ell' \in \textsc{Locations}.\ell \longrightarrow \ell' \Rightarrow \forall v' \in \textsc{Memory}.P_\ell(v) \wedge cond_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v \mapsto f_{\ell,\ell'}(v))$
- $\forall v, v' \in \textsc{Memory}.\forall \ell, \ell' \in \textsc{Locations}.\ell \longrightarrow \ell' \Rightarrow P_\ell(v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v')$ est équivalent à $\forall \ell, \ell' \in \textsc{Locations}.\ell \longrightarrow \ell' \Rightarrow \forall v' \in \textsc{Memory}.(\exists v \in \textsc{Memory}.P_\ell(v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v)) \Rightarrow P_{\ell'}(v')$

Condition de vérification pour l'affectation

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

$$\boxed{\begin{array}{l} \ell : P_\ell(v) \\ v := f_{\ell,\ell'}(v) \\ \ell' : P_{\ell'}(v) \end{array}}$$

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

Conditions de vérification pour l'itération

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

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

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

Conditions de vérification pour la conditionnelle

$$
\begin{array}{l}
\ell_1 : P_{\ell_1}(v) \\
\textbf{IF} \quad B(v) \quad \textbf{THEN} \\
\quad \ell_2 : P_{\ell_2}(v) \\
\quad \ldots \\
\quad \ell_3 : P_{\ell_3}(v) \\
\textbf{ELSE} \\
\quad m_2 : P_{\ell_2}(v) \\
\quad \ldots \\
\quad m_3 : P_{\ell_3}(v) \\
\textbf{FI} \\
\ell_4 : P_{\ell_4}(v)
\end{array}
$$

Pour la structure de conditionnelle, les conditions
suivantes :

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

Propriétés de correction des programmes

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**$(\mathrm{P})(v)$ est la postcondition de P pour v ; elle caractérise les valeurs finales de v.

Exemple

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

$$\boxed{\begin{array}{l} \forall \underline{x}, \underline{y}, \underline{r}, \underline{q}, \overline{x}, \overline{y}, \overline{r}, \overline{q}. \\ \quad \mathbf{pre}(\mathrm{P})(\underline{x}, \underline{y}, \underline{r}, \underline{q}) \wedge (\underline{x}, \underline{y}, \underline{r}, \underline{q}) \xrightarrow{\mathrm{P}} (\overline{x}, \overline{y}, \overline{r}, \overline{q}) \\ \qquad \Rightarrow \mathbf{post}(\mathrm{P})(\underline{x}, \underline{y}, \underline{r}, \underline{q}, \overline{x}, \overline{y}, \overline{r}, \overline{q}) \end{array}}$$

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**$(\mathrm{P})(\mathrm{v})$
- la spécification des résultats de P **post**$(\mathrm{P})(\mathrm{v})$
- une famille d'annotations de propriétés $\{P_\ell(v) : \ell \in \textsc{Locations}\}$ pour ce programme.
- une propriété de sûreté définissant la correction partielle $\ell = \mathit{output} \Rightarrow \mathbf{post}(\mathrm{P})(v)$ où *output* est l'étiquette marquant la fin du programme P

Definition

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

Si les conditions suivantes sont vérifiées :

- $\forall \ell \in \textsc{InputLocations}.\forall v, v' \in \textsc{Memory}.\mathbf{pre}(\mathrm{P})(v) \Rightarrow P_\ell(v)$
- $\forall \ell \in \textsc{Outputs}.\forall v, v' \in \textsc{Memory}.P_\ell(v) \Rightarrow \mathbf{post}(\mathrm{P})(v)$
- $\forall \ell, \ell' \in \textsc{Locations} : \ell \longrightarrow \ell' : \forall v, v' \in$ $\textsc{Memory}.\,(P_\ell(v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v'))$,

alors le programme P est partiellement correct par rapport à $\mathbf{pre}(\mathrm{P})(\mathrm{v})$ et $\mathbf{post}(\mathrm{P})(\mathrm{v})$.

