Exemples de preuves d'algo : correction, terminaison.

927

Intro : On présente des méthodes permettant de montrer qu'un algorithme termine et est correct relativement à sa spécification.

Ex de spécification (tri)
- **entrée** tableau $t[1..n]$ d'entiers
- **sortie** tableau $t'[1..n]$ d'entiers triés dans l'ordre croissant **et** qui est une permutation de $t[1..n]$

(I) Terminaison

Un algorithme **termine** s'il s'exécute en temps fini pour toute entrée conforme à sa spécification.

1/ Algorithmes itératifs : variant de boucle

Déf : Un ensemble ordonné $(\mathcal{E}, \leq)$ est dit **bien fondé** s'il n'admet pas de suite strictement décroissantes infinies.

Ex : $(\mathbb{N}^k, \leq_{lex})$

Déf : Un **variant** d'une boucle est un élément $t \in \mathcal{E}$ d'un ensemble bien fondé, exprimé en fonction des variables de la boucle, et décroissant strictement à chaque itération.

Prop : Si une boucle admet un variant elle termine.

Ex : Pour une boucle for : t = indice final − indice courant.

Ex : EUCLIDE = while $(x \neq y)$ do
if $x \leq y$ then $y \leftarrow y - x$
else $x \leftarrow x - y$
$(x, y > 0)$ termine. $(t = \max(x,y))$

2/ Algorithmes récursifs

Thm : Soit f un programme récursif, it l'ensemble de ses arguments, $\varphi : it \to \mathcal{E}$ avec $(\mathcal{E}, \leq)$ bien fondé, $\mathcal{B} = \varphi^{-1}(\{\text{éléments minimaux de } \varphi(it)\})$.
Si $\forall b \in \mathcal{B}$, $f(b)$ termine et si, dans la définition de $f(x)$ n'apparaissent qu'un nombre fini de $f(y)$ et qui sont tels que $\varphi(y) < \varphi(x)$.
Alors $f(x)$ termine $\forall x \in it$.

Ex :
$ACK(0,p) = p+1$
$ACK(n,0) = ACK(n-1, 1)$
$ACK(n,p) = ACK(n-1, ACK(n, p-1))$
termine. $(\mathcal{E} = (\mathbb{N}^2, \leq_{lex}))$

3/ Non terminaison

Ex d'algo ne terminant pas :
$MORRIS(0,n) = 1$
$MORRIS(m,n) = MORRIS(m-1, MORRIS(m,n))$
ne termine pas (appel par valeur).

Ex d'algo dont on ne sait pas s'il termine :
$$SYRACUSE(n) = \begin{cases} 1 \text{ si } n = 1 \\ SYRACUSE(3n+1) \text{ si } n > 1 \text{ impair} \\ SYRACUSE(n/2) \text{ sinon} \end{cases}$$

Thm : le problème de la terminaison est indécidable.

4/ Problèmes liés aux algorithmes parallèles

Ex : $f : \mathbb{Z} \to \mathbb{Z}$ on cherche un zéro de f (supposé exister)

On considère deux programmes S_1 et S_2 qui énumèrent les entiers (l'un les positifs, l'autre les négatifs). On s'intéresse à la terminaison de $S_1 \| S_2$.

Problèmes qui peuvent arriver :

* S_1 et S_2 partagent une variable booléenne "trouvé" mais celle-ci peut être modifiée par S_2 alors que S_1 a trouvé un zéro → boucle infinie
* S_1 et S_2 s'exécutent à tour de rôle à l'aide d'une variable partagée "tour" mais si S_1 trouve un zéro, S_2 reste bloqué en attente.

(II) Correction

Déf : Un algorithme est dit partiellement correct si, pour toute entrée conforme à sa spécification, il produit un effet conforme à sa spécification.

Correction totale = correction partielle + terminaison.

Rq : Des algorithmes incorrects peuvent être intéressants :

* test de primalité de Miller-Rabin
* algorithmes d'approximation
* heuristiques

1/ Algorithmes itératifs : invariant de boucle

La correction des algorithmes itératifs est rendue compliquée par la présence de boucles (for, while), Pour prouver la correction on s'intéresse aux propriétés préservée par les boucles.

Déf : Un invariant d'une boucle est un prédicat portant sur les variable de la boucle qui vérifie :

* initialisation il est vrai avant la première itération
* conservation s'il est vrai avant une itération de la boucle alors il reste vrai après celle-ci.

