

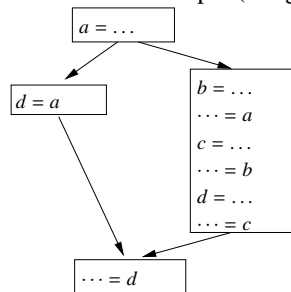
Le problème d'allocation de registres sous ordonnancement fixe: utilisation des propriétés de la forme SSA

Fabrice Rastello

1. Définitions et prérequis

1.1. Allocation de registres

- **Allocation de registre:** Allouer les variables sur un nombre fini de ressources physiques, les registres.
 ➔ On suppose ici un seul type de registres.
- Allocation \approx **coloriage**
- Sauvegarder une variable en mémoire pour une utilisation future: **spiller**.
- Regrouper deux variables qui représentent la même valeur en une variable commune: **coalescer**. *Ca peut transformer un code initialement colorable en non colorable.* Par exemple (2 registres, coalescing de a et d):



- L'opération inverse du coalescing: **splitting**.
- allocation de registres \leftrightarrow ordonnancement des instructions. Ainsi:

```

a = exp1
store a
b = exp2
store b
  
```

est 1-colorable
Symétriquement,

```

a = exp1
b = exp2
store a
store b
  
```

n'est pas 1-colorable, il faut 2 registres.

$R1 = exp_1$
 store $R1$
 $R1 = exp_2$
 store $R1$

$R1 = exp_1$
 store $R1$
 $R2 = exp_2$
 store $R2$

n'est pas réordonnancable

peut être réordonnancé pour par exemple être parallélisé:

$R1 = exp_1$
 $R2 = exp_2$
 store $R1$
 store $R2$

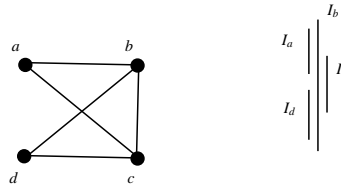
→ On suppose un ordonnancement fixe.

1.2. Graphes parfaits

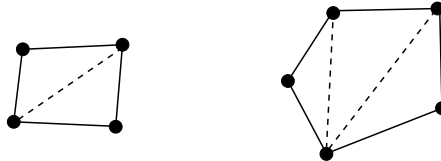
$w(G)$: nombre de clique. $\chi(G)$: nombre chromatique.

$G = (V, E)$ **Grphe parfait**: $\forall A \subset V, \chi(G_A) = w(G_A)$

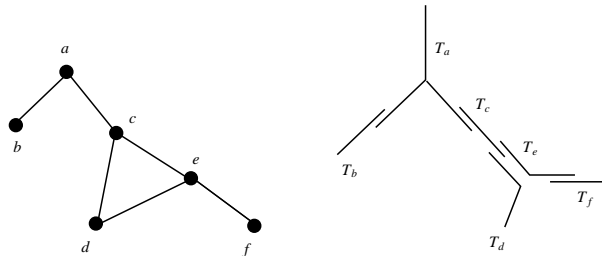
1.2.1. graphe d'intervalles graphe d'intersection d'une famille d'intervalles sur une droite:



1.2.2. graphe triangulé graphe non orienté où tout cycle de longueur > 3 contient une corde (chordal graph):



C'est le graphe d'intersection d'une famille de sous-arbres d'un arbre:

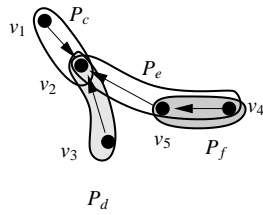


- **Glouton_Col**: Soit $v \in V$, tq $d^o(v) = \delta(G)$
 Colorie G_v à l'aide de **Glouton_Col**
 Assigne à v une couleur différente de celle de ses voisins.
- **Glouton-kColorable**: peut être k-coloré grâce à **Glouton_Col**.
- **Glouton_Col** est optimal ($w(G) = \chi(G)$) sur les graphes triangulés.

