La méthode B

Cours donné à l'Ecole des Jeunes Chercheurs en Programmation
Dinard - 21 mai 2010

Marie-Laure Potet *Didier Bert*

VERIMAG, Grenoble, France

Plan du cours

1. Introduction aux méthodes formelles- Méthode B
2. Formalisme de modélisation
3. Spécification des opérations : substitutions
4. Les machines abstraites
5. Raffinement et implémentation
6. Modularité : raffinement et composition
7. Applications industrielles

Particularités du logiciel

- produit intellectuel
  - coût de fabrication nul
  - conception complexe
- logiciel pour la sécurité
  - pas d'usure
  - duplication à coût nul
  - fonctionnalités complexes
  - rapidité, réactivité

$\Rightarrow$ coût Validation/Vérification élevé

Contraintes

- Fiabilité (transport ferroviaire) :
  - Système : 10^{-9} pannes par rame/heure
  - Logiciel : 10^{-11} pannes par rame/heure

  $\Rightarrow$ non vérifiable par expérimentation
- Coût du développement (aérospaciale) :
  - $\times 3$ les fonctionnalités embarquées
  - $\times 60$ l'effort de production de code

  $\Rightarrow$ maîtrise du processus

Systèmes critiques

- Intérêt limité de la redondance
  - double développement
  - double support matériel
  - absence de mode d'erreur commun
- Pas de principe de sécurité intrinsèque
  - panne $\not\Rightarrow$ état dangereux
  - système discret

$\Rightarrow$ Vers des techniques formelles

Le ferroviaire et B

Logiciel pour les fonctions critiques de sécurité (fin 80)

1) développement non redondé avec validation à l'aide de méthodes formelles

- correction du code vis-a-vis des spécifications fontionnelles

2) utilisation de la technique du Processeur Sécuritaire Codé pour la détection des pannes matérielles

- codage des données et vérification à l'exécution
- etat sûr si non conformité à l'exécution

Le ferroviaire et B ... suite

1. vérification a posteriori :
   - ajout d'assertions dans le code
   - vérification semi-automatique
2. lien avec la spécification :
   - réexpression formelle
   - conformité (manuelle) du code

$\Rightarrow$ Méthode B (J-R Abrial)

- développement correct par construction

Météor : B + PSC $\Rightarrow$ suppression des tests unitaires

Pourquoi étudier la méthode B ?

- des notions communes à toute approche formelle
  - spécification, vérification, preuve
- processus de développement en son entier
  - raffinement, génération automatique de code prouvé
- outil et méthode permettant le passage à l'échelle
  - composition des spécifications et des développements, vérification incrémentale
- applications industrielles et processus métier
  - AtelierB, Rodin, Leirios test Generator, Bart . . .

Spécification

Une machine abstraite :

- un état
- une initialisation
- des opérations
- des propriétés invariantes

Données	ensembles
initialisation opérations	substitutions généralisées
propriétés	prédicats du premier ordre

Vérification

Obligations de preuve :

- Les propriétés invariantes sont vérifiées par la dynamique
- Les raffinements préservent la correction totale
- Le code est exempt d'erreur à l'exécution

L'atelier B (ClearSy)

- Analyseur
- Générateur d'obligations de preuve
- Démonstrateur automatique
- Démonstrateur interactif
- Générateur de code (C et Ada)
- Gestionnaire de projets

AtelierB 4.0 (Windows, Linux, Mac OS, Solaris) :
http://www.atelierb.eu/php/telecharger-atelier-b-fr.php

Partie 2 : Modélisation

1. Introduction à la méthode B
2. Formalisme de modélisation
3. Spécification des opérations : substitutions
4. Les machines abstraites
5. Raffinement et implémentation
6. Modularité : raffinement et composition
7. Applications industrielles

Formalisme logique :Prédicats

- Logique du premier ordre :
  $P \wedge Q$, $P \Rightarrow Q$, $\neg P$
  $\forall x \cdot P$ quantification
  $[x := E]\, P$ substitution dans un prédicat
- Prédicats de base :
  $x \in S$ appartenance
  $E_1 = E_2$ égalité
- Les autres constructeurs sont dérivés :
  $P \vee Q$, $\exists x \cdot P$, $x \notin S$, etc.

Modélisation : Expressions et Ensembles

- Les ensembles (typés) :
  $S_1 \times S_2$ produit
  $\mathbb{P}(S)$ ensemble des parties
  $\{\, x \mid P\}$ ensemble en compréhension
  BIG un ensemble infini
- Les expressions :
  x variable
  $[x := E_1]E_2$ substitution dans une expression
  (E_1, E_2) paire d'expressions
  choice(S) fonction de choix
  S ensemble

Quelques notations

La substitution :

$[x := E]\, P$

les occurrences libres de x sont remplacées par E dans P.

Autre notation :

$x \backslash P$

qui signifie : x n'est pas libre dans P.

Axiomes de base

	Axiome
SET1	$(E, F) \in s \times t \;\Leftrightarrow\; E \in s \;\wedge\; F \in t$
SET2	$s \in \mathbb{P}(t) \;\Leftrightarrow\; \forall x \cdot (x \in s \Rightarrow x \in t)$
SET3	$E \in \{x \mid x \in s \wedge P\} \;\Leftrightarrow\; (E \in s \wedge [x := E]\, P)$
SET4	$\forall x \cdot (x \in s \Leftrightarrow x \in t) \;\Rightarrow\; s = t$
SET5	$\exists x \cdot (x \in s) \;\Rightarrow\;$ choice$(s) \in s$
SET6	infinite(BIG)

Constructions dérivées

Les autres opérateurs et notations sur les ensembles sont dérivés du jeu de base donné.
Les propriétés usuelles sur ces opérateurs peuvent être démontrées à partir des axiomes. Exemples :

$s \subseteq t$	$s \in \mathbb{P}(t)$	
$s \cup t$	$\{a \mid a \in u \ \wedge \ (a \in s \ \vee \ a \in t)\}$	$s \subseteq u \wedge t \subseteq u$
$s \cap t$	$\{a \mid a \in u \ \wedge \ (a \in s \ \wedge \ a \in t)\}$	$s \subseteq u \wedge t \subseteq u$
$s - t$	$\{a \mid a \in u \ \wedge \ (a \in s \ \wedge \ a \notin t)\}$	$s \subseteq u \wedge t \subseteq u$
$\{E\}$	$\{a \mid a \in u \ \wedge \ a = E\}$	$E \in u$
$\{E, F\}$	$\{E\} \cup \{F\}$	$E \in u \wedge F \in u$
$\varnothing$	BIG − BIG	

Construction des relations

Relation entre deux ensembles $s \leftrightarrow t \ \widehat{=} \ \mathbb{P}(s \times t)$

Construction des relations

Relation entre deux ensembles $s \leftrightarrow t \ \widehat{=} \ \mathbb{P}(s \times t)$

Opérateurs classiques sur les relations :

Condition	Expression	Définition
$r \in s \leftrightarrow t$	$\mathsf{dom}(r)$	$\{x \mid x \in s \ \wedge$ $\exists y \cdot (y \in t \ \wedge \ (x, y) \in r)\}$
$r \in s \leftrightarrow t$	$\mathsf{ran}(r)$	$\{y \mid y \in t \ \wedge$ $\exists x \cdot (x \in s \ \wedge \ (x, y) \in r)\}$
$r \in s \leftrightarrow t$ $\wedge \ u \subseteq s$	$r[u]$	$\{y \mid y \in t \ \wedge$ $\exists x \cdot (x \in u \ \wedge \ (x, y) \in r)\}$
$r \in s \leftrightarrow t$	r^{-1}	$\{(y, x) \mid (y, x) \in t \times s \ \wedge \ (x, y) \in r\}$

Autres opérateurs sur les relations

Condition	Expr	Définition
	$\mathsf{id}(s)$	$\{x, y \mid (x, y) \in s \times s \ \wedge \ x = y\}$
$r_1 \in s \leftrightarrow t \ \wedge$ $r_2 \in t \leftrightarrow u$	$r_1 \ ; r_2$	$\{x, z \mid (x, z) \in s \times u \ \wedge$ $\exists y \cdot (y \in t \ \wedge \ (x, y) \in r_1 \ \wedge \ (y, z) \in r_2)\}$
$r \in s \leftrightarrow t \ \wedge$ $q \in s \leftrightarrow t$	$r \mathbin{<\!\!+} q$	$\{x, y \mid (x, y) \in s \times t \ \wedge$ $(((x, y) \in r \ \wedge \ x \notin \mathsf{dom}(q))$ $\vee \ (x, y) \in q)\}$

Construction des fonctions

Les fonctions sont un cas particulier de relations :

Signification	Notation	Définition
f. partielles	$s \mapsto t$	$\{r \mid r \in s \leftrightarrow t \ \wedge \ \forall x, y, z \cdot (x, y \in r \ \wedge \ x, z \in r \Rightarrow y = z)\}$
f. totales	$s \rightarrow t$	$\{f \mid f \in s \mapsto t \ \wedge \ \mathsf{dom}(f) = s\}$
injectives part.	$s \rightarrowtail\!\!\!\mapsto t$	$\{f \mid f \in s \mapsto t \ \wedge \ f^{-1} \in t \mapsto s\}$
injectives tot.	$s \rightarrowtail t$	$s \rightarrowtail\!\!\!\mapsto t \ \cap \ s \rightarrow t$
evaluation	$f(E)$	$\mathsf{choice}(f[\{E\}])$ si $f \in s \mapsto t \ \wedge \ E \in \mathsf{dom}(f)$

Construction des ensembles inductifs

Comment définir les ensembles tels que :

- les entiers naturels $\mathbb{N}$
- les parties finies d'un ensemble $\mathbb{F}(s)$
- la fermeture réflexive transitive d'une relation r^*

. . . tout en restant dans la théorie de base B. . .

Définition par induction

On dispose d'un schéma d'induction pour caractériser un sous-ensemble E de s :

- un élément de base $a \in E$
- une règle $x \in E \Rightarrow f(x) \in E$
- une clause de fermeture : E est le plus petit sous-ensemble de s finiment engendré par *la règle* à partir de *la base*.

Définition par induction

On dispose d'un schéma d'induction pour caractériser un sous-ensemble E de s :

- un élément de base $a \in E$
- une règle $x \in E \Rightarrow f(x) \in E$
- une clause de fermeture : E est le plus petit sous-ensemble de s finiment engendré par *la règle* à partir de *la base*.

