SORBONNE UNIVERSITÉ

Projet compilation avancée

Objectif(s)

- ✯ Développement d'un interpréteur pour le langage UM de la machine universelle.
- ✯ Développement d'un compilateur du langage S-UM vers le langage UM.

1 Rendu

— **Quand ?** : 4 avril 2018 à 23h59.
— **À qui ?** : Un unique mail avec comme destinataires `Emmanuel.Chailloux@lip6.fr` et `talbot@ircam.fr`.
— **Quoi ?** : Une archive contenant 5 éléments :
  — `um/` : Les sources de l'interpréteur de la machine universelle (Section 2).
  — `sum/` : Les sources du compilateur du langage S-UM (Section 3).
  — `tests/` : Les fichiers de tests écrit en S-UM (Section 4).
  — `rapport.pdf` : Un rapport de 10 à 15 pages (Section 5).
  — `README.md` : Fichier indiquant comment compiler et tester le projet.
— **De plus...** : Les deux code sources `um` et `sum` pourront être compilé aisément :
  — Si c'est en C alors il faut un `Makefile`.
  — Si c'est en OCaml alors il faut un `Makefile` ou un projet OPAM.
  — Si c'est en Java, il faut un projet `Maven` ou `Ant`.
  — Si c'est en Rust, il faut un projet `Cargo`.

2 Machine universelle

2.1 Introduction

Cette partie est inspirée d'un sujet de la conférence *ACM International Conference on Functional Programming* (ICFP) de 2006. La version originale du sujet peut se trouver via le lien : `http://www.boundvariable.org/task.shtml`. L'interpréteur de cette section sera contenu dans le répertoire `um` de votre projet.

2.2 La machine

On nous demande de créer la machine universelle qui est composée de :
— Une quantité infinie de plateaux de sable avec des cases pour 32 marques (en langage plus usuel : 32 *bits*). Un plateau ressemble à :

```
                    least meaningful bit
                                       |
                                       v
.--------------------------------.
|VUTSRQPONMLKJIHGFEDCBA9876543210|
`--------------------------------'
 ^
```

```
        |
        most meaningful bit
```

Chaque bit peut prendre les valeurs 0 ou 1. En utilisant le système de numération 32 bits, ces marques peuvent aussi encoder des entiers.

— 8 registres à usage général capables chacun de contenir un plateau.
— Une collection de tableaux de plateaux, chacun étant référencé par un identificateur 32 bits. On distinguera le tableau 0 des autres, il contiendra le programme de la machine. Ce tableau sera référencé comme étant le tableau '0'.
— Une console de 1 caractère capable d'afficher les glyphes du code ASCII et faisant l'input et l'output de `unsigned 8-bit characters`.

2.3 Comportement

La machine sera initialisée avec un tableau '0' dont le contenu sera lu depuis le parchemin de programme. Tous les registres seront initialisés à '0'. L'indice d'exécution pointera sur le premier plateau de sable qui devra avoir l'indice zéro.

Quand on lit des programmes depuis des parchemins codés sur 8 bits, une série de 4 bytes (A,B,C,D) devra être interprétée comme suit :

— A sera le byte de poids fort.
— D le byte de poids faible.
— B et C seront intercalés dans cet ordre.

Une fois initialisée, la machine débute son cycle. À chaque cycle de la machine universelle, un opérateur devra être lu depuis le plateau indiqué par l'indice d'exécution. Les sections ci-après décriront les opérateurs que l'on peut récupérer. Avant que cet opérateur ne soit déchargé, l'indice d'exécution devra être avancé jusqu'au plateau suivant (s'il existe).

2.4 Opérateurs

La machine universelle dispose de 14 opérateurs. Le numéro des opérateurs est décrit par les 4 bit de poids fort du plateau d'instructions.

```
.--------------------------------.
|VUTSRQPONMLKJIHGFEDCBA9876543210|
`--------------------------------'
 ^^^^
 |
 operator number
```

2.4.1 Opérateurs standards

Chaque opérateur standard effectue un calcul en utilisant trois registres nommés : A,B et C. Chaque registre est décrit par un segment de 3 bits du plateau d'instruction. Le registre C est décrit par les 3 bits de poids faibles du plateau, le registre B par les 3 suivant et le registre A se trouve à la suite de B.

```
                          A     C
                          |     |
                          vvv   vvv
.--------------------------------.
|VUTSRQPONMLKJIHGFEDCBA9876543210|
`--------------------------------'
 ^^^^                        ^^^
 |                           |
 operator number             B
```