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 $y := x+1$:
   $\mathbf{DOM}(x+1)(x,y) \stackrel{def}{=} \mathbf{DOM}(x)(x,y) \wedge x+2 \in int$

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 $\mathbf{pre}(\mathrm{P})(\mathsf{v})$
- la spécification des résultats de P $\mathbf{post}(\mathrm{P})(\mathsf{v})$
- 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 $\mathbf{pre}(\mathrm{P})(\mathsf{v})$ et $\mathbf{post}(\mathrm{P})(\mathsf{v})$, 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.

Si les conditions suivantes sont vérifiées :

$$\begin{cases} (1)\ \forall \ell \in \mathrm{INPUTLOCATIONS}.\forall v, v' \in \mathrm{MEMORY}.\mathbf{pre}(\mathrm{P})(v) \Rightarrow P_\ell(v) \\ (2)\ \forall m \in \mathrm{LOCATIONS}-\{output\}, n \in \mathrm{LOCATIONS}, \forall v, v' \in \mathrm{MEMORY} : \\ \qquad\qquad m \longrightarrow n : P_m(v) \Rightarrow \mathbf{DOM}(m,n)(v) \\ (3)\ \forall \ell, \ell' \in \mathrm{LOCATIONS}, \forall v, v' \in \mathrm{MEMORY} : \ell \longrightarrow \ell' : (P_\ell(v) \wedge cond_{\ell,\ell'}(v) \wedge \end{cases}$$

alors le programme P ne produira pas d'erreurs à l'exécution par rapport à **pre**$(\mathrm{P})(\mathrm{v})$ et **post**$(\mathrm{P})(\mathrm{v})$.

Traduction des conditions en TLA+

Sommaire général

Sommaire

Sommaire

Substitution généralisée

EVENT e
  ANY t
  WHERE
    $G(c, s, t, x)$
  THEN
    $x :| (P(c, s, t, x, x'))$
  END

- c et s désignent les constantes et les ensembles visibles par l'événement e et sont définis dans un contexte.
- x est la variable d'état ou une liste de variables.
- $G(c, s, t, x)$ est la condition d'activation de e.
- $P(c, s, t, x, x')$ est le prédicat établissant la relation entre la valeur avant de x, notée x, et la valeur après de x, notée x'.
- $BA(e)(c, s, x, x')$ est la relation *before-after* associée à e et définie par $\exists t.G(c, s, t, x) \ \wedge P(c, s, t, x, x')$.

Propriétés d'invariance en Event-B

- L'invariant $I(x)$ d'un modèle est une propriété invariante pour tous les événements du système modélisé y compris l'événement initial.
- Si e est un événement du modèle, alors la condition de préservation de cet invariant par cet événement est la suivante :
  $I(x) \ \wedge \ BA(e)(c,s,x,x') \ \Rightarrow \ I(x') \quad (INV)$.
- $I(x)$ est écrit sous la forme d'une liste de prédicats étiquettés $inv_1, \ldots inv_n$ et est interprétée comme une conjonction.
- Conditions initiales : $Init(x,s,c) \ \Rightarrow \ I(x) \quad (INIT)$.
- Faisabilité : $I(x) \ \wedge \ \mathrm{grd}\,(e) \ \Rightarrow \ \exists x' \cdot BA(e)(c,s,x,x') \qquad (FIS)$.
- Propriétés de sûreté : $C(s,c) \ \wedge \ I(x) \ \Rightarrow \ A(s,c,x) \qquad (THM)$.

Forme simple d'un événement

- Un événement simple est de la forme suivante :

$$
\boxed{
\begin{array}{l}
< event\_name > \; \widehat{=} \\
\quad \textbf{WHEN} \\
\qquad < condition > \\
\quad \textbf{THEN} \\
\qquad < action > \\
\quad \textbf{END}
\end{array}
}
$$

where
- $<$ *event_name* $>$ est un identificateur
- $<$ *condition* $>$ est la condition de déclencehement de l'événement
- $<$ *action* $>$ est une subtitution généralisée d'affectation

