Cours de compilation

Chapitre 8. Sélection d'instructions

Master informatique fondamentale 1 – ENS-Lyon

Christophe Alias
Christophe.Alias@ens-lyon.fr
http://perso.ens-lyon.fr/christophe.alias

Introduction

- Le front-end produit un graphe de flot de contrôle entre blocs de base.
- Les blocs de base sont représentés par des DAGs qui explicitent les calculs communs.
- Il s'agit donc de générer du code machine pour évaluer un DAG, en exploitant au mieux le jeu d'instructions et les registres.

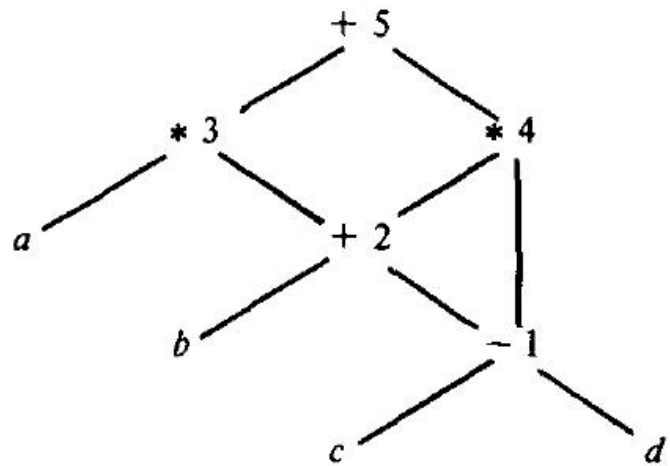
$$u_1 \leftarrow c - d$$

$$u_2 \leftarrow b + u_1$$

$$u_3 \leftarrow a * u_2$$

$$u_4 \leftarrow u_2 * u_1$$

$$u_5 \leftarrow u_3 + u_4$$



Génération de code

1. On partitionne le DAG en parties réalisables par une instruction machine. (Sélection d'instructions)
2. On construit un ordre d'évaluation de ces instructions. (Ordonnancement).
3. On alloue les temporaires. (Allocation de registres)

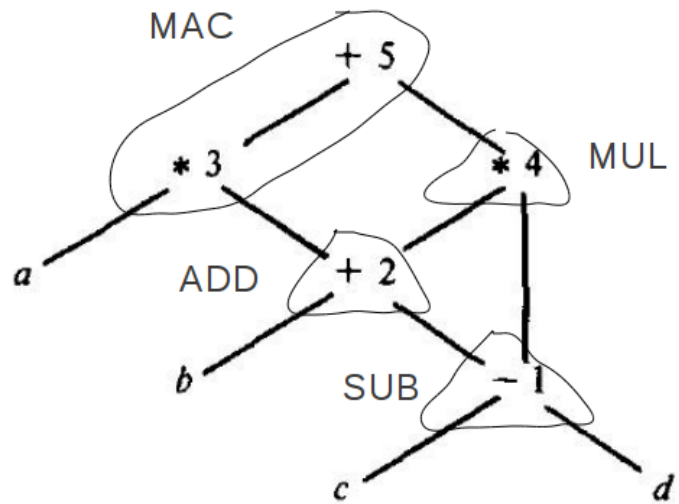
$$u_1 \leftarrow c - d$$

$$u_2 \leftarrow b + u_1$$

$$u_3 \leftarrow a * u_2$$

$$u_4 \leftarrow u_2 * u_1$$

$$u_5 \leftarrow u_3 + u_4$$



Sélection d'instructions

Les meilleures instructions sont celles qui:

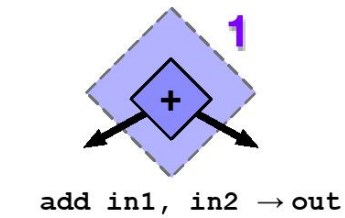
- couvrent le plus de calcul.
- provoquent le moins d'accès mémoire.

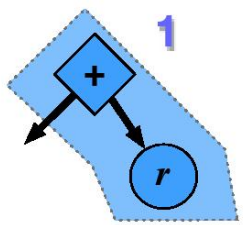
On tente de satisfaire ces contraintes (souvent contradictoires) en associant un coût à chaque instruction. Il s'agit alors de minimiser le coût global.

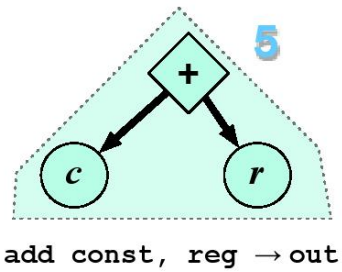
Par exemple, on peut noter le mode d'adressage:
immédiat $= 0$, registre $= 1$, mémoire $= 10$
Puis, définir le coût d'une instruction comme la somme de ses modes d'adressage $+ 1$.

