Cours Modélisation et vérification des systèmes informatiques

Dominique Méry
Telecom Nancy
Université de Lorraine

Année universitaires 2015-2016
2 septembre 2015

Plan

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
  - RODIN
  - PAT
  - Spec#, DAFNY
- Contrôle des connaissances : deux écrits et un TP

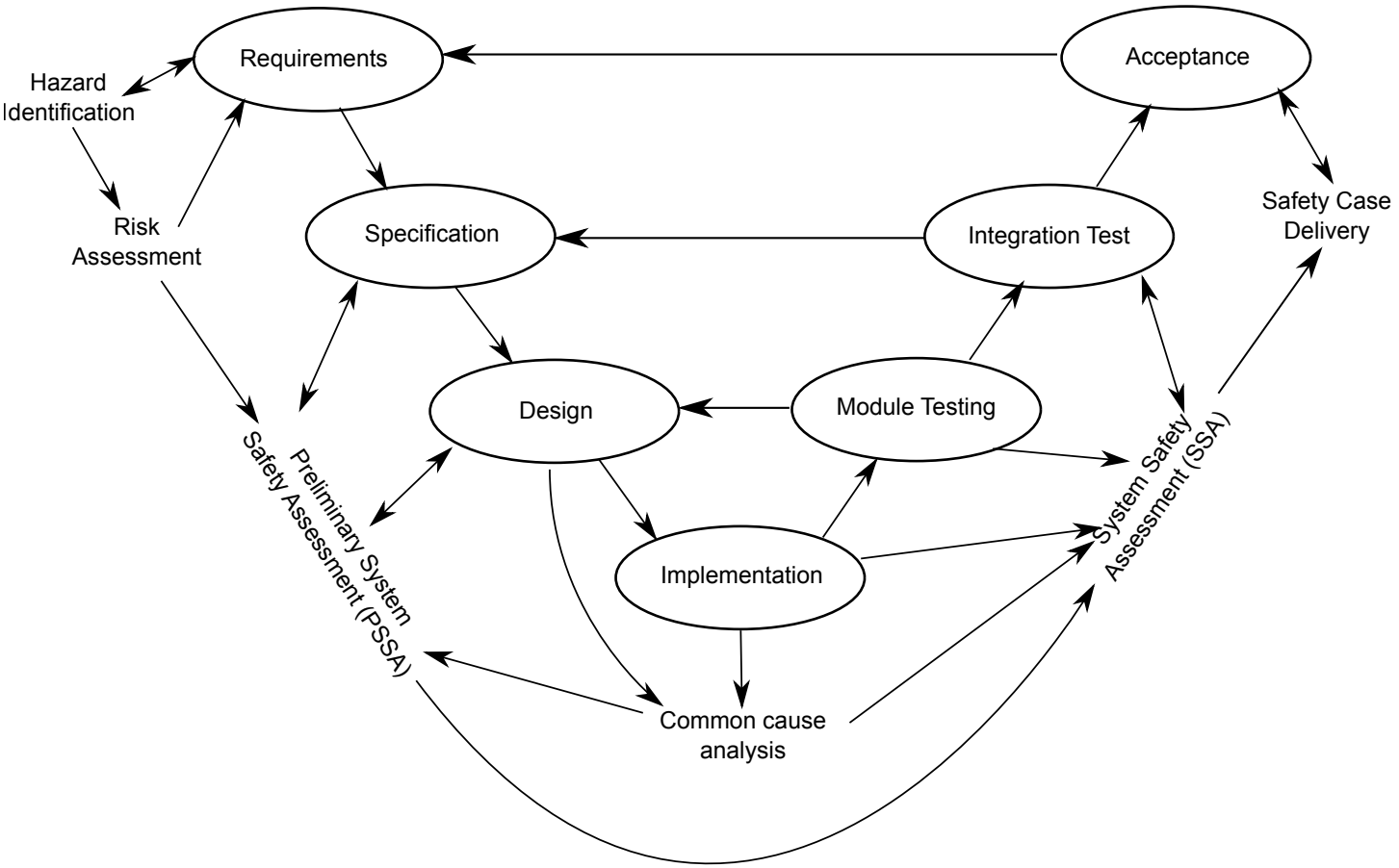
Plan

Modélisation, spécification et vérification

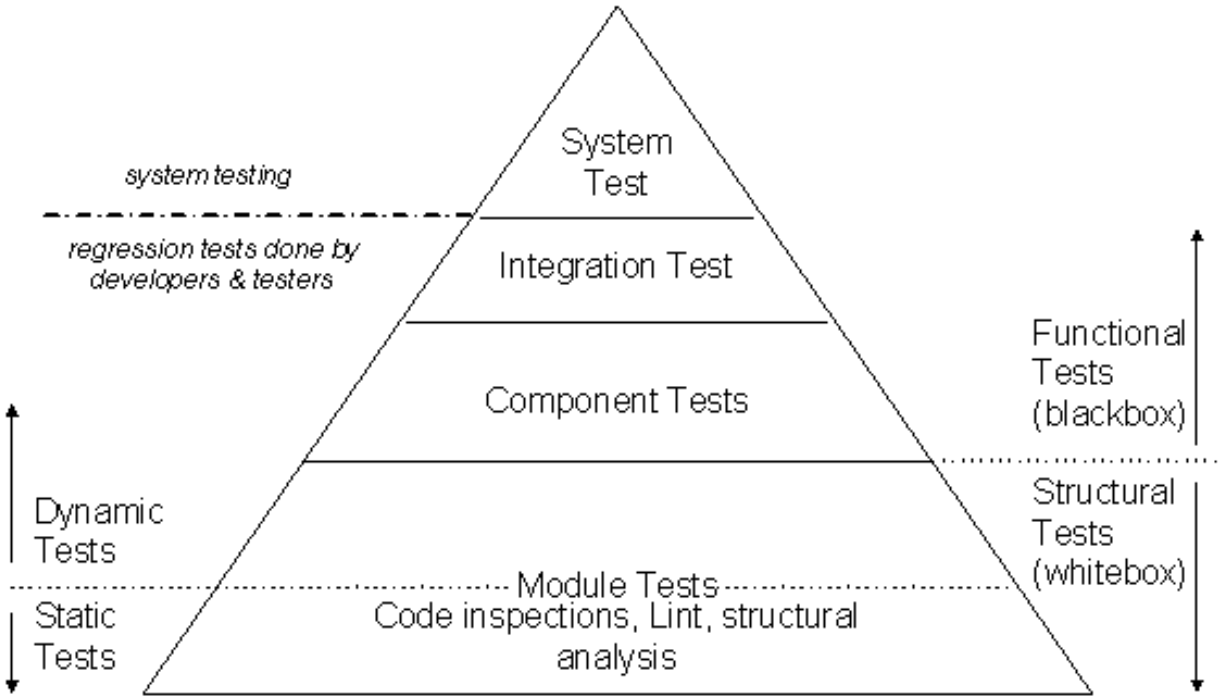
- Modélisation relationnelle d'un programme
- Notations ensemblistes
- Propriétés de programme et pré/post-conditions
- Méthodes de preuve de programmes et principes d'induction
- Techniques de Model-Checking
- Propriétés de système comme sûreté, vivacité, équité, ...
- Outils : Plate-forme TLC, RODIN,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

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 :

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

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 :

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

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

$Q = 0; R = X;$
while $R \geq Y$ **do**
    $Q = Q + 1;$
    $R := R - Y;$

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 : $x, y, q, r \in \mathbb{I}$
- $\mathcal{D} = \mathit{POWERSET}(\mathbb{I}^4)$

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**
  $\ell_3 : \{\ldots\}$
  $Q = Q + 1\,;$
  $\ell_4 : \{\ldots\}$
  $R := R - Y\,;$
;
$\ell_5 : \{\ldots\}$

...

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**
  $\ell_3 : \{\ldots\}$
  $Q = Q + 1\,;$
  $\ell_4 : \{\ldots\}$
  $R := R - Y\,;$
;
$\ell_5 : \{\ldots\}$

...

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**
  $\ell_3 : \{\ldots\}$
  $Q = Q + 1\,;$
  $\ell_4 : \{\ldots\}$
  $R := R - Y\,;$
;
$\ell_5 : \{\ldots\}$

...