Forme non-déterminististe d'un événement

- Un événement non-déterminste est de la forme suivante :

$$
\begin{array}{l}
< event\_name > \ \widehat{=} \\
\quad \textbf{ANY} \ < variable > \ \textbf{WHERE} \\
\qquad < condition > \\
\quad \textbf{THEN} \\
\qquad < action > \\
\quad \textbf{END}
\end{array}
$$

where
- $< event\_name >$ est u identificateur
- $< variable >$ est une liste de variables
- $< condition >$ est la condition de déclenchement de l'événement
- $< action >$ est une substitution généralsée

Forme d'une susbstitution généralisée

A generalized substitution can be

- affectation simple : $x := E$
- affectation généralisée : $x : |P(x, x')$
- affectation ensembliste : $x :\in S$

T

,

- composition parallèle : $\cdots$

U

Vérification de la préservation d'invariant(0)

$$\text{INVARIANT} \quad \wedge \quad \text{GUARD}$$
$$\Longrightarrow$$
$$\text{ACTION } \text{establishes } \text{INVARIANT}$$

Vérification de la préservation d'invariant (1)

- Soit un événement de la forme :

$$\begin{array}{l} \mathrm{EVENT}\ \textsf{EVENT} \quad \widehat{=} \\ \quad \textbf{WHEN} \\ \qquad G(x) \\ \quad \textbf{THEN} \\ \qquad x := E(x) \\ \quad \textbf{END} \end{array}$$

et l'invariant $I(x)$ à préserver, l'instruction à prouver est :

$$\boxed{I(x) \ \wedge \ G(x) \ \Longrightarrow \ I(E(x))}$$

Vérification de la préservation d'invariant (2)

- Soit un événement de la forme :

$$
\begin{array}{l}
\mathrm{EVENT}\ \mathsf{EVENT}\ \ \widehat{=} \\
\quad \textbf{WHEN} \\
\qquad G(x) \\
\quad \textbf{THEN} \\
\qquad x :| P(x, x') \\
\quad \textbf{END}
\end{array}
$$

et un invariant $I(x)$ à préserver, l'instruction à prouver est :

$$\boxed{I(x) \ \wedge \ G(x) \ \wedge \ P(x, x') \ \Longrightarrow \ I(x')}$$

Vérification de la préservation d'invariant (3)

- Soit un événement de la forme :

$$\begin{array}{l} \mathrm{EVENT}\ \mathsf{EVENT} \quad \widehat{=} \\ \quad \textbf{WHEN} \\ \qquad G(x) \\ \quad \textbf{THEN} \\ \qquad x :\in S(x) \\ \quad \textbf{END} \end{array}$$

et l'invariant $I(x)$ à préserver, la propriététét à prouver est :

$$\boxed{I(x) \quad \wedge \quad G(x) \quad \wedge \quad x' \in S(x) \quad \Longrightarrow \quad I(x')}$$

Vérification de la condition de vérification (4)

- soit un événement de la forme non-déterministe :

$$\boxed{\begin{array}{l} \mathrm{EVENT}\ \mathsf{EVENT} \quad \widehat{=} \\ \quad \textbf{ANY}\ v\ \textbf{WHERE} \\ \qquad G(x,v) \\ \quad \textbf{THEN} \\ \qquad x := E(x,v) \\ \quad \textbf{END} \end{array}}$$

et l'invariant $I(x)$ à préserver, la condition à prouver est :

$$\boxed{I(x) \quad \wedge \quad G(x,v) \quad \Longrightarrow \quad I(E(x,v))}$$

Définir le contexte mathématique et logique

CONTEXT $\mathcal{D}$
EXTENDS $\mathcal{AD}$
SETS
$S_1, \ldots S_n$
CONSTANTS
$C_1, \ldots, C_m$
AXIOMS
$ax_1 : P_1(S_1, \ldots S_n, C_1, \ldots, C_m)$
$\ldots$
$ax_p : P_p(S_1, \ldots S_n, C_1, \ldots, C_m)$
THEOREMS
$th_1 : Q_1(S_1, \ldots S_n, C_1, \ldots, C_m)$
$\ldots$
$th_q : Q_q(S_1, \ldots S_n, C_1, \ldots, C_m)$

- Les constantes sont déclarées dans la clause CONSTANTS.
- Les axiomes sont énumérés dans la clause AXIOMS et définissent les propriétés des constantes.
- Les théorèmes sont des propriétés déclarées dans la clauseTHEOREMS et doivent être démontrées valides en fonction des axiomes.
- Le contexte définit une théorie logico-mathématique qui doit être consistante.
- La clause EXTENDS étend le contexte mentionné et étend donc la théorie définie par le contexte de cette clause.