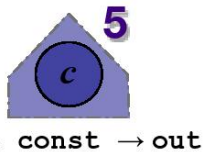
Tuilage

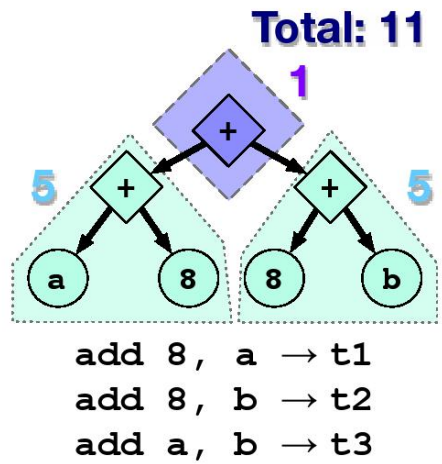
- Conceptuellement, chaque instruction peut être vue comme un sous-arbre (une tuile) du DAG.
- Il s'agit donc de trouver un tuilage de coût minimum.
- Problème NP-complet.









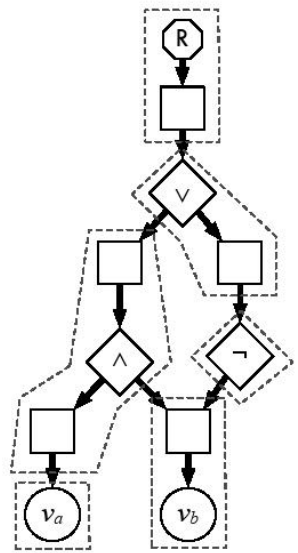


Complexité

Théorème. Le problème de la sélection d'instruction sur un DAG est NP-complet.

Preuve. Par reduction de SAT.
A partir d'une expression booléenne, on construit un DAG avec les règles suivantes.

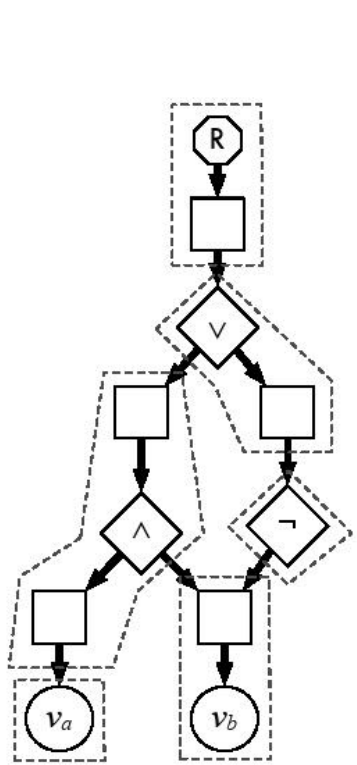
- Pour chaque variable x, on ajoute $\square \rightarrow x$
- Pour chaque operation x op y, on ajoute l'arc $\square \rightarrow op$, puis:
  - un arc de *op* vers le $\square$ associée à x
  - un arc de *op* vers le $\square$ associée à y
- Pour l'opération "racine" op, on ajoute un arc $R \rightarrow op$.



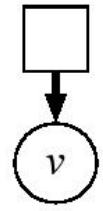
(a ∧ b) ∨ (¬b)

Complexité

- On considère maintenant les tuiles suivantes (de coût 1).
- Il existe un tuilage du DAG ssi l'expression booléenne est satisfiable.

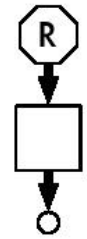


(a ∧ b) ∨ (¬b)