Ex : tri par insertion : "à la i^e itération $t[1..i]$ est une permutation triée de $t_{init}[1..i]$".

Rq : Dans le cas des boucles imbriquées, on commence par travailler sur les plus profondes.

2/ Algorithmes récursifs

Les algorithmes récursifs traitent en général des structures inductives. On peut alors vérifier la correction sur les cas de bases puis la correction du traitement des opérateurs, pour prouver la correction.

Exs : taille d'une liste, évaluation d'une formule logique, ...

Thm Avec les mêmes hypothèses qu'au I.2.

Si P_ξ est un prédicat sur $\mathcal{X}$ vérifiant :

* $\forall b \in \mathcal{B}$, $P_\xi(b)$
* $\forall x, (\forall y, \varphi(y) < \varphi(x) \Rightarrow P_\xi(y)) \Rightarrow P_\xi(x)$

Alors P_2 est vérifié sur $\mathbb{N}$ entiers.

Ex : BINOME(n,0) = 1
BINOME(0,k) = 0 (k>0)
BINOME(n,k) = BINOME(n-1,k) + BINOME(n-1,k-1)
Alors $\forall n,k \in \mathbb{N}$, BINOME(n,k) = $\binom{n}{k}$ $(\mathcal{E} = (\mathbb{N}^2, \leq_{lex}))$.

Rq : Pour les structures définies par induction on prend souvent q = hauteur.

Rq : Il arrive que l'on doive prouver la correction et la terminaison en même temps.

Ex : * Fonction 91 de MacCarthy.
$$g(n) = \begin{cases} n-10 & \text{si } n > 100 \\ g(g(n+11)) & \text{sinon} \end{cases}$$
* Algorithme du lièvre et de la tortue

III Formalisation avec la logique de Hoare

1/ le language WHILE

Syntaxe $p ::=$ skip $\mid X := a \mid p_1 ; p_2 \mid$ if b then p_1 else $p_2 \mid$ while b do p

où • p programme • X variable entière
• a expression arithmétique • b express° booléenne

Ex : FACT =
```
res := 1;
while n ≥ 1 do
    res := res × n;
    n := n - 1
```

Sémantique

Un état est $\sigma : Var \to \mathbb{Z}$, Σ l'ensemble des états.

Evaluation d'une expression arithmétique (induction) :
- $ev_\sigma(n) = n$ pour $n \in \mathbb{Z}$
- $ev_\sigma(X) = \sigma(X)$ pour $X \in Var$
- $ev_\sigma(a_1 + a_2) = ev_\sigma(a_1) + ev_\sigma(a_2)$ ...

Idem pour les expressions booléennes.

Ex : $ev_\sigma(4 \times (X-3) \geq Y) = \begin{cases} \text{true si } 4(\sigma(X)-3) \geq \sigma(Y) \\ \text{false sinon} \end{cases}$

Enfin on définit l'exécution d'un programme (par induction).

* $\dfrac{}{\langle skip, \sigma \rangle \to \sigma}$ * $\dfrac{ev_\sigma(a) = n}{\langle X := a, \sigma \rangle \to \sigma[n/X]}$

* $\dfrac{\langle p_1, \sigma \rangle \to \sigma' \quad \langle p_2, \sigma' \rangle \to \sigma''}{\langle p_1 ; p_2, \sigma \rangle \to \sigma''}$

* $\dfrac{ev_\sigma(b) = true \quad \langle p_1, \sigma \rangle \to \sigma'}{\langle \text{if } b \text{ then } p_1 \text{ else } p_2, \sigma \rangle \to \sigma'}$ * $\dfrac{ev_\sigma(b) = false \quad \langle p_2, \sigma \rangle \to \sigma'}{''}$

* $\dfrac{ev_\sigma(b) = false}{\langle \text{while } b \text{ do } p, \sigma \rangle \to \sigma}$ * $\dfrac{ev_\sigma(b) = true \quad \langle p, \sigma \rangle \to \sigma' \quad \langle \text{while} \ldots, \sigma' \rangle \to \sigma''}{\langle \text{while} \ldots, \sigma \rangle \to \sigma''}$

App° : On peut définir l'équivalence de programmes :
$p_1 \sim p_2$ ssi $\forall \sigma, \sigma', \langle p_1, \sigma \rangle \to \sigma' \Leftrightarrow \langle p_2, \sigma \rangle \to \sigma'$

Exs : • skip ; skip $\sim$ skip
• si W = while b do p, alors W $\sim$ if b then p ; W else skip

Prop: La sémantique est déterministe, $\langle p, \cdot \rangle$ est donc une fonction partielle notée $[\![p]\!]$.