Définir la machine

$$
\begin{array}{l}
\text{MACHINE } \mathcal{M} \\
\text{REFINES } \quad \mathcal{AM} \\
\text{SEES } \quad \mathcal{D} \\
\text{VARIABLES } \quad x \\
\text{INVARIANTS} \\
\quad inv_1 : I_1(x, S_1, \ldots S_n, C_1, \ldots, C_m) \\
\quad \ldots \\
\quad inv_r : I_r(x, S_1, \ldots S_n, C_1, \ldots, C_m) \\
\text{THEOREMS} \\
\quad th_1 : SAFE_1(x, S_1, \ldots S_n, C_1, \ldots, C_m) \\
\quad \ldots \\
\quad th_s : SAFE_s(x, S_1, \ldots S_n, C_1, \ldots, C_m) \\
\quad \ldots \\
\text{END}
\end{array}
$$

$$
\begin{array}{l}
\text{MACHINE } \mathcal{M} \\
\quad \ldots \\
\text{EVENTS} \\
\quad \text{EVENT initialisation} \\
\quad\quad \textbf{BEGIN} \\
\quad\quad\quad x : |(P(x')) \\
\quad\quad \textbf{END} \\
\quad \ldots \\
\quad \text{EVENT e} \\
\quad\quad \textbf{ANY } t \\
\quad\quad \textbf{WHERE} \\
\quad\quad\quad G(x, t) \\
\quad\quad \textbf{THEN} \\
\quad\quad\quad x : |(P(x, x', t)) \\
\quad\quad \textbf{END} \\
\quad \ldots \\
\text{END}
\end{array}
$$

Définir les événements

$$\begin{array}{l}
\mathcal{E}(\ell,\ell') \\
\quad \textbf{WHEN} \\
\qquad c = \ell \\
\qquad cond_{\ell,\ell'}(v) \\
\quad \textbf{THEN} \\
\qquad c := \ell' \\
\qquad v := f_{\ell,\ell'}(v) \\
\quad \textbf{END}
\end{array}$$

- v est la variable de l'état mémoire ou la liste des variables de l'tat mémoire ; v inclut les variables locales et les variables résultat.
- c est une nouvelle variable qui modélise le flôt de contrôle de type LOCATIONS.
- $\mathcal{E}(\ell,\ell')$ simule le calcul débutant en ℓ et terminant en o ℓ' ; v est mise à jour.

Définir l'invariant et les propriétés de sûreté

INVARIANTS

  $inv_i : c \in \textsc{Locations}$

  $inv_j : v \in Type$

...

  $inv_\ell : c = \ell \Rightarrow P_\ell(v)$

  $inv_{\ell'} : c = \ell' \Rightarrow P_{\ell'}(v)$

...

  $th_n : S(c, v)$

- *Type* est le type des variables v et est un ensemble de valeurs possibles définies dans le contexte C.
- L'annotation donne gratuitement les conditions satisfaites par v qyand le contrôle est en ℓ, (resp. en ℓ').
- $S(c, v)$ est une propriété de sûreté à vérifier et est une théorème dans le cas de *Event-B*.

