

TP de Compilation n° 3 : De Cminor à RTLabs

20 Février 2014

Vous pouvez trouver tous les sujets au format électronique sur internet à l'adresse :

<http://www.pps.univ-paris-diderot.fr/~pboutill/CompilM1/>.

C sujet propose d'étudier la traduction des programmes Cminor en des programmes RTLabs.

Pensez à tester régulièrement votre code !

I) Présentation

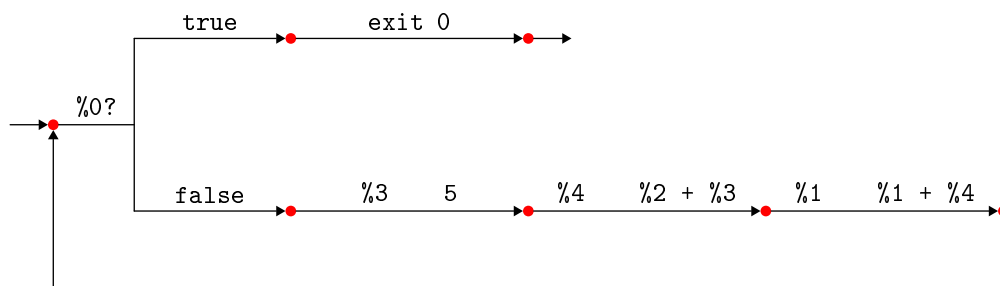
La syntaxe abstraite du langage Cminor se trouve dans le fichier `src/cminor/cminor.mli` ; celle du langage RTLabs se trouve dans le fichier `src/RTLabs/RTLabs.mli`. Dans un premier temps, nous vous conseillons de lire tout d'essai le commentaire des deux fichiers d'introduction.

Ainsi, vous verrez apparaître deux différences principales entre le langage Cminor et le langage RTLabs. La première est le passage à la représentation du programme sous forme de *graphe de flot de contrôle* : les sommets du graphe sont les *points de contrôle* du programme, les transitions entre sommets sont étiquetées par les instructions du programme. La seconde différence est une simplification des expressions : il n'y a plus de variables mais des *pseudo-registres*, et les expressions ne sont plus imbriquées.

Exemple : le bout de programme Cminor

```
block {
  loop {
    if (b) exit 0;
    x = x + y + 5;
  }
}
```

aura la forme suivante en RTLabs :



où on suppose que `%0`, `%1` et `%2` sont les pseudo-registres respectivement associés aux variables `b`, `x` et `y`, et `%3` et `%4` sont des nouveaux pseudo-registres.

Une dernière subtilité différencie Cminor et RTLabs : en RTLabs, les variables globales ne peuvent plus être initialisées en même temps qu'elles sont déclarées comme c'était le cas en Cminor. Leur initialisation doit être déplacée au début du `main` s'il existe.

Nous étudions dans ce sujet la phase de compilation de CerCo qui consiste à traduire un programme Cminor de type OCaml `Cminor.progr.m`, en un programme RTLabs de type OCaml `RTLabs.progr.m`. Cette traduction est faite par la fonction `translate_progr.m` du module `CminorToRTLabs` défini dans le répertoire `src/cminor`. Le fichier qui nous intéresse donc est `src/cminor/cminorToRTLabs.ml`.

Dans ce fichier, des portions de code ont été remplacées par des `assert false`. Vous pouvez les identifier facilement car ils sont suivis du commentaire `TODO M1`. La suite de ce sujet vous expliquera comment rétablir les portions de code désactivées.

II) Fonctions sur les graphes

1. La première fonction que nous vous conseillons de rétablir est la fonction `dd_graph`. Elle doit ajouter un sommet et son instruction associée au début du graph d'une fonction `RTLabs`. Vous pouvez accéder au graph d'une fonction par le champ de structure `RTLbs.f_graph`. De plus, vous pouvez utiliser les fonctions suivantes déjà définies :
 - `LabelMap.dd` : permet d'ajouter un sommet dans un graph tout en lui associant une instruction ;
 - `change_entry` : change le sommet initial du graph d'une fonction `RTLabs`.
2. L'autre fonction sur les graphes à rétablir est la fonction `generate` qui doit ajouter une instruction au début du graph d'une fonction `RTLabs` à partir d'un nouveau sommet (qui deviendra donc le nouveau sommet initial du graph). Vous pouvez vous servir de la fonction déjà définie `fresh_label` qui retourne un nouveau nom de sommet, la fonction `dd_graph` définie précédemment peut également être utilisée.

III) Instructions

La traduction des instructions `Cminor` se fait dans la fonction `translate_stmt`. La fonction retourne un graph équivalent à celui de la fonction `RTLabs` qui lui est passée en argument, mais où la traduction d'une instruction `Cminor` a été ajoutée au début du graph. La traduction d'un programme `Cminor` se fait donc en commençant par la fin et en remontant vers le début.

Remarque : la plupart des cas à rétablir nécessitent probablement une utilisation récursive de la fonction `translate_stmt`.

1. **Affectation** (constructeur `Cminor.St_assign`). Pour traduire une affectation `Cminor`, vous pouvez utiliser la fonction déjà (partiellement) définie `translate_expr` qui affecte à un pseudo-registre `RTLabs` la traduction d'une expression `Cminor`. La fonction déjà définie `find_label` vous permettra de trouver le pseudo-registre associé à une variable `Cminor`.
2. **Séquence** (constructeur `Cminor.St_seq`). Traduire la séquence de deux instructions `Cminor` consiste à ajouter au début du graph la traduction de la *seconde* instruction de la séquence d'abord, puis à ajouter au début du graph résultant la traduction de la *première* instruction de la séquence.
3. **Conditionnelle** (constructeur `Cminor.St_ifthenelse`). Soit ℓ le sommet du graph. Traduire une conditionnelle `Cminor` consiste à ne traduire chaque branch que si sa traduction mène à ℓ . Soient ℓ_{true} et ℓ_{false} les prochains sommets des deux branches traduites. Il faut alors ajouter au début du graph une conditionnelle `RTLabs` qui conduit à ℓ_{true} ou à ℓ_{false} (selon l'expression conditionnelle). Vous pouvez vous appuyer sur les fonctions `translate_branch` et `change_entry` déjà définies.
4. **Boucle** (constructeur `Cminor.St_loop`). Pour traduire une boucle, il faut ajouter le corps de la boucle au début du graph et faire en sorte que la dernière instruction permette de revenir au début de la boucle. Vous pouvez utiliser les fonctions déjà définies `fresh_label` et `change_entry`. Le constructeur d'instruction `RTLabs` `RTLbs.St_skip` pourrait également avoir une utilité.

I) Expressions

offsets. Ces décalages ne sont pas quelconques : ils dépendent de la taille de la variable et peuvent être construits grâce à la fonction `Memory.ll_offsets`.