1.2.3. graphe de chemins graphe d'intersection d'une famille de chemins d'un arbre orienté:



La matrice d'indidence d'un hypographe de chemins d'une graphe orienté est totalement unimodulaire:

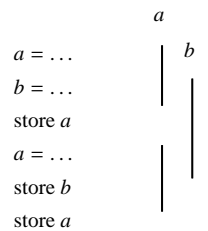
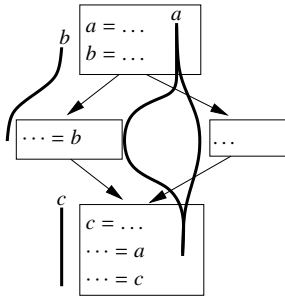


	P_c	P_d	P_e	P_f
v_1	1	0	0	0
v_2	1	1	1	0
v_3	0	1	0	0
v_4	0	0	1	1
v_5	0	0	1	1

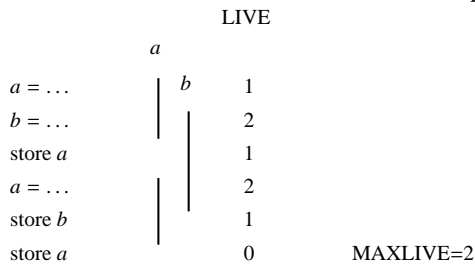
$\text{graphes d'intervalles} \subset \text{graphes de chemins} \subset \text{graphes triangulés} \subset \text{graphes parfaits}$

1.3. Formalisation du problème d'allocation de registres

- **durée de vie:** union des intervalles du CFG qui relient les définitions de cette variable à leurs utilisations.



- Deux variables **interfèrent** si la durée de vie de l'un contient un point de définition de l'autre.
- **MAXLIVE:** maximum sur l'ensemble des points du CFG du nombre de variables en vie.



- **graphe d'interférence:** $G = (V, E)$, à chaque variable est associée un sommet; $(u, v) \in E$ si et seulement si u et v interfèrent.
- **graphe d'affinité:** il y a une affinité entre u et v si dans le code `move u, v` ou `move v, u`.

1.4. Forme SSA

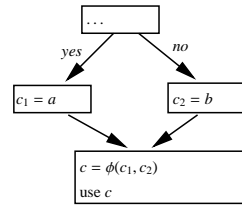
La forme SSA est utilisée dans de nombreux compilateurs. Elimination entre autre des fausses dépendances. Optimisation rendues simples et efficaces:

1. propagation de constantes
2. PRE
3. reconnaissance de variables d'inductions
4. analyse d'intervalle de valeurs
5. etc.

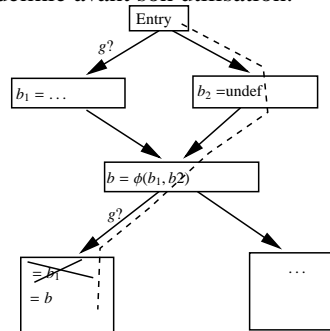
Chaque assignation n'apparaît **textuellement** qu'une seule fois:

```

if (condition)
  then c = a
  else c = b
use c
  
```



Toute variable utilisée est nécessairement définie avant son utilisation.



- **A domine B**: si tout chemin du CFG de Entry à B passe par A.
La notion de dominance est transitive, antisymétrique & réflexive (ordre partiel). Le diagramme de Hasse est **un arbre** dont les arêtes représentent les **dominateurs immédiats**.

