Compilateur

RENDU FINAL

YANNICK ET PE

1. Partie I — PROLOGUE

- Remerciements. -
On a été ravi de faire ce modeste compilateur. On a certes passé des moments de solitude face à des bugs dit incompréhensible mais on s'est tout de même fait plaisir avec ce projet. Voici une entrée en matière sincère qui nous vaudra +1. On est conscient que notre rapport n'est pas très long mais il n'y a pas grand chose à dire de plus que les anciens rapports.

- Exemples. -
On a, cette fois-ci, fait une banque d'exemples plus complète (Rapport_Exemples.yannick_pe). Vous pourrez constater que cela fonctionne bien. Lorsque notre façon de faire les choses est conventionelles (propre à tous), nous ne rentrons pas trop dans les détails (boucle for, ...).

- Passage aux aveux. -
Passage aux aveux :

1) On ne peut faire de fonctions mutuellements récursives (oublié !)

2) On ne vérifie pas si les tableaux débordent. (voir tableau.pas) (bienvenue aux hackers)

3) c'est tout !

2. Partie II — LIVENESS

- Modifcation de liveness. -
On a tenu à faire les enregistrements de registres de manière assez "éxotique". On se donne 16 (on pourrait s'en donner plus) registres pour la coloration. Il faut savoir que fp, a0 et ra ne sont jamais utilisés pour autre chose que leur utilisation conventionelle.

Pour la coloration avec les fonctions, tous les arguments sont dans un temporaire différent. Ils sont pris en compte lors de la coloration dans la colonne USE. Il n'a pas falut modifier beaucoup la coloration pour introduire les fonctions.

On a décidé de colorier indépendamment chaque fonction une et une seule fois. Pour cela, le manuel a dû énumérer toutes les fonctions du programme et appeler l'intellectuel pour chacune des fonctions.

- Oui, mais l'exotisme, c'est quoi ? -
Lors de la coloration d'une fonction. On se rend compte que l'on a besoin d'un certain nombre de registres (au plu 16). Si l'on a besoin de 4 registres, on a décidé de donné 4 registres aléatoirement pris dans les 16 disponibles. C'est dans un souci d'équité de l'utilisation des registres. En réalité, on pensait réduire le nombre de registres à sauvegarder lors d'un appel de fonction. Au final, on est juste en callee saved. Ce qui est aussi rigolo, c'est que notre programme génère un code différent lors de 2 compilations (on attend les hackers !).

3. Partie III — DEBUGS

- Beaucoup ? Pas beaucoup ? De quel type ? -
La dernière ligne droite fut un peu pénible car on a eu pas mal de bugs non fondamentaux et difficile a déceler. On est maintenant rodé et les derniers bugs se sont résolus plus vite que les premiers.

La chose qui nous a le plus perturbé est le signe de l'offset des fonctions. En fait, l'offset (qui se trouve dans la table des symboles) est une valeur positive. Cela ne pose de problème (quoique) lorsque l'on génère le code intermédiaire avec STACK(descripteur_fonction.offset). Cependant, dans le gencode et même dans la génération du code intermédiaire, il faut bien faire attention car la pile descend. On s'est un peu cafouillé sur les +4 et -4 ou autre... On ne se rappelle plus de tous les bugs.

- Méthode pour les corriger. -
Pour les corriger, la meilleure méthode a été d'essayer de faire le plus petit programme qui comportait ce bug. Puis, on se lancait dans le code spim qui était donc réduit. Il nous était plus facile de voir à quoi il correspondait.

4. Partie IV — Les tableaux

- Nos tableaux. -
Les tableaux possibles dans notre pascal sont les tableaux multidimensionnels dont les tailles sont fixés : hyper-parallèlepipèdes. La manière de les stocker en memoire est la suivante : En en-tête, on donne la taille d'une case du tableau (de tableau...). Directement à la suite sont mis les cases de ce tableau (qui peuvent etre des tableaux.). On avait besoin de ça pour passer des tableaux de tailles differentes à une fonction. A la base, lors de la déclaration d'un tableau de grande taille, on le remplissait explicitement, sans boucle. Il se trouve que c'était très inefficace lors de la coloration (et pour la taille de l'executable bien sur). Les tableaux sont donc maintenant remplis à l'aide d'une boucle. (big_array.pas)

- Vérification de débordement. -
Ici il n'y a pas de verification de debordement de tableau (de la même manière on ne verifie pas si on divise par zero !). On aurait pu en rajoutant dans l'entete la taille du tableau mais on aimait l'idée qu'on puisse faire des choses crades grâce à ça !