Rq: On peut définir $[\![p]\!]$ directement...

2/ Triplets de Hoare

Déf: Triplet de Hoare : $\{A\}p\{B\}$ (précondition, programme, postcondition)

Ex: $\{i+12 \leq X\}\ X := X+100\ \{i \leq X\}$

(variable du programme ; variable logique, c'est sur celles-ci que portent les quantificateurs.)

Déf: $I : \mathrm{Var_L} \to \mathbb{Z}$ interprétation des variables logiques.
On dit que $\{A\}p\{B\}$ est vérifié par I et σ ($\sigma \vDash^I \{A\}p\{B\}$) si : $(\sigma \vDash^I A \wedge [\![p]\!](\sigma)$ défini$) \Rightarrow ([\![p]\!](\sigma) \vDash^I B)$.
On dit que $\{A\}p\{B\}$ est valide ($\vDash \{A\}p\{B\}$) s'il est vérifié par tous I et σ.

Système de déduction

Règles

- SKIP $\dfrac{}{\vdash \{A\}\ skip\ \{A\}}$
- $\dfrac{\vdash \{A\}p_1\{B\} \quad \vdash \{B\}p_2\{C\}}{\vdash \{A\}p_1;p_2\{C\}}$
- ASSIG $\dfrac{}{\vdash \{B[a/X]\}\ X := a\ \{B\}}$
- IF $\dfrac{\vdash \{A \wedge b\}p_1\{B\} \quad \vdash \{A \wedge \neg b\}p_2\{B\}}{\vdash \{A\}\ \text{if } b \text{ then } p_1 \text{ else } p_2\ \{B\}}$
- WHILE $\dfrac{\vdash \{A \wedge b\}p\{A\}}{\vdash \{A\}\ \text{while } b \text{ do } p\ \{A \wedge \neg b\}}$
- CSQ $\dfrac{\vDash A \Rightarrow A' \quad \vdash \{A'\}p\{B'\} \quad \vDash B' \Rightarrow B}{\vdash \{A\}p\{B\}}$

Thm: Le système de déduction est consistant :
si $\vdash \{A\}p\{B\}$ alors $\vDash \{A\}p\{B\}$.

App°: $\vDash \{n = N \wedge n \geq 0\}\ FACT\ \{res = N!\}$

Déf: A est une plus faible précondition de (p, B) si
$\forall I, \forall \sigma,\ \sigma \vDash^I A$ ssi $([\![p]\!](\sigma)$ défini $\Rightarrow [\![p]\!](\sigma) \vDash B)$.

Thm: le système de déduction est relativement complet :
si $\vDash \{A\}p\{B\}$ alors $\vdash \{A\}p\{B\}$.

⚠ $\vDash \{true\}p\{false\}$ ssi p ne termine jamais
$\vDash \{true\}\ skip\ \{B\}$ ssi $\vDash B$

3/ Correction totale

On note $[A]p[B]$ si p termine pour les états vérifiant A et si $\{A\}p\{B\}$.
Pour prouver la correction totale il faut changer la règle WHILE :

$$\frac{\vdash [A \wedge b]\,p\,[A] \quad \vdash [A \wedge b \wedge (t = z)]\,p\,[t < z] \quad \vDash A \Rightarrow t \geq 0}{\vdash [A]\ \text{while } b \text{ do } p\ [A \wedge \neg b]}$$

où z est une variable logique fraîche et t une expression arithmétique (le variant).

App°: Corection totale de FACT.

4/Automatisation des preuves

Les preuves dans cette logique sont très lourdes
↳ on les automatise

* On ne donne pas des preuves entières mais on annote les programmes.
* Des outils en déduisent une preuve.

Ex: le logiciel WHY prend en entrée un programme annoté, en déduit les formules du 1er ordre à prouver, et fait appel à un démonstrateur automatique ou à un assistant de preuve pour prouver celles-ci.

