

École Normale Supérieure
Langages de programmation et compilation
devoir à la maison 2020–2021

Jean-Christophe Filliâtre
22 janvier 2021 — 8h30–11h30

Les deux parties sont indépendantes mais les questions de la partie 2 peuvent faire appel à des définitions introduites dans les questions précédentes.

Les figures sont regroupées en fin de sujet, pages 6 à 8.

Cela peut être une bonne idée d’ouvrir une seconde fois ce document, uniquement pour les figures.

1 Une autre allocation

Dans cette partie, on étudie un algorithme d’allocation de registres alternatif, différent de celui étudié en cours (par coloration du graphe d’interférence). Ce nouvel algorithme a notamment le mérite d’être moins coûteux, ce qui permet de l’utiliser par exemple pendant de la compilation à la volée. On réalise cette allocation de registres dans le contexte d’un langage RTL, qui a été simplifié à l’extrême uniquement pour les besoins de ce sujet. L’allocation de registres est réalisée indépendamment pour chaque fonction du programme.

La figure 1 décrit les cinq instructions de ce langage RTL, où n désigne une constante entière, v désigne un pseudo-registre (qu’on appellera une *variable* par la suite), L désigne une étiquette et f désigne un nom de fonction. L’instruction `mov n v` charge la constante n dans la variable v ; l’instruction `mov u v` copie la variable u dans la variable v ; l’instruction `sub u v` soustrait la valeur de la variable u à la variable v (attention au sens) ; l’instruction `jnz $\rightarrow L_1, L_2$` saute à l’étiquette L_1 si le résultat de la dernière soustraction est non nul et à l’étiquette L_2 sinon ; enfin, l’instruction `$u \leftarrow$ call $f(v, w)$` affecte à la variable u le résultat de l’appel $f(v, w)$.

Une fonction est définie par son graphe de flot de contrôle. Les N instructions du graphe sont étiquetées avec des entiers, de 0 à $N - 1$. Le point d’entrée est l’étiquette 0 et le point de sortie est l’étiquette N . On suppose que chaque fonction a exactement deux arguments, reçus dans les variables x et y , et qu’elle renvoie un résultat dans la variable z . La figure 2 contient un exemple de graphe RTL avec $N = 8$ instructions.

Question 1 Que calcule la fonction de la figure 2 ? On pourra faire l’hypothèse que $x \geq 0$ à l’entrée de cette fonction.

Variables vivantes. Le premier ingrédient dont nous avons besoin est la détermination des variables vivantes en entrée de chaque instruction. On les calcule comme expliqué en cours, à partir des définitions et des utilisations de chaque instruction puis d’un calcul de point fixe sur le graphe de flot de contrôle.

Question 2 Pour chaque instruction RTL de la figure 1, indiquer quelles sont les variables que cette instruction définit (*def*) et quelles sont les variables que cette instruction utilise (*use*).

Question 3 En supposant que la variable z est vivante à la sortie du graphe de la figure 2, donner les variables vivantes en *entrée* de chacune des instructions.

Intervalles. Pour chaque variable v du graphe de flot de contrôle, on définit son *intervalle* comme le plus petit intervalle d'entiers $[i, j]$, avec $0 \leq i \leq j \leq N$, tel que la variable v n'est pas vivante à l'entrée de l'instruction k pour tout k en dehors de cet intervalle. En particulier, la variable v est donc vivante en entrée de l'instruction i et de l'instruction j . Mais elle n'est pas nécessairement vivante en entrée de toute instruction k pour $i < k < j$. On fait l'hypothèse que toute variable apparaissant dans le graphe de flot de contrôle est vivante à l'entrée d'au moins une instruction. On note qu'il est tout à fait possible que $i = j$, c'est-à-dire qu'un intervalle soit réduit à une seule instruction.

