Représentations intermédiaires pour la compilation : s'affranchir du graphe de flot de contrôle

Delphine Demange
Univ Rennes, Inria, CNRS, IRISA
delphine.demange@irisa.fr

15 février 2024 – Collège de France, Paris

Représentations intermédiaires

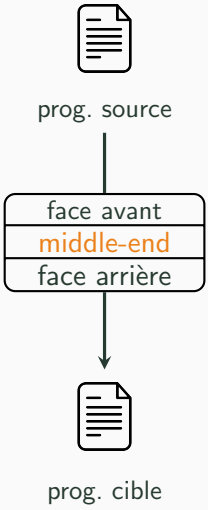
Structure d'un compilateur

- Face avant : du texte à l'AST du langage source
- Face arrière : production de code pour l'architecture cible
- Au milieu : le *middle-end*
  traitements (+/-) indépendants du source et de la cible

Représentations intermédiaires de programmes

- Une étape de la compilation
- Analyses et optimisations nombreuses
- Transformer le programme sans en changer la sémantique

Des langages d'optimisation de programmes



Représentations intermédiaires

Des langages d'optimisation de programmes

Des enjeux sémantiques importants

- Expliciter l'information présente dans le programme
  La rendre accessible à l'analyse et l'optimisation
- Plus qu'une structure de données !
  Reflet fidèle d'un programme, doté d'une sémantique
- Marge de manœuvre sémantique contenue
  Concilier les langages source et cible
- Correction sémantique des transformations

Critère de raffinement sémantique

Maintenu au fur et à mesure des phases de compilation

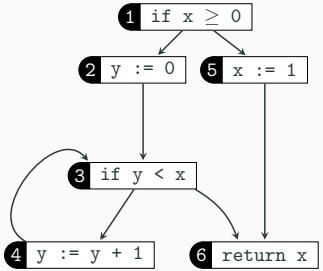
$$[\![P_{src}]\!]_{source} \supseteq [\![P]\!]_{IR} \supseteq [\![P_{opt}]\!]_{IR} \supseteq [\![P_{asm}]\!]_{cible}$$

CFG : Graphe de Flot de Contrôle - Frances E. Allen (1970)

Incontournable dans les compilateurs optimisants de langages impératifs

```
if (x >= 0) then
    y := 0 ;
    while (y < x) do y := y + 1 end
else
    x := 1 ;
return x
```

Programme impératif exemple



Son graphe de flot de contrôle

- Sommets : instructions, arcs : vers le(s) successeur(s) à l'exécution
- Contrôle séquentiel explicite : un sommet, puis son successeur
- Représente tous les chemins possibles d'exécution du programme
- Ici : graphe intraprocédural, où appel de fonction $\simeq$ instruction élémentaire

Compilation : s'affranchir du graphe de flot de contrôle

Notre point de départ

- ∼ **1970** CFG intraprocédural et optimisations
  Écueils : coûts, précision des analyses,
  et rigidité sémantique

Représentations intermédiaires emblématiques

- ∼ **1990** Conçues pour l'optimisation de programmes
- ∼ **2005** Utilisées dans les compilateurs de référence
  Temps long d'application de travaux de recherche

Prisme : vérification formelle de compilateurs

- ∼ **2010** Compréhension sémantique fine récente

CompCertSSA

Compilation vérifiée

Étudier les principes fondamentaux des
techniques et algorithmes utilisés en compilation

Au travers d'une formalisation

- Sémantiques des langages et des représentations
- Propriétes sémantiques, principes de raisonnement
- Souvent dans un assistant de preuve

Dans quel but ?

- Compréhension fine et profonde des techniques du folklore
- Très peu d'angles morts dans les raisonnements
- Renforcer la confiance en les compilateurs, même s'ils sont non-vérifiés

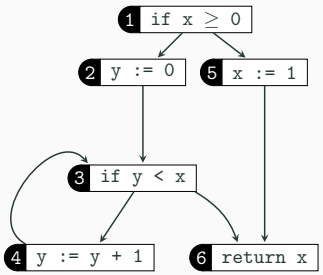
Le graphe de flot de contrôle

CFG : Graphe de Flot de Contrôle

Définition du CFG

$(g : N \hookrightarrow (Instr, [N]), pc_e)$

- Liste des successeurs $[N]$
- Point d'entrée $pc_e \in N$



Sémantique opérationnelle petit-pas

- Compteur ordinal $pc \in N$
- Environnement $\rho : E = Var \hookrightarrow Val$
- Transitions $\rightarrow : (N \times E) \times ((N \times E) + Val)$

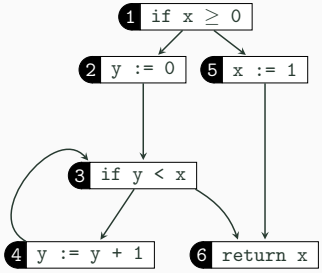
$$\frac{g(pc_1) = (\texttt{goto}, [pc_2])}{(pc_1, \rho) \rightarrow (pc_2, \rho)} \qquad \frac{g(pc_1) = (\texttt{x := e}, [pc_2])}{(pc_1, \rho) \rightarrow (pc_2, \rho[x \mapsto [\![e]\!]\rho])}$$

$$\frac{g(pc_1) = (\texttt{if e}, [pc_t, pc_f]) \quad pc_2 = \begin{cases} pc_t & \text{si } [\![e]\!]\rho = true \\ pc_f & \text{sinon} \end{cases}}{(pc_1, \rho) \rightarrow (pc_2, \rho)} \qquad \frac{g(pc) = (\texttt{return e}, [\,])}{(pc, \rho) \rightarrow [\![e]\!]\rho}$$

