Minigloca

Vladislas de Haldat

31 janvier 2023

Table des matières

1 Un langage impératif simple

Pour commencer, définissons une syntaxe abstraite minimale que nous utiliserons tout le long de cette étude. Cette syntaxe se composera de trois blocs fondamentaux que sont les expressions arithmétiques, les expressions booléennes ainsi que les déclarations. Nous n'incluons pas pour le moment la déclaration de routines au sein de cette syntaxe.

1.1 Expressions arithmétiques

Les expressions arithmétiques sont définies sur l'ensemble des entiers relatifs. On se donne les opérateurs de l'addition, de la soustraction ainsi que de la multiplication. À ces opérateurs l'on pourra appliquer des entiers ainsi que des identifiants de variables.

$$
\begin{aligned}
Int &\to n && n \in \mathbb{Z} \\
Id &\to x \mid y \mid z \mid \cdots \\
Exp_a &\to Int \mid Id \mid op_A(a_1, a_2) && op_A \in \{+, -, \times\}
\end{aligned}
$$

L'identifiant d'une variable est, de manière générale, une chaîne de caractères.

1.2 Expressions booléennes

Les expressions booléennes nous permettent d'introduire la comparaison entre deux expressions arithmétiques, ainsi que les opérateurs booléens sur les expressions booléennes.

$$
\begin{aligned}
Exp_b \to\ & \textbf{true} \mid \textbf{false} \\
& \mid op_R(a_1, a_2) && op_R \in \{<, =\} \\
& \mid op_B(b_1, b_2) && op_B \in \{\wedge, \vee\} \\
& \mid \neg b
\end{aligned}
$$

1.3 Déclarations

Les déclarations sont définies de la manière suivante :

$$
\begin{aligned}
Stm \rightarrow\ & Id := Exp_a \\
| \ & s_1 \,;\, s_2 \\
| \ & \\
| \ & \text{if } b \text{ then } s_1 \text{ else } s_2 \\
| \ & \text{while } b \text{ do } s_1
\end{aligned}
$$

1.4 Exemple

Voici un premier exemple sur ce langage

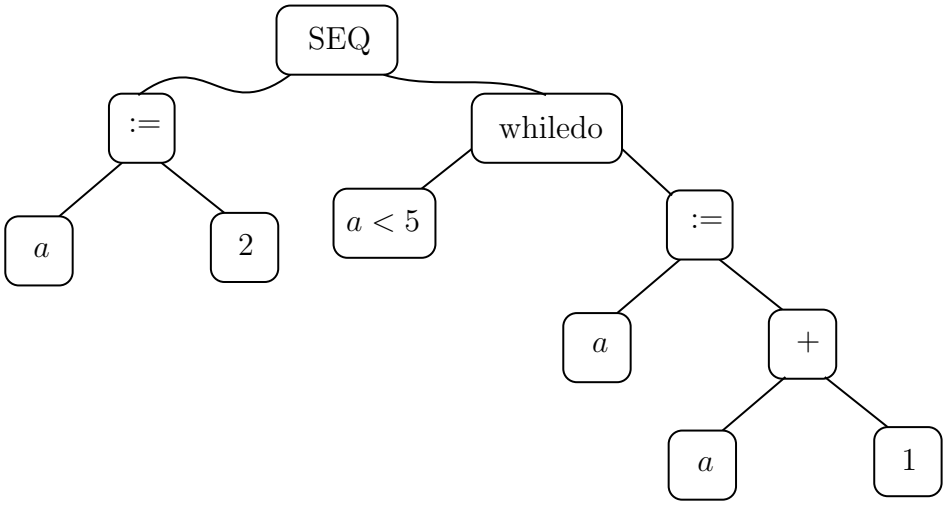
```
a := 1;
b := 20;

if a = 3 then
        c := 4
else
        c := 6
endif;

while b < 100 do
        b := b + 1
done
```