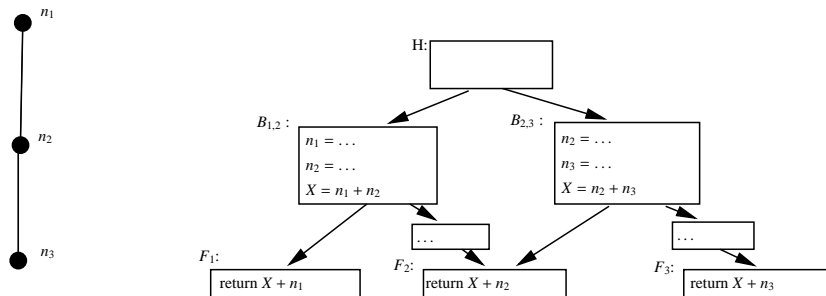
- Sous SSA, toute définition de variable domine l'ensemble de ses utilisation.

2. Allocation de registres: spiller, coalescer & colorier. Complexité des différents problèmes

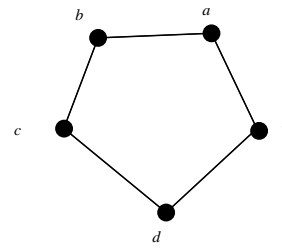
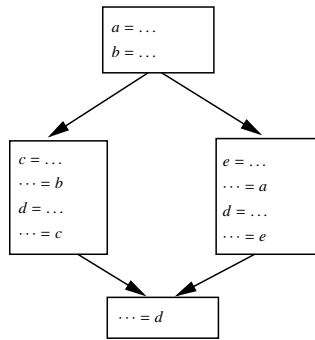
2.1. Colorier avec MAXLIVE registres

“A-t-on assez de registres pour un code donné?”

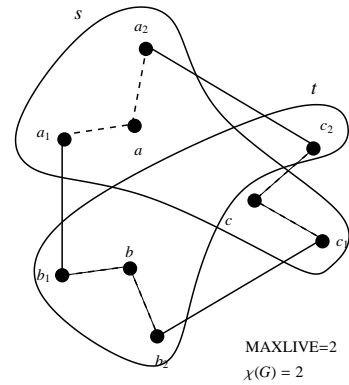
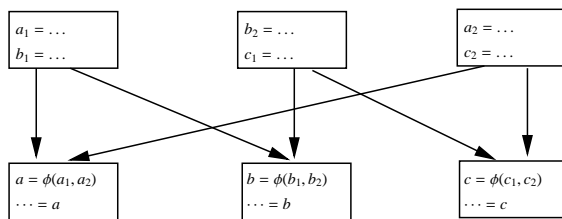
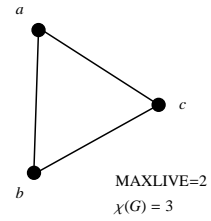
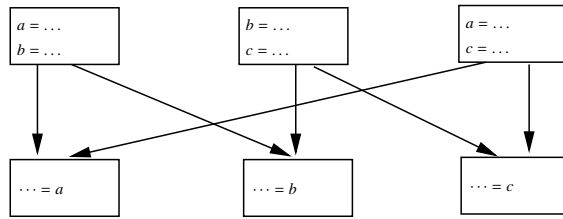
- Chaitin (81) a tout graphe G , on peut construire un code dont le graphe d'interférence est G .



- $\chi(G) \geq MAXLIVE$
- dans le cas général on a pas l'égalité

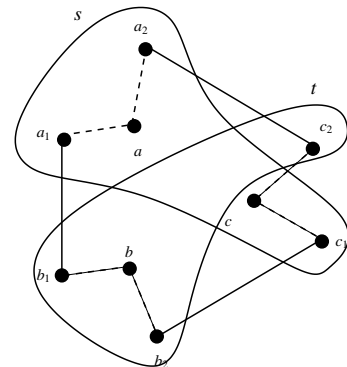
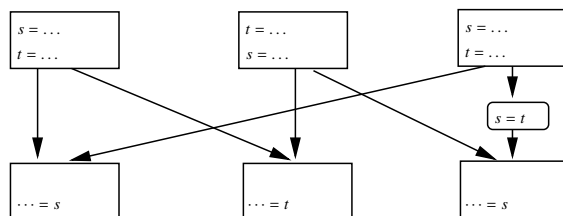