Les opérateurs standards sont les suivants :

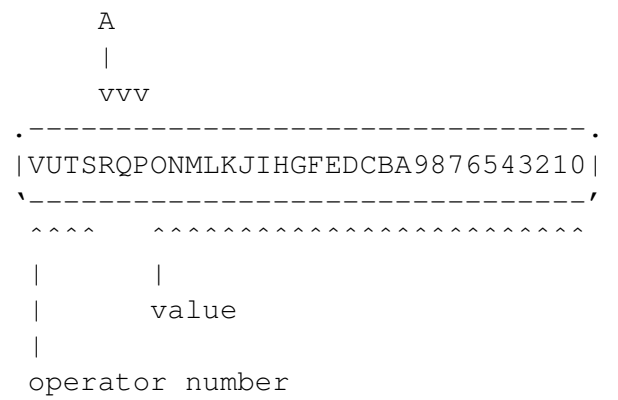
Opérateur 0 Mouvement conditionnel
Le registre A reçoit la valeur du registre B sauf si le registre C contient 0.
Opérateur 1 Indice de tableau
Le registre A reçoit la valeur à l'offset contenu dans le registre C dans le tableau identifié par registre B.
Opérateur 2 Modification de tableau
Le tableau identifié par le registre A est modifié à l'offset B par la valeur contenu dans le registre C.
Opérateur 3 Addition
Le registre A reçoit la valeur contenu dans le registre B plus la valeur contenu dans le registre C modulo 2^{32}.
Opérateur 4 Multiplication
Le registre A reçoit la valeur du registre B multipliée part la valeur du registre C modulo 2^{32}.
Opérateur 5 Division
Le registre A reçoit la valeur du registre B divisée par la valeur du registre C.
Opérateur 6 Not-And
Le registre A reçoit le `NAND` des registres B et C.

2.4.2 Autres opérateurs

D'autres opérateurs sont disponibles :
Opérateur 7 Stop
Arrête la machine universelle.
Opérateur 8 Allocation
Un nouveau tableau est créé avec une capacité égale à la valeur du registre C. Ce nouveau tableau est initialisé entièrement avec des plateaux à 0. Le registre B reçoit l'identificateur du nouveau tableau.
Opérateur 9 Abandon
Le tableau identifié par le registre C est abandonné et une future allocation peut réutiliser son identificateur.
Opérateur 10 Sortie
Écrit sur la console la valeur du registre C, seules des valeurs comprises entre 0 et 255 sont tolérées.
Opérateur 11 Entrée
La machine universelle attend une entrée sur la console. Quand une donnée arrive, le registre C est chargé avec :
— Tous ses bits positionnés à 1, si la fin de l'entrée est signalée.
— La valeur de cette entrée sinon.
Opérateur 12 Chargement de programme
Le tableau dont l'identificateur est B est dupliqué et sa copie remplace le tableau '0', quelle que soit sa taille. L'indice d'exécution doit être placé sur le plateau dont l'indice est contenu par le registre C.

2.4.3 Opérateurs spéciaux

Pour les opérateurs spéciaux, les registres utilisés ne sont pas décrits de la même façon. Les trois bits immédiatement après ceux décrivant l'opérateur donnent le numéro du registre à utiliser. Les 25 bits restant donnent une valeur qui devra être chargée dans le registre A.



Opérateur 13 Orthographe
La valeur indiquée doit être chargée dans le registre A.

2.5 Mesures de réduction de couts

Tout comportement non décrit par le schéma précédent pourra mettre la machine en panne.

3 Langage S-UM

Écrire soi-même des programmes dans le langage binaire de la machine universelle n'est pas chose aisée... Il est donc difficile de savoir si votre machine fonctionne bien. On vous propose d'écrire un compilateur de S-UM (Specification of Universal Machine) vers le langage binaire UM compris par la machine universelle. Il sera contenu dans le répertoire sum/.