Soit g tel que $g \ : \ e \mapsto \{a\} \cup f[e]$

La fonction g est monotone :

$e_1 \subseteq e_2 \ \Rightarrow \ g(e_1) \subseteq g(e_2)$

Plus petit point fixe

D'après le théorème de Tarski :

Si g est monotone, la plus petite solution de $X = g(X)$ est le *plus petit point fixe* de g, qui est défini par :

$$\mathsf{fix}(g) = \bigcap\{e \mid e \in \mathbb{P}(s) \ \wedge\ g(e) \subseteq e\}$$

Plus petit point fixe

D'après le théorème de Tarski :

Si g est monotone, la plus petite solution de $X = g(X)$ est le *plus petit point fixe* de g, qui est défini par :

$$\mathsf{fix}(g) = \bigcap\{e \mid e \in \mathbb{P}(s) \ \wedge\ g(e) \subseteq e\}$$

Avec $E = \mathsf{fix}(g)$, E satisfait les axiomes :

$$\begin{array}{l} g[E] \subseteq E \ \wedge \\ \forall S \cdot (S \subseteq \mathbb{P}(s) \ \wedge\ g[S] \subseteq S \Rightarrow E \subseteq S) \end{array}$$

Ecole des Jeunes Chercheurs en Programmation - mai 2010 – p.23/10

Plus petit point fixe

D'après le théorème de Tarski :

Si g est monotone, la plus petite solution de $X = g(X)$ est le *plus petit point fixe* de g, qui est défini par :

$$\mathsf{fix}(g) = \bigcap\{e \mid e \in \mathbb{P}(s) \ \wedge\ g(e) \subseteq e\}$$

Avec $E = \mathsf{fix}(g)$, E satisfait les axiomes :

$$\begin{array}{l} g[E] \subseteq E \ \wedge \\ \forall S \cdot (S \subseteq \mathbb{P}(s) \ \wedge\ g[S] \subseteq S \Rightarrow E \subseteq S) \end{array}$$

Schéma d'induction : $\forall x \cdot (x \in E \Rightarrow P(x))$

$$\begin{array}{l} g[\{x \mid x \in E \wedge P(x)\}] \subseteq \{x \mid x \in E \wedge P(x)\} \Rightarrow E \subseteq \{x \mid x \in E \wedge P(x)\} \\ \{a\} \cup f[\{x \mid x \in E \wedge P(x)\}] \subseteq \{x \mid x \in E \wedge P(x)\} \Rightarrow \ldots \\ \{a\} \cup \{f(x) \mid x \in E \wedge P(x)\} \subseteq \{x \mid x \in E \wedge P(x)\} \Rightarrow \ldots \end{array}$$

$$P(a) \ \wedge\ \forall x \cdot (x \in E \wedge P(x) \Rightarrow P(f(x))) \Rightarrow \forall x \cdot (x \in E \Rightarrow P(x))$$

Ecole des Jeunes Chercheurs en Programmation - mai 2010 – p.23/10

Exemple : définition de r^*

On a une relation $r \in s \leftrightarrow s$

r^* est réflexive	$\mathsf{id}(s) \subseteq r^*$
elle contient r	$r \subseteq r^*$
et est fermée par composition	$v \subseteq r^* \Rightarrow (r\ ;\ v) \subseteq r^*$

La fonction g de génération de r^* est :

$$g\ : e \mapsto \mathsf{id}(s) \cup (r\ ;\ e)$$

Exemple : définition de $\mathbb{N}$

On doit avoir (axiomes de Peano):

1- $0 \in \mathbb{N}$

2- $n \in \mathbb{N} \Rightarrow succ(n) \in \mathbb{N}$

3- $0 \neq succ(n)$

4- $succ(n) = succ(m) \Rightarrow n = m$

5- $[n := 0]P \ \wedge \ \forall n \cdot (n \in \mathbb{N} \wedge P \Rightarrow [n := succ(n)]P)$
$\Rightarrow \forall n \cdot (n \in \mathbb{N} \Rightarrow P)$

Exemple : définition de $\mathbb{N}$

On doit avoir (axiomes de Peano):

1- $0 \in \mathbb{N}$

2- $n \in \mathbb{N} \Rightarrow succ(n) \in \mathbb{N}$

3- $0 \neq succ(n)$

4- $succ(n) = succ(m) \Rightarrow n = m$

5- $[n := 0]P \ \wedge \ \forall n \cdot (n \in \mathbb{N} \wedge P \Rightarrow [n := succ(n)]P)$
$\Rightarrow \forall n \cdot (n \in \mathbb{N} \Rightarrow P)$

Codage des opérateurs :

$$0 \ \widehat{=} \ \emptyset$$
$$succ \ \widehat{=} \ \lambda n \cdot (n \in \mathbb{F}(\mathsf{BIG}) \mid n \cup \{\mathsf{choice}(\mathsf{BIG} - n)\})$$

La fonction g de génération de $\mathbb{N}$ est :

$$g \ : e \mapsto \{\emptyset\} \cup succ[e]$$

Et on a $\mathbb{N} \ \widehat{=} \ \mathsf{fix}(g)$

Les ensembles offerts par le langage

Notations d'ensembles prédéfinis en B avec, pour chacun, un jeu d'opérateurs usuels. Ce sont :

- les ensembles *donnés* : ce sont des ensembles finis, non vides
- les ensembles finis *énumérés*
- les entiers relatifs $\mathbb{Z}$ (avec les sous-ensembles $\mathbb{N}$ et $\mathbb{N}_1$)
- les entiers relatifs bornés INT (avec le sous-ensemble NAT)
- les séquences de s (fonctions de $1..n \rightarrow s$)
- les arbres n-aires (avec le sous-ensemble des arbres binaires)

$INT = -2147483647..214748364$ (modifiable)

Exercice de modélisation

Soit un ensemble $personne \subseteq PERSONNE$:

R1 : toute personne est soit un homme, soit une femme

R2 : une personne ne peut être à la fois un homme et une femme

R3 : seules les femmes peuvent avoir un mari qui est un homme

R4 : les femmes ont au plus un mari

R5 : les hommes ne peuvent être mariés qu'à au plus une femme

R6 : les mères d'une personne sont des femmes mariées

Solution

R1 : toute personne est soit un homme, soit une femme

R2 : une personne ne peut être à la fois un homme et une femme

R3 : seules les femmes peuvent avoir un mari qui est un homme

R4 : les femmes ont au plus un mari

R5 : les hommes ne peuvent être mariés qu'à au plus une femme

R6 : les mères d'une personne sont des femmes mariées

Solution

R1 : toute personne est soit un homme, soit une femme

$homme \subseteq personne$

$femme \subseteq personne$

$homme \cup femme = personne$

R2 : une personne ne peut être à la fois un homme et une femme

R3 : seules les femmes peuvent avoir un mari qui est un homme

R4 : les femmes ont au plus un mari

R5 : les hommes ne peuvent être mariés qu'à au plus une femme

R6 : les mères d'une personne sont des femmes mariées

Solution

R1 : toute personne est soit un homme, soit une femme

$homme \subseteq personne$

$femme \subseteq personne$

$homme \cup femme = personne$

R2 : une personne ne peut être à la fois un homme et une femme

$homme \cap femme = \varnothing$

R3 : seules les femmes peuvent avoir un mari qui est un homme

R4 : les femmes ont au plus un mari

R5 : les hommes ne peuvent être mariés qu'à au plus une femme

R6 : les mères d'une personne sont des femmes mariées

Solution

R1 : toute personne est soit un homme, soit une femme

$homme \subseteq personne$

$femme \subseteq personne$

$homme \cup femme = personne$

R2 : une personne ne peut être à la fois un homme et une femme

$homme \cap femme = \varnothing$

R3 : seules les femmes peuvent avoir un mari qui est un homme

$mari \in femme \leftrightarrow homme$

R4 : les femmes ont au plus un mari

R5 : les hommes ne peuvent être mariés qu'à au plus une femme

R6 : les mères d'une personne sont des femmes mariées

Solution

R1 : toute personne est soit un homme, soit une femme

$homme \subseteq personne$

$femme \subseteq personne$

$homme \cup femme = personne$

R2 : une personne ne peut être à la fois un homme et une femme

$homme \cap femme = \varnothing$

R3 : seules les femmes peuvent avoir un mari qui est un homme

$mari \in femme \leftrightarrow homme$

R4 : les femmes ont au plus un mari

$mari \in femme \nrightarrow homme$

R5 : les hommes ne peuvent être mariés qu'à au plus une femme

R6 : les mères d'une personne sont des femmes mariées

Solution

R1 : toute personne est soit un homme, soit une femme

$homme \subseteq personne$

$femme \subseteq personne$

$homme \cup femme = personne$

R2 : une personne ne peut être à la fois un homme et une femme

$homme \cap femme = \varnothing$

R3 : seules les femmes peuvent avoir un mari qui est un homme

$mari \in femme \leftrightarrow homme$

R4 : les femmes ont au plus un mari

$mari \in femme \nrightarrow homme$

R5 : les hommes ne peuvent être mariés qu'à au plus une femme

$mari^{-1} \in homme \nrightarrow femme$

$mari \in femme \rightarrowtail homme$

R6 : les mères d'une personne sont des femmes mariées

Solution

R1 : toute personne est soit un homme, soit une femme

$homme \subseteq personne$

$femme \subseteq personne$

$homme \cup femme = personne$

R2 : une personne ne peut être à la fois un homme et une femme

$homme \cap femme = \varnothing$

R3 : seules les femmes peuvent avoir un mari qui est un homme

$mari \in femme \leftrightarrow homme$

R4 : les femmes ont au plus un mari

$mari \in femme \nrightarrow homme$

R5 : les hommes ne peuvent être mariés qu'à au plus une femme

$mari^{-1} \in homme \nrightarrow femme$

$mari \in femme \rightarrowtail homme$

R6 : les mères d'une personne sont des femmes mariées

$mere \in personne \nrightarrow \mathsf{dom}(mari)$

Exercice de modélisation (suite)

A l'aide des définitions précédentes, définir les notions de :

R7 : père

R8 : parent

R9 : enfant

R10 : grand-parent et ancêtre

R11 : frère-sœur

R12 : Démontrer $mere = pere\ ;\ mari^{-1}$

Solution des exercices (suite)

R7 : père

R8 : parent

R9 : enfant

R10 : grand-parent et ancêtre

R11 : frère-sœur

R12 : Démontrer $mere = pere \,;\, mari^{-1}$

Solution des exercices (suite)