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
```

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 $S(x_0, y_0, t_0) \overset{R(x,y)}{\longrightarrow} S(x_f, y_f, t_f)$ où f désigne le dernier état de ce calcul
  - $P(x_0) \wedge (S(x_0, y_0, t_0) \overset{R(x,y)}{\longrightarrow} S(x_f, y_f, t_f)) \Rightarrow Q(x_0, y_0, t_0, x_f, y_f, t_f)$

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;  // Déclarer les variables
int somme = 0;          // Déclarer 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

$//\ell_1 : \{P_{\ell_1}(x)\}$
FOR $i := 1$ TO n DO
$\ell_2 : \{P_{\ell_2}(i, x)\}$
$S(x)$;
$\ell_3 : \{P_{\ell_3}(i, x)\}$
ENDFOR
$\ell_4 : \{P_{\ell_4}(x)\}$

Programme annoté

$//\ell_1 : \{P_{\ell_1}(x)\}$
FOR $i := 1$ TO n DO
$\ell_2 : \{P_{\ell_2}(i, x)\}$
$S(x)$;
$\ell_3 : \{P_{\ell_3}(i, x)\}$
ENDFOR
$\ell_4 : \{P_{\ell_4}(x)\}$

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

Programme annoté

$//\ell_1 : \{P_{\ell_1}(x)\}$
FOR $i := 1$ TO n DO
$\ell_2 : \{P_{\ell_2}(i, x)\}$
$S(x)$;
$\ell_3 : \{P_{\ell_3}(i, x)\}$
ENDFOR
$\ell_4 : \{P_{\ell_4}(x)\}$

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

(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) $c = \ell_3 \wedge P_{\ell_3}(x, i) \wedge i + 1 \leq n \wedge c' = \ell_2 \wedge i' = i + 1 \wedge x' = x \Rightarrow c' = \ell_2 \wedge P_{\ell_2}(i', x')$
(4) $c = \ell_2 \wedge P_{\ell_3}(x, i) \wedge \neg(i + 1 \leq n) \wedge c' = \ell_4 \wedge x' = x \wedge i' = i + 1 \Rightarrow c' = \ell_4 \wedge P_{\ell_4}(x')$

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,. . .

Variables et états d'un système

- un système S est modélisé par un ensemble d'états Σ_{S}, noté Σ,
- $\Sigma \stackrel{def}{=} \textsc{Variables} \longrightarrow \textsc{Valeurs}$.
- L'écriture $s \in A$ se traduit en une expression de la forme $s[\![\varphi(x)]\!]$ où x est une liste dont les éléments sont toutes les variables de $\textsc{Variables}$ et exclusivement ces variables ;
- $s \in A$ signifie de manière équivalente que $\varphi(x)$ est vraie en s.
- La définition de la validité d'une formule ou d'un prédicat peut être donnée sous une forme inductive de $s[\![\varphi(x)]\!]$.

Exemples

1. $s[\![x]\!]$ est la valeur de s en x c'est-à-dire $s(x)$ ou encore la valeur de x en s..
2. $s[\![\varphi(x) \wedge \psi(x)]\!] \stackrel{def}{=} s[\![\varphi(x)]\!]$ et $s[\![\psi(x)]\!]$.
3. $s[\![x = 6 \wedge y = x + 8]\!] \stackrel{def}{=} s[\![x]\!] = 6$ et $s[\![y]\!] = s[\![x]\!] + 8$.

Notations simplifiant la référence aux états

- $s[\![x]\!]$ est la valeur de x en s
- le nom de la variable x et sa valeur ne seront pas distinguées.
- $s'[\![x]\!]$ est la valeur de x en s' et sera notée x'.
- $s[\![x = 6]\!] \wedge s'[\![y = x + 8]\!]$ se simplifiera en $x = 6 \wedge y' = x' + 8$.
- La conséquence est que l'on pourra écrire la relation de transition comme une relation liant l'état des variables en s et l'état des variables en s'.

Notations simplifiant la référence aux états

relation de calcul

Soient s, s' deux états de l'ensemble $\textsc{Variables} \longrightarrow \textsc{Vals}$.
$s \xrightarrow[R]{} s'$ s'écrira comme une relation $\mathrm{R}(x, x')$ où x et x' désignent des valeurs de x.

- Introduction de la notion de variable primée de la logique temporelle des actions de Lamport
- x' est la valeur après la transition considérée et x est la valeur avant la transition considérée.
- la condition $\exists y.\mathrm{R}(x, y)$ définit la condition de transition ou la garde ;
- Intérêt des expressions particulières de la forme suivante $cond(x) \wedge x' = f(x)$ où $cond$ est une condition sur x et f est une fonction.
- L'ensemble des états est Σ pour un système donné et nous identifions cet ensemble à l'ensemble des valeurs possibles des variables flexibles x : $\textsc{Vals}$