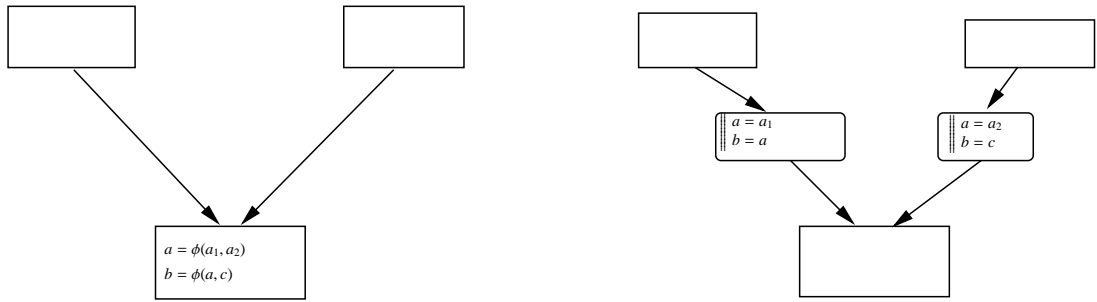
- Un programme sous forme SSA est colorable en MAXLIVE couleurs.



- Car SSA split les variables aux bon endroits: points de fusion.

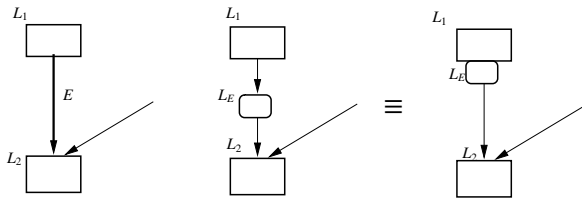
- Mais, la forme SSA n'est pas implémentable. Il faut remplacer les ϕ par des copies (parallèles) sur les arêtes entrantes (en ajoutant éventuellement des blocs):



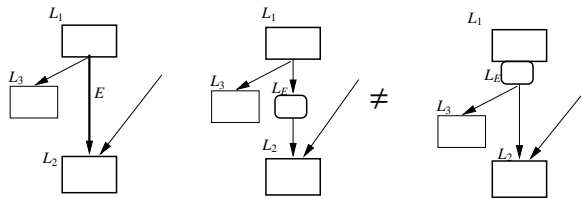


- → Moyennant la possibilité de rajouter des blocs & des copies parallèles, le problème de coloriage avec MAXLIVE registres est un problème polynomial.
- copie parallèle \approx permutation. Eventuellement MAXLIVE+1 registres

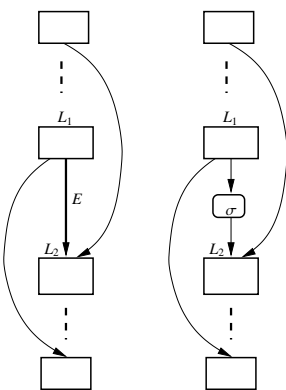
2.1.1. Arc (fortement) critique, arc faiblement critique. Le problème de rajout de blocs. **arc critique:** le prédécesseur a plusieurs successeurs & le successeur a plusieurs prédécesseur: un bloc sur cet arc ne peut être agrégé ni au bloc prédécesseur ni au bloc successeur.



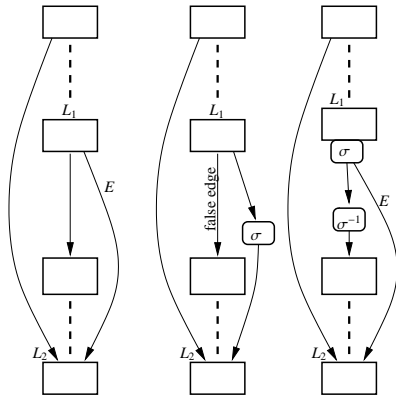
E n'est pas critique: L_1 et L_E peuvent être agrégés.



