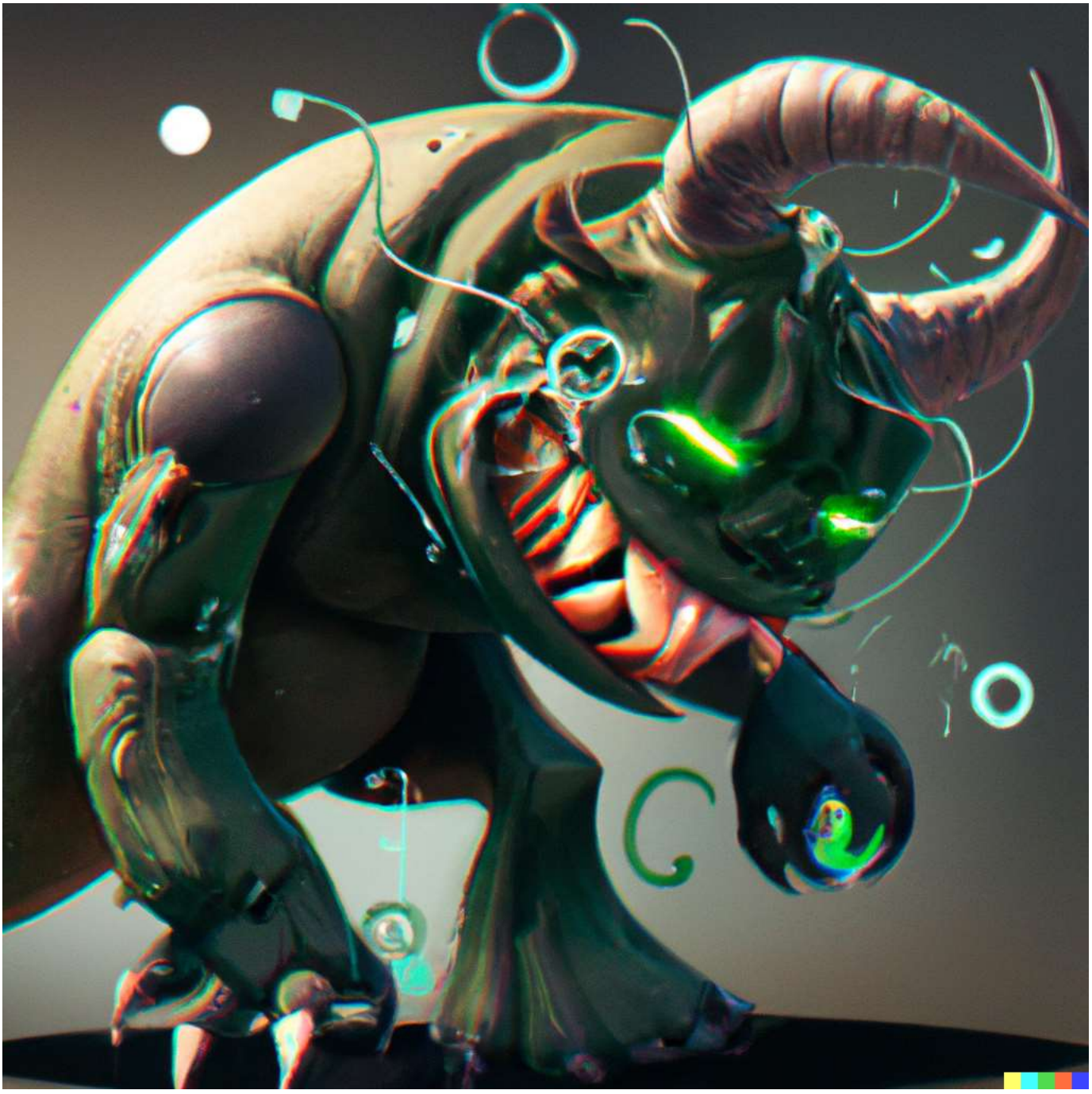
Error : segmentation fault, depicted as a monster, digital art

Rapport de Projet de Compilation RPCR

Clément DUMAS



Points manquants

- Copie récursive des structures pour passage par valeur
- Comparaison structurelle des structures
- Passage par valeur des structures Pas facile à implémenter et requérant beaucoup de debugging, j'ai préféré les laisser de côté pour me concentrer sur la robustesse du reste du code.

Méthodologie

Structure du projet

J'ai adapté le projet en l'organisant en librairies afin d'utiliser pleinement le potentiel de `dune` et de le rendre plus lisible. J'ai ajouté les flags `--trace` et `--run`. `--trace` permet de voir la stack trace des erreurs qui sont levées. C'est très utile pour le debugging. `--run` permet de lancer le programme après la compilation, ce qui évite de devoir appeler `gcc` à la main. Le flag `--debug` désactive les erreurs de variables non utilisées afin d'avoir des exemples minimaux permettant d'identifier la cause des segfaults.