Modèle relationnel d'un système (I)

Definition

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

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

où

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

Modèle relationnel d'un système (II)

- Un modèle relationnel $\mathcal{MS} = (Th(s, c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ pour un système $\mathcal{S}$ est une structure permettant d'étudier un système au travers d'un modèle.
- Nous supposons que la relation r_0 est la relation $Id[\textsc{Vals}]$, identité sur $\textsc{Vals}$.

Relation de transition

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

Modélisation d'un système

- La modélisation d'un système comprend la donnée des variables x, du prédicat caractériant les valeurs initiales des variables et une relation $\textsc{Next}$ modélisant la relation entre les valeurs avant et les valeurs après.
- Les principes d'induction sont transcrits dans le cadre des modèles relationnels et nous introduisons la défintion de la sûreté dans un modèle relationnel.
- Les modèles peuvent prendre en compte des aspects très divers : états, traces, probabilités, risques, ...

- Soit $(Th(s,c), x, \textsc{Vals}, \textsc{Init}(x), \{r_0, \ldots, r_n\})$ un modèle relationnel d'un système $\mathcal{S}$.
- La théorie $Th(s,c)$ est définie dans un langage d'assertions qui permet de décrire un certain nombre de propriétés et de définir des ensembles.
- Un exemple est la théorie des ensembles du langage B ou du langage TLA$^+$.
- Quand nous parlerons d'une propriété φ, il s'agira de ce langage implicite dans notre exposé.

Propriétés de sûreté

Les propriétés de sûreté expriment que *rien de mauvais ne peut arriver*.

- la valeur de la variable x est toujours comprise entre 0 et 567
- la somme des valeurs courantes des variables x et y est égale à la valeur courante de la valeur z.

On se donne un langage d'assertions $(\mathcal{P}(\Sigma), \subseteq)$ et une interprétation ensembliste de la relation de satisfaction.

Definition

Une propriété φ est une propriété de sûreté pour le système S, si

$$\forall s, s' \in \Sigma . s \in \mathit{Init}_{\mathrm{S}} \wedge s \xrightarrow[R]{\star} s' \Rightarrow s' \in \varphi.$$

Principe d'induction

Principe d'induction

Une propriété φ est une propriété de sûreté pour le programme P si, et seulement si, il existe une propriété *INV* telle que

$$\begin{cases} (1) & \mathit{Init}_{\mathrm{P}} \subseteq \mathit{INV} \\ (2) & \mathit{INV} \subseteq \varphi \\ (3) & \forall s, s' \in \Sigma_{\mathrm{P}}.s \in \mathit{INV} \wedge s \underset{R}{\longrightarrow} s' \Rightarrow s' \in \mathit{INV} \end{cases}$$

La propriété *INV* est appelée un invariant du programme et est une propriété de sûreté particulière plus forte que les autres propriétés de sûreté. La justification de ce principe d'induction est assez simple.

Application de ce principe

Definition

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

(Principe d'induction pour un modèle relationnel)

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

$$\begin{cases} (1) & \forall x \in \textsc{Vals}. Init(x) \Rightarrow i(x) \\ (2) & \forall x \in \textsc{Vals}. i(x) \Rightarrow \varphi(x) \\ (3) & \forall x, x' \in \textsc{Vals}. i(x) \wedge \textsc{Next}(x, x') \Rightarrow i(x') \end{cases}$$

Equivalence

Simplification des conditions

Les deux énoncés suivants sont équivalents :

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

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

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

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

Application à des langages de programmation et de modélisation