E est critique: il existe un chemin de L_1 à L_3 ne passant pas par L_E ; L_1 et L_E ne peuvent être agrégés.



arc faiblement critique: l'arc critique E est un "faux" arc qui peut être splitté gratuitement afin d'ajouter la permutation σ .



compensation de permutation: l'arc E est faiblement critique si bien que σ peut être déplacé et compensé localement.

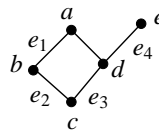
- Pas d'arc fortement critique \implies minimiser nombre de registres, moyennant rajout de copies (nombre notable): polynomial.
- Un code linéarisé sans sauts indirects n'a pas d'arc fortement critique.

2.2. Le problème de coalescing/splitting

2.2.1. NP-complétude du problème de coalescing conservatif P_b : Un graphe d'interférence k -colorable. Un ensemble d'arêtes d'affinité. Coalescer un nombre maximum d'arêtes tq le graphe résultant soit toujours k -colorable.

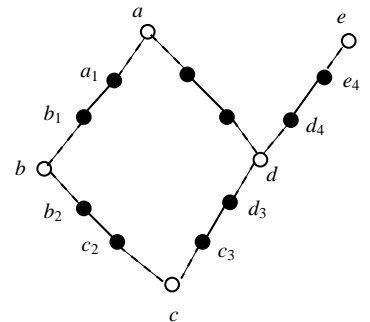
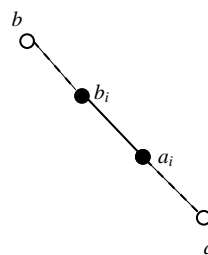
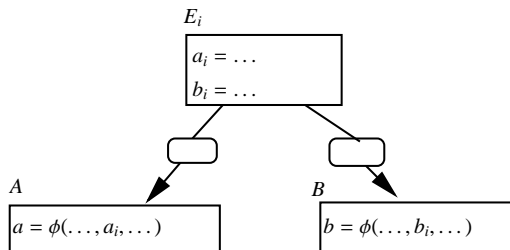
Preuve de NP-complétude:

- Instance k -colorability: graphe G qlque



Est-il k -colorable?

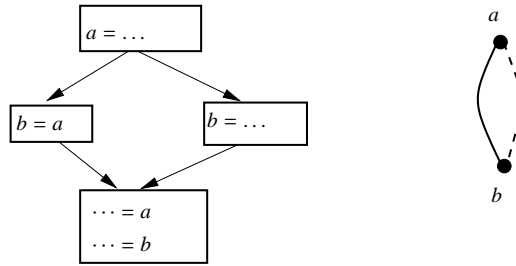
- A chaque arête $e_i = (a, b)$ un bloc E_i définissant a_i et b_i . A chaque sommet a un bloc A définissant a . Successeurs de E_i : A & B . Prédecesseur de A , ensemble des arêtes adjacentes.



Restriction Glouton-KColorabilité: NP-complet. Même preuve, rajout d'une k -clique (le coloriage) reliée par affinité à tous les autres sommets.

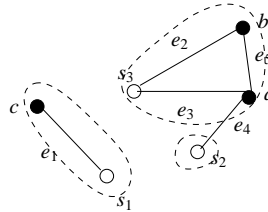
2.2.2. NP-complétude du problème de coalescing agressif P_b : Un graphe d'interférence & d'affinité. Coalescer un maximum d'arêtes sans tenir compte de la colorabilité.