R7 : père

$pere = mere \,;\, mari$

R8 : parent

R9 : enfant

R10 : grand-parent et ancêtre

R11 : frère-sœur

R12 : Démontrer $mere = pere \,;\, mari^{-1}$

Solution des exercices (suite)

R7 : père

$pere = mere \,;\, mari$

R8 : parent

$parent = mere \cup pere$

R9 : enfant

R10 : grand-parent et ancêtre

R11 : frère-sœur

R12 : Démontrer $mere = pere \,;\, mari^{-1}$

Solution des exercices (suite)

R7 : père

$pere = mere \,;\, mari$

R8 : parent

$parent = mere \cup pere$

R9 : enfant

$enfant = parent^{-1}$

R10 : grand-parent et ancêtre

R11 : frère-sœur

R12 : Démontrer $mere = pere \,;\, mari^{-1}$

Solution des exercices (suite)

R7 : père

$pere = mere \mathbin{;} mari$

R8 : parent

$parent = mere \cup pere$

R9 : enfant

$enfant = parent^{-1}$

R10 : grand-parent et ancêtre

$grand\_parent = parent \mathbin{;} parent$

$ancetre = parent \mathbin{;} parent^{*}$

R11 : frère-sœur

R12 : Démontrer $mere = pere \mathbin{;} mari^{-1}$

Solution des exercices (suite)

R7 : père

$pere = mere \mathbin{;} mari$

R8 : parent

$parent = mere \cup pere$

R9 : enfant

$enfant = parent^{-1}$

R10 : grand-parent et ancêtre

$grand\_parent = parent \mathbin{;} parent$

$ancetre = parent \mathbin{;} parent^{*}$

R11 : frère-sœur

$frere\_soeur = (mere \mathbin{;} mere^{-1}) - \mathrm{id}(personne)$

R12 : Démontrer $mere = pere \mathbin{;} mari^{-1}$

Solution des exercices (suite)

R7 : père

$pere = mere \mathbin{;} mari$

R8 : parent

$parent = mere \cup pere$

R9 : enfant

$enfant = parent^{-1}$

R10 : grand-parent et ancêtre

$grand\_parent = parent \mathbin{;} parent$

$ancetre = parent \mathbin{;} parent^{*}$

R11 : frère-sœur

$frere\_soeur = (mere \mathbin{;} mere^{-1}) - \mathrm{id}(personne)$

R12 : Démontrer $mere = pere \mathbin{;} mari^{-1}$

$$= (mere \mathbin{;} mari) \mathbin{;} mari^{-1} \qquad = mere \mathbin{;} (mari \mathbin{;} mari^{-1})$$

$$= mere \mathbin{;} \mathrm{id}(\mathrm{dom}(mari)) \qquad = mere$$

Partie 3 : Les substitutions généralisées

1. Introduction à la méthode B
2. Formalisme de modélisation
3. Spécification des opérations : substitutions
4. Les machines abstraites
5. Raffinement et implémentation
6. Modularité : raffinement et composition
7. Applications industrielles

Substitutions primitives

$x := E$	substitution simple
$x, y := E, F$	substitution multiple simple
skip	substitution sans effet
$P \mid S$	substitution préconditionnée
$P \Longrightarrow S$	substitution gardée
$S \parallel T$	substitution de choix borné
$@z \cdot S$	substitution de choix non borné
$S\ ;\ T$	séquencement de substitutions
$\mathcal{W}(P, S, J, V)$	substitution d'itération

Spécification des instructions

- Logique des programmes : correction partielle

  $P\{S\}Q$

  Si l'état satisfait P avant S et si S termine,
  alors l'état satisfait Q après.

Spécification des instructions

- Logique des programmes : correction partielle

  $P\{S\}Q$

  Si l'état satisfait P avant S et si S termine,
  alors l'état satisfait Q après.

- Plus faible précondition : correction totale

  $wp(S, Q)$

  Si l'état satisfait $wp(S, Q)$ avant S
  alors S termine et l'état satisfait Q après.

  Remarque : $P\{S\}Q$ en correction totale est équivalent à $P \Rightarrow wp(S, Q)$

Spécification des instructions

- Logique des programmes : correction partielle

  $P\{S\}Q$

  Si l'état satisfait P avant S et si S termine,
  alors l'état satisfait Q après.

- Plus faible précondition : correction totale

  $wp(S, Q)$

  Si l'état satisfait $wp(S, Q)$ avant S
  alors S termine et l'état satisfait Q après.

  Remarque : $P\{S\}Q$ en correction totale est équivalent à $P \Rightarrow wp(S, Q)$

- En B, la plus faible précondition $wp(S, Q)$ est notée sous la forme d'une substitution $[S]\,Q$

WP des substitutions primitives

Cas de substitution	Réduction	Condition
$[x := E]\, R$	$[x := E]\, R$	
$[x, y := E, F]\, R$	$[z := F][x := E][y := z]\, R$	$z \backslash E, F, R$
$[\text{skip}]\, R$	R	
$[P \mid S]\, R$	$P \;\wedge\; [S]\, R$	
$[P \Longrightarrow S]\, R$	$P \Rightarrow [S]\, R$	
$[S \parallel T]\, R$	$[S]R \;\wedge\; [T]\, R$	
$[@z \cdot S]\, R$	$\forall z \cdot [S]\, R$	$z \backslash R$
$[S\, ;\, T]\, R$	$[S]\, ([T]\, R)$	

La notation

Dans les programmes B, les substitutions s'écrivent avec des mots réservés :

Substitution	Notation
$P \mid S$	PRE P THEN S END
$P \Longrightarrow S$	SELECT P THEN S END
$S \parallel T$	CHOICE S OR T END
$@z \cdot S$	VAR z IN S END
$\mathcal{W}(P, S, J, V)$	WHILE P DO S INVARIANT J VARIANT V END

La notation (suite)

$x := E \mid\mid y := F$	$x, y := E, F$
BEGIN S END	(S)
IF P THEN S ELSE T END	$(P \Longrightarrow S) \parallel (\neg\, P \Longrightarrow T)$
CHOICE S OR $T \ldots$ OR U END	$S \parallel T \parallel \cdots \parallel U$
ANY z WHERE P THEN S END	$@z \cdot (P \Longrightarrow S)$
$x :\in E$	ANY x' WHERE $x' \in E$ THEN $x := x'$ END
$x := \text{bool}((P)$	$x :=$ IF P THEN TRUE ELSE FALSE END
$f(x) := E$	$f := f \lhd\!\!- \{(x, E)\}$

Exemples de calcul

- $[x := z + 1; y := x + z](y \in 0..5) = [x := z + 1]([y := x + z](y \in 0..5))$
  $= [x := z + 1](x + z \in 0..5) = (z + 1 + z \in 0..5)$

Exemples de calcul

- $[x := z + 1; y := x + z](y \in 0..5) = [x := z + 1]([y := x + z](y \in 0..5))$
  $= [x := z + 1](x + z \in 0..5) = (z + 1 + z \in 0..5)$

- $[\text{ IF } P \text{ THEN } S \text{ ELSE } T \text{ END}]\, R$
  $= [(P \Longrightarrow S) \,[\!]\, (\neg\, P \Longrightarrow T)]\, R$
  $= [P \Longrightarrow S]\, R \;\wedge\; [\neg\, P \Longrightarrow T]\, R$
  $= (P \Rightarrow [S]\, R) \;\wedge\; (\neg\, P \Rightarrow [T]\, R)$

Exemples de calcul

- $[x := z + 1; y := x + z](y \in 0..5) = [x := z + 1]([y := x + z](y \in 0..5))$
  $= [x := z + 1](x + z \in 0..5) = (z + 1 + z \in 0..5)$

- $[\text{ IF } P \text{ THEN } S \text{ ELSE } T \text{ END}]\, R$
  $= [(P \Longrightarrow S) \,[\!]\, (\neg\, P \Longrightarrow T)]\, R$
  $= [P \Longrightarrow S]\, R \;\wedge\; [\neg\, P \Longrightarrow T]\, R$
  $= (P \Rightarrow [S]\, R) \;\wedge\; (\neg\, P \Rightarrow [T]\, R)$

- $[x :\in E]\, R$
  $= [\text{ ANY } x' \text{ WHERE } x' \in E \text{ THEN } x := x' \text{ END}]\, R$
  $= \forall x' \cdot (x' \in E \Rightarrow [x := x']\, R)$

Un exemple de modélisation

Problème :
On veut spécifier une opération qui alloue un mot dans une mémoire et retourne l'adresse de l'emplacement alloué, s'il y a de la place en mémoire.

Quelques préliminaires de modélisation :

$ADRESSES$	ensemble abstrait d'adresses
$memoire \subseteq ADRESSES$	les adresses de la mémoire à allouer
$libres \subseteq memoire$	l'ensemble des adresses libres
$null \in ADRESSES$	une adresse particulière
$null \notin memoire$	l'adresse $null$ n'est pas en mémoire

Solution 1

Cas 1 : L'opération $allouer$ ne peut agir que s'il reste des adresses libres.
Première modélisation : une précondition assure qu'il reste de la place.

$r \longleftarrow allouer =$	entête de l'opération
PRE $libres \neq \varnothing$ THEN	précondition
ANY v WHERE	
$v \in libres$	choix d'une adresse libre
THEN	
$libres := libres - \{v\} \;\|\|$	modification de l'état
$r := v$	retour de l'adresse allouée
END	
END	

Solution 1 (suite)

Cas 1 : Dans la méthode B, lorsqu'on appelle une opération, il y a une obligation de preuve qui permet d'assurer que la précondition est vérifiée à l'appel.

D'un point de méthode de spécification, il faut, dans ce cas, fournir à l'utilisateur des opérations pour tester *de l'extérieur* si une précondition est vérifiée. On aura ici :

$$b \longleftarrow n\_est\_pas\_pleine = b := \mathsf{bool}(libres \neq \varnothing)$$

Solution 2

Cas 2 : Autre manière de spécifier : l'utilisateur n'a pas à tester la précondition. Si l'adresse de retour est $null$, cela signifie à l'utilisateur que la mémoire est pleine et que l'allocation n'a pas été possible

$r \longleftarrow allouer =$	entête de l'opération
IF $libres \neq \varnothing$ THEN	test dynamique
ANY v WHERE	
$v \in libres$	choix d'une adresse libre
THEN	
$libres := libres - \{v\} \parallel$	modification de l'état
$r := v$	retour de l'adresse allouée
END	
ELSE	il n'y a plus d'adresse libre
$r := null$	retour de la valeur de non allocation
END	

Solution 3

Cas 3 : On pourrait simplement spécifier avec les deux cas possibles de retour de valeur de r :

$r \longleftarrow allouer =$	entête de l'opération
CHOICE	choix interne
ANY v WHERE	
$v \in libres$	
THEN	
$libres := libres - \{v\} \parallel$	modification de l'état
$r := v$	retour de l'adresse allouée
END	
OR	autre possibilité
$r := null$	retour de la valeur de non allocation
END	

Comparez cette solution avec la précédente. Que peut-on dire ?