3.1 Spécification de S-UM

Le langage S-UM est un simple langage impératif contenant les instructions suivantes :

Instruction	Action
$stmt$ `;` $stmt$	Instructions mises en séquences.
`let` var `=` $expr$	Affectation de la variable var par la valeur de $expr$.
`print` $expr$	Affiche la valeur de l'expression e.
`scan` var	Lit au clavier une valeur qui sera affectée à `var`.
`if` $expr$ `then {` $stmt$ `} else {` $stmt'$ `}`	Si $expr$ est vraie, exécuter le code décrit par $stmt$ sinon par $stmt'$.
Bonus	
`procedure` $ident$ `{` $stmt$ `}`	Déclare une procédure $ident$ dont le corps est spécifié par $stmt$.
$ident$`()`	Appel la procédure $ident$.

Finalement, les expressions arithmétiques, relationnelles et logiques manipulées sont les suivantes :

Expression	Rôle
`0,1,...,N`	Entier
`"une chaine"`	Chaine de caractère
`e [*+/] e'`	Expression arithmétique
`( e )`	Expression parenthésée
`e (<|=|>) e'`	Expressions relationnelles
`e (AND|OR) e'`	Expressions logiques binaires
`NOT e`	Expressions logiques unaires

Les chaines de caractères ne peuvent pas être manipulées via ces opérateurs, en fait, elles servent juste à être affichée via `print`.

Conseil : Vous commencerez par implémenter un langage très minimal, par exemple juste `print` pour les entiers (sans expression), testerez qu'il compile vers un fichier UM valide et que votre machine universelle peut l'exécuter. Ajouter les expressions et instructions **incrémentalement**.

3.2 Bonus : notes sur les procédures

On remarque que les procédures n'ont ni argument ni valeur de retour. L'utilisateur du langage utilisera des variables globales pour récupérer le résultat de ces procédures. Notez qu'elles peuvent néanmoins avoir des variables locales, et qu'elles peuvent être récursives. La difficulté est de simuler une pile d'appel : lorsqu'on appelle une procédure comment faire, à la fin de l'exécution, pour revenir sur le lieu de l'appel ? Nous proposons la méthode suivante :

— Étape initiale : stocker toutes les procédures dans un tableau différent.
— L'appel de procédure effectuera les étapes suivantes :

1. Copier le tableau de la procédure dans un nouveau tableau i.
2. Copier le programme courant (tableau 0) dans un nouveau tableau j.
3. Pour le retour de la procédure : Ajouter à la fin du tableau i une instruction qui charge le programme j (opérateur 12) avec le pointeur d'instruction à l'instruction suivante de l'appel de la procédure.
4. Pour l'appel de la procédure : Charger le programme i avec le pointeur d'instruction au début de la procédure.

Notez que la taille d'une procédure est une information statique que vous pouvez obtenir à la compilation. C'est notamment utile pour générer les instructions qui ajoutent une instruction à la fin d'un tableau.

En bonus du bonus, vous pouvez concevoir une méthode pour passer des paramètres aux procédures.

4 Infrastructure de fichiers de tests

On propose d'automatiser le test de la machine universelle. Dans le répertoire `tests/`, vous fournirez une série de fichiers de tests tels qu'un test est composé de trois fichiers :

— `test-add.sum` contient le fichier spécifié en S-UM. Ce fichier sera compilé vers un fichier binaire `test-add.um` par votre compilateur S-UM.
— `test-add.input` contient l'entrée attendue par `test-add.um` (ou vide si pas d'entrée utilisateur attendue).
— `test-add.output` contient la sortie attendue par `test-add.um` : ce que produit votre machine universelle quand on exécute ce fichier test.um.

Vous pouvez développer un utilitaire qui va automatiser la vérification de ces fichiers de tests et vérifier que la sortie produite est bien la sortie attendue.

5 Rapport de rendu

Le rapport de rendu sera constitué d'un unique document `rapport.pdf` de 10 à 15 pages contenant les sections suivantes :

— Description du travail produit : qu'avez-vous réalisé par rapport à l'énoncé, qu'est-ce qui marche (ou ne marche pas).
— Une vision globale de l'architecture logicielle du projet.
— La description détaillée de vos choix techniques pour la machine universelle sans répéter le sujet, par exemple : représentation des éléments constituant la machine, la représentation des données (dans le programme), ...
— La description du compilateur du langage S-UM et vos schémas de compilation : comment compilez-vous les instructions S-UM vers UM ? Quelles difficultées rencontrées ?
— L'architecture de votre système de tests.
— Toutes autres informations pertinentes.