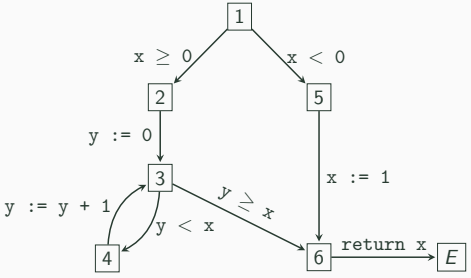
CFG : Variantes employées

Variantes employées et discutées (Knoop, Koschützkil et Steffen, 1998)

- Sommets : 1 instruction VS. 1 bloc d'instructions
- Arcs : relation successeur VS. instruction



CFG - Sommet = 1 instruction



CFG - Arc = 1 instruction / condition

CFG : Analyse de flot de données (Kildall, 1973 ; Kam et Ullman, 1976)

Optimisations du CFG basées sur une analyse de flot de données

Inférer statiquement des ensembles de faits sur valeurs calculées

Ex : variables constantes, variables actives, définitions atteignantes...

Cadre classique, parfois relâché

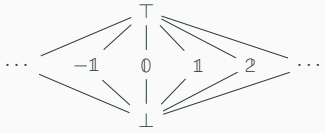
- Approximation : treillis $(D, \leq, \bot, \top, \cup, \cap)$
- Fonctions de transfert des instructions $f_{pc} : D \to D$ pour chaque sommet pc
- Système d'inéquations, inconnues A_{pc}
  $f_{pc}(A_{pc}) \leq A_{pc'}$ pour tout arc (pc, pc')
  $a_e \leq A_{pc_e}$ au sommet d'entrée pc_e
- Algorithme itératif de *work-list*

```
while worklist ≠ empty do
    pc = pick(worklist);
    out_pc = f_pc(A_pc);
    foreach pc' ∈ succs(pc) do
        in_pc' = A_pc' ∪ out_pc;
        if in_pc' ≠ A_pc' then
            A_pc' = in_pc';
            add(worklist, pc')
        end
    end
end
```

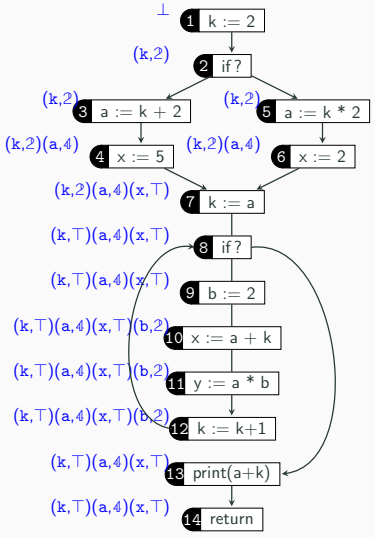
CFG : Analyse de flot de données - Exemple

Analyse de constantes

- Optimisations : propagation de constantes, élimination de code inutile...
- Treillis $\mathbb{C}$ des constantes, relevé à $Var \mapsto \mathbb{C}$



- Constantes "simples" : conditions inexploitées littéraux, et opérations sur constantes simples
- Analyse correcte : invariant sémantique
  Dans un état (pc, ρ), on a $A_{pc}(x) \gtrsim \rho(x)$:
  $$\mathbb{3} \gtrsim 3 \qquad \forall v, \top \gtrsim v \qquad \forall v, \bot \not\gtrsim v$$



Graphe de flot de contrôle - Discussion

Les points fort du CFG

- Sémantique opérationnelle limpide
- Analyse flot de données : avant ou arrière, graphe réductible ou non
  Correction intuitive de l'analyse : suit le modèle d'exécution

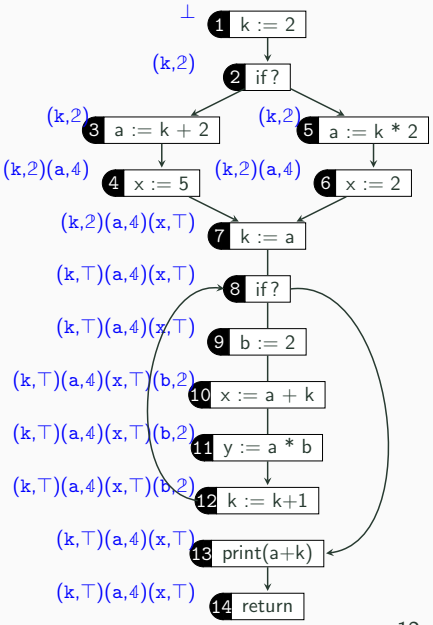
Technique mature et éprouvée

- Permet de réaliser beaucoup d'optimisations !
  propagation de constantes, propagation de copies, analyse de vivacité (allocation de registres), élimination de sous-expression communes, élimination de redondance partielle, loop-invariant code motion, lazy code motion, ordonnancement d'instructions...
- CompCert Compilateur C formellement vérifié (Leroy, 2009)
  Chamois : optimisations sur Block Transfer Language (Gourdin et al., 2023)

