Compilation : théorie, techniques et outils

Thème 7 : Les représentations intermédiaires

Habib ABDULRAB (Institut National des Sciences Appliquées de Rouen)

Claude MOULIN (Université de Technologie de Compiègne)

Sid TOUATI (Université de Versailles Saint-Quentin en Yvelines)

Table des matières

Objectifs 5

I - Présentation générale 7

A.Portage d'un compilateur.......7

B.Code intermédiaire.......7

II - Codes intermédiaires structurés (graphes) 13

A.Arbre abstrait.......13

B.Graphes Acycliques (DAG).......14

C.Graphe de flot de données.......15

III - Codes intermédiaires linéaire (textuels) 17

A.Forme pré-fixée et post-fixée.......17

B.Code de machine à pile.......18

C.Code trois adresses.......18

D.Forme SSA.......21

IV - Code intermédiaire hybride 23

A.Graphe de flot de contrôle.......23

B.Graphe de dépendances d'un programme.......24

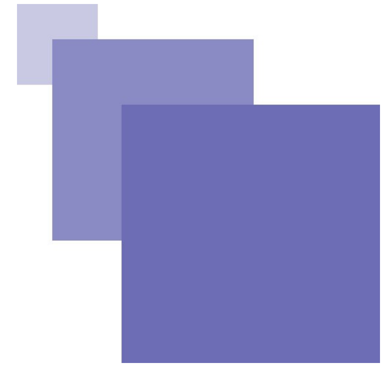
V - Les modèles mémoire 25

A.Les modèles mémoire.......25

B.Le reste de l'histoire.......25

Conclusion 27

Objectifs



Définition des structures et langages intermédiaires utilisés en compilation: structures de graphes (dépendances de données, dépendances de contrôle), formes textuelles (codes 3 adresses, codes 2 adresses, notation préfixées et postfixées, SSA, etc.).

Lecture conseillée :

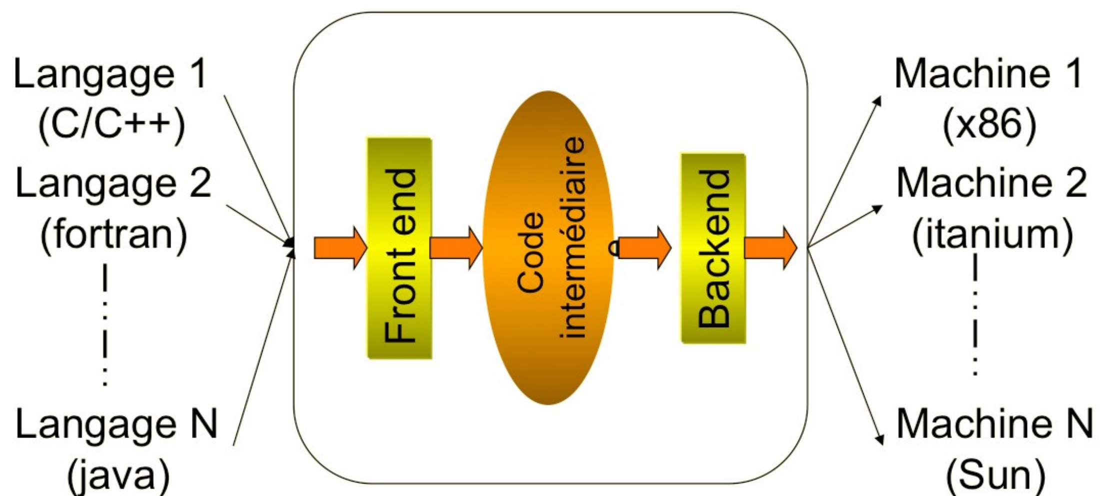
- Aho, Sethi et Ullman : chapitre 8

Présentation générale

I

Portage d'un compilateur 7
Code intermédiaire 7

A. Portage d'un compilateur



B. Code intermédiaire

- La notion d'intermédiaire est floue et ambiguë: intermédiaire entre quoi et quoi? tout est relatif !
- Le langage C peut être considéré comme un langage intermédiaire si on compile des langages de plus haut niveau (orienté objet, fonctionnel, data flow).