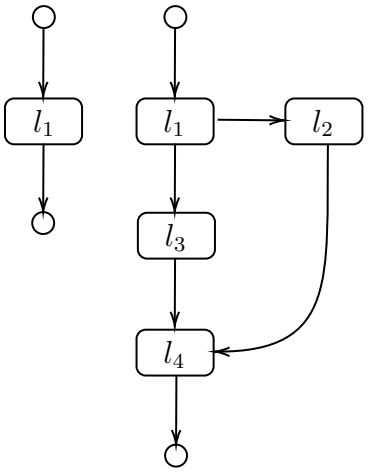
1.5 Arbre de syntaxe abstraite

Ce modèle de représentation permet de construire un arbre de syntaxe abstraite (AST) du programme en question, cette structure étant un moyen pratique de le représenter. Cela nous servira particulièrement lors des différentes analyses qui seront effectuées, et notamment l'analyse par flot de contrôle. Cette dernière ne requièrant pas la connaissance de l'ordre d'exécution du programme, les AST sont tout-à-fait adaptés !



1.6 Graphe de flot de contrôle

Un graphe de flot de contrôle est un graphe orienté dont les noeuds représentent une déclaration et les arcs un flot de contrôle. Chaque noeud est une unique opération de notre programme. Dans notre cas, le graphe de flot de contrôle aura un seul point d'entrée et un seul point de sortie. Ils sont considérés comme des noeuds "sans opération". De manière à construire ce graphe, on peut assigner à chaques déclarations procurant une unique exécution, un label. Dans notre cas, ces déclarations sont l'assignation, la condition ainsi que la boucle. En voici quelques exemples :



1.7 Implémentation

Détails sur l'implémentation.

2 Interpréteur et sémantique

Dans ce chapitre, on s'atèle à décrire l'interpréteur ainsi que la sémantique sur notre petit langage impératif. Dans ce cadre là, on définit l'état du programme par une bijection entre l'identifiant des variables et leur valeur, ici un entier :

$$\sigma : \mathbb{V} \longrightarrow \mathbb{Z}$$

On introduire aussi la bijection μ_B,

$$\mu_B : \mathcal{B} \longrightarrow \mathbb{B}$$

où $\mathcal{B} = \{\textbf{true}, \textbf{false}\}$ les valeurs booléennes de notre langage et $\mathbb{B} = \{\top, \bot\}$ l'ensemble des valeurs booléennes natives à OCaml. Pour la suite, on considèrera l'ensemble des états $\mathcal{S} = \mathbb{Z}^{\mathbb{V}}$. Les états décriront l'évolution de l'exécution du programme.

2.1 Interpréteur

Sur les expressions arithmétiques, définies dans le chapitre 1, on se donne la fonction suivante :

$$\begin{aligned}
[\![Exp_a]\!]^A &: \mathcal{S} \longrightarrow \mathbb{Z} \\
[\![Id]\!]^A &: \sigma \longmapsto \sigma(Id) \\
[\![a_1 + a_2]\!]^A &: \sigma \longmapsto [\![a_1]\!]^A(\sigma) \mathbin{\hat{+}} [\![a_2]\!]^A(\sigma)
\end{aligned}$$

Cette dernière est aussi définie respectivement aux autres opérateurs arithmétiques, introduit dans le chapitre précédent. De la même manière, on définit la fonction qui suit sur les expressions booléennes :

$$\begin{aligned}
[\![Exp_b]\!]^B &: \mathcal{S} \longrightarrow \mathbb{B} \\
[\![b]\!]^B &: \sigma \longmapsto \mu(b) \\
[\![a_1 = a_2]\!]^B &: \sigma \longmapsto [\![a_1]\!]^A(\sigma) \mathbin{\hat{=}} [\![a_2]\!]^A(\sigma) \\
[\![b_1 \wedge b_2]\!]^B &: \sigma \longmapsto [\![b_1]\!]^B(\sigma) \mathbin{\hat{\wedge}} [\![b_2]\!]^B(\sigma) \\
[\![\neg b]\!]^B &: \sigma \longmapsto \hat{\neg}[\![b]\!]^B(\sigma)
\end{aligned}$$

Les opérateurs de la forme $\hat{op}$ représentent les opérateurs natifs à OCaml.