Graphe de flot de contrôle - Discussion

Les points faibles du CFG

- Sémantique opérationnelle et déplacements globaux d'instructions
- Analyse flot de données sur le flot de contrôle !
  Propager : chaque sommet, arête, et variable
  Portions de chemins non pertinentes



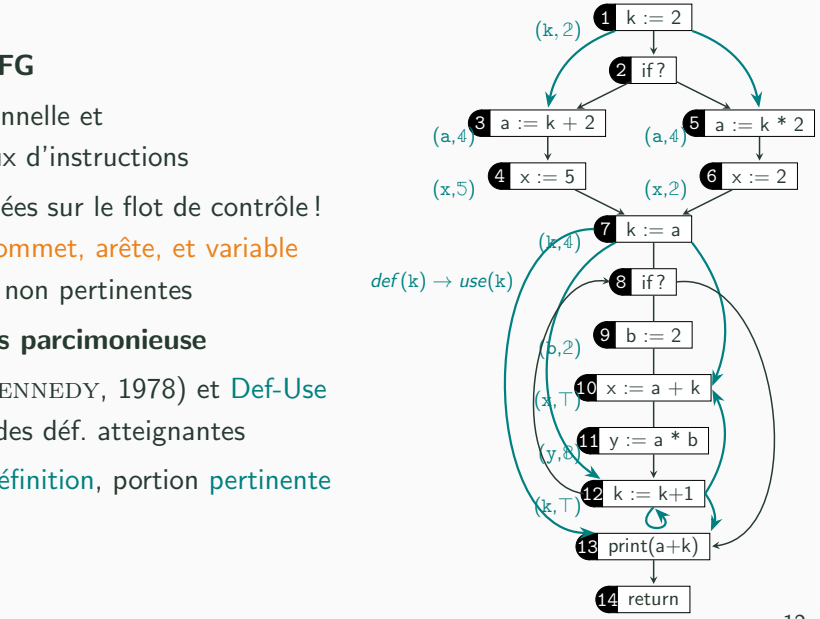
Graphe de flot de contrôle - Discussion

Les points faibles du CFG

- Sémantique opérationnelle et déplacements globaux d'instructions
- Analyse flot de données sur le flot de contrôle ! Propager : chaque sommet, arête, et variable Portions de chemins non pertinentes

Analyse flot de données parcimonieuse

- Chaînes Use-Def (KENNEDY, 1978) et Def-Use Construites à l'aide des déf. atteignantes
- Propager : chaque définition, portion pertinente



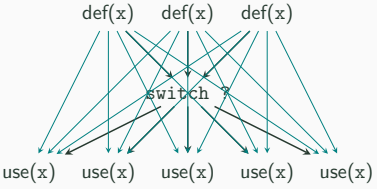
Graphe de flot de contrôle - Discussion

Les points faibles du CFG

- Sémantique opérationnelle et déplacements globaux d'instructions
- Analyse flot de données sur le flot de contrôle !
  Propager : chaque sommet, arête, et variable
  Portions de chemins non pertinentes

Analyse flot de données parcimonieuse

- Chaînes Use-Def (KENNEDY, 1978) et Def-Use
  Construites à l'aide des déf. atteignantes
- Propager : chaque définition, portion pertinente
- Pire cas : m def, n use, $m \cdot n$ Def-Uses
  Mieux factoriser les calculs ?



Forme SSA
(Static Single Assignment)

SSA : Static Single Assignment

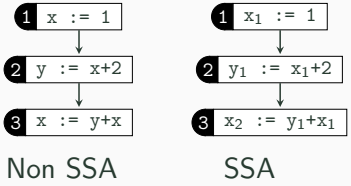
Une petite révolution dans les compilateurs

Par Alpern, Wegman et Zadeck (1988) et Cytron et al. (1991)

Aujourd'hui largement adoptée : GCC, LLVM, JIT Java HotSpot...

SSA : variables ont une unique définition

- **Code linéaire**
  Définition : un nom frais, nº de version
  Utilisation : choisir la dernière version



Non SSA SSA

SSA : Static Single Assignment

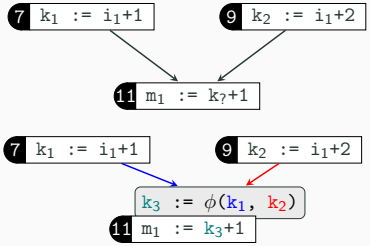
Une petite révolution dans les compilateurs

Par Alpern, Wegman et Zadeck (1988) et Cytron et al. (1991)

Aujourd'hui largement adoptée : GCC, LLVM, JIT Java HotSpot...

SSA : variables ont une unique définition

- **Code linéaire**
  Définition : un nom frais, nº de version
  Utilisation : choisir la dernière version
- **Points de jonction du CFG**
  Quelle version utiliser ? Ça dépend !
  Instruction spéciale : ϕ-fonction
  Sélectionne l'argument selon l'arc exécuté



SSA : Static Single Assignment

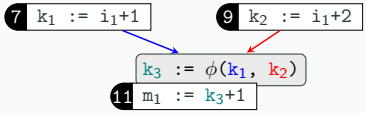
Une petite révolution dans les compilateurs

Par Alpern, Wegman et Zadeck (1988) et Cytron et al. (1991)

Aujourd'hui largement adoptée : GCC, LLVM, JIT Java HotSpot...