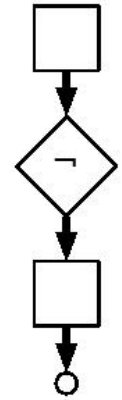


v: F

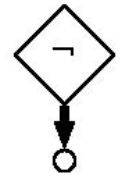
v: T



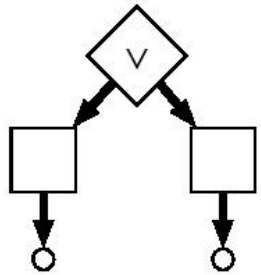
satisfied



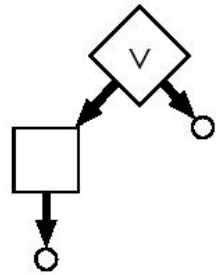
F: ¬T



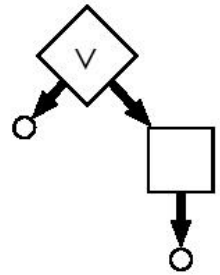
T: ¬F



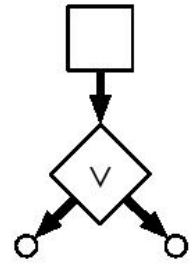
T: T ∨ T



T: T ∨ F



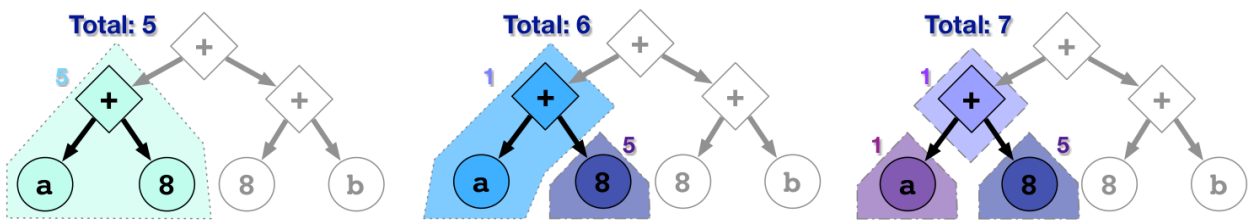
T: F ∨ T



F: F ∨ F

Sélection par programmation dynamique

- On ne considére non plus des DAGs, mais des arbres.
- Dans ce cas, il existe un algorithme optimal en temps linéaire.
- Dans l'ordre post-fixe (bottom-up), on calcule le tuilage optimal d'un noeud à partir des tuilages optimaux des sous-arbres.
- Les tuilages optimaux des sous-arbres sont connus, parce que déjà calculés (ordre post-fixe).
- Pour chaque noeud, on retient donc:
  - le coût optimal
  - la tuile correspondante.
- On garde ensuite la tuile optimale de la racine, les tuiles optimales de ses sous-arbres, etc.



Et sur les DAGs?

La solution optimale peut être trouvée par programmation en 0-1.

On note cout(t) le coût d'une tuile t, et input_nodes(i,t) l'ensemble des noeuds qui fournissent les entrées de la tuile t quand elle est enracinée au noeud i.

M_{it} vaut 1 lorsque la tuile t est enracinée en i (et 0 sinon).

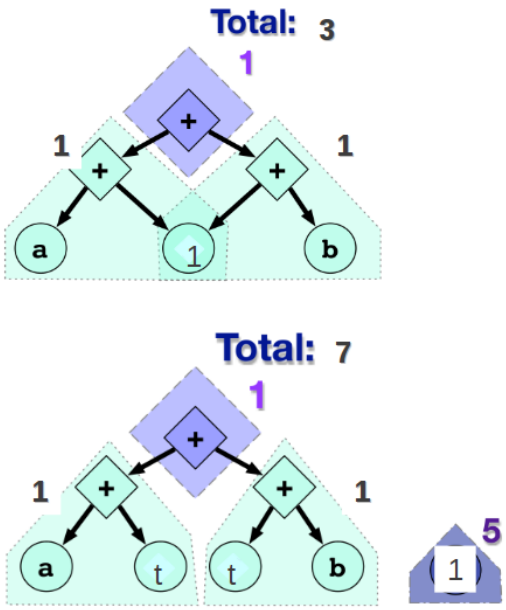
$$\begin{aligned}
&\max \textstyle\sum_t \mathrm{cout}(t) M_{it} \\
&\text{s.t.} \\
&\forall i \text{ racine du DAG :} \\
&\qquad \textstyle\sum_t M_{it} = 1 \\
&\forall i, t \; \forall i' \in \mathrm{input_nodes}(i,t) : \\
&\qquad \textstyle\sum_{t'} M_{i't'} \geq M_{it}
\end{aligned}$$

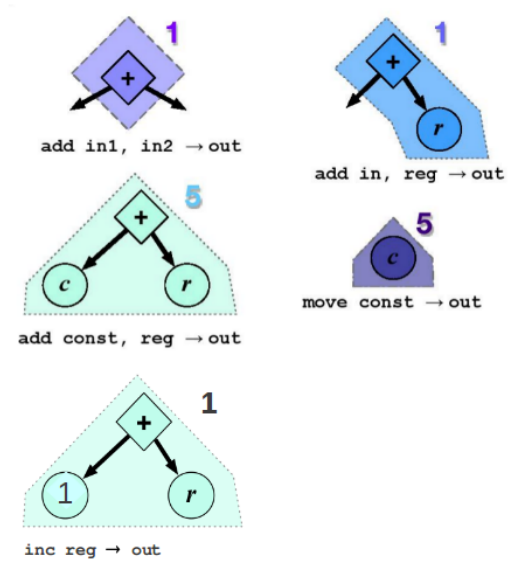
La première contrainte exprime qu'exactement une tuile est enracinée sur chaque racine du DAG. La seconde contrainte exprime que la tuile t est enracinable au noeud i s'il existe au moins une tuile t' enracinable sur chaque input i' de t.

