Documentation de conception

Projet GL

ELHAJ-LAHSEN Hugo
GONDET Matthias
LAVAL Jean
MAZOUZ Mahmoud
NADJA Mohammad

GRENOBLE INP UGA Ensimag

Table des matières

Génération de code

Les classes correspondantes se trouvent dans le dossier deca/codegen. Ceci comporte trois interfaces importantes pour le fonctionnement du reste du projet.

1. OutputProgram

Cette interface présente une abstraction par rapport au programme assembleur généré, donc elle ne contient que des méthodes pour l'afficher sur un `PrintStream` et pour ajouter des commentaires. Le reste des fonctionnalités est gardé intentionnellement ouvert pour assurer la flexibilité du compilateur.

2. CodeGen

En plus de l'interface `CodeGenDisplay`, cette interface permet à un nœud de l'arbre abstraite de Deca de modifier un `OutputProgram` et y ajouter les lignes de code suffisantes pour décrire son comportement.

Parmi les classes du package deca/tree qui implémentent `CodeGen` il existe `AbstractExpr` et toutes ses sous classes. Dans le cas de IMA elles sont conçues pour générer un code assembleur qui produit la valeur de l'expression en question de le dernier registre alloué.

Analyse Contextuelle

La classe helper `Context` du package codegen/context continent des méthodes statiques très utiles pour l'implémentation de l'analyse contextuelle pour les différents nœuds de l'arbre abstraite de Deca.

Les méthodes existantes implémentent respectivement les fonctions `subType`, `assignCompatible` et `castCompatible` décrite dans les pages 75 et 76 de la spécification.

Nous pensons qu'il est important de séparer de tels éléments de logique afin de limiter l'effort de maintenance et réutiliser les mêmes fonctions partout dans le projet.

Génération de code

Calcul des Not à la compilation

Une expression du type `!(a && b)` se simplifie en `!a || !b` par les lois de de Morgan. De la même manière, une instruction `if(!a)` peut se traduire en inversant les conditions dans le if, et de même pour le while.

En s'inspirant de l'inversion faite par l'algorithme *Code()* des expressions booléennes page 221, nous avons étendu cet algorithme à tous les types d'expressions.

Pour l'implémenter, nous utilisons une interface Invert, avec une fonction *invert()* qui peut invertir tout nœud de l'arbre. Cette interface est implémentée par la plupart des nœuds booléens, qui modifient l'arbre localement avant d'appliquer l'algo de branchement.

Chargement des variables globales dans les registres

Nous avons modifié ExpDefinition pour qu'il puisse à la fois contenir une adresse et (optionnellement) un registre associé à cette adresse. Elle expose maintenant deux fonctions:

- getDVal(), renvoyant une DVal (registre ou adresse)
- getAdress(), renvoyant l'adresse du registre, qui doit être toujours mise

Le code utilise systématiquement getDVal, avec parfois une disjonction de cas. Par exemple, pour charger dans une variable, il faut maintenant utiliser soit STORE, soit LOAD en fonction de si le getDVal retourne une adresse ou un registre.

Cependant, quand un flush des registres vers les variables doit être fait, lors par exemple d'un appel de fonction ou de la fin du programme, pour respecter la convention d'appel, toutes les variables globales chargées dans des registres sont remises sur la pile.

Toolchain ARM

Le script `arm-env.sh` permet d'installer les divers outils de compilation et de débogage des programmes ARM32 sur Linux (par exemple le compilateur GCC, l'assembleur et l'éditeur de liens GNU ...).

Le script permet aussi d'exécuter les programmes assembleur ARM32 sur un environnement QEMU et spécifie le CPU Cortex-A15. Ceci est facilement changeable.

Suite de tests

Le script Python `test.py` se trouve dans le répertoire se trouvent dans src/test. Il permet de lancer automatiquement les tests d'intégration du compilateur Deca.

La suite de tests est subdivisée selon la partie testé du compilateur en plus de leur résultat attendu. Ceci permet un affichage clair et informatif de l'ensemble des tests exécutés.

Le script produit un résumé des tests échoués vers sa fin et offre la possibilité de produire un output plus détaillé avec l'option -X sur la ligne de commande. Il est

également facile de commenter et commenter les appels de fonctions suite_test_* en bas du script afin de désactiver des portions de la suite des tests.

Une “magic header” de la forme `//! <OPTIONS>` permet d’envoyer des options supplémentaires sur la ligne de commandes des programmes `decac` et `ima` afin de faciliter l'exécution des tests délicats (par exemple les tests de débordement de la pile).