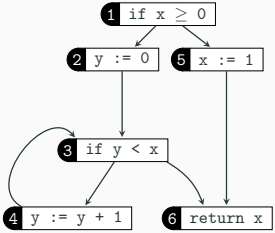
SSA : variables ont une unique définition

- **Code linéaire**
  Définition : un nom frais, nº de version
  Utilisation : choisir la dernière version
- **Points de jonction du CFG**
  Quelle version utiliser ? Ça dépend !
  Instruction spéciale : ϕ-fonction
  Sélectionne l'argument selon l'arc exécuté

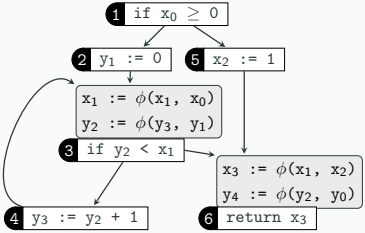


$$\begin{pmatrix} g : N \hookrightarrow (Instr, [N]) \\ \varphi : N \hookrightarrow [Instr_\phi] \\ pc_e : N \end{pmatrix}$$

SSA : Placement des ϕ-fonctions



CFG non SSA

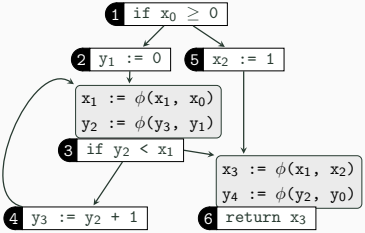


SSA maximale

Toutes les variables

Tous les points de jonction

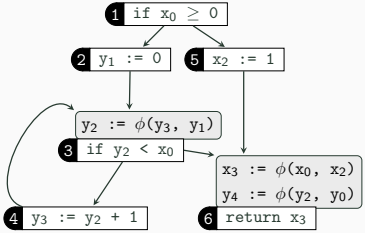
SSA : Placement des ϕ-fonctions



SSA maximale

Toutes les variables

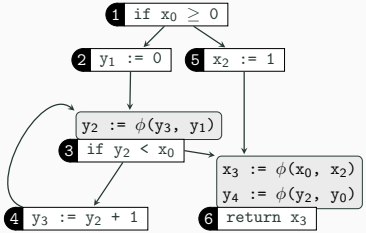
Tous les points de jonction



SSA minimale

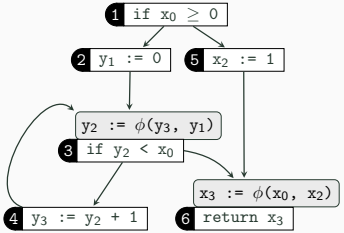
Frontières de dominance

SSA : Placement des ϕ-fonctions



SSA minimale

Frontières de dominance



SSA élaguée

SSA minimale + vivacité

SSA : Sémantique ? !

1991 SSA encore vue comme une structure de données

"The operands to the ϕ-function indicate which assignments to x reach the join point." (Cytron et al., 1991)

2009 Un verrou en compilation vérifiée

"Since the beginning of CompCert we [...] were held off by two difficulties. First, the dynamic semantics for SSA is not obvious to formalize. Second, the SSA property is global to the code of a whole function and not straightforward to exploit locally within proofs." (Leroy, 2009)

2012 Génération spontanée d'efforts de formalisation

- Sémantique opérationnelle simple, en accord avec l'intuition
- Lemme équationnel : les définitions sont des équations
- Premières vérifications d'optimisations :
  Sparse Conditional Constant Propagation, *Global Value Numbering*, *Promote Memory to Register* (`mem2reg`)

CompCertSSA

SSA : Sémantique (Barthe, Demange et Pichardie, 2014)

Sémantique des instructions

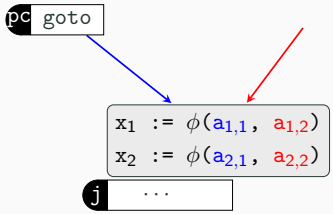
Exactement celles du CFG sans SSA, quand le successeur n'est pas un point de jonction

Sémantique d'un bloc de ϕ-fonctions

- Exécuter l'instruction en *pc*
- Puis les ϕ-fonctions au successeur
- Les ϕ-fonctions sont non-strictes

Points notables

- État : sans trace du prédécesseur
- Évaluation parallèle des ϕ-fonctions
- Seul `goto` précède un point de jonction (coupe des arcs critiques)



$$g(pc) = (\texttt{goto}, [j])$$

$$\varphi(j) = \begin{cases} \texttt{x}_1 := \phi(\texttt{a}_{1,1}, \texttt{a}_{1,2}) \\ \cdots \\ \texttt{x}_n := \phi(\texttt{a}_{n,1}, \texttt{a}_{n,2}) \end{cases}$$

pc est le k^e prédecesseur de j

$$\frac{\rho' = \rho[\texttt{x}_1 \mapsto \rho(\texttt{a}_{1,k}); \ldots; \texttt{x}_n \mapsto \rho(\texttt{a}_{n,k})]}{(pc, \rho) \to (j, \rho')}$$

SSA : Propriétés fondamentales (Barthe, Demange et Pichardie, 2014)

Deux ingrédients de base pour raisonner sur la sémantique et les optimisations.

Propriété (SSA Forme Stricte)
Toute utilisation d'une variable x définie en pc_x est strictement dominée par pc_x.
(un ϕ-argument est virtuellement utilisé au prédécesseur.)