Toute la difficulté est dans la recherche d'invariants de boucle. La génération automatique d'invariants de boucle est aujourd'hui un sujet de recherche actif...

Réfs
* I.1 pas de réf...
* I.2, I.3, II.2 → Luc Albert
* II.1 → Cormen
* I.4 → Apt-Olderog
* III → Apt-Olderog & Winskel

Complétude de la logique de Hoare

Ref : Winskel

Thm : Si $\vDash \{A\}\, p\, \{B\}$ alors $\vdash \{A\}\, p\, \{B\}$.

Déf : On dit que A est une plus faible précondition (pfp) de (p, B) si pour tous I et σ on a :

$$\sigma \vDash^I A \quad \text{ssi} \quad [\![p]\!](\sigma) \vDash^I B$$

Lm 0 : Si $\vDash \{A\}\, p\, \{B\}$ et si A_0 est une pfp de (p, B) alors $\vDash A \Rightarrow A_0$.

Preuve Soient I et σ tels que $\sigma \vDash^I A$.
Comme $\vDash \{A\}\, p\, \{B\}$, $[\![p]\!](\sigma) \vDash^I B$, d'où $\sigma \vDash^I A_0$.

Lm 1 : Pour tous p, B il existe $\varphi(p, B)$ une pfp de (p, B).

Lm 2 : Pour tous p, B, $\vdash \{\varphi(p, B)\}\, p\, \{B\}$.

Cl° Soit $\vDash \{A\}\, p\, \{B\}$.
Alors (lm 0) $\vDash A \Rightarrow \varphi(p, B)$ et (lm 2) $\vdash \{\varphi(p, B)\}\, p\, \{B\}$.
D'où $\vdash \{A\}\, p\, \{B\}$ par CSQ.

Preuve lm 1 On construit $\varphi(p, B)$ par induction sur p.

* $p = \text{skip}$ $\quad \varphi(p, B) = B$
* $p = X := a$ $\quad \varphi(p, B) = B[a/X]$
* $p = p_1 ; p_2$ $\quad \varphi(p, B) = \varphi(p_1, \varphi(p_2, B))$

* $p =$ if b then p_1 else p_2

$$\varphi(p, B) = (b \Rightarrow \varphi(p_1, B)) \wedge (\neg b \Rightarrow \varphi(p_2, B))$$

* $p =$ while b do p_1

C'est là que ça fait mal.

On voudrait encoder : $\forall I, \sigma,\ \sigma \models^I \varphi(p, B)$ ssi

$$\forall k \in \mathbb{N},\ \forall \sigma_0, \dots \sigma_k \in \Sigma,\ (\sigma_0 = \sigma \wedge \forall i \in \{1..k-1\},\ \sigma_i \models^I b \wedge [\![p]\!](\sigma_i) = \sigma_{i+1}) \Rightarrow (\sigma_k \models^I b \vee B)$$

Il faut encoder les états dans la logique du premier ordre...

↳ Gödelisation

⇒ admis.

Ensuite il faut vérifier par induction que $\varphi(p, B)$ est bien une pfp de (p, B).

Preuve lm 2 On montre $\vdash \{\varphi(p, B)\}\ p\ \{B\}$ par induction sur p.

* $p =$ skip $\quad \varphi(p, B) = B$ et $\overline{\vdash \{B\}\ p\ \{B\}}$ (SKIP)

* $p = X := a \quad \varphi(p, B) = B[a/X]$ et $\overline{\vdash \{B[a/X]\}\ p\ \{B\}}$ (ASSIG)

* $p = p_1; p_2 \quad \varphi(p, B) = \varphi(p_1, \varphi(p_2, B))$

Par ind : $\vdash \{\varphi(p_1, \varphi(p_2, B))\}\ p_1\ \{\varphi(p_2, B)\}$ et $\vdash \{\varphi(p_2, B)\}\ p_2\ \{B\}$.

D'où $\vdash \{\varphi(p, B)\}\ p\ \{B\}$ (SEQ).

* $p =$ if b then p_1 else $p_2 \quad \varphi(p, B) = (b \Rightarrow \varphi(p_1, B)) \wedge (\neg b \Rightarrow \varphi(p_2, B))$

Par ind : $\vdash \{\varphi(p_1, B)\}\ p_1\ \{B\}$ et $\vdash \{\varphi(p_2, B)\}\ p_2\ \{B\}$.

