Université Côte d'Azur
Enrico FORMENTI

Année 2021/22
M1 Info
Intro à la calculabilité

Examen sur projet

Table des matières

1 Le projet et ses finalités

Le projet de cette année voudrais être représentatif d'un cas d'application réelle des notions vues en cours. Il s'agira de concevoir un logiciel qui puisse interpréter un programme RAM et afficher le résultat du calcul de ce programme. La finalité ultime d'un tel logiciel est d'avoir une plateforme avec laquelle expérimenter autour de la calculabilité.

L'architecture du programme interpréteur est celle vue en cours (je vous fournirai un draft en pseudo code pour avancer plus vite). Il s'agira d'insérer cela à l'intérieur d'une interface (en utilisant `tkinter` si vous programmez en Python, par exemple) avec un minumum de fonctionnalités qui seront détaillées à la section suivante.

2 Fonctionnalités

Les fonctionnalités à ajouter à l'interface se divisent en deux groupes : les nécessaires et les optionnelles. La signification de chaque groupe est claire de le nom du groupe.

Par mis les fonctionnalités nécessaires on trouve :

Barre de menu : avec au minumum un menu 'File' permettant charger un programme à partir d'un fichier et de le sauvegarder dans un fichier en plus de la possibilité de quitter le logiciel ; à cela il faudrait ajouter un menu 'Run' qui permet de mettre en exécution un programme RAM avec deux modalités : exécution d'une seule instruction ou exécution du programme tout entier ; il faudrait aussi ajouter un menu 'Stop' qui permet d'arrêter l'exécution d'un programme et enfin un menu 'Help' qui donne une aide à la fois sur l'utilisation du logiciel et sur les instructions RAM pour l'utilisateur non expérimenté.

Editeur de texte : l'utilisateur doit avoir la possibilité d'éditer un programme RAM (qui a été chargé du disque ou qui est complètement nouveau) ; le programme ainsi édité doit pouvoir être mis en exécution ou sauvegardé sur le disque.

Par mis les fonctionnalités optionnelles on peut trouver :

Barre d'icones : il s'agit d'une barre d'icones à positionner juste en dessous de la barre de menu ; chaque icone est un bouton qui aidera de manière rapide certaines actions comme par exemple le chargement d'un programme ou encore sa sauvegarde. On pourra aussi ajouter des boutons pour contrôler l'exécution (une instruction à la fois, tout le programme), etc.

Debougueur : il s'agit de pouvoir marquer des lignes de code (sur l'éditeur on pourra changer la couleur de fond de la ligne en question par exemple) et faire en sorte que l'exécution du code s'arrête à cette ligne et en même temps de pouvoir marquer certain registres et afficher leur contenu dans une fenêtre séparé de manière à pouvoir les observer pendant l'exécution du programme

Macros : il s'agit de donner la possibilité de définir des macros. Le macros sont introduites par l'instruction `begin macro nom(`$R_x, \ldots, R_z$`)` et terminées par l'instruction `end macro` ; `nom` est le nom de la macro et R_x à R_z est la liste des registres passés à la macro. Pour simplifier on considérera que le passage des paramètres se fait exclusivement par valeur.

push/pop : il s'agira d'implémenter une couple d'instructions `push R` qui pousse la valeur du registre R dans un registre spécial (on pourra imaginer d'utiliser le registre R_2 mais que se passe-t il si un programme chargé depuis le disque utilise ce registre ?) et la macro `pop R` qui prend la valeur du registre R_2 et le copie dans `R`.

Preprocessing on pourrait imaginer d'ajouter des directives de preprocessing comme, par exemple, `#include file` qui est remplacée par le contenu du fichier `file` ou encore `#define nom chose` qui remplace dans la suite du fichier toute occurrence du mot `nom` par la séquence de caractères `chose`. Si vous en avez l'envie (et le temps) vous pouvez aussi prévoir des directives de sélection du style `#ifundef/#else/#endif` typiques des compilateurs C/C++.

3 Langage de programmation

Le choix du langage de programmation est libre mais je vous suggère très fortement d'utiliser Python (3.9 ou 3.10) car il intègre naturellement les grands entiers (et vous vous rappelez depuis le cours qu'on en a grand besoin assez vite) et aussi vous mes à disposition un certain nombre de bibliothèques simples à prendre en main pour créer des interfaces utilisateur (voir `tkinter` par exemple).

4 Comment se déroule l'examen

Lors de l'examen il faudra faire une démonstration du logiciel à votre professeur. Pendant la démo il faudra montrer toutes les fonctionnalités que vous avez développé. Le professeur pourra à l'occasion vous poser des question à la fois sur le logiciel et sur la partie de théorie afférente.

L'examen durera 1 heure maxi.

A Pseudo-code pour la simulation d'une machine RAM

Le code s'attends en entrée un entier `n` qui est vu comme une couple `p,x` (rappelez-vous de la fonction couple de Cantor) où `p` est le programme à simuler et `x` est la donnée en entrée pour ce programme. Quelques commentaires sur le code.

L'entrée est décodée par les lignes 2 et 3. Puis, les lignes 3-10 initialisent les variables nécessaires à la simulation. Le conteur de programme `intPC` de la machine RAM ; la longueur du programme (la macro `len` prend en entrée un entier i.e. un programme RAM et en donne sa longueur) et la "mémoire" de la machine RAM (la macro `allouer` un vecteur d'entiers d'une longueur passée en paramètre).