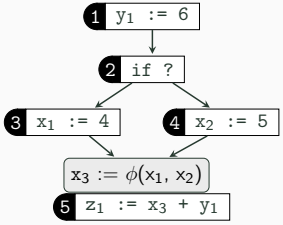
Lemme (Lemme équationnel)
À toute instruction $g(pc_x) = (x := e, [\_])$ correspond une égalité de valeurs entre x et e, dans la région du graphe strictement dominée par pc_x.

$$\forall pc_x, x, e, pc, \rho. \quad \left.\begin{array}{r} g(pc_x) = (x := e, [\_]) \\ (pc, \rho) \text{ état atteignable} \\ pc_x \text{ sdom } pc \end{array}\right\} \Rightarrow \rho(x) = [\![e]\!]\rho$$

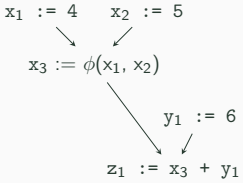
Cadre d'analyse parcimonieuse SSA

(Wegman et Zadeck, 1985 ; Novillo, 2005)

- Treillis des approximations $(D, \bot, \top, \cup, \cap)$
- Deux informations calculées : $(exec, avars)$
  - Insensible au flot $\quad avars : Var \mapsto D$
  - Exécutabilité des arcs $\quad exec : \mathcal{P}(N)$
- Algorithme : deux *work-lists*
  Def-Use SSA + arcs du CFG
- Accélérer la convergence : prioriser les $\mathtt{x}$ telles que $avars(\mathtt{x}) = \top$



CFG SSA



Def-Use SSA

SSA : Analyse flot de données parcimonieuse - Correction

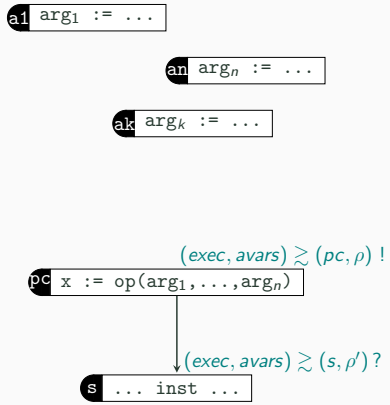
(Demange, Pichardie et Stefanesco, 2015)

Invariant de correction sémantique $(exec, avars) \gtrsim (pc, \rho)$

$\forall \mathtt{x}, pc \in exec \wedge def(\mathtt{x})\ sdom\ pc \Rightarrow avars(\mathtt{x}) \gtrsim \rho(\mathtt{x})$

Preuve de correction :

1. Supposons $(exec, avars) \gtrsim (pc, \rho)$
2. SSA stricte : $def(\mathtt{arg}_k)\ sdom\ pc$
3. D'après 1., en pc : $avars(\mathtt{arg}_k) \gtrsim \rho(\mathtt{arg}_k)$
4. Fonction transfert f_{pc} prouvée correcte avec le lemme équationnel
5. Au point s :
   Correction pour $\mathtt{x}$: $avars(\mathtt{x}) \gtrsim \rho'(\mathtt{x})$,
   Préservation pour $\mathtt{y} \neq \mathtt{x}$, par trans. de dom
6. On conclut $(exec, avars) \gtrsim (s, \rho')$



SSA : Bilan

Un tournant important en compilation : nommage + ϕ-instructions

- Contourne certains écueils du CFG originel
- Apports considérables pour l'optimisation
  Simplification conceptuelle et performances
- Sémantique formelle aujourd'hui bien comprise
- Lemme équationnel : raisonner localement sur les valeurs

Et pourtant... SSA reste basée sur un CFG

- Les ϕ-fonctions sont dirigées par le contrôle
- Sém. opér. et déplacements globaux d'instructions
- Et l'analyse de dépendances d'instructions ?

Rastello et Bouchez-Tichadou, 2022

Sea-of-Nodes

Sea-of-Nodes

Proposée par Click et Paleczny (1995)

Java HotSpot Server C2, Graal Oracle, LibFirm, V8...

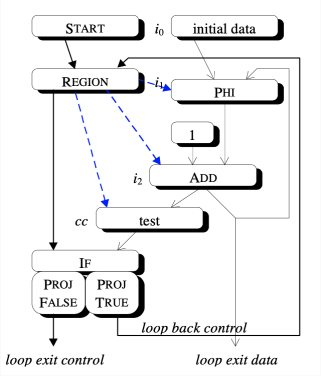
Gommer les dépendances syntaxiques

Graphe de dépendances

- Données, contrôle et effets
- Basée sur la forme SSA

Optimisations

- Code inutile : sous-graphe isolé
- Normalisation, réécriture sous-graphes
- Élimination redondances : partage
- Ordonnancement de code : en sortie !