Définition

- C'est toute réécriture d'un programme P1 d'un langage L1 vers un autre programme P2 d'un langage L2 tel que :
  - P1 calcule la même chose que P2
  - L2 est plus "près" de la machine cible. En d'autres termes, L2 est un langage "simplifié" par rapport à L1.

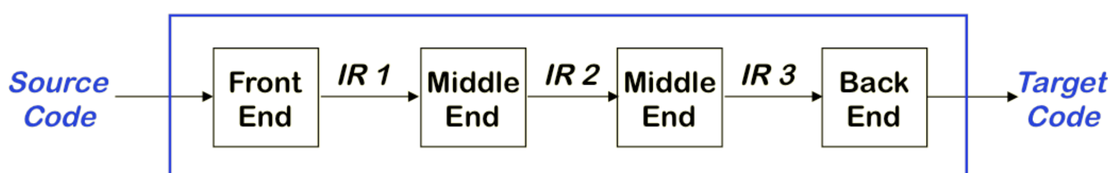
Relativité

- Dans le cas général, les machines exécutent des programmes impératifs (ressemblant au modèle von-Neumann)
- Donc, les représentations intermédiaires dans ce genre de machines doivent être de langage impératif.
- Si la machine est data flow par ex, on aurait choisi une représentation intermédiaire data flow ou fonctionnelle.

Plusieurs couches

- Les compilateurs de nos jours sont devenus très complexes. Plusieurs niveaux de codes intermédiaires peuvent cohabiter.
- Le but est de procéder par étapes successives afin d'optimiser le code intermédiaire avant d'arriver au code binaire de la machine cible.
- On peut voir un compilateur comme une succession de compilateurs en cascade.

Utilisation de plusieurs formes intermédiaires



- Abaisser à plusieurs reprises le niveau d'abstraction de la forme intermédiaire
  - Chaque représentation intermédiaire est appropriée pour une certaine optimisation
- Ex: compilateur Open64
  - forme intermédiaire appelé WHIRL
    - consiste en cinq représentations intermédiaires progressivement détaillées

Complément

Rien n'empêche de générer directement un code natif à partir d'un programme source.

Si le programme source est assez bas niveau, on peut se passer de forme intermédiaire : ex, si on compile du code asm, ou du code 3 adresses.

Le code passé d'une étape à une autre se fait soit par des fichiers intermédiaires, soit par des structures de données en mémoire. En général, on utilise des fichiers intermédiaires car

- le développement compilateurs implique beaucoup d'équipes
- l'interfaçage entre équipes est facilité
- plus facile à débuguer/tracer

Comment choisir ?

- La conception d'une forme intermédiaire affecte l'efficacité et la rapidité d'un compilateur, ainsi que la qualité du programme généré.
- Quelques critères de sélection
  - Facilité de génération
  - Facilité de manipulation
  - Taille de code induite
  - Liberté et puissance d'expression d'informations
  - Niveau d'abstraction
- L'importance de ces critères diffère selon les compilateurs
  - Sélectionner une forme intermédiaire pour un compilateur est une décision de conception importante!

Types des représentations intermédiaires

On peut les classer en trois catégories majeures

- FI structurée
  - Utilise les graphes
  - Beaucoup utilisée dans les traducteurs source à source
  - Facilite une vision abstraite et globale d'un programme
  - Nécessite une présence en mémoire qui peut être large

Exemples : Arbres, DAGs

- FI Linéaire (code textuel)
  - Pseudo-code pour une machine abstraite
  - le niveau d'abstraction varie
  - simple et de taille plus compacte
  - facile à réécrire et manipuler