Ainsi, pour le graphe donné à gauche, on obtient les intervalles donnés à droite :

graphe	variables vivantes en entrée	intervalles
0 : <code>mov 0 t → 1</code>	0 : $\{x, y\}$	
1 : <code>sub t x → 2</code>	1 : $\{t, x, y\}$	x : $[0, 3]$
2 : <code>jnz → 3, 4</code>	2 : $\{x, y\}$	y : $[0, 4]$
3 : <code>mov x z → 5</code>	3 : $\{x\}$	z : $[5, 5]$
4 : <code>mov y z → 5</code>	4 : $\{y\}$	t : $[1, 1]$
	5 : $\{z\}$	

Les variables vivantes en entrée de chaque instruction sont données au centre. La variable z , supposée vivante à la sortie du graphe, est considérée comme vivante à l'entrée de l'instruction N (ici $N = 5$ sur cet exemple), même s'il n'y a pas vraiment d'instruction N .

Algorithme d'allocation des registres. On introduit maintenant un algorithme d'allocation des registres utilisant ces intervalles. On suppose qu'il y a K registres physiques, notés R_1, \dots, R_K . L'algorithme va affecter à chaque variable v une localisation, notée $loc[v]$, qui est soit un registre physique R_i , soit la valeur **Spill** pour indiquer que cette variable est vidée en mémoire.

L'algorithme est donné figure 3. Il est composé de trois fonctions : **allocation**, **expiration** et **vidage**. Le point d'entrée est la fonction **allocation**. On prendra le temps de bien lire et de bien comprendre cet algorithme. On note que cet algorithme ne dépend *que* de la valeur de K et des intervalles associés à chaque variable.

Question 4 Donner le résultat de cet algorithme pour $K = 3$ et les intervalles suivants :

$$x : [0, 7] \quad y : [0, 2] \quad z : [1, 8] \quad a : [2, 7] \quad t : [6, 6]$$

En cas d'égalité sur une valeur de i ou de j , quand l'algorithme indique de procéder selon un certain ordre ou de sélectionner une valeur maximale, on pourra choisir arbitrairement.

Question 5 Dans la fonction **vidage**, on pourrait se contenter de la partie **sinon**, c'est-à-dire de toujours effectuer $loc[v] \leftarrow \text{Spill}$. Expliquer l'intérêt de la partie **alors** de la fonction **vidage**.

Question 6 Cet algorithme d'allocation des registres permet-il d'allouer un même registre physique à deux variables u et v pour lesquelles il existe une instruction `mov u v`, à l'instar de ce que permettent (parfois) les arêtes de préférences dans l'allocation par coloration de graphe ?

Question 7 Montrer que cet algorithme est *correct*, au sens où il n'affecte jamais le même registre physique à deux variables distinctes vivantes en entrée d'une même instruction. On s'attachera notamment à énoncer clairement les *invariants* de cet algorithme.

Question 8 Le résultat de la question précédente ne suffit pas en pratique, car si le code appelle une fonction avec `call` alors les registres physiques utilisés par cette fonction vont être écrasés. Proposer une solution pour y remédier.

Question 9 Quelle est la complexité de cet algorithme, en fonction du nombre V de variables et de la valeur K ?

Numérotation des instructions. Comme on l'a compris, le résultat de l'algorithme dépend fortement de la numérotation des instructions. Or il existe $(N - 1)!$ façons différentes de numéroter les instructions, car seul le numéro de l'instruction 0 est imposé.

Question 10 Donner un exemple de graphe de flot de contrôle et deux numérotations possibles pour les instructions de ce graphe, l'une permettant une allocation avec seulement deux registres physiques et l'autre nécessitant au moins trois registres physiques (sans rien vider en mémoire à chaque fois).