©Click, Paleczny

Sea-of-Nodes : Exemple (simplifié)

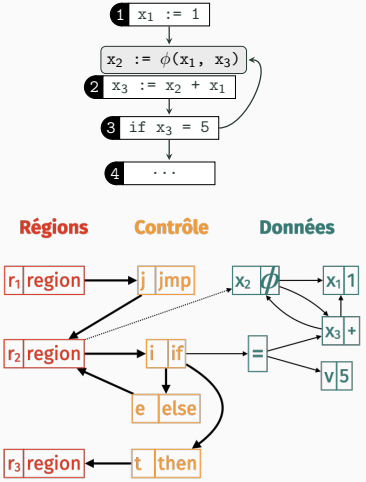
Catégories de noeuds

- Régions : ossature du contrôle
- Contrôle : flot d'exécution entre régions
- Données : calcul de valeurs
  Graphe de dépendances
  Flottent librement, sauf les noeuds ϕ

Variantes rencontrées

- Noeuds : boucles, aiguillage...
- Information dynamique : contexte, déoptimisation...
- Orientation des arcs

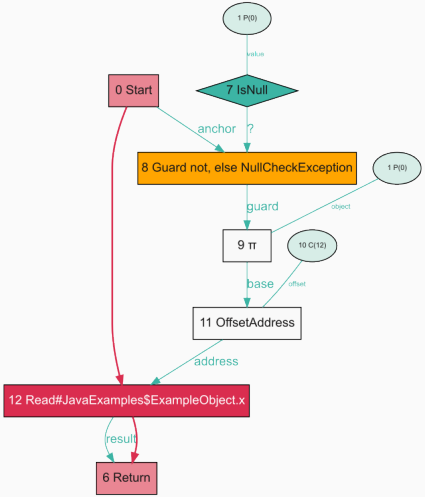
Modèles d'exécution informels



Sea-of-Nodes : Exemple (spécimen réel)

Graal IR - Lecture d'un champ objet en Java
(Duboscq et al., 2013)

- Après quelques phases de compilation
- Chargement mémoire et calcul d'offset
- Exceptions, vérifications de bornes, nullité
- Informations typage et analyse (*stamps*)
- Pas de noeuds région stricto sensu



© Chris Seaton – Oracle

Sea-of-Nodes : Sémantique (Demange, Retana et Pichardie, 2018)

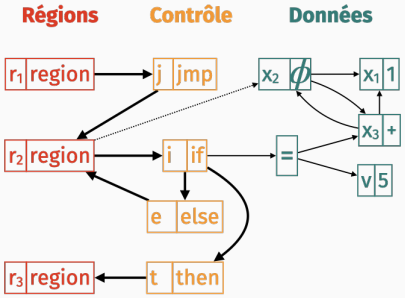
Données : évaluer le sous-graphe

- Noeuds ϕ : cycles de dépendance
- Environnement $\rho : \phi_{id} \mapsto \mathit{Val}$

$$[\![x]\!]\rho = \begin{cases} c & \text{si } g(x) = \texttt{cst } c \\ [\![\texttt{n}_1]\!]\rho \ \mathit{op} \ [\![\texttt{n}_2]\!]\rho & \text{si } g(x) = \texttt{op n}_1 \ \texttt{n}_2 \\ \rho(x) & \text{si } g(x) = \phi \ \_ \end{cases}$$

Contrôle : d'une région à la suivante

- Sémantique opérationnelle à la SSA
  État d'exécution (r, ρ)
- Env. ρ mis à jour sur les arcs
- Noeuds données évalués "au besoin"



Sea-of-Nodes : Propriétes sémantiques (Demange, Retana et Pichardie, 2018)

Optimisations globales

Dominance entre régions et ϕ-dépendances des noeuds de données

Lemme (Valeur définie)
Dans un état atteignable (r, ρ), un noeud de donnée est évaluable si toutes ses ϕ-dépendances dominent r.

Lemme (Préservation de valeurs)
La valeur d'un noeud de donnée est préservée dans les régions dominées par ses ϕ-dépendances.

Preuve de correction d'une optimisation importante

Élimination de vérifications redondantes (*nullchecks unzipping*)

Du contrôle... aux données

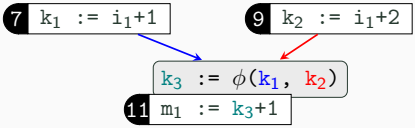
SSA : Retour sur les ϕ-fonctions

SSA : sémantique des ϕ-fonctions

- Dépend du flot de contrôle
- Sélection de l'argument : critère implicite et global

Gated SSA : ϕ-fonctions prédiquées

- Sélection de l'argument : prédicats
- Critère explicite et local
- Conversion dép. de contrôle en dép. de données



Gated SSA : Instructions spécifiques

Gated SSA = mieux expliciter dépendances de données et contrôle

Point de jonction simple : ϕ + prédicats

Les prédicats p_i discriminent les x_i $\quad x \leftarrow \gamma((p_1, x_1), (p_2, x_2), \ldots (p_n, x_n))$

Sémantique locale guidée par les données

Point de jonction en tête de boucle : ϕ + contrôle explicite

Aucun prédicat ne peut distinguer démarrage et itérations ! $\quad x_\mu \leftarrow \mu(x_{init}, x_{loop})$

Boucles imbriquées et réinitialisation

Sémantique reste guidée par le contrôle

Sortie de boucle : dépendances explicites

Distinguer usages dans boucle (x_μ) et hors de boucle (x_η) $\quad x_\eta \leftarrow \eta(p, x_\mu)$

Prédicat p : x_μ a atteint sa valeur finale, après dernière itération

Gated SSA : État de l'art

Nombreuses variantes

Program Dependence Graph (Ferrante, Ottenstein et Warren, 1987), Program Dependence Web (Ottenstein, Ballance et MacCabe, 1990), Gated SSA (Tu et Padua, 1995a ; Tu et Padua, 1995b), Thinned Gated SSA (Havlak, 1994)