Mais $\models (\varphi(p, B) \wedge b) \Rightarrow \varphi(p_1, B)$ et $\models (\varphi(p, B) \wedge \neg b) \Rightarrow \varphi(p_2, B)$.

Donc (CSQ) $\vdash \{\varphi(p, B) \wedge b\}\ p_1\ \{B\}$ et $\vdash \{\varphi(p, B) \wedge \neg b\}\ p_2\ \{B\}$.

D'où (IF) $\vdash \{\varphi(p, B)\}\ p\ \{B\}$.

* $p = \text{while } b \text{ do } p_1$

Dans ce cas $\mathcal{P}(p, B)$ est immonde mais on sait quand même que c'est une pfp de (p, B).

Montrons ① $\vdash A \wedge b \Rightarrow \mathcal{P}(p_1, A)$

② $\vdash A \wedge \neg b \Rightarrow B$

Ccl : Par ind : $\vdash \{\mathcal{P}(p_1, A)\}\, p_1 \,\{A\}$

CSQ $\rightsquigarrow \vdash \{A \wedge b\}\, p_1 \,\{A\}$

WHILE $\rightsquigarrow \vdash \{A\}\, p \,\{A \wedge \neg b\}$

CSQ $\rightsquigarrow \underline{\vdash \{A\}\, p \,\{B\}}$.

① Soient I et σ tq $\sigma \models^I A \wedge b$.

Alors $\sigma \models^I A$ donc $[\![p]\!](\sigma) \models^I B$. Or $[\![p]\!](\sigma) = [\![p]\!]([\![p_1]\!](\sigma))$ car $ev_\sigma(b) = \text{true}$.

D'où $[\![p]\!]([\![p_1]\!](\sigma)) \models^I B$, d'où $[\![p_1]\!](\sigma) \models^I A$, d'où $\sigma \models^I \mathcal{P}(p_1, A)$.

② Soient I et σ tq $\sigma \models^I \wedge \neg b$.

Alors $\sigma \models^I A$ donc $[\![p]\!](\sigma) \models^I B$. Mais $[\![p]\!](\sigma) = \sigma$, d'où le résultat.

Rq pour la route : Si A_1 et A_2 sont deux pfp de (p, B) alors $\vdash A_1 \Leftrightarrow A_2$!

Rq pour la vie : Si $[\![p]\!](\sigma) = \bot$, on considère que $[\![p]\!](\sigma) \vdash B$ est vraie.

Algorithme d'unification

On cherche à unifier deux formules atomiques
$R\,t_1 \dots t_m$ et $R\,t'_1 \dots t'_m$, c'est à dire à trouver une substitution σ des variables présentes telle que :
$\forall i \in [\![1, m]\!] \quad \sigma(t_i) = \sigma(t'_i)$

On utilise pour cela l'algorithme suivant :

$E \leftarrow \bigcup_{i=1}^{m} \{(t_i, t'_i)\}$
$\sigma \leftarrow Id$
tant que $E \neq \emptyset$ faire :
  Soit (t, t') un élément de E :
  3 cas possibles :

Cas 1 • si $(t, t') = (f\,t_1 \dots t_m,\ g\,t'_1 \dots t'_m)$
  (avec f et g symboles de fonctions)
  - si $f = g$, • $E \leftarrow E - \{(f\,t_1 \dots t_m, f\,t'_1 \dots t'_m)\} + \bigcup_{i=1}^{m} \{(t_i, t'_i)\}$ • $\sigma \leftarrow \sigma$
  - si $f \neq g$, erreur "pas d'unification possible"

Cas 2 • si $(t, t') = (x, x)$, • $E \leftarrow E - \{(x, x)\}$ • $\sigma \leftarrow \sigma$

Cas 3 • si $(t, t') = (x, u)$ ou (u, x) $\quad u \neq x$
  - si x apparaît dans u, erreur "pas d'unif possible"
  - sinon • $E \leftarrow [u/x](E - \{(x, u)\})$
    • $\sigma \leftarrow [u/x] \circ \sigma$

x variable
u terme

fin tant que

Dans toute la suite, on notera E_n et σ_n les valeurs de E et σ après la n^{e} itération de la boucle.

- **Théorème 1 :** L'algorithme termine toujours, soit par un échec, soit par $E_n = \emptyset$.

