Optimisations de code dans GCC

Sébastian Pop

Université Louis Pasteur

Strasbourg

Introduction

`GCC : GNU Compiler Collection`

- Un ensemble de compilateurs : C, C++, Java, Ada, Frotran, Mercury, ...

Introduction

`GCC : GNU Compiler Collection`

- Un ensemble de compilateurs : C, C++, Java, Ada, Frotran, Mercury, . . .
- Génération de code pour 43 architectures : i386, ia64, m68k, sparc, . . .

Introduction

`GCC : GNU Compiler Collection`

- Un ensemble de compilateurs : C, C++, Java, Ada, Frotran, Mercury, . . .
- Génération de code pour 43 architectures : i386, ia64, m68k, sparc, . . .
- Plus de 1 million de lignes de code.

Introduction

`GCC : GNU Compiler Collection`

- Un ensemble de compilateurs : C, C++, Java, Ada, Frotran, Mercury, . . .
- Génération de code pour 43 architectures : i386, ia64, m68k, sparc, . . .
- Plus de 1 million de lignes de code.
- Clef de voûte des logiciels GNU.

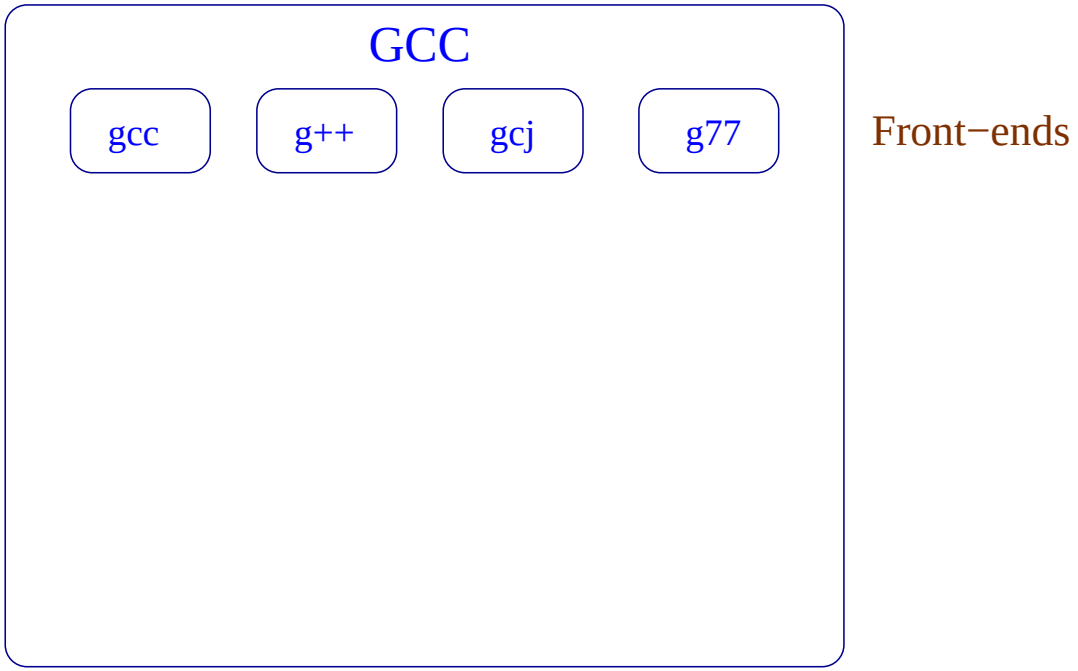
Introduction

`GCC : GNU Compiler Collection`

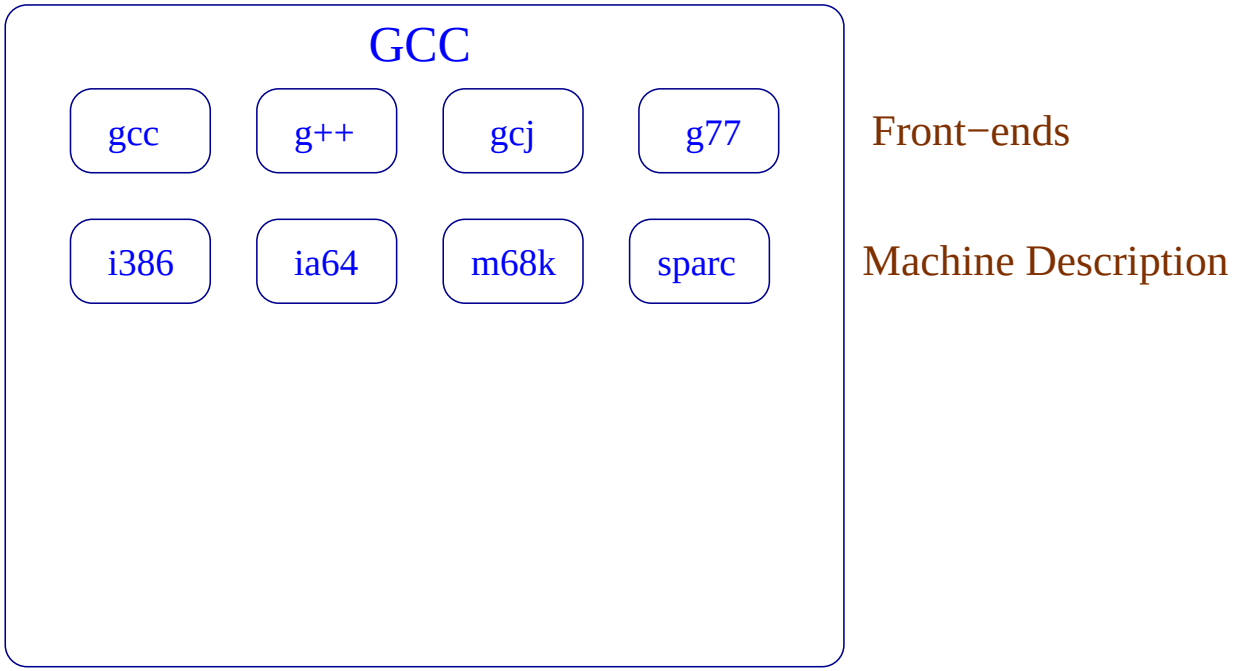
- Un ensemble de compilateurs : C, C++, Java, Ada, Frotran, Mercury, . . .
- Génération de code pour 43 architectures : i386, ia64, m68k, sparc, . . .
- Plus de 1 million de lignes de code.
- Clef de voûte des logiciels GNU.
- Compilateur industriel.