Applications variées

- Optimisations standards (Arenaz, Amoedo et Touriño, 2008)
- Validation d'optimisation (Tristan, Govereau et Morrisett, 2011)
- Analyse de divergence pour SIMD (Sampaio et al., 2012)
- Synthèse de haut-niveau (Derrien et al., 2020 ; Elakhras et al., 2022)

Principales difficultés en compilation vérifiée

- Définition et construction des prédicats
- Sémantique formelle adaptée
- Correction sémantique de la construction

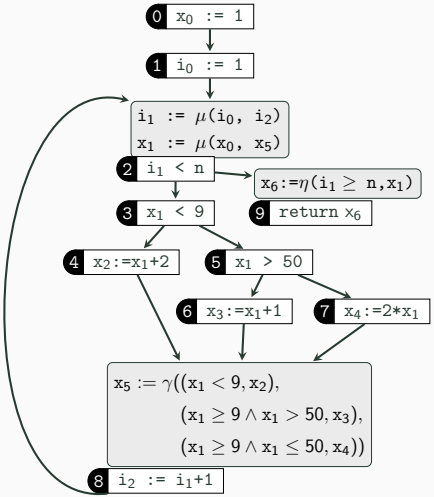
Gated SSA : Exemple et difficultés

Caractéristiques

- CFG est préservé
- Tête de boucle : ϕ devient μ
- Sortie de boucle : ajout des η, renommage
- Points de jonction simples : ϕ devient γ

Prédicats syntaxiques

- Arguments des γ ne sont pas ordonnés
- Conditions doivent être “préservées”
- La dominance ne suffit pas toujours !

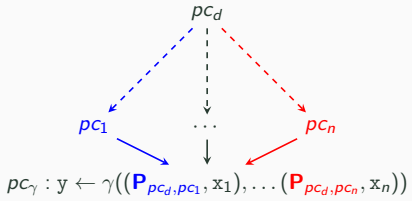


Gated SSA : Correction des prédicats (Herklotz, Demange et Blazy, 2023)

Construction des prédicats (Tarjan, 1981)
Single-source Path-Expression Problem

Sémantique des prédicats

- Cohérence vis-à-vis du CFG :
  Au moins un des prédicats satisfait
- Exclusion mutuelle :
  Ils distinguent les arguments
- Chemins non-exécutés, var. indéfinies :
  Interprétés en logique 3 valeurs
- Cohérence + exclusivité = validité !
  Correction sémantique : $[\![P_{gsa}]\!] \subseteq [\![P_{ssa}]\!]$



En pc_γ, expressions de chemins

- entre pc_d dominant pc_γ
- et préd. immédiats de pc_γ

+ l'arc (pc_i, pc_γ), si besoin

Du flot de contrôle... au flot de données

Gated SSA et Program Dependence Web :

- Parallèlisation de code
- Compilation pour les architectures flots de données

Un pont entre différents modèles d'exécution ?

> *There is the control-driven von Neumann massively parallel camp. There is the data-driven dataflow machine camp. And there is the demand-driven graph reduction machine camp. [...] Moving computations between camps is difficult. The Program Dependence Web helps to eliminate this barrier.*
> *(*Campbell, Krishna *et* Ballance, *1993)*

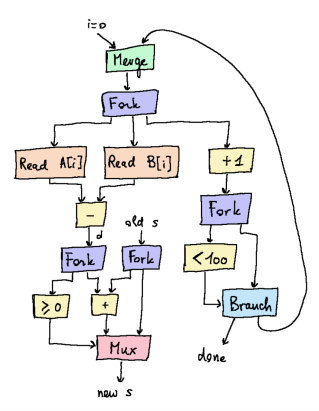
Du flot de contrôle... au flot de données

Application récente du PDW : synthèse de haut-niveau (Elakhras et al., 2022)

- Programmes impératifs vers circuits élastiques
- Ordonnancement dynamique, dirigé par les données
- Protocole de négociation entre composants

Travaux de recherche en cours

- Basé sur notre formalisation Gated SSA
- Représentation intermédiaire des circuits élastiques
- Preuve de correspondance sémantique



©Tony Law – IRISA

Conclusions

Compilation : s'affranchir du graphe de flot contrôle

Représentations intermédiaires en compilation

- Langages impératifs : base historique du CFG
- Enrichi selon besoins d'analyses et optimisations
  Grandes familles : SSA, graphes de dépendances
- Représenter syntaxiquement dép. sémantiques analysées
  Y compris celles de contrôle : relâcher des contraintes
- Enjeu fort : rester "exécutable"

En compilation vérifiée

- Progrès considérables ces 15 dernières années
  Langages impératifs, fonctionnels, concurrents, réactifs...
- Formalisation des représentations intermédiaires
  Des jalons plantés (CFG, SSA, ...) et d'autres en ligne de mire !

Représentations intermédiaires pour la compilation : s'affranchir du graphe de flot de contrôle

Delphine Demange
Univ Rennes, Inria, CNRS, IRISA
delphine.demange@irisa.fr

15 février 2024 – Collège de France, Paris

Références bibliographiques

Références bibliographiques (1/6)


Allen, F. E. (1970). **"Control flow analysis".** In : *Proceedings of a Symposium on Compiler Optimization*, 1–19.


Alpern, B., M. N. Wegman et F. K. Zadeck (1988). **"Detecting equality of variables in programs".** In : *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '88, 1–11.