... il est plus raisonnable d'utiliser des heuristiques, qui approchent l'optimal en temps raisonnable.

Heuristique naïve

On décompose le DAG en une forêt d'arbres, et on applique séparement la méthode par programmation dynamique.

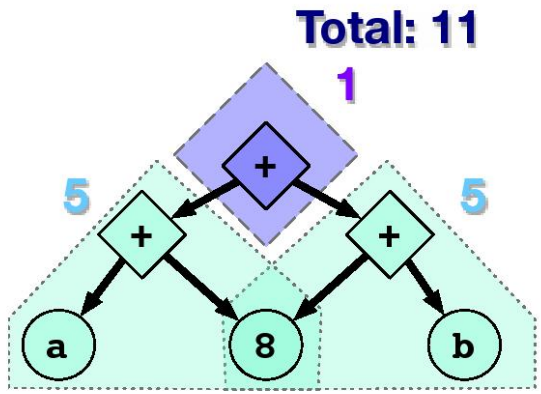




Un compromis...

Etape 1.

- Appliquer la méthode par programmation dynamique en ignorant les noeuds partagés (implicitement dupliqués).
- On obtient un tuilage, qui va permettre de décider quels noeuds partagés dupliquer.
- Les noeuds partagés peuvent être couverts par plusieurs tuiles.



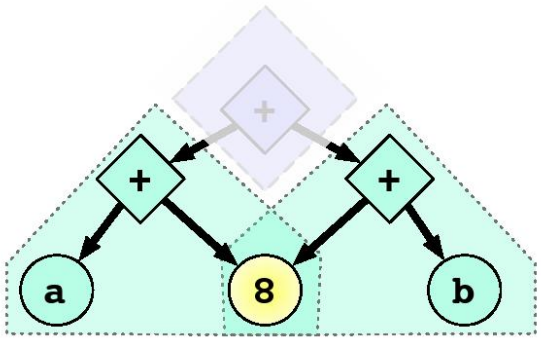
Un compromis...

Etape 2.

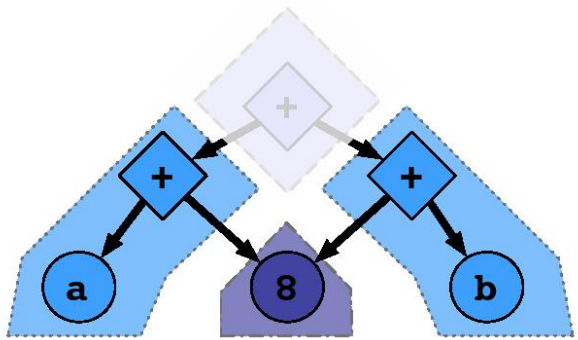
On cherche les noeuds partagés à ne pas dupliquer.

Pour chaque noeud partagé n,

- On calcule le coût en cas de duplication
  - Somme les coûts des tuiles qui recouvrent n.
- On calcule le coût en cas de non-duplication
  - Le coût de la tuile optimale enraciné en n est donné par la programmation dynamique.
  - On ajoute le coût de la coupure des tuiles qui couvrent n.
    Pour chaque père de n, on choisi la meilleure tuile qui ne recouvre pas n.
- Si le coût de non-duplication est inférieur, on marque le noeud.



cost = 5 + 5 = 10

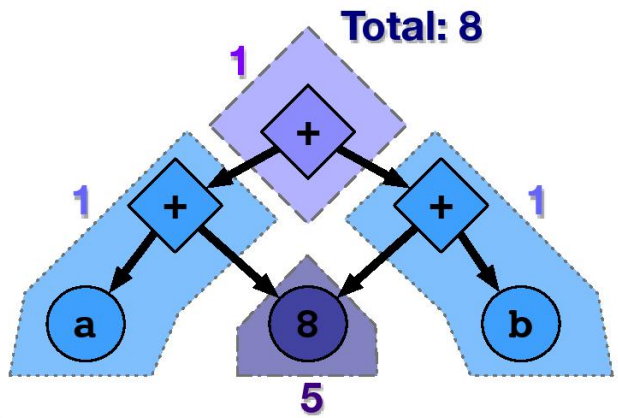


cost = 5 + 1 + 1 = 7

Un compromis...

Etape 3.

- On relance la méthode par programmation dynamique, modifiée pour ne pas recouvrir les noeuds marqués.



Conclusion

- Il existe des générateurs de sélecteurs d'instruction à partir d'une description du jeu d'instructions de la machine cible
- La sélection d'instructions pour une machine séquentielle est globalement résolue, il existe des heuristiques efficaces.
- La reconnaissance/sélection d'instructions complexes est toujours un problème ouvert (instructions vectorielles, réductions, etc).
- Extension: reconnaissance de fonctions, puis substitution par un appel vers une bibliothèque optimisée.