Caractérisation des substitutions

- Le langage des substitutions généralisées est conçu pour décrire des changement d'états.
- Il y a une grande variété de substitutions.
- Que peut-on dire de commun à toutes les substitutions ?
- Peut-on “ représenter ” les substitutions par l'effet qu'elle produisent comme une relation entre les états ?

Terminaison d'une substitution

La terminaison est un prédicat $\mathsf{trm}(S)$ qui caractérise la terminaison de la substitution S. Définition :

$$\mathsf{trm}(S) \mathrel{\widehat{=}} [S]\,\mathsf{true}$$

Quelques résultats :

$\mathsf{trm}(x := E)$	$\Leftrightarrow$	true
$\mathsf{trm}(\mathsf{skip})$	$\Leftrightarrow$	true
$\mathsf{trm}(P \mid S)$	$\Leftrightarrow$	$P \ \wedge\ \mathsf{trm}(S)$
$\mathsf{trm}(P \Longrightarrow S)$	$\Leftrightarrow$	$P \Rightarrow \mathsf{trm}(S)$
$\mathsf{trm}(S \mathbin{[\!]} T)$	$\Leftrightarrow$	$\mathsf{trm}(S) \ \wedge\ \mathsf{trm}(T)$
$\mathsf{trm}(@z \cdot S)$	$\Leftrightarrow$	$\forall z \cdot \mathsf{trm}(S)$

Prédicat avant-après

Le prédicat avant-après $\mathsf{prd}_x(S)$ donne la relation entre les valeurs avant et après la substitution S pour les variables x. Définition :

$\mathsf{prd}_x(S) \mathrel{\widehat{=}} \neg\,[S]\,(x' \neq x)$ (et $\mathsf{fis}(S) \Leftrightarrow \exists x' \cdot \mathsf{prd}_x(S)$)

$\mathsf{prd}_x(x := E)$	$\Leftrightarrow$	$x' = E$	
$\mathsf{prd}_{x,y}(x := E)$	$\Leftrightarrow$	$x', y' = E, y$	
$\mathsf{prd}_x(\mathsf{skip})$	$\Leftrightarrow$	$x' = x$	
$\mathsf{prd}_x(P \mid S)$	$\Leftrightarrow$	$P \Rightarrow \mathsf{prd}_x(S)$	
$\mathsf{prd}_x(P \Longrightarrow S)$	$\Leftrightarrow$	$P \ \wedge\ \mathsf{prd}_x(S)$	
$\mathsf{prd}_x(S \mathbin{[\!]} T)$	$\Leftrightarrow$	$\mathsf{prd}_x(S) \ \vee\ \mathsf{prd}_x(T)$	
$\mathsf{prd}_x(@z \cdot S)$	$\Leftrightarrow$	$\exists z \cdot \mathsf{prd}_x(S)$	si $z \backslash x'$
$\mathsf{prd}_x(@y \cdot T)$	$\Leftrightarrow$	$\exists (y, y') \cdot \mathsf{prd}_{x,y}(T)$	si $y \backslash x'$

Forme normalisée

Toute substitution peut se mettre sous la forme :

$$S \ = \ \mathsf{trm}(S) \mid @x' \cdot (\mathsf{prd}_x(S) \Longrightarrow x := x')$$

Deux substitutions sont égales si elles ont le même effet sur tout prédicat :

$S = T \ \mathrel{\widehat{=}}\ [S]\,R \Leftrightarrow [T]\,R$ pour tout prédicat R

Les substitutions généralisées satisfont les propriétés :

$[S]\,(R \ \wedge\ Q) \ \Leftrightarrow\ [S]\,R \ \wedge\ [S]\,Q$	Distributivité
$\forall x \cdot (R \Rightarrow Q) \ \Rightarrow\ ([S]\,R \Rightarrow [S]Q)$	Monotonie

On particulier on a (terminaison) : $(R \Rightarrow \mathit{true}) \ \Rightarrow\ ([S]\,R \Rightarrow \mathsf{trm}(S))$

Substitutions généralisées vs prédicats

On a vu que l'on peut passer des substitutions généralisées aux prédicats avant-après et terminaison et vice-versa. Pourquoi choisir les SG pour spécifier, plutôt que les prédicats comme en Z, OCL, JML,. . . ?

- le style d'écriture est plus proche de la programmation
- par défaut, les variables ne sont pas modifiées ($y' = y$)
- l'utilisation des substitutions est plus efficace pour les preuves :

  $$[x := 1 \,;\, x := x + 1]\,(x > 0) \longrightarrow 2 > 0$$

  avec les prédicats : $\exists x_2 \cdot (x_2 = 1 \wedge x' = x_2 + 1) \Rightarrow x' > 0$
- Il y a un continuum entre les spécifications et les programmes à l'aide du raffinement.

Extension : langage avec exceptions

On étend le langage B avec une notion d'exceptions EXC. Valeur prédéfinie : $no \in EXC$ (non utilisable dans le langage)

Nouvelles constructions :

RAISE e	déclenchement d'une exception e
BEGIN S CATCH WHEN e_1 THEN S_1 $\dots$ WHEN e_n THEN S_n END	bloc avec récupération le corps du bloc séquence de traitement des exceptions

Extension du calcul de plus faible précondition : $wpe(S, F)$

avec : $F \in EXC \nrightarrow PostCondition$

Axiomatisation de wpe

$wpe(\text{skip}, F)$	=	$F(no)$
$wpe(x := v, F)$	=	$[x := v]\, F(no)$
$wpe(raise\ e, F)$	=	$F(e)$
$wpe(S_1 \parallel S_2, F)$	=	$wpe(S_1, F) \ \wedge \ wpe(S_2, F)$
$wpe(S_1 \ ; S_2, F)$	=	$wpe(S_1, F \lhd\!\!\!- \{no \mapsto wpe(S_2, F)\})$
$wpe($ BEGIN S CATCH WHEN e_1 THEN S_1 $\dots$ WHEN e_n THEN S_n END, $F)$	=	$wpe(S,$ $F \lhd\!\!\!- \{e_1 \mapsto wpe(S_1, F),$ $\dots$ $e_n \mapsto wpe(S_n, F)\}$ $)$

Exemple de calcul

BEGIN

$x := 1$

; IF $y > 0$ THEN RAISE $stop$

END

; $x := 2$

END

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

Exemple de calcul

BEGIN

$x := 1$

; IF $y > 0$ THEN RAISE $stop$

END

; $x := 2$

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

END

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

Exemple de calcul

BEGIN

$x := 1$

; IF $y > 0$ THEN RAISE $stop$

END

$\{no \mapsto true, stop \mapsto x = 1\}$

; $x := 2$

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

END

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

Exemple de calcul

BEGIN

$x := 1$

; IF $y > 0$ THEN RAISE $stop$

ELSE skip

END

$\{no \mapsto true, stop \mapsto x = 1\}$

; $x := 2$

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

END

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

Exemple de calcul

BEGIN

$x := 1$

; IF $y > 0$ THEN $x = 1$ RAISE $stop$

ELSE $true$ skip

END

$\{no \mapsto true, stop \mapsto x = 1\}$

; $x := 2$

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

END

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

Exemple de calcul

BEGIN

$x := 1$

$\{no \mapsto (y > 0 \Rightarrow x = 1) \ \wedge \ (\neg(y > 0) \Rightarrow true),$
$stop \mapsto x = 1\}$

; IF $y > 0$ THEN $x = 1$ RAISE $stop$

ELSE $true$ skip

END

$\{no \mapsto true, stop \mapsto x = 1\}$

; $x := 2$

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

END

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

Exemple de calcul

BEGIN

$(y > 0 \Rightarrow true) \ \wedge \ (\neg(y > 0) \Rightarrow true)$

$x := 1$

$\{no \mapsto (y > 0 \Rightarrow x = 1) \ \wedge \ (\neg(y > 0) \Rightarrow true),$

$stop \mapsto x = 1\}$

; IF $y > 0$ THEN $x = 1$ RAISE $stop$

ELSE $true$ skip

END

$\{no \mapsto true, stop \mapsto x = 1\}$

; $x := 2$

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

END

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

Exemple de calcul

BEGIN

$true$

$x := 1$

$\{no \mapsto (y > 0 \Rightarrow x = 1) \ \wedge \ (\neg(y > 0) \Rightarrow true),$

$stop \mapsto x = 1\}$

; IF $y > 0$ THEN $x = 1$ RAISE $stop$

ELSE $true$ skip

END

$\{no \mapsto true, stop \mapsto x = 1\}$

; $x := 2$

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

END

$\{no \mapsto x = 2, stop \mapsto x = 1\}$

Justification de la sémantique

Equivalence de sémantique entre un programme avec exceptions et un programme sans exception : ajout d'une variable exc qui simule l'exception courante .

Définition de $\mathcal{C}(S)$:

$$
\begin{array}{lcl}
\mathcal{C}(x := v) & = & x := v \ ; \ exc := no \\
\mathcal{C}(\text{skip}) & = & exc := no \\
\mathcal{C}(\text{RAISE } e) & = & exc := e \\
\mathcal{C}(S1 \ ; \ S2) & = & \mathcal{C}(S1) \ ; \ \text{IF } exc = no \text{ THEN } \mathcal{C}(S2) \text{ END} \\
\ldots & &
\end{array}
$$

Quelle postcondition R pour que $wp(S,F) \equiv wpe(\mathcal{C}(S), R)$?

Justification de la sémantique

Equivalence de sémantique entre un programme avec exceptions et un programme sans exception : ajout d'une variable exc qui simule l'exception courante .