Propriété

Pour toute paire d'étiquettes successives ℓ, ℓ', les trois énoncés suivants sont équivalents :

- $P_{\ell}(v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v')$
- $I(c,v) \wedge c = \ell \wedge cond_{\ell,\ell'}(v) \wedge c' = \ell' \wedge v' = f_{\ell,\ell'}(v) \Rightarrow (c' = \ell' \Rightarrow P_{\ell'}(v'))$
- $I(c,v) \wedge BA(\mathcal{E}(\ell,\ell'))(c,v,c',v') \Rightarrow (c' = \ell' \Rightarrow P_{\ell'}(v'))$

Propriété

Soit $\textsc{Alg}$ un algorithme annoté avec comme précondition **pre**($\textsc{Alg}$)(v) et postcondition **post**($\textsc{Alg}$)(v_0,v). Soit le contexte C et la machine M engendrée à partir de $\textsc{Alg}$ en utilisant la construction présentée précédemment. Nous supposons que ℓ_0 est l'étiquette *input* et ℓ_e est l'étiquette *output*. Nous ajoutons les propriétés de sûreté suivantes dans la machine M :

- $c = \ell_0 \wedge \mathbf{pre}(\textsc{Alg})(v) \Rightarrow P_{\ell_0}(v)$
- $c = \ell_e \Rightarrow (P_{\ell_e}(v) \Rightarrow \mathbf{post}(\textsc{Alg})(v_0, v))$

Si les conditions de vérification de M sont vérifiées et démontrées, alors l'algorithme annoté $\textsc{Alg}$ est partiellement correct par rapport à la pré/post spécification.

Problèmes à résoudre

- Vérifier les énoncés de la forme $\Gamma \vdash P$
- Enoncer ou calculer les invariants

TLA$^+$ versus Event-B

- Plate-formes : TLA$^+$ avec TLAPS et Toolbox, Event-B avec Rodin
- Langage de la théorie des ensembles avec quelques différences
- Fonctionnalités des outils
  - Editeurs de modèles : TLA$^+$ et Event-B
  - Model-Checking : TLA$^+$ et Event-B
  - Assistant de preuve : Event-B

Sommaire général

Sommaire

Sommaire

Sommaire

Introduction

Contexte des cours

Analyse statique

Cours 2

Spécification et annotation d'un programme

Modélisation relationnelle

Cours 3

Méthode de preuves de propriétés d'invariance

General form for processes

```
---- MODULE module_name ----
\* TLA+ code

(* --algorithm algorithm_name
variables global_variables

process p_name = ident
variables local_variables
begin
  \* pluscal code
end process

process p_group \in  set
variables local_variables
begin
  \* pluscal code
end process

end algorithm; *)
```

Example 1

```
process pro = "test"
begin
  print<<"test">>;
end process
```

Process in PLusCal

- A multiprocess algorithm contains one or more processes.
- A process begins in one of two ways :
  - defining a set of processes : `process ( ProcName â IdSet )`
  - defining one process with an identifier `process ( ProcName = Id )`
- `self` designates the current procees

A process S sends a mlessage to a process R

```
--algorithm ex_process {
  variables
    input = <<>>, output = <<>>,
    msgChan = <<>>, ackChan = <<>>,
    newChan = <<>>;
\* defining macros
  process (Sender = "S")
  {

  }; \* end Sender process block
  process (Receiver = "R")
  {

  }; \* end Receiver process block

} \* end algorithm
```

Using macros for defining sending and receiving prilmitives

```
--algorithm ex_process {
  variables
    input = <<>>, output = <<>>,
    msgChan = <<>>, ackChan = <<>>,
    newChan = <<>>;
  macro Send(m, chan) {
    chan := Append(chan, m);
  }
  macro Recv(v, chan) {
    await chan # <<>>;
    v := Head(chan);
    chan := Tail(chan);
  }

* Processes S and R

} \* end algorithm
```

Defining processes S and R

```
--algorithm ex_process {
  variables
    input = <<>>, output = <<>>,
    msgChan = <<>>, ackChan = <<>>,
    newChan = <<>>;
 \* defining macros
  process (Sender = "S")
  variables msg;
  {
  sending:  Send("Hello", msgChan);
  printing:    print <<"Sender", input>>;
  }; \* end Sender process block
  process (Receiver = "R")
  {
  waiting: Recv(msg, msgChan);
  adding: output := Append(output, msg);
  printing:  print <<"Receiver", output>>;
  }; \* end Receiver process block
} \* end algorithm
```

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

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