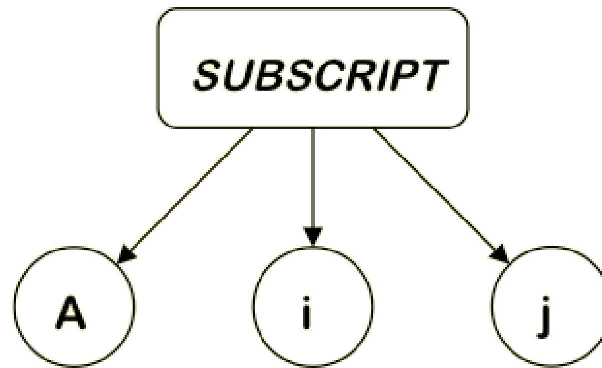
Exemples: Code 3 adresses, code machine à pile

- Hybride
  - Combinaison entre graphes et codes linéaires
  - Je pense que c'est le cas le plus fréquent

Exemple: Graphe de flot de contrôle

Niveau d'abstraction

- Le niveau de détails exposés dans une FI influence la profitabilité et la faisabilité de plusieurs optimisations.
- Ex : deux représentations possibles d'une référence à un élément de tableau A[i,j].



Arbre syntaxique haut niveau :
favorable pour une désambiguation mémoire

loadI	1	=> r_1
sub	r_j, r_1	=> r_2
loadI	10	=> r_3
mult	r_2, r_3	=> r_4
sub	r_i, r_1	=> r_5
add	r_4, r_5	=> r_6
loadI	@A	=> r_7
Add	r_7, r_6	=> r_8
load	r_8	=> r_{Aij}

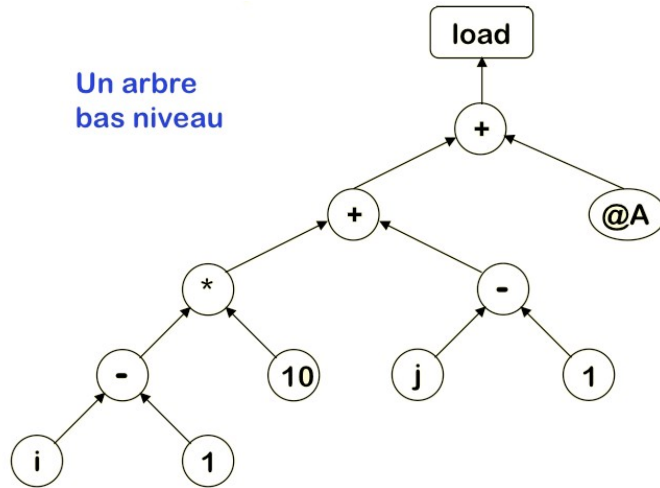
Code linéaire bas niveau:
Favorable pour le calcul d'adresse d'un élément

Complément

désambiguation mémoire : processus effectué par un compilateur pour détecter si deux variables (adresses mémoire) sont distinctes.

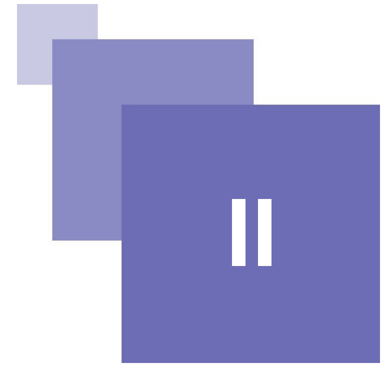
Niveau d'abstraction

- Une FI structurée est souvent considérée comme haut niveau
- Une FI linéaire est souvent considérée bas niveau.
- Ceci n'est pas nécessairement vrai!




```
loadArray A,i,j
```