Définition de $\mathcal{C}(S)$:

$$
\begin{array}{lcl}
\mathcal{C}(x := v) & = & x := v \ ; \ exc := no \\
\mathcal{C}(\text{skip}) & = & exc := no \\
\mathcal{C}(\text{RAISE } e) & = & exc := e \\
\mathcal{C}(S1 \ ; \ S2) & = & \mathcal{C}(S1) \ ; \ \text{IF } exc = no \text{ THEN } \mathcal{C}(S2) \text{ END} \\
\ldots & &
\end{array}
$$

Quelle postcondition R pour que $wp(S,F) \equiv wpe(\mathcal{C}(S), R)$?

$$\boxed{wpe(S,F) \ \Leftrightarrow \ [\mathcal{C}(S)] \, (\bigwedge_{e_i \in dom(F)} (exc = e_i \Rightarrow F(e_i)))}$$

Exemple de preuve

$$wpe(S,F) \;\Leftrightarrow\; [\mathcal{C}(S)]\,(\bigwedge_{e_i \in dom(F)}(exc = e_i \Rightarrow F(e_i)))\ ?$$

- Affectation ($wpe(x := v, F) = F(no)$) :

Exemple de preuve

$$wpe(S,F) \;\Leftrightarrow\; [\mathcal{C}(S)]\,(\bigwedge_{e_i \in dom(F)}(exc = e_i \Rightarrow F(e_i)))\ ?$$

- Affectation ($wpe(x := v, F) = F(no)$) :

$$[x := v\ ;\ exc := no] \bigwedge_{e_i \in dom(F)}(exc = e_i \Rightarrow F(e_i))$$
$$= [x := v][exc := no] \bigwedge_{e_i \in dom(F)}(exc = e_i \Rightarrow F(e_i))$$
$$= [x := v] \bigwedge_{e_i \in dom(F)}(no = e_i \Rightarrow F(e_i))$$
$$= [x := v]F(no) \wedge \bigwedge_{e_i \in dom(F)-\{no\}}(no = e_i \Rightarrow F(e_i))$$
$$= [x := v]F(no)$$

Exemple de preuve (2)

$$wpe(S,F) \;\Leftrightarrow\; [\mathcal{C}(S)]\,(\bigwedge_{e_i \in dom(F)}(exc = e_i \Rightarrow F(e_i)))\ ?$$

- Séquencement ($wpe(S_1, F \lhd\!\!- \{no \mapsto wpe(S_2, F)\})$) :

Exemple de preuve (2)

$$wpe(S,F) \;\Leftrightarrow\; [\mathcal{C}(S)]\,(\bigwedge_{e_i \in dom(F)}(exc = e_i \Rightarrow F(e_i)))\ ?$$

- Séquencement ($wpe(S_1, F \lhd\!\!- \{no \mapsto wpe(S_2, F)\})$) :

$$[\mathcal{C}(S1)\ ;\ \ \text{IF } exc = no \text{ THEN } \mathcal{C}(S2) \text{ END}] \bigwedge_{e_i \in dom(F)}(exc = e_i \Rightarrow F(e_i))$$
$$= [\mathcal{C}(S1)][\text{IF } exc = no \text{ THEN } \mathcal{C}(S2) \text{ END}] \bigwedge_{e_i \in dom(F)}(exc = e_i \Rightarrow F(e_i))$$
$$= [\mathcal{C}(S1)]\ (exc = no \Rightarrow ([\mathcal{C}(S2)] \bigwedge_{e_i \in dom(F)}(exc = e_i \Rightarrow F(e_i))))$$
$$\wedge\ \ (exc \neq no \Rightarrow \bigwedge_{e_i \in dom(F)}(exc = e_i \Rightarrow F(e_i)))$$

Exemple de preuve (2)

$$wpe(S,F) \Leftrightarrow [\mathcal{C}(S)]\,(\bigwedge_{e_i \in dom(F)} (exc = e_i \Rightarrow F(e_i)))\ ?$$

- Séquencement ($wpe(S_1, F \lhd\!\!\!- \{no \mapsto wpe(S_2,F)\})$) :

$[\mathcal{C}(S1)\ ;\ \text{IF } exc = no \text{ THEN } \mathcal{C}(S2) \text{ END}] \bigwedge_{e_i \in dom(F)} (exc = e_i \Rightarrow F(e_i))$

$= [\mathcal{C}(S1)][\text{IF } exc = no \text{ THEN } \mathcal{C}(S2) \text{ END}] \bigwedge_{e_i \in dom(F)} (exc = e_i \Rightarrow F(e_i))$

$= [\mathcal{C}(S1)]\ (exc = no \Rightarrow ([\mathcal{C}(S2)] \bigwedge_{e_i \in dom(F)} (exc = e_i \Rightarrow F(e_i))))$
$\quad \wedge\ (exc \neq no \Rightarrow \bigwedge_{e_i \in dom(F)} (exc = e_i \Rightarrow F(e_i)))$

$= [\mathcal{C}(S1)](exc = no \Rightarrow wpe(S2,F))$
$\quad \wedge\ \bigwedge_{e_i \in dom(F) - \{no\}} (exc = e_i \Rightarrow F(e_i))$

$= [\mathcal{C}(S1)] \bigwedge_{e_i \in dom(F)} (exc = e_i \Rightarrow F \lhd\!\!\!- \{no \mapsto wpe(S_2,F)\}(e_i))$

$= wpe(S1, F \lhd\!\!\!- \{no \mapsto wpe(S_2,F)\})$

Partie 4 : Les machines abstraites

Composant machine

MACHINE
- *Partie entête :*
  - *nom de la machine*
- *Partie statique :*
  - *déclaration d'ensembles et de constantes*
  - *propriétés des constantes*
  - *variables (état)*
  - *invariant (caractérisation de l'état)*
- *Partie dynamique :*
  - *initialisation de l'état*
  - *opérations*

END

Rubriques d'une machine

MACHINE M	
SETS S;	*/* ensembles donnés */*
$T = \{a, b\}$	*/* ensembles énumérés */*
CONSTANTS c	*/* liste de constantes (concrètes) */*
PROPERTIES C	*/* spécification des constantes */*
VARIABLES x	*/* liste de variables (abstraites) */*
INVARIANT I	*/* spécification des variables */*
INITIALISATION U	*/* substitution d'initialisation */*
OPERATIONS	*/* liste des opérations */*
$r \longleftarrow nom\_op(p)$ = PRE P THEN K END; ...	
END	

Obligations de preuves d'une machine

- L'initialisation établit l'invariant :
  B : ensembles déclarés sont finis et non vides et les constantes énumérées sont distinctes.

$$B \ \wedge \ C \ \Rightarrow \ [U]I$$

- Chaque opération préserve l'invariant :

$$B \ \wedge \ C \ \wedge \ I \ \wedge \ P \Rightarrow [K]I$$

$\Rightarrow$ Par la propriété de terminaison, on assure que K termine :

$$B \ \wedge \ C \ \wedge \ I \ \wedge \ P \Rightarrow \mathsf{trm}(K)$$

Atelier B : production des obligations de preuve (initialisation et un ensemble d'OPs par opération), preuve automatique ou interactive.

Exemple de l'ascenseur

- exprimer des propriétés
  - par des spécifications
  - par des invariants
- utiliser la preuve pour détecter des problèmes
  - incohérence entre invariant et comportement
  - invariant non inductif
- exemple d'utilisation de l'atelier B

Ascenseur

On souhaite spécifier le fonctionnement simplifié d'un ascenseur.

- une porte à chaque étage
- l'appel intérieur et l'appel extérieur ne sont pas distingués
- il n'y a pas de panne
- une constante donne le nombre d'étages : max_etage (> 0)

Les opérations sont :

- ouvrir, fermer une porte
- appeler l'ascenseur
- déplacement de l'ascenseur

Propriétés de l'ascenseur

- l'ascenseur reste dans la limite des étages
- si une porte est ouverte l'ascenseur est arrêté à l'étage correspondant
- chaque appel est traité en un temps raisonnable
- si l'ascenseur est arrêté à un étage, l'appel à cet étage est considéré comme traité
- ...

Modélisation de l'ascenseur

MACHINE $ASCENSEUR$

SETS $MODE = \{arret, mouv\}$

CONSTANTS $max\_etage, ETAGES$

PROPERTIES $max\_etage \in \mathsf{NAT}_1 \wedge ETAGES = 0..max\_etage$

VARIABLES $appels, ouvertes, pos, mode$

INVARIANT

$$ouvertes \subseteq ETAGES \ \wedge \ appels \subseteq ETAGES$$
$$\wedge \ pos \in ETAGES \ \wedge \ mode \in MODE$$

Modélisation de l'ascenseur

MACHINE $ASCENSEUR$

SETS $MODE = \{arret, mouv\}$

CONSTANTS $max\_etage, ETAGES$

PROPERTIES $max\_etage \in \mathsf{NAT}_1 \wedge ETAGES = 0..max\_etage$

VARIABLES $appels, ouvertes, pos, mode$

INVARIANT

$$ouvertes \subseteq ETAGES \ \wedge \ appels \subseteq ETAGES$$
$$\wedge \ pos \in ETAGES \ \wedge \ mode \in MODE$$
$$\wedge \ (ouvertes \neq \emptyset \Rightarrow ouvertes = \{pos\} \wedge mode = arret)$$
$$\wedge \ (mode = arret \Rightarrow pos \notin appels)$$

Plan de la démo

- spécification des invariants
- spécification des opérations
- obligations de preuve (appeler, fermer)
- preuve
- erreur de spécification (ASCENSEUR_FAUX)
- détection des erreurs à partir des obligations de preuve

Opération : déclaration et appel

Une déclaration d'opération est de la forme :

$$r \longleftarrow op(p) \quad = \quad \text{PRE } P \text{ THEN } K \text{ END}$$

avec r affecté dans S (plus formellement $r \backslash \mathsf{prd}_{x,r}(S)$)

Un appel d'opération se présente sous la forme $v \leftarrow op(e)$ avec :

- e un n-uplet d'expressions
- v un n-uplet de variables ne contient pas de doublon ;
- les variables v sont disjointes des variables de la machine dans laquelle l'opération est définie

$\Rightarrow$ utilisation encapsulée des machines.

Opération : déclaration et appel

Une déclaration d'opération est de la forme :

$$r \longleftarrow op(p) \quad = \quad \text{PRE } P \text{ THEN } K \text{ END}$$

avec r affecté dans S (plus formellement $r \backslash \mathsf{prd}_{x,r}(S)$)

Un appel d'opération se présente sous la forme $v \leftarrow op(e)$ avec :

- e un n-uplet d'expressions
- v un n-uplet de variables ne contient pas de doublon ;
- les variables v sont disjointes des variables de la machine dans laquelle l'opération est définie

$\Rightarrow$ utilisation encapsulée des machines.

On veut montrer que si la préservation de l'invariant est établie sur la définition alors il est préservé par les appels.

