Examen de compilation

M1 Informatique Fondamentale, ENS Lyon, Jeudi 7 janvier 2010

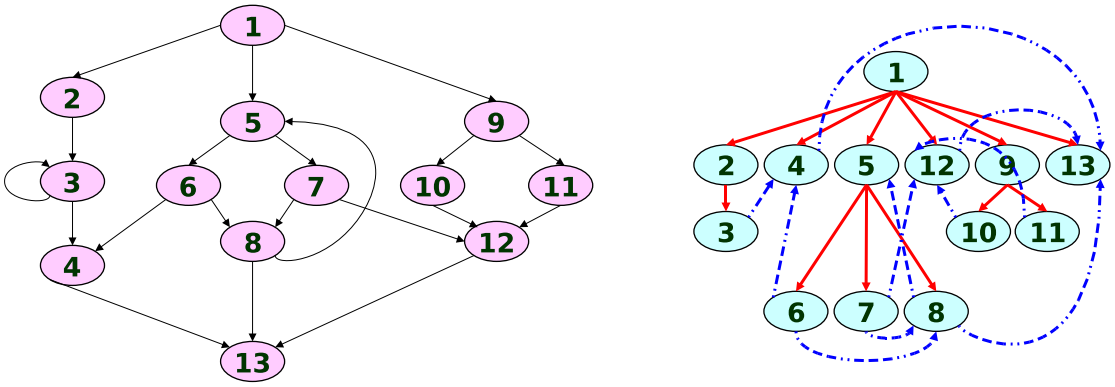
Durée: 3 heures. Tous les documents sont autorisés.
La réponse à l'exercice 3 est à rendre sur une copie **séparée**.
L'exercice 2 dépend de l'exercice 1, qui devra etre traité en premier.
L'exercice 3 est plus facile, et pourra être traité en priorité.

Exercice 1. Frontières de dominance

Le but de cet exercice est de construire un algorithme alternatif pour le calcul d'une frontière de dominance. L'exercice suivant reprendra ensuite l'idée de cet algorithme, ainsi que certains résultats montrés ici, pour construire un algorithme linéaire de placement des ϕ-fonctions dans un graphe de flot de contrôle.

Etant donné un graphe de flot de contrôle et deux noeuds x et y, on note x **dom** y pour indiquer que x domine y et x !**dom** y pour indiquer que x ne domine pas y. Si x **dom** y et $x \neq y$, on dit que x domine strictement y, et on note x **sdom** y. Si x est un parent de y dans l'arbre de dominance, on notera indifféremment $x =$ **idom** (y) ou x **idom** y. On note $SubTree(x)$ l'ensemble des noeuds dominés par x. A chaque noeud de l'arbre de dominance, on associe un niveau: sa profondeur dans l'arbre de dominance. On écrit $x.level$ pour indiquer le niveau de x. On note $DF(x)$ la frontière de dominance d'un noeud x.

Dans cet exercice comme dans l'exercice suivant, on ne travaillera sur une extension du graphe de flot de contrôle appelée DJ-graphe. Un DJ-graphe possède le même ensemble de noeud que la graphe de flot de contrôle, et deux types d'arcs appelés D-arcs et J-arcs. Un D-arc est un arc de l'arbre de dominance, et un J-arc est un arc du graphe de flot de contrôle qui n'est pas un D-arc. La figure suivante montre un graphe de flot de contrôle et le DJ-graphe correspondant. Les D-arcs sont pleins et les J-arcs sont en pointillés :



Dans la suite, on considère un graphe de flot de contrôle $G = (N, E)$ et son DJ-graphe $DJ = (N_{dj}, E_{dj})$. Le cardinal d'un ensemble X est noté $|X|$.

Q1. Montrer que $|N_{dj}| = |N|$ et que $|E_{dj}| < |N| + |E|$.

Q2. Soit x un noeud du DJ-graphe et $y \in DF(x)$. En utilisant **idom** (y), montrer que $x.level \geq y.level$.

Q3. En déduire que z est dans $DF(x)$ si et seulement si il existe $y \in SubTree(x)$ tel que $y \rightarrow z$ soit un J-arc et $z.level \leq x.level$. Pour la partie $\Leftarrow$, on distinguera le cas $z \in SubTree(x)$ de $z \notin SubTree(x)$.

Ces remarques nous amènent à construire l'algorithme suivant, le pseudo-code `y -> z == J_arc` testant s'il existe un J-arc de y vers z :

```
DF(x)
{
  DF = {}
  foreach y in SubTree(x) do
    if((y->z == J_arc) and
       (z.level <= x.level))
      DF = DF U {z}
}
```

Q4. Utiliser cet algorithme pour calculer $DF(5)$ dans le graphe de flot de contrôle donné ci-dessus. On détaillera les étapes de l'algorithme.

Q5. Avec cet algorithme, quelle est la complexité au pire de la méthode de mise sous forme SSA vue en cours ?