On ne peut pas en général tout coalescer, un coalesce peut en bloquer deux autres:

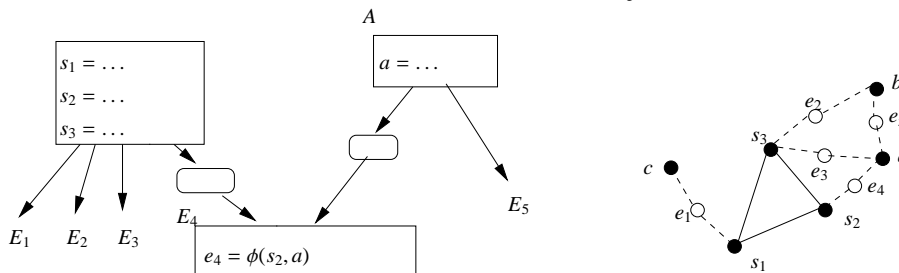


Preuve de NP-complétude:

- Instance de MULTIWAY-CUT: $G = (V, E)$. $S = \{s_1, \dots, s_k\} \subset V$, $k \geq 3$ fixe. Partitionner V en S_1, \dots, S_k tq $s_i \in S_i$; minimiser les nombre d'arêtes entre parties.



- A chaque sommet a un bloc A définissant a . s_i regroupés dans le même bloc. A chaque arête $e_i = (a, b)$, un bloc E_i définissant e_i . Successeurs de A , ensemble des arêtes d'adjacence. Prédécesseur de E_i : A & B .



- Les variables coalescées ensemble correspondent aux parties du partitionnement.

Si on split les variables à la manière du 2.1.1, l'heuristique classique (iterated register coalescing) n'est pas satisfaisante. La partie "difficile" de l'allocation de registre est le coalescing. Il faut porter ses efforts la dessus.

2.3. Le problème du spill

Supposons que l'on ait résolu les problèmes de coloriage et de coalescing, reste le problème de réduire MAXLIVE (+1) au nombre de ressources physiques: c'est le spill.

2.3.1. $\hat{c}(\text{load})=1$ & $\hat{c}(\text{store})=0$ L'algorithme glouton est optimal sur un basic-bloc. L'idée directrice est: "spiller celui utilisé le plus tard". Exple:

code	R1	R2	
$a = \dots$	a	-	
$c = \dots$	a	c	
$b = \dots$	b	c	c est utilisé avant a
$d = c + 1$	b	d	b est utilisé avant c
$e = a + d$	a	d	a & d sont nécessaires, donc a écrase b
	a	e	d n'est plus utilisé
$\dots = b$	a	b	e est utilisé après a
$\dots = a$	a	b	
$\dots = c + e$	c	e	c & e sont nécessaires

2.3.2. $\hat{c}(\text{load})=1$ & $\hat{c}(\text{store})=1$ C'est NP-complet, même sur un BB (réduction à MAX-COVER: Liberatore).

2.3.3. Approche par réduction de noeuds Pb: graphe d'interférence. Pondération: coût de spiller la variable correspondante. Enlever un minimum de sommet tq le graphe résultant soit k-Colorable.

C'est trivialement NP-complet à cause de K-COLORABILITY.

Restriction à Glouton-kColorable Tout problème de réduction de noeuds (node deletion pb) sur un graphe qlque avec une propriété non triviale, héréditaire & intéressante est NP-complet (Reduction à MAX-COVER: Yannikakis).

Notre problème est NP-complet.

Avec des graphes particuliers (SSA...)?

- Le résultat précédant reste vrai sur un graphe planaire!
- Problème ouvert pour un graphe triangulé
- Polynomial sur un graphe de chemins:
 - C : ensemble des points du programme. V : ensemble des variables. $A = (a_{c,v})$ la matrice de cliques: $a_{c,v} = 1$ ssi v est en vie au point c . Si les durées de vie sont splittées afin d'être des chemins du graphe de dominance, A est totalement unimodulaire (graphe d'incidence d'un hypergraphe de chemins).
 - Problème d'optimisation (si on a k registres): n variables, I_n matrice identité. $AX \leq \begin{pmatrix} k \\ \vdots \\ k \end{pmatrix}, I_n X \leq \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, -I_n \leq \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$. p_i le poids de spiller v_i :
maximiser $XP = \sum x_i p_i$.
 - Résolution dans \mathbb{Q}^n (polynomial), solution dans \mathbb{Z}^n . Donc le problème est polynomial.