Sémantique par copie

Soit $r \leftarrow op(p) \hat{=}$ PRE P THEN S END la définition d'une opération de nom op et soit l'appel $v \leftarrow op(e)$. Sa définition est :

PRE $([p := e]P)$

THEN VAR p, r IN $p := e$; S; $v := r$ END

END

et on a :

$$\forall r, p \ \ (I \wedge P \Rightarrow [S]I)$$

$$\Rightarrow$$

$$(I \wedge [p := e]P \Rightarrow [\text{VAR } p, r \ \text{ IN } p := e \ ; \ S; \ v := r \ \text{ END}]I)$$

Preuve

On déduit de $\forall p, r\ (I \wedge P \Rightarrow [S]I)$:

$$\forall r\ (I \wedge [p := e]P \Rightarrow [p := e][S]I) \qquad (1)$$

par instanciation de p par e et p non libre dans I.

De plus $[var\ p, r\ in\ p := e\ ;\ S;\ v := r\ end]I$ devient, par définition du calcul de plus faible précondition :

$$\forall\ p, r\ [p := e][S][v := r]I \qquad (2)$$

qui se réduit, puisque v est non libre dans I, à $\forall\ p, r\ [p := e][S]I$.

Or p n'apparait pas dans $[p := e][S]I$ puisque p n'apparaît pas dans e. Donc $I \wedge [p := e]P \Rightarrow (2)$ se déduit de $\forall p, r\ (I \wedge P \Rightarrow [S]I)$.

Sémantique par référence

L'appel par référence (le remplacement) ne permet pas de préserver les propriétés.

Soit par exemple l'opération suivante :

$op(y) \hat{=}$

PRE $pair(y)$

THEN $x := x + 1; x := x + y + 1$

END

Cette opération préserve l'invariant $pair(x)$. Par contre l'appel $op(x)$ devient $x := x + 1; x := x + x + 1$ qui ne préserve pas l'invariant.

Partie 5 : Raffinement

1. Introduction à la méthode B
2. Formalisme de modélisation
3. Spécification des opérations : substitutions
4. Les machines abstraites
5. RAFFINEMENT
6. Modularité : raffinement et composition
7. Applications industrielles

Raffinement : principe

- Le raffinement est le fait de transformer une spécification abstraite en un texte plus proche de la programmation, pour finalement obtenir un programme
- L'effet des appels d'opérations de la machine abstraite doit être préservé, vu de l'utilisateur
- Le raffinement de machine se fait opération par opération
- Il y a (éventuellement) raffinement de l'état
- Pour chaque opération :
  - reformulation en fonction du changement d'état
  - affaiblissement des préconditions
  - réduction du non-déterminisme

Exemple : une machine

MACHINE $RAFF\_EX1$

VARIABLES yy

INVARIANT $yy \subseteq \mathsf{NAT}_1$

INITIALISATION $yy := \varnothing$

OPERATIONS

$ajouter(nn)$ = PRE $nn \in \mathsf{NAT}_1$
THEN $yy := yy \cup \{nn\}$
END;

$vv \longleftarrow choix$ = PRE $yy \neq \varnothing$
THEN $vv :\in yy$
END

END

Un raffinement

Une machine qui “ fait presque la même chose ” :

MACHINE $RAFF\_EX2$

VARIABLES zz

INVARIANT $zz \in \mathsf{NAT}$

INITIALISATION $zz := 0$

OPERATIONS

$ajouter(nn)$ = PRE $nn \in \mathsf{NAT}_1$
THEN $zz := nn$
END;

$vv \longleftarrow choix$ = $vv := zz$

END

Un raffinement

Une machine qui “ fait presque la même chose ” :

MACHINE $RAFF\_EX2$

VARIABLES zz

INVARIANT $zz \in$ NAT

INITIALISATION $zz := 0$

OPERATIONS

$ajouter(nn)$ = PRE $nn \in$ NAT$_1$

THEN $zz := nn$

END;

$vv \longleftarrow choix$ = $vv := zz$

END

Relation de simulation : $zz \in yy \ \vee \ yy = \emptyset$

Raffinement de substitution

Sans changement de représentation :

Définition du raffinement de S par T :

$S \sqsubseteq T$	$\forall R \cdot ([S]\, R \ \Rightarrow \ [T]\, R)$

Si S préserve l’invariant, alors le raffinement le préserve.

Raffinement de substitution

Sans changement de représentation :

Définition du raffinement de S par T :

$S \sqsubseteq T$	$\forall R \cdot ([S]\, R \ \Rightarrow \ [T]\, R)$

Si S préserve l’invariant, alors le raffinement le préserve.

Autre définition :

$S \sqsubseteq T$	$\mathsf{trm}(S) \ \Rightarrow \ \mathsf{trm}(T)$ $\mathsf{trm}(S) \ \wedge \ \mathsf{prd}_x(T) \ \Rightarrow \ \mathsf{prd}_x(S)$

Avec changement de représentation

Définition :

$S \sqsubseteq_L T$	$L \ \wedge \ \mathsf{trm}(S) \ \Rightarrow \ \mathsf{trm}(T)$ $L \ \wedge \ \mathsf{prd}_y(T) \ \Rightarrow \ \exists x' \cdot (\mathsf{prd}_x(S) \ \wedge \ [x, y := x', y']\, L)$

Commutation de diagramme :

$$\begin{tikzcd}
X \arrow[r, "S"] \arrow[d, shift left] & X' \arrow[d, shift left] \\
Y \arrow[u, shift left, "L"] \arrow[r, "T"'] & Y' \arrow[u, shift left, "L"]
\end{tikzcd}$$

Obligations de preuve

Définition de $S \sqsubseteq_L T$:

$$L \wedge \mathsf{trm}(S) \Rightarrow \mathsf{trm}(T)$$
$$L \wedge \mathsf{prd}_y(T) \Rightarrow \exists x' \cdot (\mathsf{prd}_x(S) \wedge [x, y := x', y']\, L)$$

Obligations de preuve

Définition de $S \sqsubseteq_L T$:

$$L \wedge \mathsf{trm}(S) \Rightarrow \mathsf{trm}(T)$$
$$L \wedge \mathsf{prd}_y(T) \Rightarrow \exists x' \cdot (\mathsf{prd}_x(S) \wedge [x, y := x', y']\, L)$$

Formulation utilisée pour la preuve :

$$L \wedge \mathsf{trm}(S) \Rightarrow [T]\neg[S]\neg L$$

Obligations de preuve

Définition de $S \sqsubseteq_L T$:

$$L \wedge \mathsf{trm}(S) \Rightarrow \mathsf{trm}(T)$$
$$L \wedge \mathsf{prd}_y(T) \Rightarrow \exists x' \cdot (\mathsf{prd}_x(S) \wedge [x, y := x', y']\, L)$$

Formulation utilisée pour la preuve :

$$L \wedge \mathsf{trm}(S) \Rightarrow [T]\neg[S]\neg L$$

Exemple : $x :\in e \sqsubseteq_{z \in e} x := z$

$$z \in e \Rightarrow [x_1 := z]\neg\forall\, v\ (v \in e \Rightarrow [x_2 := v]\neg(x_1 = x_2))$$
$$z \in e \Rightarrow [x_1 := z]\exists\, v\ (v \in e \wedge\ \neg\neg(x_1 = v))$$
$$z \in e \Rightarrow \exists\, v\ (v \in e \wedge\ z = v)$$

Equivalence des deux formulations

$L \wedge \mathsf{trm}(S) \Rightarrow [T]\neg[S]\neg L$

forme normale de T : $\mathsf{trm}(T) \mid @y' \cdot (\mathsf{prd}_y(T) \Longrightarrow y := y')$

(a) $L \wedge \mathsf{trm}(S) \Rightarrow \mathsf{trm}(T)$

(b) $L \wedge \mathsf{trm}(S) \wedge \mathsf{prd}_y(T) \Rightarrow [y := y'](\neg[S]\neg L)$

forme normale de S : $\mathsf{trm}(S) \mid @y' \cdot (\mathsf{prd}_x(S) \Longrightarrow x := x')$

$\neg[S]\neg L \Leftrightarrow (\mathsf{trm}(S) \Rightarrow \exists x' \cdot (\mathsf{prd}_x(S) \wedge [x := x']\, L))$

(b) $L \wedge \mathsf{trm}(S) \wedge \mathsf{prd}_y(T) \Rightarrow \exists x' \cdot (\mathsf{prd}_x(S) \wedge [x, y := x', y']\, L)$

on peut montrer :

$L \wedge \neg\mathsf{trm}(S) \Rightarrow \exists x' \cdot (\mathsf{prd}_x(S) \wedge [x, y := x', y']\, L)$

D'où le résultat.

Raffinement de machines : Syntaxe

MACHINE M
VARIABLES x
INVARIANT I
INITIALISATION U
OPERATIONS
  $r \longleftarrow nom\_op(w)$ =
    PRE P THEN K END
END

REFINEMENT N REFINES M
VARIABLES y
INVARIANT J
INITIALISATION V
OPERATIONS
  $r \longleftarrow nom\_op(w)$ =
    PRE Q THEN L END
END

Raffinement de machines : Syntaxe

MACHINE M
VARIABLES x
INVARIANT I
INITIALISATION U
OPERATIONS
  $r \longleftarrow nom\_op(w)$ =
    PRE P THEN K END
END

REFINEMENT N REFINES M
VARIABLES y
INVARIANT J
INITIALISATION V
OPERATIONS
  $r \longleftarrow nom\_op(w)$ =
    PRE Q THEN L END
END

Pour chaque opération	Initialisation
$I \wedge J \wedge P \Rightarrow Q$	$[V] \neg [U] \neg J$
$I \wedge J \wedge P \Rightarrow [L] \neg [K] \neg J$ $I \wedge J \wedge P \Rightarrow [[r := r']L] \neg [K] \neg (J \wedge r = r')$	

Raffinement : un exemple (1)

MACHINE $ATTENTE$
VARIABLES $attente, nb\_elem$
INVARIANT $attente \subseteq INT \wedge nb\_elem \in NAT \wedge nb\_elem = card(attente)$
INITIALISATION $nb\_elem := 0 \mid\mid attente := \emptyset$
OPERATIONS

$nb \longleftarrow nb\_elem \hat{=}$ BEGIN $nb := nb\_elem$ END ;

$ajouter(v) \hat{=}$ PRE $v \in INT \wedge v \notin attente \wedge nb\_elem < MAXINT$
  THEN $attente := attente \cup \{v\} \mid\mid nb\_elem := nb\_elem + 1$ END ;