Numérotation en profondeur. On se propose de re-numéroter les instructions à l'aide d'un parcours en profondeur du graphe de flot de contrôle. À chaque instruction i , on va associer un nouveau numéro $num[i]$, toujours entre 0 et $N - 1$. On utilise une variable globale $next$ et une marque sur chaque instruction. On initialise $next$ à $N - 1$ et on lance $dfs(0)$, où dfs est la fonction suivante :

```
dfs(i)
    marquer  $i$  comme visitée
    pour chaque successeur  $j < N$ , non visité, de l'instruction  $i$ 
        appeler  $dfs(j)$ 
     $num[i] \leftarrow next$ 
     $next \leftarrow next - 1$ 
```

En supposant que toutes les instructions sont atteignables à partir de l'instruction 0, l'appel à $dfs(0)$ va visiter toutes les instructions du graphe et terminer en donnant la valeur 0 à $num[0]$.

Question 11 Donner le résultat de cette numérotation sur le graphe de la figure 2.

Question 12 Donner un exemple de graphe de flot de contrôle montrant que la numérotation en profondeur ne donne pas forcément un résultat optimal, au sens où, après numérotation en profondeur, l'algorithme d'allocation a besoin de plus de registres physiques qu'avec la numérotation initiale.

2 Compilation d'un micro langage

Dans cette partie, on considère un tout petit fragment du langage OCaml dont la syntaxe abstraite est donnée figure 4. Chaque fonction a exactement deux arguments, notés x et y . Comme en OCaml, les fonctions sont récursives, mais pas mutuellement récursives : le corps d'une fonction f ne peut appeler que des fonctions définies préalablement ou la fonction f elle-même. Voici un programme dans ce fragment :

```
let rec mul x y =
  if x = 0 then 0 else y - (0 - mul (x - 1) y)
let rec fact x y =
  if x = 0 then y else fact (x - 1) (mul x y)
```

Avec ce programme, un appel à `fact n m` , pour deux entiers n et m , va calculer $n! \times m$, en supposant une absence de débordement arithmétique et $n \geq 0$. On peut supposer qu'une fonction `main` est ajoutée systématiquement à un tel programme, mais cet aspect-là ne nous intéresse pas ici.

Question 13 Indiquer ce que calcule un appel à `myst n m` , pour deux entiers n et m , avec $n \geq 0$ et la définition suivante :

```
let rec myst x y =
  if x = 0 then
    y
  else
    if x - 1 = 0 then 1 - (0 - y) else myst (x - 2) (myst (x - 1) y)
```

Question 14 Donner le code d'une fonction `lt` qui reçoit en arguments deux entiers n et m , en supposant $n \geq 0$ et $m \geq 0$, et qui renvoie un entier non nul si et seulement si $n < m$.

Sémantique opérationnelle à grands pas. On souhaite munir notre langage d'une sémantique opérationnelle à grands pas sous la forme d'un jugement

$$E, e \rightarrow n$$

où E est une fonction donnant la valeur associée aux variables x et y . Ce jugement signifie « dans l'environnement E , l'évaluation de l'expression e termine, avec la valeur n ». On suppose que les fonctions du programme sont données sous la forme d'un ensemble F de paires (f, e) où f est le nom d'une fonction et e l'expression qui constitue le corps de cette fonction.

Question 15 Donner les règles d'inférence définissant la relation $E, e \rightarrow n$.

Question 16 Donner un ensemble de fonctions F , un environnement E et une expression e pour lesquels il n'existe pas de valeur n telle que $E, e \rightarrow n$.

Analyse syntaxique. La figure 5 contient le fragment d'un fichier Menhir pour l'analyse syntaxique de notre langage, en se limitant ici aux expressions.

Question 17 Lorsque l'outil Menhir est lancé sur ce fichier, plusieurs conflits sont signalés. Identifier ces conflits et leur nature (*shift/reduce* ou *reduce/reduce*). Proposer des règles de priorités et d'associativités à ajouter au fichier Menhir pour résoudre ces conflits.