2.2 Sémantique

Maintenant que nous avons correctement défini l'interpréteur, il est possible de construire la sémantique du langage. La logique de Hoare est un modèle formel qui nous permet la correction rigoureuse de nos programmes. Soient P, Q des prédicats et C une déclaration. Alors le triplet de Hoare est défini comme tel,

$$\{P\}C\{Q\}$$

Il est ensuite possible de développer un ensemble de règles logiques. Pour des raisons de commodités, on adaptera légèrement la syntaxe de la déclaration des règles, en prenant en compte nos état et déclaration, cela prendra donc la forme suivante,

$$Stm, \mathcal{S} \longrightarrow Stm', \mathcal{S}'$$

où Stm est la première déclaration, Stm celle qui suit après son exécution, $\mathcal{S}$ l'état initial et $\mathcal{S}'$ l'état successeur. Commençons par la déclaration vide.

$$\frac{}{Skip, \sigma_A \longrightarrow \emptyset, \sigma_A}$$

Poursuivons avec la déclaration de l'assignation,

$$\frac{}{Id := a, \sigma_A \longrightarrow \emptyset, \sigma'_A : Id \longmapsto [\![a]\!]^A(\sigma_A)}$$

Ici, le nouvel état σ'_A lie l'identifiant de la variable assignée, à la valeur de l'expression arithmétique $[\![a]\!]^A$. Poursuivons avec la règle de la séquence entre deux déclarations, d'une part si la première termine,

$$\frac{Stm_1, \sigma \longrightarrow \emptyset, \sigma'}{Stm_1; Stm_2, \sigma \longrightarrow Stm_2, \sigma'}$$

D'autre part, si la première ne termine pas,

$$\frac{Stm_1, \sigma \longrightarrow Stm'_1, \sigma'}{Stm_1; Stm_2, \sigma \longrightarrow Stm'_1; Stm_2, \sigma'}$$

La condition peut se formaliser de la sorte dans le cas où la garde est vérifiée,

$$\frac{[\![b]\!]^B(\sigma_B)}{\text{if } b \text{ then } Stm_1 \text{ else } Stm_2, \sigma_B \longrightarrow Stm_1, \sigma_B}$$

Dans le cas où elle ne l'est pas,

$$\frac{\neg[\![b]\!]^B(\sigma_B)}{\text{if } b \text{ then } Stm_1 \text{ else } Stm_2, \sigma_B \longrightarrow Stm_2, \sigma_B}$$

La dernière de nos déclarations est la boucle while, celle-ci peut en fait être décrite grâce à la déclaration de la condition de la manière suivante,

$$\text{while } b \text{ do } Stm \equiv_d \text{if } b \text{ then } Stm \text{ while } b \text{ do } Stm \text{ else } \mathit{Skip}$$

Ainsi, on peut déclarer la règle qui suit pour la déclaration while,

$$\frac{[\![b]\!]^B(\sigma_B)}{\text{while } b \text{ do } Stm, \sigma_B \longrightarrow Stm\,;\ \text{while } b \text{ do } Stm, \sigma_B}$$

3 Prérequis à l'analyse

De manière à pouvoir travailler avec les analyses de flot de données et de flot de contrôle, il nous est au préalable nécessaire d'approfondir le principe d'étiquettage déjà abordé à la fin du premier chapitre.

3.1 Étiquettage

Avant tout, introduisons la fonction suivante,

$$\lambda : Stm \longrightarrow (\mathbb{N})^{\mathbb{N}}$$

qui prend une déclaration s et retourne une suite d'étiquettes rencontrées à partir de s.

Définition – Soient $s \in Stm$ et $(l_n)_{n\in\mathbb{N}} = \lambda(s)$, s est dite bien formée si et seulement si $\forall i \in \mathbb{N}, \forall j \in \mathbb{N}$ tel que $i \neq j$ alors $l_i \neq l_j$.