$v \longleftarrow traiter \hat{=}$ PRE $attente \neq \emptyset$ THEN
  ANY val WHERE $val \in INT \wedge val \in attente$
  THEN $v := val \mid\mid attente := attente - val$
    $\mid\mid nb\_elem := nb\_elem - 1$ END
  END
END

Raffinement : un exemple (2)

REFINEMENT $ATTENTE\_R1$
REFINES $ATTENTE$
VARIABLES $file, b1, b2$

INVARIANT $file : NAT \nrightarrow INT \wedge b1 \in NAT \wedge b2 \in NAT$
  $\wedge \ \ldots$

INITIALISATION $b1 := 1 \mid\mid b2 := 1 \mid\mid file := \emptyset$
OPERATIONS
  $nb \longleftarrow nb\_elem \hat{=}$ BEGIN $nb := b2 - b1$ END ;
  $ajouter(v) \hat{=}$ BEGIN $file(b2) := v \mid\mid b2 := b2 + 1$ END ;
  $v \longleftarrow traiter \hat{=}$ BEGIN $v := file(b1) \mid\mid b1 := b1 + 1$ END
END

Raffinement : un exemple (2)

REFINEMENT $ATTENTE\_R1$

REFINES $ATTENTE$

VARIABLES $file, b1, b2$

INVARIANT $file : NAT \nrightarrow INT \wedge\ b1 \in\ NAT \wedge\ b2 \in\ NAT$

$\wedge\ file[b1..b2-1] = attente \wedge\ b2 - b1 = nb\_elem$

INITIALISATION $b1 := 1 \mid\mid b2 := 0 \mid\mid file := \emptyset$

OPERATIONS

$nb \longleftarrow nb\_elem \hat{=}$ BEGIN $nb := b2 - b1 + 1$ END ;

$ajouter(v) \hat{=}$ BEGIN $file(b2+1) := v \mid\mid b2 := b2 + 1$ END ;

$v \longleftarrow traiter \hat{=}$ BEGIN $v := file(b1) \mid\mid b1 := b1 + 1$ END

END

Partie 6 : résultats généraux et raffinemen

Raffinement et Simulation

Soit M un composant raffiné par R. Une substitution U est dite externe pour M et R si elle ne contient aucune référence directe aux variables v_M ou v_R.

Le principe de substitution de M par R peut de définir par :

- R offre les mêmes opérations que M avec les mêmes signatures
- Toute substitution externe U pour M et R est telle que :

$\Rightarrow$ Cette définition n'est pas opérationnelle : on raisonne sur toutes les utilisations possibles d'une interface.

Simulation = le raffinement opération par opération (imposant donc un invariant de représentation). Est-il correct ? Complet ?

Principe

Méthode effective :

1. une relation d'abstraction α qui lie les valeurs abstraites et les valeurs concrètes
2. Une notion de commutativité du raffinement ($\subseteq$)

$$\begin{tikzcd}
X \arrow[r, "A"] \arrow[d, shift left, "\alpha^{-1}"] & X' \arrow[d, shift left, "\alpha^{-1}"] \\
Y \arrow[r, "C"'] \arrow[u, shift left, "\alpha"] & Y' \arrow[u, shift left, "\alpha"]
\end{tikzcd}$$

$\Rightarrow$ plusieurs façons de faire commuter le diagramme.

L et L^{-1} simulations

$$\begin{tikzcd} X \arrow[r, "A"] \arrow[d, shift left, "\alpha^{-1}"] & X' \arrow[d, shift left, "\alpha^{-1}"] \\ Y \arrow[u, shift left, "\alpha"] \arrow[r, "C"'] & Y' \arrow[u, shift left, "\alpha"] \end{tikzcd}$$

- L-simulation (forward ou downward simulation) :

$$\alpha^{-1} \; ; \; C \; \subseteq \; A \; ; \; \alpha^{-1}$$

i.e : $\forall a, c' \; (\exists c \; (\alpha \wedge C) \Rightarrow \exists a' (A \wedge \alpha'))$

- L^{-1}-simulation (backward ou upward simulation) :

$$C \; ; \; \alpha \; \subseteq \; \alpha \; ; \; A$$

i.e : $\forall c, a' \; (\exists c' \; (C \wedge \alpha') \Rightarrow \exists a \; (\alpha \wedge A))$

Correction

S'il existe α tel que :

- $init_M \sqsubseteq_\alpha init_R$
- $S \sqsubseteq_\alpha T$ pour chaque opération

alors pour chaque utilisation externe U pour M et R :

- $@\, v_M \; . \; init_M \; ; U_M \;\; \sqsubseteq_{id} \;\; @\, v_R \; . \; init_R \; ; U_R$

$\Rightarrow$ correction des L et L^{-1} simulations

Complétude

Pour chaque utilisation externe U pour M et R :

- $@\, v_M \; . \; init_M \; ; U_M \;\; \sqsubseteq_{id} \;\; @\, v_R \; . \; init_R \; ; U_R$

alors il existe α tel que :

- $init_M \sqsubseteq_\alpha init_R$
- $S \sqsubseteq_\alpha T$ pour chaque opération

$\Rightarrow$ incomplétude de chaque simulation
$\Rightarrow$ complétude des deux simulations L et L^{-1}

Exemple

MACHINE $CASINO1$ VARIABLES i INVARIANT $i \in 0..36$ INITIALISATION $i :\in 0..36$ OPERATIONS $r1 \longleftarrow spin \widehat{=} r1 := i \;\|\|\; i :\in 0..36$ END	MACHINE $CASINO2$ OPERATIONS $r2 \longleftarrow spin \widehat{=} r2 :\in 0..36$ END

Même interface produisant les mêmes résultats :

CASINO1 : $@\, i \; . \; init \; ; \; v_1 \; \longleftarrow \; spin \; ; \; \ldots \; ; \; v_n \; \longleftarrow \; spin$
CASINO2 : $v_1 \; \longleftarrow \; spin \; ; \; \ldots \; ; \; v_n \; \longleftarrow \; spin$

Exemple (2)

$\Rightarrow$ casino2 $\sqsubseteq^L$ casino1 :

$$i \in 0..36 \Rightarrow \; [r1 := i \mid\mid i :\in 0..36]\neg[r2 :\in 0..36]\neg(r1 = r2)$$
$$i \in 0..36 \Rightarrow \; [r1 := i \mid\mid i :\in 0..36]\exists r2(r2 \in 0..36 \wedge r1 = r2)$$
$$i \in 0..36 \Rightarrow \; [r1 := i \mid\mid i :\in 0..36]r1 \in 0..36$$
$$i \in 0..36 \Rightarrow \; i \in 0..36$$

$\Rightarrow$ casino1 $\not\sqsubseteq^L$ casino2 :

$$i \in 0..36 \Rightarrow \; [r2 :\in 0..36]\neg[r1 := i \mid\mid ii :\in 0..36]\neg(r1 = r2)$$
$$i \in 0..36 \Rightarrow \; \forall r2(r2 \in 0..36 \Rightarrow \; i = r2)$$
$$i \in 0..36 \wedge \; r2 \in 0..36 \Rightarrow \; i = r2$$
$$false$$

S. Dunne, ZB 2003

Exemple (3)

$\Rightarrow$ casino1 $\sqsubseteq^{L^{-1}}$ casino2 :

$$\forall r1'(\exists r2'(r2' \in 0..36 \wedge r1' = r2') \Rightarrow \exists i(i \in 0..36 \wedge r1' = i))$$
$$\forall r1'(r1' \in 0..36 \Rightarrow \exists i(i \in 0..36 \wedge r1' = i))$$
$$\forall r1'(r1' \in 0..36 \Rightarrow r1' \in 0..36)$$
$$true$$

Rappel : $\forall c, a' \; (\exists c' \; (C \wedge \alpha') \Rightarrow \exists a \; (\alpha \wedge A))$

Autres simulations

$$\begin{tikzcd} X \arrow[r, "A"] \arrow[d, shift right, "\alpha^{-1}"'] & X' \arrow[d, shift left, "\alpha^{-1}"] \\ Y \arrow[r, "C"'] \arrow[u, shift right, "\alpha"] & Y' \arrow[u, shift left, "\alpha"'] \end{tikzcd}$$

- U-simulation : $\alpha^{-1} \; ; \; C \; ; \; \alpha \; \subseteq \; A$
- U^{-1}-simulation : $C \; \subseteq \; \alpha; A; \alpha^{-1}$

U simulation correcte si α est totale ($id_C \subseteq \alpha \; ; \; \alpha^{-1}$)

U^{-1} simulation correcte si α est fonctionnelle ($\alpha \; ; \; \alpha^{-1} \subseteq id_A$)

$\Rightarrow$ Si α est une fonction totale alors équivalence de ces différentes notions.

Propriétés importantes

- La transitivité (raffinements successifs)

$$\boxed{S \sqsubseteq_{\alpha_1} T \wedge \; T \sqsubseteq_{\alpha_2} U \Rightarrow S \sqsubseteq_{\alpha_1 \; ; \; \alpha_2} U}$$

- La monotonie (raffinement par partie)

$$\begin{array}{|lcl|}
\hline
S \sqsubseteq_\alpha T & \Rightarrow & (P \mid S) \sqsubseteq_\alpha (P \mid T) \\
(U \sqsubseteq_\alpha V) \; \wedge \; (S \sqsubseteq_\alpha T) & \Rightarrow & (U \parallel S) \sqsubseteq_\alpha (V \parallel T) \\
S \sqsubseteq_\alpha T & \Rightarrow & (P \Longrightarrow S) \sqsubseteq_\alpha (P \Longrightarrow T) \\
\forall z \cdot (S \sqsubseteq_\alpha T) & \Rightarrow & @z \cdot S \sqsubseteq_\alpha @z \cdot T \\
(U \sqsubseteq_\alpha V) \; \wedge \; (S \sqsubseteq_\alpha T) & \Rightarrow & (U \; ; S) \sqsubseteq_\alpha (V \; ; T) \\
\hline
\end{array}$$

Monotonie de l’appel d’opération

Préservation des preuves de raffinement

- définition abstraite : $r \leftarrow op(p) \hat{=} S_1$
- définition concrète : $r \leftarrow op(p) \hat{=} S_2$
- appel : $v \leftarrow op(e)$ et e ne contient aucune occurrence des variables de la machine (ni de son raffinement)

alors :

$$[r := r_1]S_1 \sqsubseteq_{J \wedge r_1 = r_2} [r := r_2]S_2$$
$$\Rightarrow$$
$$[v := v_1][p, r := e, v]S_1 \sqsubseteq_{J \wedge v_1 = v_2} [v := v_2][p, r := e, v_2]S_2$$

Preuve basée sur les formes normalisées.