Code linéaire haut niveau

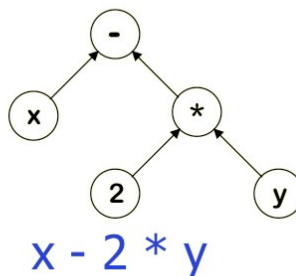


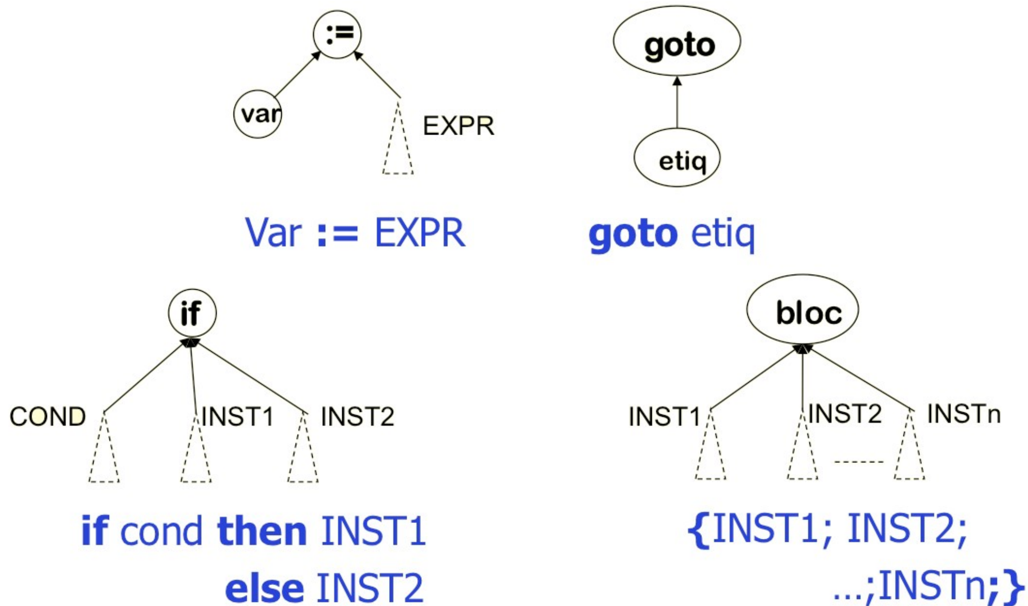
Codes intermédiaires structurés (graphes)

Arbre abstrait	13
Graphes Acycliques (DAG)	14
Graphe de flot de données	15

A.Arbre abstrait

L'arbre abstrait est un arbre syntaxique après avoir enlevé les nœuds des non-terminaux. Les nœuds internes sont les opérateurs, et les feuilles sont les opérandes.





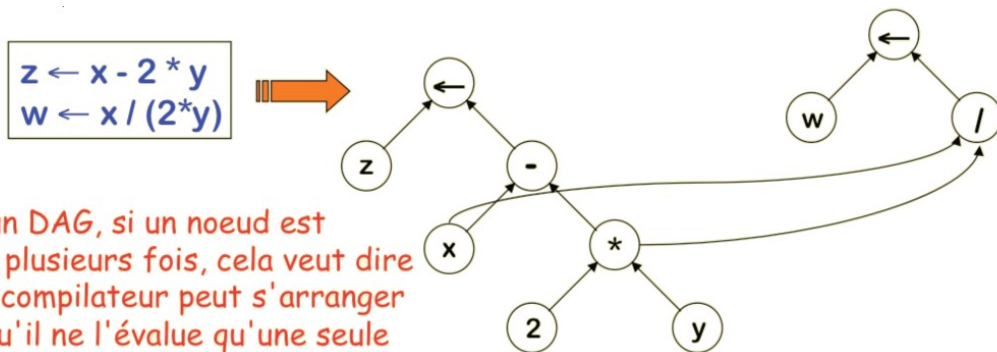
Complément

Les arcs peuvent ne pas être orientés, ou peuvent être orientés de haut en bas.
Une grammaire S-attribuée construit les arcs de bas en haut.

B.Graphes Acycliques (DAG)

Directed Acyclic Graph (DAG)

- C'est une forme optimisée de l'arbre abstrait.
- Chaque valeur calculée a un seul nœud.



