

Méthodes formelles

Daniel Sanz
Université de Fribourg,
`daniel.sanz@unifr.ch`

April 17, 2020

Abstract

Ceci est un résumé non officiel du cours de méthodes formelles du professeur Ultes Nietzsche. Il s'agit principalement de ces slides traduites en français ainsi que quelques exos en guise d'exemple.

Introduction

Les formules logiques, les prédicats entre autres peuvent être utilisés afin d'exprimer de l'information sur l'état d'un programme. $x = 10$; indique que x doit impérativement avoir la valeur 10.

Pré & post-condition

Définition Une précondition P nous indique ce qui peut être considéré comme vrai avant même l'exécution d'une séquence d'instructions S .

Une postcondition Q nous indique ce qui sera vrai après l'exécution des instructions S .

Notation On écrit : $\{P\} S \{Q\}$ qui veut dire que si P est vrai alors, après l'exécution de S , Q est vrai. Il s'agit d'un triplet d'Hoare.

Exemple

$$\{x = 2;\} x = x \cdot 3; \{x = 6;\}$$

On utilise \hat{x} comme notation de la variable x pour indiquer la valeur de x après l'exécution du programme et x avant l'exécution.

Exemple

$$\begin{aligned} &\{true\} x = x + 1; \{\hat{x} > x\} \\ &\{true\} x = x + 1; \{\hat{x} = x + 1\} \end{aligned}$$

Les assertions

Il est possible d'écrire des prédicats entre deux lignes de code. On présume alors que ce prédicats est la postcondition de la ligne de code précédente et qu'il est la précondition de la ligne suivante.

Définition De tels prédicats sont dits *assertions* ou *annotations*. Pour savoir si des triplets sont corrects il faut tout d'abord transformer le programme S en une formule ϕ_S . Ainsi il est possible de prouver l'exactitude d'un triplet:

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

en verifiant la formule:

$$P \wedge \phi_S \rightarrow Q$$

ou de façon analogue:

$$\phi_S \rightarrow (P \rightarrow Q)$$

Exemple

$$\begin{aligned} &\{true\} x = 10; \{x > 0\} \\ &\phi_S \equiv x = 10 \end{aligned}$$

donc, $(true \wedge (x = 10)) \rightarrow (x > 10)$.

Exemple

$$\begin{aligned} \{x \neq 0\} \quad x = 1/x; \\ x = 1/x; \quad \{\hat{x} = x\} \end{aligned}$$

Soit $x'' = 1/x$ et $\hat{x} = 1/x''$ alors on vérifie:

$$(x \neq 0) \wedge (x'' = 1/x) \wedge (\hat{x} = 1/x'') \rightarrow \hat{x} = x$$

Ce qui est vrai par du calcul élémentaire. Bien sûr, ici on ne tient pas compte de la précision limitée des *floats*.

Les clauses If

Soit la clause générale *If* suivante. Avec la précondition P et la postcondition Q.

$$\begin{aligned} \{P\} \text{ if}(\textit{condition}) \{ \text{progI} \} \\ \text{ else } \{ \text{progII} \} \\ \{Q\} \end{aligned}$$

Afin de prouver cette clause il faut procéder aux transformation suivantes (et les prouver indépendamment):

$$\begin{aligned} \{P \wedge \textit{condition}\} \text{ progI} \quad \{Q\} \quad \underline{\text{et}} \\ \{P \wedge \neg \textit{condition}\} \text{ progII} \quad \{Q\} \end{aligned}$$

Les deux triplets doivent être vrais pour vérifier la clause.

Exemple Soit la clause *If* suivant avec sa précondition P et postcondition Q respective.

$$\begin{aligned} \overbrace{\{true\}}^P \text{ if}(x < 0) \quad \{x = -x\} \\ \text{ else } \{x = x\} \\ \underbrace{\{\hat{x} \geq 0\}}_Q \end{aligned}$$

Ainsi on fait la transformation :

$$\begin{aligned} & \{true \wedge (x < 0)\} x = -x \{ \hat{x} \geq 0 \} \quad \text{et} \\ & \{true \wedge \neg(x < 0)\} x = x \{ \hat{x} \geq 0 \} \end{aligned}$$

Formellement, il faudrait encore faire la transformation du programme en une formule logique ϕ_S afin de prouver les deux triplets. Mais il est évident que c'est juste.

Les boucles

On s'intéresse ici à comment prouver les boucles. En particulier la boucle `while`. Mais tout d'abord nous avons besoin de quelques définitions et d'outils supplémentaires.

Définitions

- Quand une boucle se termine et que le résultat espéré est atteint on dit qu'il s'agit d'une *exactitude partielle*.
- Quand il est garanti qu'une boucle se atteint une fin on dit : *termination*
- Quand les deux conditions précédentes sont vérifiées on parle d'*exactitude total*

Invariant de boucle

Il s'agit d'une formule logique qui est vraie dans les cas suivants:

- avant la boucle
- avant chaque execution du corps de la boucle
- après chaque execution du corps de la boucle
- après la boucle

et elle doit rendre la postcondition vraie. Donc grosso merdo c'est vrai tout le temps, d'où l'invariance.

Variant de boucle

Il s'agit d'une expression évaluée dans les entiers \mathbb{N}^+ qui

- est décrémentée de 1 à chaque itération
- ne peut pas aller en dessous de 0

Donc en d'autres termes $\text{int } i = \text{CST}$; et dans la boucle $i = i - 1$;

Boucle While

Soit la boucle suivante:

$$\begin{array}{l} \{P\} \text{ initialisation;} \\ \text{while}(\text{condition}) \{ \text{loop body} \}; \{Q\} \end{array}$$

On vérifie l'exactitude partielle et la terminaison séparément.