Compilation vers RTL. Enfin, on souhaite compiler ce petit langage vers le langage RTL de la partie 1 (voir figure 1). Chaque fonction f est compilée vers un graphe de flot de contrôle où les deux arguments x et y sont supposés être contenus dans les variables x et y en entrée et où le résultat de la fonction doit être placé dans la variable z au final. La compilation est écrite sous la forme d'une fonction $C(e, v, L)$ où e est l'expression à compiler, où v est la variable destinée à recevoir la valeur de e et où L est l'étiquette où le contrôle doit être transmis après l'évaluation de e . La fonction C construit le morceau de graphe de flot de contrôle correspondant et renvoie son étiquette d'entrée (exactement comme dans le cours 11). Pour compiler la fonction

$$\text{let rec } f \text{ x y} = e$$

on appelle donc $C(e, z, \textit{Exit})$ où \textit{Exit} est l'étiquette de sortie du graphe de flot de contrôle.

Question 18 Donner la définition de la fonction C , pour chacune des six constructions de la syntaxe abstraite du langage. On pourra supposer qu'on peut construire une variable fraîche avec $\textit{NewVar}()$ et une étiquette fraîche avec $\textit{NewLabel}()$.

instruction RTL ::= $\text{mov } n \ v \rightarrow L$
 | $\text{mov } v \ v \rightarrow L$
 | $\text{sub } v \ v \rightarrow L$
 | $\text{jnz } \rightarrow L, L$
 | $v \leftarrow \text{call } f(v, v) \rightarrow L$

FIGURE 1 – Un langage RTL minimal.

0 : $\text{mov } 0 \ z \rightarrow 1$
1 : $\text{mov } 0 \ a \rightarrow 2$
2 : $\text{sub } y \ a \rightarrow 3$
3 : $\text{mov } 0 \ t \rightarrow 6$
4 : $\text{sub } a \ z \rightarrow 5$
5 : $\text{mov } 1 \ t \rightarrow 6$
6 : $\text{sub } t \ x \rightarrow 7$
7 : $\text{jnz } \rightarrow 4, 8$

FIGURE 2 – Un exemple de graphe RTL.

allocation()

$actifs \leftarrow \emptyset$

$libres \leftarrow \{R_1, R_2, \dots, R_K\}$

pour chaque intervalle $v : [i, j]$, par ordre de i croissant

expiration($v : [i, j]$)

si $actifs$ contient K éléments **alors**

vidage($v : [i, j]$)

sinon

$loc[v] \leftarrow$ un registre retiré de $libres$

ajouter $v : [i, j]$ à l'ensemble $actifs$

expiration($v : [i, j]$)

pour chaque intervalle $v' : [i', j']$ de l'ensemble $actifs$, par ordre de j' croissant

si $j' \geq i$ **alors sortir** de **expiration**

enlever $v' : [i', j']$ de l'ensemble $actifs$

ajouter $loc[v']$ à l'ensemble $libres$

vidage($v : [i, j]$)

soit $v' : [i', j']$ un élément de $actifs$ avec j' maximal

si $j' > j$ **alors**

$loc[v] \leftarrow loc[v']$

$loc[v'] \leftarrow \text{Spill}$

enlever $v' : [i', j']$ de l'ensemble $actifs$

ajouter $v : [i, j]$ à l'ensemble $actifs$

sinon

$loc[v] \leftarrow \text{Spill}$

FIGURE 3 – Algorithme d'allocation des registres.

$e ::=$	expression
n	constante entière
x	la variable x
y	la variable y
$e - e$	soustraction
$f e e$	appel de fonction
$\text{if } e = 0 \text{ then } e \text{ else } e$	conditionnelle
$d ::=$	définition de fonction
$\text{let rec } f \ x \ y = e$	
$p ::=$	programme
$d \dots d$	

FIGURE 4 – Syntaxe abstraite.

```

expr :
| ZERO           {...}
| CST            {...}
| X              {...}
| Y              {...}
| IDENT expr expr {...}
| expr MINUS expr {...}
| IF expr EQUAL ZERO THEN expr ELSE expr {...}
| LPAR expr RPAR {...}

```

FIGURE 5 – Fichier Menhir (extrait).