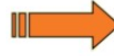
- Réutilisation des sous-expressions communes
- Encode la redondance de calcul

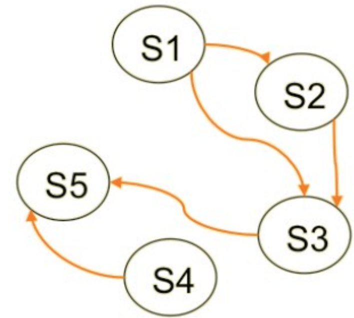
C.Graphe de flot de données

- Les nœuds sont les instructions.
- Un arc de a vers b veut dire que l'instruction a calcule un résultat lu par

l'instruction b

S1: $t_1 \leftarrow x+y$
S2: $t_2 \leftarrow t_1 + 2$
S3: $t_2 \leftarrow t_2 * t_1$
S4: $z \leftarrow$ load x
S5: $z \leftarrow z - t_2$





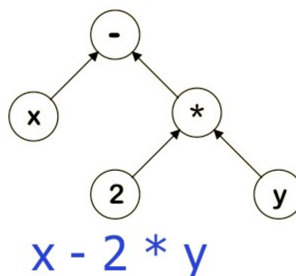
Codes intermédiaires linéaire (textuels)

III

Forme pré-fixée et post-fixée	17
Code de machine à pile	18
Code trois adresses	18
Forme SSA	21

A. Forme pré-fixée et post-fixée

- C'est une linéarisation de l'arbre abstrait. Elles sont définies par un parcours récursif de cet arbre.
- Forme pré-fixée: visiter la racine ensuite le fils gauche suivi du fils droit
  - Ex: - x* 2 y
- Forme post-fixée: visiter le fils gauche suivi du fils droit, en enfin la racine
  - Ex: x 2 y * -



Complément

Appelées également "notations polonaises", car introduites par un logicien polonais

appelé J. Lukasiewicz.

B.Code de machine à pile

Utilisé à l'origine pour des machines n'ayant pas de registres, maintenant utilisé pour java (jvm).

Exemple

x - 2 * y

```
push x
push 2
push y
multiply
subtract
```

Avantages

- Les noms utilisés sont implicites, et non explicites
- Simple à générer et à exécuter.
- Exercice : montrez comment il est facile de générer du code de machine à pile à partir d'une notation post-fixée, et vice-versa.

C.Code trois adresses

Il y a plusieurs représentations d'un code à trois adresses

- En général, les instructions à trois adresses ont la forme:
  $x \longleftarrow y op z$
  avec un seul opérateur (op) et au plus trois opérandes(x, y, z)

Exemple

z ← x - 2 * y

```
t ← 2 * y
z ← x - t
```

Caractéristiques:

- Ce code ressemble à celui de plusieurs machines (de moins en moins vrai).
- Introduit un ensemble de variables temporaires
- Forme compacte

Code trois adresses: Quadruplets

Représentation naïve de code trois adresses

- Table de $k * 4$ entiers

- Structure simple
- Réordonnancer le code plus aisé
- Noms (et numéros de temp) explicites

Le compilateur FORTRAN d'origine utilisait les "quads"

```
load   r1, y
loadI  r2, 2
mult   r3, r2, r1
load   r4, x
sub    r5, r4, r3
```

assembeur RISC

opérateur	dest	src1	src2
load	1	Y	
loadi	2	2	
mult	3	2	1
load	4	x	
sub	5	4	3

Quadruplets

Complément

Cette notation n'est pas optimale : nous n'avons pas toujours trois opérandes.

Code trois adresses : les triplets

- L'indexe de la table est utilisé pour implicitement indiquer les numéros du temporaire contenant le résultat de l'instruction.
- économie de 25% d'espace comparé aux quads (on a éliminé une colonne)
- Plus difficile de réordonnancer le code

load	1	Y	
loadi	2	2	
mult	3	2	1
load	4	X	
sub	5	4	2

Quadruplets

(1)	load	y	
(2)	loadi	2	
(3)	mult	(1)	(2)
(4)	load	x	
(5)	sub	(4)	(2)

Triplets

Complément

Dans les triplets, les numéros (noms) implicites des temporaires destination ne prennent pas de place en mémoire.