Étant donné un programme P défini par l'ensemble de ses déclarations atomiques, on a alors que $\pi \in \mathcal{P}(P)$, un sous bloc de ce programme, n'admet qu'une seule entrée et une ou plusieurs sorties. On peut maintenant définir la fonction ι, qui retournera la première étiquette rencontrée dans une déclaration,

$$
\begin{aligned}
\iota : Stm &\longrightarrow \mathbb{N} \\
(Id := a)^l &\longmapsto l \\
s_1; s_2 &\longmapsto \iota(s_1) \\
Skip^l &\longmapsto l \\
(\text{if } b \text{ then } s_1 \text{ else } s_2)^l &\longmapsto l \\
(\text{while } b \text{ do } s)^l &\longmapsto l
\end{aligned}
$$

Comme expliqué précédemment, il est aussi nécessaire de déclarer une fonction ϕ qui retournera l'ensemble des étiquettes finales à la fin d'un bloc π.

$$
\begin{aligned}
\phi : Stm &\longrightarrow \mathcal{P}(\mathbb{N}) \\
(Id := a)^l &\longmapsto \{l\} \\
s_1; s_2 &\longmapsto \phi(s_2) \\
Skip^l &\longmapsto \{l\} \\
(\text{if } b \text{ then } s_1 \text{ else } s_2)^l &\longmapsto \phi(s_1) \cup \phi(s_2) \\
(\text{while } b \text{ do } s)^l &\longmapsto \{l\}
\end{aligned}
$$

3.2 Blocs

De manière à faciliter l'analyse d'un programme, il est utile de partitionner et de factoriser notre représentation du code. Les blocs ont été évoqués dans la partie précédente comme des sous-parties de l'ensemble des déclarations atomiques d'un programme. On peut désormais les définir de la sorte,

$$
\begin{aligned}
Block \rightarrow\ & Id := Exp_a \\
\mid\ & Exp_b \\
\mid\ & \text{Skip}
\end{aligned}
$$

Maintenant les blocs correctement définis, il nous faut pouvoir les lier à une étiquette. Pour ce faire, on se donne β définie par,

$$
\begin{aligned}
\beta : Stm &\longrightarrow \mathcal{P}(\mathbb{N} \times Block) \\
(Id := Exp_a)^l &\longmapsto \{(l, Id := Exp_a)\} \\
s_1; s_2 &\longmapsto \beta(s_1) \cup \beta(s_2) \\
Skip^l &\longmapsto \{(l, Skip)\} \\
(\text{if } b \text{ then } s_1 \text{ else } s_2)^l &\longmapsto \{(l, b)\} \cup \beta(s_1) \cup \beta(s_2) \\
(\text{while } b \text{ do } s)^l &\longmapsto \{(l, b)\} \cup \beta(s)
\end{aligned}
$$

À partir de là, il est possible de formaliser les graphes orientés de flot. Les blocs en représentent les noeuds, et le passage vers le bloc suivant est représenté par un arc.

3.3 Flots

Nous avons désormais toutes les structures nécéssaires à la construction de notre graphe de flot. Pour le moment, il s'agira de le construire naïvement, l'on reviendra plus tard sur les optimisations possibles. Ainsi, une manière simple de représenter ce graphe est de considérer l'ensemble des couples des étiquettes de blocs qui indiqueront un arc du premier élément au second. Pour ce faire, considérons l'application suivante,

$$
\begin{aligned}
\varrho : Stm &\longrightarrow \mathbb{N}^2 \\
Id := Exp_a &\longmapsto \emptyset \\
Skip &\longmapsto \emptyset \\
s_1; s_2 &\longmapsto \varrho(s_1) \cup \varrho(s_2) \cup [\phi(s_1) \times \{\iota(s_2)\}] \\
(\text{if } b \text{ then } s_1 \text{ else } s_2)^l &\longmapsto \varrho(s_1) \cup \varrho(s_2) \cup (l, \iota(s_1)) \cup (l, \iota(s_2)) \\
(\text{while } b \text{ do } s)^l &\longmapsto \varrho(s) \cup (l, \iota(s)) \cup [\phi(s) \times \{l\}]
\end{aligned}
$$