Sémantique de Niklaus

Frédéric Boulanger

CentraleSupélec

Rappel : Niklaus

Niklaus : langage utilisé dans le cours de traitement des langages

- instructions de lecture et d'écriture sur une console
- affectation, conditionnelle, boucle *while*
- instructions exécutées en séquence

Exemple :

```
program PGCD;
var a, b;
{
    read a;
    read b;
    while(a <> b) {
        if(a > b) {
            a := a - b;
        } else {
            b := b - a;
        }
    }
    write a;
}
```

CentraleSupélec

Définition d'une sémantique

Ingrédients :

- syntaxe concrète (vue dans les cours précédents)
- syntaxe abstraite
- domaine sémantique
- correspondance entre la syntaxe abstraite et le domaine sémantique.

Domaine sémantique

Logique d'ordre supérieure

- possibilité de quantifier les fonctions et les prédicats
- cadre mathématique formel
- outillé (assistant de preuve Isabelle)

Pour des raisons pratiques, on définira la syntaxe abstraite et la correspondance sémantique dans le même cadre par des types de données, fonctions et prédicats.

CentraleSupélec

Isabelle

Assistant de preuve fondé sur un noyau (*Pure*) :

- $\Rightarrow$ constructeur de type fonction :
  $\mathrm{nat} \Rightarrow \mathrm{nat}$ = fonction de $\mathbb{N}$ vers $\mathbb{N}$.
- $\bigwedge$ quantificateur universel : $\bigwedge x.\ P(x)$
- $\Longrightarrow$ implication : $P \Longrightarrow Q$ ou $P \Longrightarrow Q \Longrightarrow R$

Curryfication

$f : (x, y) \mapsto xy$ est une fonction de deux arguments.

On peut l'écrire comme : $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$
Mais également sous forme curryfiée : $f : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})$
On interprète alors $f(x, y)$ comme : $f(x)(y)$
$f(x)$ rendant une fonction que l'on applique à y.

En Isabelle, on aura : $f :: nat \Rightarrow nat \Rightarrow nat$

Exemple de curryfication

Isabelle/Pure

Règles de déduction

$$\frac{\begin{array}{c}[x :: \alpha] \\ \vdots \\ b\ x :: \beta\end{array}}{\lambda x.\ b\ x :: \alpha \Rightarrow \beta}\ (\Rightarrow I)$$

$$\frac{b :: \alpha \Rightarrow \beta \quad a :: \alpha}{b\ a :: \beta}\ (\Rightarrow E)$$

$$\frac{\begin{array}{c}[x] \\ \vdots \\ B(x)\end{array}}{\bigwedge x.\ B(x)}\ (\bigwedge I)$$

$$\frac{\bigwedge x.\ B(x)}{B(a)}\ (\bigwedge E)$$

$$\frac{\begin{array}{c}[A] \\ \vdots \\ B\end{array}}{A \Longrightarrow B}\ (\Longrightarrow I)$$

$$\frac{A \Longrightarrow B \quad A}{B}\ (\Longrightarrow E)$$

Égalité

$\equiv$

$t \equiv u$ est définie selon :

- α-conversion :
  $\lambda x.\ 2 \times x \equiv \lambda y.\ 2 \times y$
- β-reduction :
  $(\lambda x.\ 2 \times x)\ 4 \equiv 2 \times 4$
- η-conversion :
  $\lambda x.\ f\ x \equiv f$
- $\mathit{refl} : t \equiv t$, $\mathit{subst} : [\![s \equiv t; P(s)]\!] \Longrightarrow P(t)$,
  $\mathit{ext} : (\bigwedge x.\ f\ x \equiv g\ x) \Longrightarrow f \equiv g$,
  $\mathit{iff} : [\![P \Longrightarrow Q; Q \Longrightarrow P]\!] \Longrightarrow P \equiv Q$

Pure est une métalogique

C'est un cadre pour raisonner sur ce que l'on peut déduire de faits.
Il est inspiré de la Déduction Naturelle, qui définit les opérations par des règles d'introduction et d'élimination, contrairement aux systèmes à la Hibert fondés sur des axiomes.

Il est possible de construire différentes logiques sur Pure :

- Logique du premier ordre (FOL : First Order Logic)
- Théorie des ensembles de Zermelo et Fraenkel (ZF)
- ZF avec axiome du choix (ZFC)
- Logique d'ordre supérieur (HOL : Higher Order Logic)

Exemple : Logique du premier ordre

CentraleSupélec

Isabelle HOL

Logique d'ordre supérieur fondée sur *Pure*.

Calcul des propositions :

- type *bool*, avec constantes *True* et *False*
- opérateurs logiques $\wedge$, $\vee$, $\neg$, $\longrightarrow$, $\longleftrightarrow$
- égalité $=$, et tiers exclu : $a = \mathrm{True} \vee a = \mathrm{False}$

Logique du premier ordre :

- quantificateurs $\forall$ et $\exists$

Logique d'ordre supérieur : les quantificateurs s'appliquent aux fonctions et aux prédicats, les fonctions et prédicats peuvent prendre pour argument des fonctions et des prédicats.

Définition de types : entiers, listes, ensembles etc.

Exemple Faire le tutoriel et les exercices

CentraleSupélec

Syntaxe abstraite – Concepts de Niklaus

Expressions

- entier, variable
- opérations arithmétiques : $+$, $-$, $\times$, $\div$

Syntaxe abstraite en Isabelle

Expressions booléennes

- booléen : `true`, `false`
- comparaison d'expressions : $=$, $\neq$, $<$, $\leqslant$, $>$, $\geqslant$

Syntaxe abstraite en Isabelle

Instructions

- déclaration de variable : `var a;`
- affectation : `a := a - b;`
- mise en séquence : `read a; read b;`
- alternative et boucle while : `if (c) {} else {}` et `while (c) {}`

Syntaxe abstraite en Isabelle

Domaine sémantique et correspondance

Valeurs et opérations

Entiers de Niklaus $\rightarrow$ type `int` d'Isabelle/HOL
Noms de variable $\rightarrow$ type `string` d'Isabelle/HOL
Opérations arithmétiques $\rightarrow$ arithmétique d'Isabelle/HOL

État de la machine d'exécution

Pour simplifier, on ignore les instructions `read` et `write`
État de la machine = valeur des variables (string $\Rightarrow$ int)

L'instruction en cours d'exécution dans un programme n'est pas modélisée explicitement. C'est l'opérateur de séquencement qui fait passer d'une instruction à la suivante.

L'instruction `Nop` correspond à un programme vide.

Exercices

Simplification d'expressions

Écrire une fonction qui simplifie une expression :

- les opérations sur les constantes sont remplacées par leur résultats
- l'addition à zéro et la soustraction de zéro sont supprimées
- la multiplication par 0 et 0 divisé par x sont remplacés par 0
- la multiplication et la division par 1 sont supprimées

Prouvez que la simplification ne change pas la valeur d'une expression

Optimalité de la simplification

Pour les plus aventureux : spécifiez ce qu'est une expression non simplifiable.
Prouvez que la fonction de simplification est optimale

Il n'est pas nécessaire que les preuves aboutissent.
Exprimer la propriété sous forme d'un théorème est déjà un travail intéressant.