L'exactitude partielle à l'aide de l'invariant de boucle nous donne les formules suivantes:

$$\begin{array}{ll} \{P\} \text{ initialisation;} & \{Inv\} \\ \{Inv \wedge \text{condition}\} \text{ loop body;} & \{Inv\} \\ \{Inv \wedge \neg \text{condition}\} \text{ skip;} & \{Q\} \end{array}$$

La terminaison nous donne encore la formule suivante à vérifier:

$$\{ \text{intvar} \wedge \text{var} > 0 \} \text{ loop body}; \{ \text{var} > \text{var}' \geq 0 \}$$

Exemple Soit ce programme qui calcul de façon itérative la somme des entiers de 1 à n .

$$\begin{array}{l} \overbrace{\{n > 0 \wedge x = 1\}}^P \text{ sum} = 1; // \text{initialisation} \\ \text{while}(x < n) \{ \\ \quad x = x + 1; \\ \quad \text{sum} = \text{sum} + x; \\ \quad \}; \quad \underbrace{\{\text{sum} = n(n+1)/2\}}_Q \end{array}$$

Exactitude partielle

On pose l'invariant $\text{sum} = x(x+1)/2$.

On commence par verifier l'initialisation.

$$\{n > 0 \wedge x = 1\} \text{sum} = 1 \{ \text{sum} = x(x+1)/2 \}$$

$$(n > 0) \wedge (x = 1) \wedge (\text{sum} = 1) \rightarrow (\text{sum} = x(x+1)/2)$$

Ce qui est vrai car $1 = 1(1+1)/2$. Puis nous vérifions le corps de la boucle avec le triplet suivant:

$$\{(\text{sum} = x(x+1)/2) \wedge (x < n)\} x = x + 1;$$

$$\text{sum} = \text{sum} + x; \{ \text{sum} = x(x+1)/2 \}$$

Donc,

$$(\text{sum} = x(x+1)/2)$$

$$\wedge (x < n)$$

$$\wedge (x' = x + 1)$$

$$\wedge (\text{sum}' = \text{sum} + x') \rightarrow (\text{sum}' = x'(x'+1)/2)$$

$$\begin{aligned} \text{sum}' &= \frac{x(x+1)}{2} + x + 1 = \frac{x(x+1)}{2} + \frac{2x+2}{2} \\ &= \frac{x^2 + x + 2x + 2}{2} = \frac{x^2 + 3x + 2}{2} \\ &= \frac{(x+2)(x+1)}{2} = \frac{x'(x'+1)}{2} \end{aligned}$$

Puis la dernière formule à la fin de la boucle:

$$\text{sum} = \frac{x(x+1)}{2} \wedge x \geq n \rightarrow \text{sum} = \frac{n(n+1)}{2}$$

Qui est vraie si $x = n$ vu que les deux côtés de la formule sont les mêmes.

Terminaison

Avec $n - x$ comme variant on obtient la formule:

$$\begin{aligned}
&int\ var \wedge \\
&var > 0 \wedge \\
&x = 1 \wedge \\
&n > 0 \wedge \\
&var = n - x \wedge \\
&x' = x + 1 \wedge \\
&sum' = sum + x' \wedge \\
&var' = n - x' \rightarrow \\
&var > var' \geq 0
\end{aligned}$$

et donc

$$var' = n - x' = n - x + 1$$

$$n - x > n - x - 1 \geq 0$$

prouve la terminaison.

Notions sur les prédicats

Faiblesses et forces des prédicats

Définition : P est plus faible que Q si et seulement si $Q \rightarrow P$

Ainsi *true* est le plus faible des prédicats car il est impliqué par tout. Analogiquement, *false* est le plus fort des prédicats car il implique tout.

exemple

$$\overbrace{-10 \leq x \leq 10}^Q \rightarrow \overbrace{-100 \leq x \leq 100}^P$$

En effet, P est plus faible que Q car P décrit plus "d'état" que Q . On peut comprendre ça comme quoi Q est une implication plus stricte.

L'affaiblissement d'une précondition

Si P est plus faible que P' c-à-d que $P' \rightarrow P$ alors prouver le triplet

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

garantit la validité de $\{P'\} S \{Q\}$. En effet,

$$\begin{aligned} (P \wedge \phi_S) &\rightarrow Q \\ (P' \wedge \phi_S) &\rightarrow (P \wedge \phi_S) \rightarrow Q \end{aligned}$$

Renforcement d'une postcondition

Si Q est plus fort que Q' c-à-d que $Q \rightarrow Q'$ alors,

$$\begin{aligned} \{P\} S \{Q\} &\Rightarrow (P \wedge \phi_S) \rightarrow (Q \rightarrow Q') \\ &\quad (P \wedge \phi_S) \rightarrow Q' \\ &\quad \{P\} S \{Q'\} \end{aligned}$$

Logique propositionnelle

Syntaxe de la logique propositionnelle

Une formule propositionnelle logique peut se construire comme suit:

- Les valeurs de vérité \perp pour false et \top pour true sont aussi bien des variables propositionnelles que des formules.
- Si F et G sont des formules propo. logiques, alors
 - (F)
 - $\neg F$
 - $F \wedge G$
 - $F \vee G$
 - $F \rightarrow G$
 - $F \leftrightarrow G$

le sont aussi.

Définition: Si P est une variable, alors P et $\neg P$ sont dit littéraux.

Intérprétations des variables propositionnelles

Définition: Une interprétation I est une assignation de valeurs de vérité à des variables propositionnelles.

exemple

$$I : \{P \mapsto true, Q \mapsto false, \dots\}$$

Notation: La valeur de vérité de la variable p sous l'interprétation I s'écrit $I[p]$ avec l'exemple d'avant ça donne:

$$I[P] = true, I[Q] = false$$