Partie 6 : Implémentation

Implémentation (1)

- Dernier raffinement d’un développement
- Langage de programmation séquentiel
- Restriction sur les substitutions
  - substitutions déterministes (:=, IF, CASE, skip, “;”)
  - plus de précondition
  - prédicats des conditions = calcul booléen
- Ajout d’instructions de programmation :
  - substitution VAR
  - substitution d’itération

Un programme est un témoin : la faisabilité ($\exists\, x'$ prd(x, x')) est garantie par construction.

Implémentation (2)

- Restrictions de typage :
  - les ensembles de valeurs sont finis (ex: entiers bornés NAT et INT)
  - constantes et variables sont de type “concret”: entiers, énumérés, ensembles donnés, tableaux (fonctions totales à domaine fini)
- Les ensembles donnés et les constantes sont valués.
- Obligations de preuve pour l’absence d’erreur à l’exécution
  - $x := e$ devient PRE $e \in type(x)$ THEN $x := e$ END
  - ordre d’évaluation imposé : $x + y + z$ découpé en $temp := x + y$ et $y + temp$

$\Rightarrow$ le niveau B0 est translatable en un programme (C, Ada, ...) qui est correct vis-à-vis de la spécification initiale, termine et est sans erreur à l’exécution.

Plus faible précondition de l'itération

En correction totale, il faut assurer que la boucle se termine dans l'état de la postcondition :

$[$ WHILE P DO S
INVARIANT J
VARIANT V END $]\,R$

$\Leftrightarrow$

$J\ \wedge$	invariant
$\forall x \cdot ((J\ \wedge\ P) \Rightarrow [S]\,J)\ \wedge$	préservation de l'invariant
$\forall x \cdot (J \Rightarrow V \in \mathbb{N})\ \wedge$	variant
$\forall x \cdot ((J\ \wedge\ P) \Rightarrow [n := V][S](V < n))\ \wedge$	décroissance du variant
$\forall x \cdot ((J\ \wedge\ \neg P) \Rightarrow R)$	sortie de boucle

Programme de la division entière

```
MACHINE
   DIVISION
OPERATIONS                       /* qq et rr sont le quotient et le reste */
   qq, rr ⟵ divi(aa, bb) =       /* de la division entière de aa par bb */
      PRE
         aa ∈ NAT ∧ bb ∈ NAT1
      THEN
         ANY ss, tt WHERE
            ss ∈ NAT ∧ tt ∈ NAT ∧
            aa = bb * ss + tt ∧ tt < bb
         THEN
            qq, rr := ss, tt
         END
      END
END
```


Implémentation de la division entière

```
IMPLEMENTATION DIVISION_I REFINES DIVISION
OPERATIONS
   qq, rr ⟵ divi(aa, bb) =
      VAR ss, tt IN              /* variables locales auxiliaires */
         ss := 0 ;               /* initialisations */
         tt := aa ;
         WHILE tt ≥ bb DO
            ss := ss + 1 ;       /* corps de la boucle */
            tt := tt − bb
         INVARIANT               /* conditions invariantes */
            ...
         VARIANT                 /* valeur entière qui décroît */
            ...
         END ;
         qq := ss ; rr := tt     /* retour du résultat */
      END
END
```


Solution ... à tester !

```
qq, rr ⟵ divi(aa, bb) =
   VAR ss, tt IN
      ss := 0 ;            /* initialisations */
      tt := aa ;
      WHILE tt ≥ bb DO
         ss := ss + 1 ;
         tt := tt − bb
      INVARIANT
         ss ∈ NAT ∧ tt ∈ NAT ∧ aa = bb * ss + tt
      VARIANT
         tt
      END ;
      qq := ss ; rr := tt
   END
```

Suffisant pour garantir l'absence d'erreur à l'exécution ?

Un exemple plus compliqué

Précondition : $tab^{-1}\{[FALSE]\} \neq \emptyset$

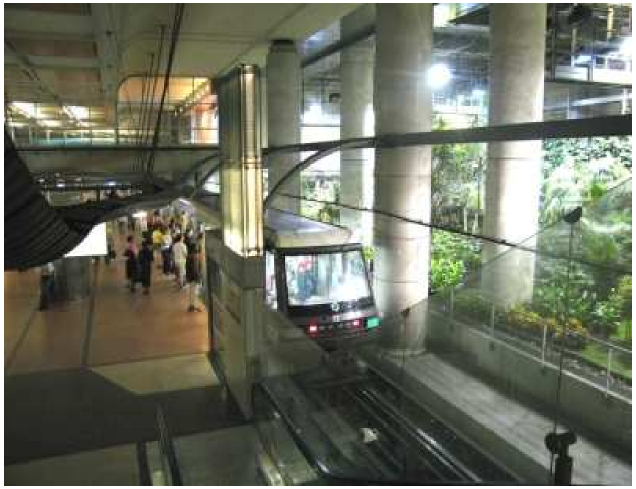
Postcondition : $tab0(place) = FALSE \wedge tab = tab0 \lhd\!\!- \{place \mapsto TRUE\}$

avec $tab : 1..tailleMax \rightarrow BOOL$

```
var ind in
     ind:=1;
     while tab(ind)=TRUE do
         ind:=ind+1
     end;
     tab(ind):=TRUE ;
     place:=ind
end
```

- Quel variant ? Quel invariant ? Quelles conditions sur $tailleMax$?
- la postcondition devient $place = min(tab^{-1}\{[FALSE]\})$. Quel invariant ?

Météor : ligne 14



Projet Météor

- Logiciel non sécuritaire : 1 million de lignes Ada
- Logiciel sécuritaire : 86 000 lignes Ada (1 000 composants)
  - 115 000 lignes B
  - 27 800 obligations de preuve
    - 81 % de preuve automatique
    - 92% après ajout de règles (550)
    - 2 254 à prouver interactivement

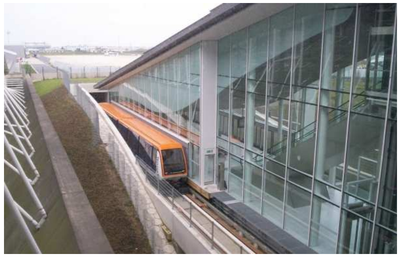
Météor (2)

- Des spécifications sûres (validation fonctionnelle)
  - modélisation dynamique
  - écarts résultats attendus / résultats obtenus
- Des logiciels exempts d'erreurs (méthode B)
  - guides méthodologiques
  - vérification des preuves et des règles
  - traçabilité des propriétés de sécurité
  - identification des limites de la preuve
- Une protection contre les erreurs à l'exécution
  - Processeur Sécuritaire Codé (PSC) : se garantir contre les perburbations electromagnétiques
  - redondance à l'exécution et vérification dynamique

Depuis Météor chez SiemensTS

- Automatisation de la preuve
  - base de règles propres
  - règles validées
- Raffinement automatique
  - schémas de raffinement de données
  - schémas de raffinement algorithmique
- Réutilisation
  - paramétrer les spécifications et les développements
  - méthodologie outillée de construction d'applications

Autre projet phare: Val de Roissy CDG

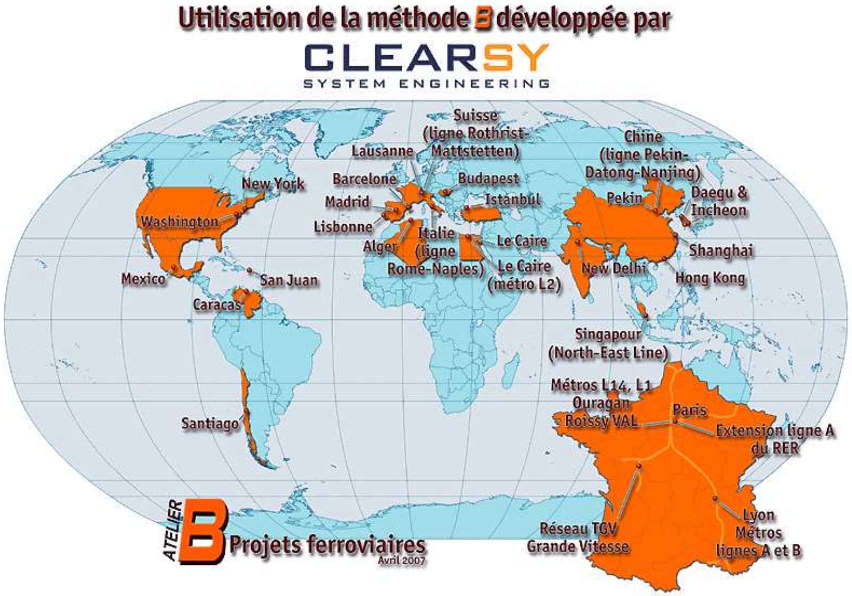


Calculateur	l. ADA ns	l. ADA s	lignes B	PO
PADS	30 632	186 440	256 653	62 056
UCA	11 662	50 085	65 722	12 811

Projet Ouragan

- Remise à niveau du réseau de la RATP
- Portes palières
- Automatisation des rames
- Début des travaux sur la ligne 1

Les projets ferroviaires B dans le monde



Autres approches

- peu d'autres approches globales basées sur le raffinement
- des approches preuve de programmes annotés (ESC Java, Spec#, Caduceus et Krakatoa) avec éventuellement un langage plus abstrait pour les assertions.
- des plate-formes de vérification de programmes faisant collaborer différentes analyses (vérification automatique mais approchée, preuve . . .) Exemple : la plate-forme Frama-C.

Autres outils d'analyse: bug checkers (pour la vérification ou la mise au point)

Des points de recherche

- automatisation de l'activité de preuve
  - procédure de décision, explication des preuves
- analyse de programmes avec allocation dynamique
  - analyses d'alias, classes de propriétés
- analyse compositionnelle
  - modules et classes, programmes concurrents
- analyse au niveau des exécutables
  - retrouver les informations (data et control flow)

Et maintenant . . . A VOUS !

- Division euclidienne
  - trouver l'invariant et le variant
  - garantir l'absence d'erreur à l'exécution
- Recherche
  - un exemple plus complexe d'invariant
- Ecluse
  - modéliser
  - développer par raffinement

~ potet/ECJP-PageB (home page Vérimag)

La preuve au premier ordre est indécidable ! Il faut essayer les différents prouveurs automatiques : pr le prouveur général, pp0 et pp1 des prouveurs combinatoires (i le niveau de profondeur d'utilisation des hypothèses).