Preuve : On note respectivement a_n, b_n et c_n le nombre de variables, de symboles de fonctions et de couples dans E_n

Posons $f(n) = (a_n, b_n, c_n)$.

Montrons que si il n'y a pas d'échec à l'étape $n+1$ $f(n+1) < f(n)$ dans l'ordre lexicographique :

On effectue une disjonction de cas :

	a_n	b_n	c_n
Cas 1	$=$	$\searrow$	
Cas 2	$\searrow$ ou $=$	$=$	$\searrow$
Cas 3	$\searrow$		

$f(n)$ est donc un variant de boucle, et le programme termine donc

- **Théorème 2 :** Si l'algorithme renvoie $E_n = \emptyset$, alors σ_n est l'unificateur principal de E, si il échoue, alors il n'y a pas d'unificateur.

Déf : σ est dit un unificateur principal de E si pour tout σ' unificateur de E, il existe une substitution β tq $\sigma' = \beta \circ \sigma$.

Preuve :

Soit H_n la propriété :

> Une substitution σ unifie E ssi $\exists \sigma'$ unifiant E_n tq $\sigma = \sigma' \circ \sigma_n$

Lemme 1 : Pour tout n tel que E_n, σ_n est défini, H_n est vraie

Admettons temporairement ce lemme et regardons la dernière étape de l'algorithme, indicée par n.

- Si $E_n = \emptyset$, l'identité unifie E_n, et, par H_n, σ_n unifie E et il s'agit bien de l'unificateur principal.
- Si l'étape échoue, E_{n-1} n'est pas une paire, et, par H_{n-1}, E non plus

Preuve du lemme 1

On montre la propriété par récurrence sur $n \in \mathbb{N}$

- $\sigma_0 = Id$ et $E_0 = E$ donc H_0 est triviale

- Soit maintenant $n \in \mathbb{N}$ tel que (E_n, σ_n) et (E_{n+1}, σ_{n+1}) soient définis :

Soit σ une substitution, montrons :

> $\exists \sigma'$, $\sigma = \sigma' \circ \sigma_n$ et σ' unifie E_n
>
> $\Leftrightarrow \exists \sigma''$, $\sigma = \sigma'' \circ \sigma_{n+1}$ et σ'' unifie E_{n+1}

Effectuons une disjonction de cas :

- Dans le cas 1 : $\mathcal{E}_n = (P(u_1, \dots u_m), P(v_1, \dots v_m)) \cup E'$
  $\mathcal{E}_{n+1} = \{(u_1, v_1), \dots (u_n, v_n)\} \cup E'$
  et $\sigma_n = \sigma_{n+1}$
  et l'équivalence est facile

- Dans le cas 2 : $\mathcal{E}_n = E' \cup \{(x, x)\}$ $\mathcal{E}_{n+1} = E'$ et
  et l'équivalence est encore facile $\sigma_n = \sigma_{n+1}$

- Dans le cas 3 : $\mathcal{E}_n = E' \cup \{(u_1, \dots)\}$
  $\mathcal{E}_{n+1} = [u/x_1](E')$
  $\sigma_{n+1} = [u/x_1] \circ \sigma_n$

  $\Leftrightarrow$ on pose $\sigma' = \sigma'' \circ [u/x_1]$

  $\Rightarrow$) Comme σ' unifie $\mathcal{E}_n$, $\sigma'(u) = \sigma'(x_1)$,

Lemme 2 : Soit x une variable, u un terme et σ une substitution
tq $\sigma(x) = \sigma(u)$ alors $\sigma = \sigma \circ [u/x]$.

En appliquant ce lemme, on a : $\sigma' = \sigma' \circ [u/x]$
et on a la propriété voulue en posant $\sigma'' = \sigma'$

L'équivalence ainsi montrée nous donne $H_n \Rightarrow H_{n+1}$,
Et on a donc prouvé le lemme 1.