Code trois adresses: triplets indirects

- Une optimisation des triplets qui utilise deux tables
  - une table contient une liste de triplets distincts
  - une table qui contient le code consistant en un ensemble de numéros de triplets.
- Ce n'est qu'une compression des triplets
- L'économie d'espace n'est pas au rendez-vous mais il est plus facile de réordonnancer le code.

(100)	load	y	
(101)	loadi	2	
(102)	mult	(100)	(101)
(103)	load	x	
(104)	sub	(103)	(102)

Table des triplets distincts

100	14	166	79	100	45	101	345

Code : numeros des triplets

Triplets vs. Quadruplets

- Le grand compromis entre les triplets et les quadruplet est la compacité versus la facilité de manipulation
  - Dans le passé, le temps de compilation et l'espace utilisé par le compilateur étaient des ressources critiques
  - De nos jours, la vitesse de compilation est un facteur plus important (on tolère plus un compilateur gourmand en espace mémoire)
  - Les triplets et quadruplets peuvent paraître bien simples par rapport à la complexité des architectures/compilateurs actuels

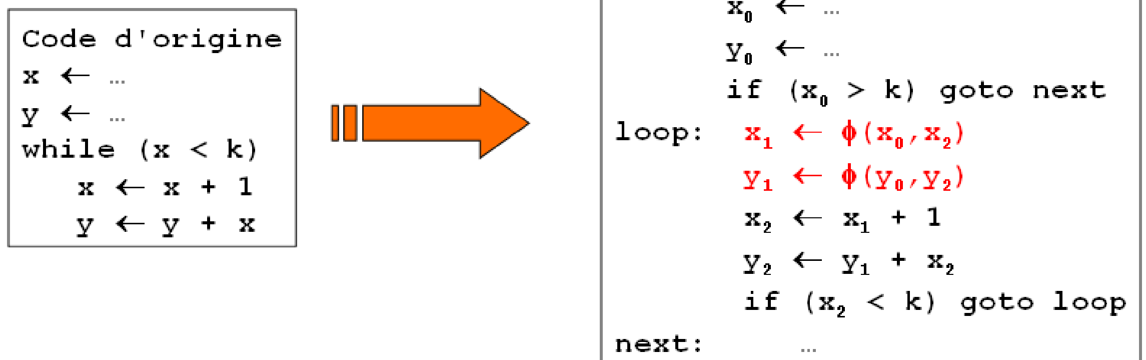
Complément

Mais on tolère moins qu'un compilateur génère des programmes gourmands en mémoire.

D.Forme SSA

Forme SSA : Static Single Assignment

- C'est une représentation textuelle du graphe de flot de données
- L'idée principale: chaque variable est définie/écrite une seule fois
- Utiliser une fonction abstraite appelée ϕ-fonction afin de pouvoir écrire un code en SSA



Caractéristiques

- Décrit le flot de données = sémantique
- Nécessite et permet une analyse de code plus pointue
- (parfois) des algorithmes d'analyse et d'optimisation plus rapides

Complément

La forme SSA renomme les variables et introduit la ϕ-fonction

Dans le cas où le compilateur ne peut pas déterminer le flot exact d'une donnée, il utilise la ϕ-fonction : c'est une fonction abstraite permettant de sélectionner la vraie valeur d'une variable. Cette fonction n'est pas évaluée à la compilation, mais à l'exécution du programme.