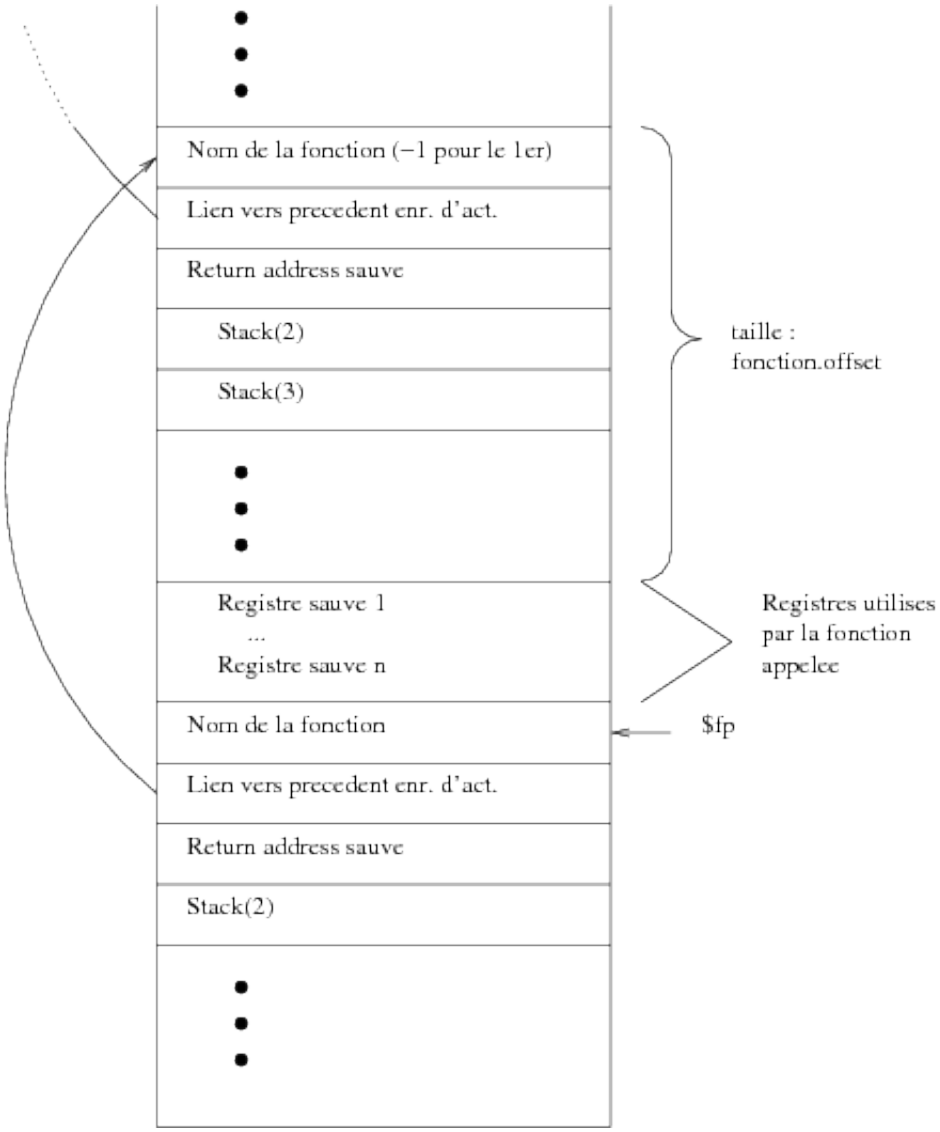
5. Partie V - Table de hachage

Cette partie a été mise dans Rapport_final.pe_yannick.

```
1  type symbol_table =
2    Root of ((string,descriptor) Hashtbl.t) (* un dictionnaire des choses
        definies directement dans le programme *)
3    | Locale of symbol_table*((string,descriptor) Hashtbl.t) (* un dictionnaire
        des choses definies dans une fonction et un lien vers la table des
        symboles dans laquelle est definie cette fonction.*)
4  and descriptor =
5      Int of descr_int
6    | Array of descr_array
7    | Function of descr_fun
8  and access = Temp of int | Stack of int (*explicite*)
9  and passage = (* suite dans le rapport txt*)
```

6. Partie VI - Enregistrement d'activation

Un bon croquis vaut mieux qu'un long discours (Napoleon 1er)



7. Partie VII - Merci encore

Vive la france, vive la république, Vive le général de Gaulle ! !