Typing

Environnement

Pour les fonctions et les structures, j'utilise une surcouche sur le module Hashtbl. La mutabilité de cette méthode rend son utilisation plus agréable qu'une Map.

Pour les variables, j'utilise une StringMap afin de gérer la localité des fonctions. Pour cela, je traite le cas PEVar à l'intérieur du cas PEBlock avec la fonction `type_in_block`.

Structures

Je détecte les structures récursives au moment de calculer la taille des structs. Pour cela, je "marque" les structures dont je calcule la taille en la mettant à `-2`. Si je rencontre une structure marquée à `-2` alors je suis dans une structure récursive et je renvoie une erreur. Si je rencontre une structure initialisée à `-1`, alors c'est que je n'ai pas encore calculé sa taille et je la calcule. Si je rencontre une structure initialisée à une valeur positive, alors c'est que je l'ai déjà calculée et je la renvoie.

Difficultés rencontrées

Comprendre la structure de la fonction `type_function_body` (ancienne `expr`), notamment le booléen retourné, m'a pris pas mal de temps. En effet, celui-ci était marqué de type `import_fmt` à cause de `open Ast` qui écrasait le type `bool`. J'ai fini par comprendre que l'on devait retourner la présence d'un `return` dans le corps de la fonction.

Le typing n'a pas grand-chose d'intéressant algorithmiquement parlant, le plus dur fût de m'approprier le code existant dans un premier temps puis de ne pas y semer des bugs comme le petit poucet semait des cailloux. J'ai encore dû corriger des oublis la veille de rendre ce rapport. Une égalité de type qui utilisait `=` au lieu de `eq_type`, un oubli de tests de types pour `PEassign` et une inversion d'arguments pour `PEFor`.

Compilation

Environnement

Un simple type record suffit. Je n'ai besoin que du label de sortie, du nombre d'arguments et de la position relative de la dernière variable locale au pointeur de variable `%rbp`.

Détails d'implémentation

Chaque type a sa fonction de `print`, seules les structures ont un print un peu élaboré qui print récursivement les différents fields de la structure. J'ai donné des noms aux labels et ai inclus des numéros de ligne correspondants aux lignes du fichier source pour faciliter le débogage. Petite modification par rapport à la disposition recommandée dans le sujet : les résultats et arguments sont dans l'ordre croissant.

```
étiquettes
     F_function      entrée fonction
     E_function      sortie fonction
     L_xxx           sauts
     S_xxx           chaîne

   expression calculée avec la pile si besoin, résultat final dans %rdi

   fonction : arguments sur la pile, résultat dans %rax ou sur la pile

            res 1
            ...
            res k
            arg 1
            ...
            arg n
            adr. retour
            ancien rbp
   rbp ---> var locales
            ...
            calculs
   rsp ---> ...
```

Je gère le cas des effets de bords de assign en mettant chaque expr à droite du égal sur la pile. L'égalité entre structure est physique et je ne les passe pas par valeurs lors des appels de fonction. Ce sont les deux faiblesses de mon compilateur. J'ai modifié le code de `new_string` afin d'éviter les doublons. Le code pourrait être rendu plus compact en créant des fonctions pour les différents `print_struct`.

Je stocke les résultats des expressions dans `%rax` et je n'ai pas d'autres callee saved que `%rbx`.

Difficultés rencontrées

Au début, je n'ai pas compris l'intêret d'utiliser `%rbp`. J'avais une solution alternative, mais elle requérait beaucoup d'effet de bords pour mon environnement, ce qui entraîna un nombre de bugs assez conséquent, notamment dû qu'au fait que dans

```
pushq %rax ++ expr env e1 ++ popq %rax
```

L'ordre d'évaluation se fait de droite à gauche. Après avoir perdu un après midi à chercher à changer cela, j'ai enfin compris l'utilité de `%rbp` et ai pu corriger mon code. Une fois tout cela compris, l'appel de fonction a été assez simple à implémenter.

J'ai eu quelque soucis avec les left values qui causaient des segmentations fault. J'avais oublié que les structures étaient déjà des pointeurs et donc j'essayais de les modifier à partir de la pile.

Conclusion

Ce projet m'aura permis d'accroitre ma compréhension du code assembleur et de me rendre compte que la syntaxe du C n'est pas si mauvaise que ça après tout. Du côté d'OCaml, j'ai appris que l'ordre d'évaluation des opérateurs est de droite à gauche et qu'il est difficile de le modifier. Ce projet m'a aussi apporté beaucoup de larmes et de douleur : "Nul bonum sine dolore" (Aucun bien sans douleur). Je suis définitivement convaincu que la compilation n'est pas pour moi.