Preuve du Lemme 2 : Posons $\sigma' = \sigma \circ [u/x]$

$\sigma'(x) = \sigma \circ (u/x)(x) = \sigma(u) = \sigma(x)$

car y variable $y \neq x$ $\sigma'(y) = \sigma \circ (u/x)(y) = \sigma(y)$ d'où $\sigma \circ [u/x] = \sigma$

DVPT

Preuve de correction du calcul de la factorielle par la logique de Hoare

Soit w le programme suivant :

$(\text{while } X > 0 \text{ do } (Y := X \times Y \ ; \ X := X - 1))$

On va prouver que w calcule la fonction factorielle, i.e :

$$\vdash \{X = n \wedge n \geq 0 \wedge Y = 1\} \ w \ \{Y = n!\}.$$

Pour cela, on souhaite utiliser la règle de la boucle "while" de la logique de Hoare :

$$\frac{\{I \wedge b\} \ c \ \{I\}}{\{I\} \text{ while } b \text{ do } c \ \{I \wedge \neg b\}}$$

Cependant, l'expression d'un I permettant d'appliquer cette règle n'est pas évidente à la lecture des préconditions et post conditions du programme. Il nous faut donc trouver un I valable, vérifiant de plus :

$$* \quad \begin{cases} (X = n \wedge n \geq 0 \wedge Y = 1) \Rightarrow I \\ (I \wedge \neg(X > 0)) \Rightarrow (Y = n!) \end{cases}$$

Sachant que l'on peut appliquer la règle de conséquence :

$$\frac{\vdash (A \Rightarrow A') \quad \{A'\} \ c \ \{B'\} \quad \vdash (B' \Rightarrow B)}{\{A\} \ c \ \{B\}}$$

Lemme : $I := (Y \times X! = n! \wedge X \geq 0)$ est un invariant de boucle pour w.

Preuve du lemme :

Il nous faut démontrer : $\vdash \{I \wedge X > 0\}\ Y := X \times Y\ ;\ X := X - 1\ \{I\}$

Comme la règle d'affectation $\boxed{\vdash \{Q[a/x]\}\ x := a\ \{Q\}}$ semble à une lecture inverse, commençons par étudier la deuxième affectation :

$$\vdash \{I\ [(X-1)/X]\}\ X := X - 1\ \{I\}$$

où $I\ [(X-1)/X] \equiv (Y \times (X-1)! = n!) \wedge (X - 1 \geq 0)$

notons A ce prédicat :

En appliquant désormais la première affectation, on obtient :

$$\vdash \{A\ [X \times Y / Y]\}\ Y := X \times Y\ \{A\}$$

où $A\ [X \times Y / Y] \equiv (X \times Y \times (X-1)! = n!) \wedge (X - 1 \geq 0)$

En notant B ce prédicat, on a par la règle de composition

$$\vdash \{B\}\ Y := X \times Y\ ;\ X := X - 1\ \{I\}$$

or $I \wedge X > 0 \equiv Y \times X! = n! \wedge X \geq 0 \wedge X > 0$

$$\Rightarrow Y \times X! = n! \wedge X \geq 1$$

$$\Rightarrow X \times Y \times (X-1)! = n! \wedge X - 1 \geq 0 \equiv B$$

Ainsi, une application de la règle de conséquence nous donne le résultat

Il reste à vérifier (*)

- $X = m \wedge m \geq 0 \wedge Y = 1 \Rightarrow Y \times X! = 1 \times m! \wedge X = m \geq 0$
  $\Rightarrow I$

- $I \wedge \neg(X > 0) \equiv Y \times X! = m! \wedge X \geq 0 \wedge \neg(X > 0)$
  $\Rightarrow Y \times X! = m! \wedge X = 0$
  $\Rightarrow Y = m!$

Rq : c'est ici qu'on utilise pour la première fois le $X \geq 0$ de l'invariant.

$$\dfrac{\dfrac{\dfrac{}{\{I \wedge X > 0\}\ Y := Y \times X\ \{A\}}\text{aff} \quad \dfrac{}{\{A\}\ X := X - 1\ \{I\}}\text{aff}}{\{I \wedge X > 0\}\ Y := Y \times X;\ X := X - 1\ \{I\}}\text{séq}}{I \wedge X > 0 \Rightarrow \exists\ \{I\}\ Y := Y \times X;\ X := X - 1\ \{I\}}\text{cons}$$

$$\dfrac{X = m \wedge Y = 1 \wedge m \geq 0 \Rightarrow I \quad \{I\}\ w\ \{I \wedge \neg(X > 0)\} \quad I \wedge \neg(X > 0) \Rightarrow Y = m!}{\{X = m \wedge Y = 1 \wedge m \geq 0\}\ w\ \{Y = m!\}}\text{cons}$$

w : while