- Langage de modélisation Event B
- Langage de modélisation TLA$^+$
- Langages de programmation : C, Spec#

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. On ne prend pas en compte la terminaison et on demande que quand le programme se termine, les valeurs des variables satisfont la postcondition.
On note les éléments suivants :

- 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 $\ell = \mathit{output} \Rightarrow$ **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 $\ell = \mathit{output} \Rightarrow$ **post**$(\mathrm{P})(v)$ est une propriété de sûreté pour ce programme.

Conditions de vérification

Si les conditions suivantes sont vérifiées :

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

,
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

Pour la preuve de l'absence d'erreur à l'exécution, nous devons définir ce que signifie une erreur à l'exécution. Pour cela, nous nouis plaçons dans le cas où 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.

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

Parmi les cas d'erreurs, on pourra citer le débordement arithmétique, la référence à une adresse non définie, la division par zéro, ... Le prédicat $\mathbf{DOM}(\ell,\ell')(v)$ doit intégrer ces éléments pour chaque cas possible qui se ramène à une affectation ou un test en C par exemple.

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. On ne prend pas en compte la terminaison et le programme peut naturellement boucler. Nous définissons donc les éléments suivants :

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

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

Definition

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

Conditions de vérification

Si les conditions suivantes sont vérifiées :

$$\begin{cases} (1)\ \forall \ell \in \textsc{InputLocations}. \forall v, v' \in \textsc{Memory}. \mathbf{pre}(\mathrm{P})(v) \Rightarrow P_\ell(v) \\ (2)\ \forall m \in \textsc{Locations} - \{output\}, n \in \textsc{Locations}, m \longrightarrow n. \\ \forall v, v' \in \textsc{Memory}. P_m(v) \Rightarrow \mathbf{DOM}(m,n)(v) \\ (3)\ \forall \ell, \ell' \in \textsc{Locations} : \ell \longrightarrow \ell' \Rightarrow \forall v, v' \in \textsc{Memory}. \\ (P_\ell(v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v')) \end{cases},$$

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

Idée de cette logique

Ces principes ont été mis en œuvre et Tony HOARE [**?**] a proposé une logique permettant de dériver des formules de la forme $\{p\}\ \mathbf{S}\ \{q\}$ où p et q sont des prédicats d'états et S est une instruction. L'idée est de pouvoir donner une signification à cette expression et d'avoir des règles pour dériver les expressions correctes. Il faut donc définir ce qui est correct.

Definition

$\{p\}\ \mathbf{S}\ \{q\}$ signifie que, S est partiellement correct par rapport à p et q.

Axiomatisation des programmes commentés

- Axiome d'affectation : $\{P(e/x)\}\mathbf{x := e}\{P\}$ où $P(e/x)$ désigne le prédicat $P(x)$ dans lequel les occurences libres de x sont remplacées par e.
- Axiome du saut : $\{P\}\mathbf{skip}\{P\}$.
- Règle de la conditionnelle : Si $\{P \wedge B\}\mathbf{S}_1\{Q\}$ et $\{P \wedge \neg B\}\mathbf{S}_2\{Q\}$, alors $\{P\}\mathbf{if\ B\ then\ S}_1\ \mathbf{then\ S}_2\ \mathbf{fi}\{Q\}$.
- Règle de l'itération : Si $\{P \wedge B\}\mathbf{S}\{P\}$, alors $\{P\}\mathbf{while\ B\ do\ S\ od}\{P \wedge \neg B\}$.
- Règle de l'implication : Si $P' \Rightarrow P$, $\{P\}\mathbf{S}\{Q\}$, $Q \Rightarrow Q'$, alors $\{P'\}\mathbf{S}\{Q'\}$.
- Règle de la séquence : Si $\{P\}\mathbf{S}_1\{Q\}$ et $\{Q\}\mathbf{S}_2\{R\}$, alors $\{P\}\mathbf{S}_1\ \mathbf{;}\ \ \mathbf{S}_2\{R\}$.