L'initialisation de la mémoire mérite quelque commentaire supplémentaire. Nous cherchons d'abord l'instruction avec le codage maximal dans le programme (la macro `seqmax` à ligne 6 calcule cette valeur). Cette valeur majeure l'indice de registre maximal qui est utilisé dans tout le programme (voir les propriétés de la fonction de Gödelisation). On majeure cette valeur encore de 1 (ligne 7) pour être sûrs d'avoir assez de

place les registres d'entrée/sortie. Ensuite on initialise la "mémoire" avec la valeur d'entrée `intArg` (lignes 9-10).

Les lignes 11-52 constituent le bloc principal de la simulation. Ce bloc est exécute tant que `intPC` est différent de zéro. La ligne 13 sélectionne la ligne de programme RAM à exécuter et la 14 décode le type d'instruction. Les trois types d'instructions sont traités dans les blocs 15-20 pour l'addition de 1 à un registre ; 21-27 pour la soustraction de 1 à un registre et 28-52 pour les deux types de saut.

Les lignes 53-57 vérifient si la valeur du conteur de programme de la machine RAM à dépassé la longueur du programme lui-même. Si oui, la simulation est arrêtée ; sinon on fait une autre étape.

La ligne 58 récupère la valeur de sortie (s'il y en a une) de la machine RAM que nous venons de simuler.

Listing 1 – L'interprète de machines RAM écrit en pseudo-code.

```
1  begin
2   intFonc = pg(entree);
3   intArg = pd(entree);
4   intPC = 1;
5   intPrgLen = len(intFonc);
6   intTmp = seqmax(intFonc);
7   intTmp = intTmp + 1;
8   intMem = allouer(intTmp);
9   intTmp = couple(intArg,intMem);
10  intMem = intTmp;
11  while intPC ≠ 0 do
12   begin
13    intExec = pro(intFonc, intPC);
14    intInstType = mod(intExec,3);
15    if intInstType == 0 then
16     begin
17      intExec = div(intExec,3);
18      intMem = proadd(intMem, intExec);
19      intPC = intPC + 1;
20     end
21    if intInstType == 1 then
22     begin
23      intExec = intExec - 1;
24      intExec = div(intExec,3);
25      intMem = prosub(intMem, intExec);
26      intPC = intPC + 1;
27     end
28    if intInstType == 2 then
29     begin
30      intExec = intExec - 2;
31      intExec = div(intExec,3);
32      intTypeSaut = pg(intExec);
33      intTmp = pd(intExec);
34      intRegNum = pg(intTmp);
35      intSautVal = pd(intTmp);
36      intRegVal = pro(intMem,intRegNum);
37      if intRegVal > 0 then
38       begin
39        if intTypeSaut == 1 then
40         begin
41          intPC = intPC + intSautVal;
42         end
```

```
43          else
44           begin
45            intPC = intPC ÷ intSautVal;
46           end
47         end
48        else
49         begin
50          intPC = intPC + 1;
51         end
52       end
53      if intPC > intPrgLen then
54       begin
55        intPC = 0;
56       end
57    end
58    sortie = pro(intMem,2);
59  end
```

La fonction `pro(n,k)` retourne le `k`-ième élément du vecteur d'entiers `n` (ici le vecteur `n` est représenté par un entier, rappelez vous de ce qu'on a dit en cours à ce propos). De manière semblable, la fonction `proadd(n,k)` (resp., `prosub(n,k)`) retourne un vecteur avec son `k`-ième élément augmenté (resp., décrémenté, avec ÷ bien sûr) de 1.

B Pseudo-code pour la macro `pd(n)`, partie droite de l'entier `n`

Dans cette section on trouve le pseudo-code pour les fonctions `pg(n)` (resp., `pd(n)`) partie gauche (resp., droite) de l'entier `n`.

Listing 2 – Pseudo-code pour la fonction auxiliaire `pz(n)` qui calcule la ligne su laquelle se trouve l'entier `n` dans l'image de la fonction couple de Cantor. Le résultat est stocké dans la variable `pzres`.

```
pzArg = n;
pzTmp = 0;
pzres = 0;
pzDiff = pdArg;
while pzDiff ≠ 0 do
 begin
  pzTmp = pzres;
  pzres = pzres + 1;
  pzTmp = mul(pzTmp,pzres);
  pzTmp = div(pzTmp,2);
  pzDiff = pzArg ÷ pzTmp;
 end
pzres = pzres ÷ 1;
```

Listing 3 – Pseudo-code pour l'opération `pd(n)`. Le résultat est stocké dans la variable `pdres`

```
pdArg = n;
pdres = pz(pdArg);
pdTmp = pdres + 1;
pdTmp = mul(pdrez, pdTmp);
pdTmp = div(pdTmp, 2);
pdTmp = pdTmp + 1
pdres = pdArg ÷ pdTmp;
```

Listing 4 – Pseudo-code pour l'opération `pg(n)`. Le résultat est stocké dans la variable `pgres`.

```
pgres = pz(n) ÷ pd(n);
```