Arenaz, M., P. Amoedo et J. Touriño (2008). **"Efficiently Building the Gated Single Assignment Form in Codes with Pointers in Modern Optimizing Compilers".** In : *Euro-Par 2008 – Parallel Processing*. Sous la dir. d'E. Luque, T. Margalef et D. Benítez, p. 360-369.


Barthe, G., D. Demange et D. Pichardie (2014). **"Formal Verification of an SSA-Based Middle-End for CompCert".** In : *ACM Trans. Program. Lang. Syst.* 36.1.


Campbell, P. L., K. Krishna et R. A. Ballance (1993). **"Refining and defining the program dependence web".** In : *Cs93-6, University of New Mexico, Albuquerque*.

Références bibliographiques (2/6)

- Click, C. et M. Paleczny (1995). **"A simple graph-based intermediate representation".** In : *Papers from the 1995 ACM SIGPLAN Workshop on Intermediate Representations*. IR '95, 35–49.
- Cytron, R. et al. (1991). **"Efficiently computing Static Single Assignment form and the Control Dependence Graph".** In : *ACM Trans. Program. Lang. Syst.* 13.4, 451–490.
- Demange, D., D. Pichardie et L. Stefanesco (2015). **"Verifying Fast and Sparse SSA-Based Optimizations in Coq".** In : *Compiler Construction*. Sous la dir. de B. Franke, p. 233-252.
- Demange, D., Y. Fernández de Retana et D. Pichardie (2018). **"Semantic reasoning about the Sea of Nodes".** In : *Proceedings of the 27th International Conference on Compiler Construction*. CC 2018, 163–173.
- Derrien, S. et al. (2020). **"Toward Speculative Loop Pipelining for High-Level Synthesis".** In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11, p. 4229-4239.

Références bibliographiques (3/6)

- Duboscq, G. et al. (2013). **"An intermediate representation for speculative optimizations in a dynamic compiler".** In : *Proceedings of the 7th ACM Workshop on Virtual Machines and Intermediate Languages*. VMIL '13, 1–10.
- Elakhras, A. et al. (2022). **"Unleashing Parallelism in Elastic Circuits with Faster Token Delivery".** In : *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*, p. 253-261.
- Ferrante, J., K. J. Ottenstein et J. D. Warren (1987). **"The Program Dependence Graph and Its Use in Optimization".** In : *ACM Trans. Program. Lang. Syst.* 9.3, 319–349.
- Gourdin, L. et al. (2023). **"Formally Verifying Optimizations with Block Simulations".** In : *Proc. ACM Program. Lang.* 7.OOPSLA2.
- Havlak, P. (1994). **"Construction of Thinned Gated Single-Assignment form".** In : *Languages and Compilers for Parallel Computing*. Sous la dir. d'U. Banerjee et al., p. 477-499.

Références bibliographiques (4/6)


Herklotz, Y., D. Demange et S. Blazy (2023). **"Mechanised Semantics for Gated Static Single Assignment".** In : *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2023, 182–196.


Kam, J. B. et J. D. Ullman (1976). **"Global Data Flow Analysis and Iterative Algorithms".** In : *J. ACM* 23.1, 158–171.


Kennedy, K. (1978). **"Use-definition chains with applications".** In : *Computer Languages* 3.3, p. 163-179.


Kildall, G. A. (1973). **"A unified approach to global program optimization".** In : *Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL '73, 194–206.


Knoop, J., D. Koschützkil et B. Steffen (1998). **"Basic-block graphs : Living dinosaurs ?"** In : *Compiler Construction*. Sous la dir. de K. Koskimies, p. 65-79.

Références bibliographiques (5/6)

- Leroy, X. (2009). **"A formally verified compiler back-end".** In : *Journal of Automated Reasoning* 43.4, p. 363-446.
- Novillo, D. (2005). **"A propagation engine for GCC".** In : *Proceedings of the 2005 GCC Developers Summit*, p. 175-184.
- Ottenstein, K. J., R. A. Ballance et A. B. MacCabe (1990). **"The Program Dependence Web : A Representation Supporting Control-, Data-, and Demand-Driven Interpretation of Imperative Languages".** In : *Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation*. PLDI '90, 257–271.
- Rastello, F. et F. Bouchez-Tichadou, éd. (2022). **SSA-based Compiler Design.**
- Sampaio, D. et al. (2012). **"Divergence Analysis with Affine Constraints".** In : *2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing*, p. 67-74.
- Tarjan, R. E. (1981). **"Fast Algorithms for Solving Path Problems".** In : *J. ACM* 28.3, 594–614.

Références bibliographiques (6/6)


Tristan, J.-B., P. Govereau et G. Morrisett (2011). **"Evaluating value-graph translation validation for LLVM".** In : *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '11, 295–305.


Tu, P. et D. Padua (1995a). **"Efficient Building and Placing of Gating Functions".** In : *Proceedings of the ACM SIGPLAN 1995 Conference on Programming Language Design and Implementation*. PLDI '95, 47–55.


— (1995b). **"Gated SSA-Based Demand-Driven Symbolic Analysis for Parallelizing Compilers".** In : *Proceedings of the 9th International Conference on Supercomputing*. ICS '95, 414–423.


Wegman, M. N. et F. K. Zadeck (1985). **"Constant propagation with conditional branches".** In : *Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL '85, 291–299.