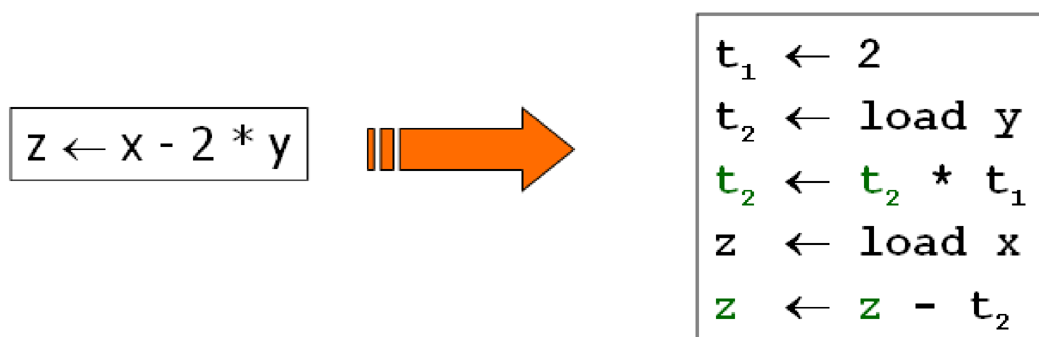
Code à deux adresses

Les instructions sont de la forme

$x \longleftarrow x op y$

Il y a un opérateur (op) et au plus deux opérandes (x et y). L'un d'eux est nécessairement détruit (affecté).

Exemple



Problème

- Beaucoup de machines ne se basent pas sur des opérations destructives (à effet de bord) : machines avec accumulateur
  - Les opérations destructives rendent difficile la réutilisation des temporaires (registres).

Code intermédiaire hybride

IV

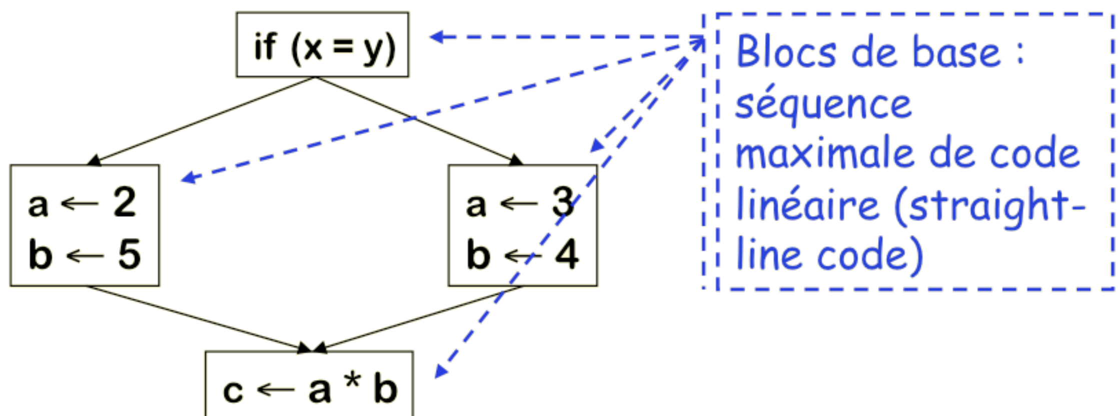
Graphe de flot de contrôle 23

Graphe de dépendances d'un programme 24

A. Graphe de flot de contrôle

Modélise le transfert de contrôle dans un programme/fonction

- Les nœuds représentent les blocs de base
  - Les instructions dans les blocs de bases peuvent être des quads, triplets, ou tout autre représentation intermédiaire
- Les arcs représentent le flot de contrôle (branchements, appels de fonctions, etc.)