Exercice 2. Calcul efficace de la forme SSA

Cet exercice propose d'étudier un algorithme de placement des ϕ-fonctions linéaire en le nombre de noeuds du graphe de flot de contrôle. Les notions, idées et résultats de l'exercice précédent seront exploitées. Cette partie n'est donc pas indépendente.

On étend la définition de la frontière de dominance $DF(S)$ à un ensemble de noeuds S:

$$DF(S) = \bigcup_{x \in S} DF(x)$$

On défini la frontière de dominance itérée $IDF(S)$ pour un ensemble de noeuds S comme la limite de la suite croissante:

$$\begin{aligned} DF_1(S) &= DF(S) \\ DF_{i+1}(S) &= DF(S \cup DF_i(S)) = DF(S) \cup DF(DF_i(S)) \end{aligned}$$

Q1. Expliquez en quoi l'algorithme de mise sous forme SSA vu en cours revient à calculer plusieurs frontières de dominance itérées.

Comme dans l'exercice précédent, on ne travaille pas sur le graphe de flot de contrôle, mais sur son DJ-graphe. Notre algorithme manipule une structure de données appelée *banque*. Une banque est un tableau dont chaque case contient une liste de noeuds du DJ-graphe. Les noeuds sont insérés de telle façon que la case i ne contient que des noeuds de niveau i.

On considère une unique banque `bank` (variable globale). L'opération d'insertion, `InsertNode(n)` ajoute le noeud `n` à la liste `bank[n.level]`. L'opération de Recherche/suppression, `GetNode()` cherche la derniere case de `bank` non vide (la case d'indice le plus grand dont la liste est non vide), et retourne (puis supprime) son premier noeud.

Chaque noeud est représenté par un enregistrement :

```
struct {
  visited: { Visited, NotVisited }; //already traversed
  inphi: {InPhi, NotInPhi};         //already added to IDF
  initial: {Initial,NotInitial};    //belong to the initial set
  level: integer;
}
```

Initialement:
n.visited = NotVisited, node.inphi = NotInPhi, n.initial = NotInitial,
n.level = profondeur dans l'arbre de dominance

Voici maintenant l'algorithme qui nous intéresse. Cet algorithme calcule la frontière de dominance itérée d'un ensemble S de noeuds passé en paramètre. Le résultat pourra être directement exploité pour placer les ϕ-fonctions dans le graphe de flot de contrôle.

```
IDF_main(S)
{
  for each node n in S {
    InsertNode(n)
    n.initial = Initial
  }

  while(x = GetNode() != NULL) {
    CurrentRoot = x;
    x.visited = Visited;
    Visit(x);
  }
}
```

```
Visit(x)
{
  for each node y successor of x
  {
    if(x -> y == J_arc and              //
       y.level <= CurrentRoot.level and // y in DF(x)...
       y.inphi == InPhi) {              // ...and not already added to IDF
      IDF = IDF U {y};
      if(y.initial != Initial)  //do not re-process initial nodes
        InsertNode(y);
    }

    if(x -> y == D_arc and
       y.visited != Visited) { // not already visited
      x.Visited = Visited
      Visit(y);
    }
  }
}
```

Initialement, on insère les noeuds de l'ensemble S dont on cherche la frontière de dominance itérée. Ensuite, pour chaque noeud x par ordre de niveau décroissant (en allant plus haut dans l'arbre de dominance), on appelle Visit(x). Visit(x) parcours en profondeur le DJ-graphe en partant de x, en évitant d'explorer les noeuds déjà visités lors d'un appel précédent (voir le for each et son deuxième if).

Suivant l'idée développée dans l'exercice précédent, le premier if collecte les noeuds dans la frontiere de dominance de x. (voir les commentaires). Ces noeuds sont ajoutés à l'ensemble resultat IDF, puis a la banque *via* insertNode(), pour être traités dans un appel ultérieur à Visit().

Q2. Détailler les étapes de l'algorithme sur le calcul de $IDF(\{6,9\})$ sur le DJ-graphe donné dans l'exercice précédent.

Q3. Expliquer pourquoi un noeud est toujours inséré (*via* InsertNode()) à un niveau $\leq$ CurrentRoot.level.

Q4. Soient x et y deux noeuds insères dans la banque, puis supprimés par GetNode(). Montrer que si $y.level > x.level$, alors Visit(y) est appelé avant Visit(x). On distinguera, au moment de la suppression de x, le cas où y est dans la banque, et le cas où y n'est pas dans la banque.

Q5. Montrer que quand Visit(x) est appele avec $x = CurrentRoot$, tous les noeuds de $DF(x)$ sont dans IDF. on prendra garde au fait que $SubTree(x)$ n'est pas toujours entièrement parcouru.

Q6. Montrer que quand l'algorithme termine, $IDF(S)$ = `IDF`. On pourra montrer l'inclusion $\subset$ par induction sur la définition de $IDF(S)$.

Q7. Montrer que chaque noeud est visité (par un appel a Visit()) au plus une fois. Pour çà, on pourra montrer qu'un noeud ne peut pas être visité à la fois dans le while de IDF_main et dans le deuxième if de Visit().

Q8. En déduire la complexité en temps de l'algorithme.