Bugperture Science

Compiler project

MinCaml -> ARM

Membres:

Astor Bizard

Gabriel Bouvier-Denoix

Jonathan Granier

Noha Wong

Thibault Lawson

Début du projet: 2 janvier 2017

Fin prévue: 25 janvier 2017

Langage choisi: Java

1er jour de travail: Mardi 3 janvier 2017

Nous avons choisi le nom du groupe: Bugperture Science et nous avons mis en place l'environnement de travail (pour la majeur partie):

Git, forge, des feutres pour tableaux et un cahier pour noter.

Nous avons ceci dit des difficultés avec le fait de cloner via forge.

Stand-up meeting: 4 janvier 2017

Repartition des tâches:

- Front-End:
  - Astor
  - Jonathan
  - Noha
- Back-End:
  - Gabriel
  - Thibault

Pour l'instant nous allons nous baser sur Git vu les problèmes que nous avons avec la forge. Nous avons choisi le langage JSon pour l'ASML.

Jeudi 5 janvier 2017

Pour git en graphique: git Kraken (ne marche finalement pas)

Back-End: décision de ne pas faire de Parser et de JSON car code non nécessaire et les avantages sont trop faibles:

[Front-End] -> {ASML} -> [Back-End]

"JSON file" -> [Parser] -> {ASML} -> [Back-End]

(schéma)

Vendredi 6 Janvier 2017

Explication au groupe du choix de l'abandon de JSON

Front-End: nécessité d'une nouvelle structure?

ASML: Discussion quand à une nouvelle structure de données plus restreinte:

-Front-End > force à se conformer aux possibilités d'ARM

-Back-End > pas plus de cas que ce que l'on peut implémenter en ARM

Environnement de travail:

Eclipse (IDE)

Git/forge (gestion de version) (en console)

Lundi 9 janvier 2017

Front End: K-normalisation finie

BackEnd: premières lignes de code ARM. Oh joie!

fonction OK. (???)

Mardi 10 janvier

Backend: gestion des variables en pile (2 registres sont reservés pour les opérations

K-normalisation corrigée

Beta-reduction finie

Mise en place d'un main commun

Gestion des floats dans ASML-traductor

Mercredi 11 janvier 2017

Modification du main pour les tests

Première gestion d'un conflit de façon propre. Yay!

ARMPrinter: stratégie: Machine à états pour écrire les cas de base.

Objectif à court termes:

- Gérer d'un bout à l'autre les cas:
  - arithmétique simple
  - appel de fonction

Jeudi 12 janvier 2017

Main: les options sont arrivés

Typecheck: ça avance.

Fin de la création de la fermeture (closure)

Backend:

Génération des premières lignes de code EXECUTABLES (Oh Joie!)

géré:

variables en pile

variables en registre

Opérations de base (Add & Sub)

Appel de fonction avec arguments & retour

(pas de vérification de signature, risque d'écrasement de donnée en registre 0-3)

Vendredi 13 janvier 2017

V1 disponible dans (avec ReadMe & tests):

projet_compilation_V1

BackEnd gestion du non-ecrasement des arguments dans un call

Fin des closures

BackEnd restructuration du code pour choisir si/ou faire des traces de débug

Lundi 16 janvier 2017

Generation d'ARM:

Fait

-Variables en registre

-Variable en pile

-Opération usuelles sur entiers

-Appel de fonction

-Déclaration de fonction

À faire:

-Variable en label

-Opération sur les vloats

-If

-(While)

-Expressions booléenes

-Tableaux et Tuples

Décision d'ajouter un arbre ASML plus contraint

Mardi 17 janvier 2017

Passage a la nouvelle structure (backEnd et translator)

Des problèmes sur le if (repenser la structure du code)

Problème de fin de fichier: pas d'OEF?

le problème viens de qemu

Changement de structure du code:

Printer --(if)--> Statmachine --> new Printer

Maintenant: appels multiples:

PrinterVisitor	Statemachine
If -->	ifcmp
	<---|
generer else	
|--->	ifelse
	<---|
generer then	
|--->	ifthen
	<---|
|--->	
	ifcontinue
	<---|
Suite	

Mercredi 18 janvier 2017

Renversement des Let x = if

Let x = [if B then e1 else e2] in Suite

==> if B then

[let x = e1 in ()]

else

[let x = e2 in ()]

Suite

Jeudi 19 janvier: Forum des entreprises

Jour de pause

Vendredi 20 janvier

ARMPrinter: gestion des if finie

-> modification de la structure du ArmprinterVisitor

-> retour plus souvent & boucle dans start

debug: double stream de sortie

Reste à faire:

Tuple

Array

Closure