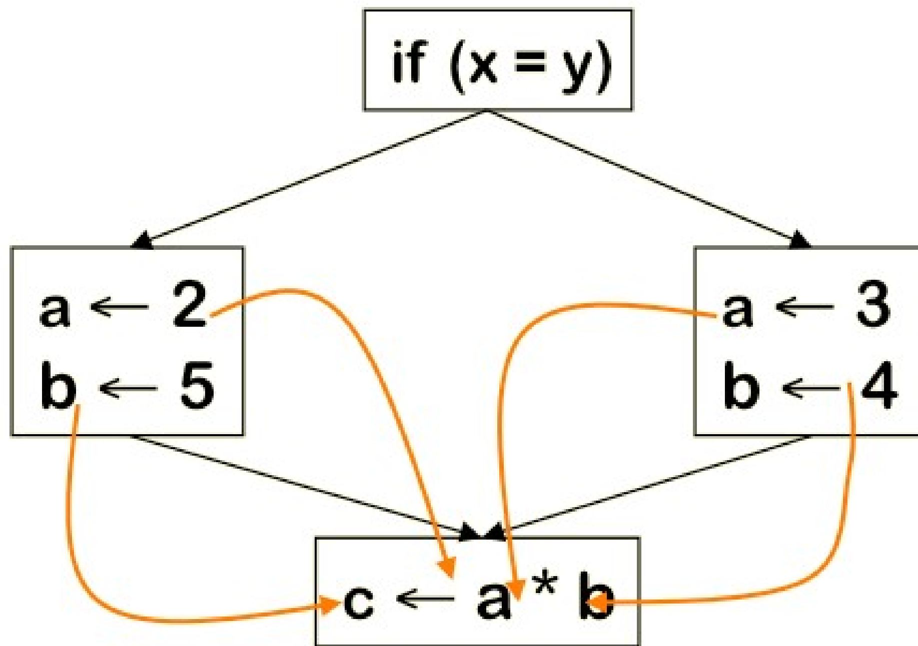


Complément

Les blocs de base dans un CFG peuvent être représentés avec n'importe quelle forme intermédiaire.

B.Graphe de dépendances d'un programme

C'est le graphe de flot de contrôle auquel on ajoute les arcs de dépendances de données.



Il évite de gérer en parallèle un graphe de contrôle et un graphe de dépendances de données

Les modèles mémoire

Les modèles mémoire 25
Le reste de l'histoire 25

A.Les modèles mémoire

On peut citer deux principaux modèles

- Modèle registre-registre
  - Garder en registres toutes les valeurs stockables en registres virtuels
  - Ignorer la limitation du nombre de registres de la machine
  - Le back-end du compilateur effectuera une allocation de registres et introduira si nécessaire du spill code (load et stores)
- Modèle mémoire-registre
  - Garder en mémoire toutes les valeurs
  - Charger les valeurs en registres (accumulateur, etc.) juste avant leur utilisation
  - Le back-end du compilateur tentera d'éliminer les loads et stores redondants.
- Les compilateurs pour machines RISC utilisent le modèle reg-reg
  - Il reflète le modèle de programmation des processeurs RISC
  - Meilleure adéquation entre registres physiques et registres virtuels

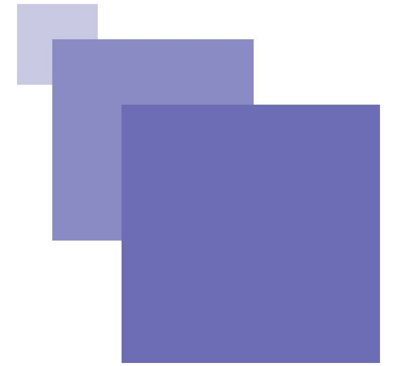
B.Le reste de l'histoire

La représentation du code intermédiaire est seulement une partie de la représentation d'un programme source.

D'autres composants peuvent être nécessaires

- Table des symboles
- Table des constantes
  - Représentation, type
- Schéma de stockage de données
  - Disposition générale de la mémoire allouée (memory layout)
  - Informations de chevauchement
  - Allocation et/ou assignation des registres virtuels sur les registres réels

Conclusion



Les codes intermédiaire ne sont que d'autres programmes sources sensés être plus facilement compilables vers la machine cible. Plus on descend vers la machine, plus on perd des informations sur le programme source. Mais, plus on a des informations susceptibles d'optimiser le programme.