Front-ends / back-end

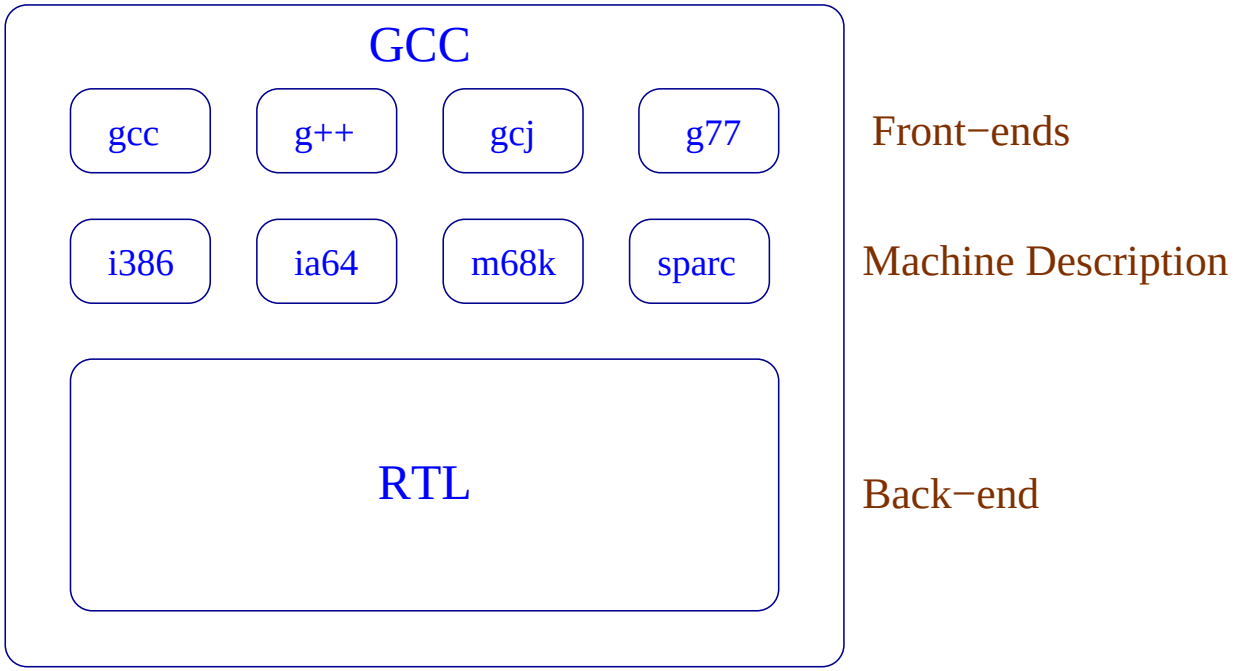
Front-ends / back-end



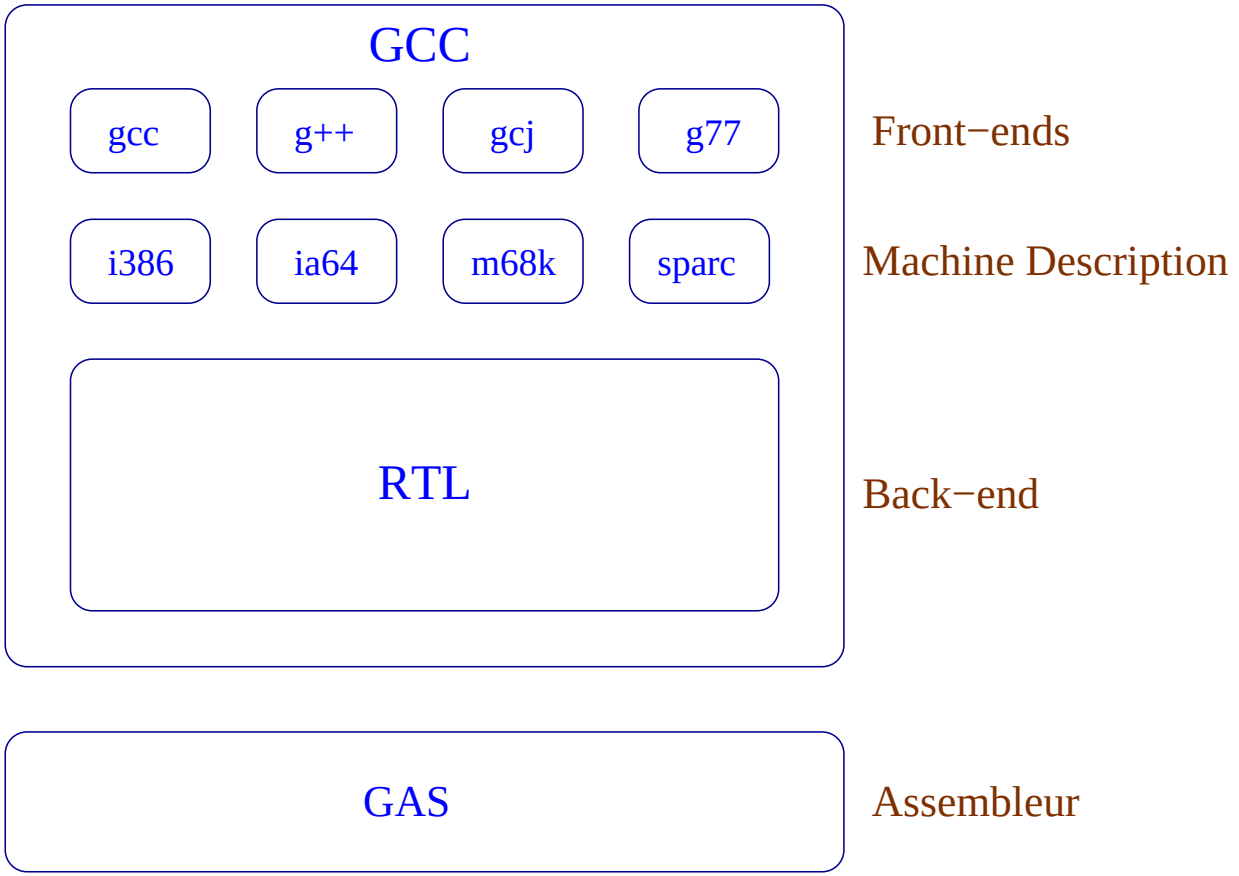
Front-ends / back-end



Front-ends / back-end



Front-ends / back-end



Exemple: cross-compilation

- Supposons qu'on veut générer du code Sparc
  `--target=sparc`

Exemple: cross-compilation

- Supposons qu'on veut générer du code Sparc

  `--target=sparc`

- on construit GCC sur mon portable :

  `--build=i586`

Exemple: cross-compilation

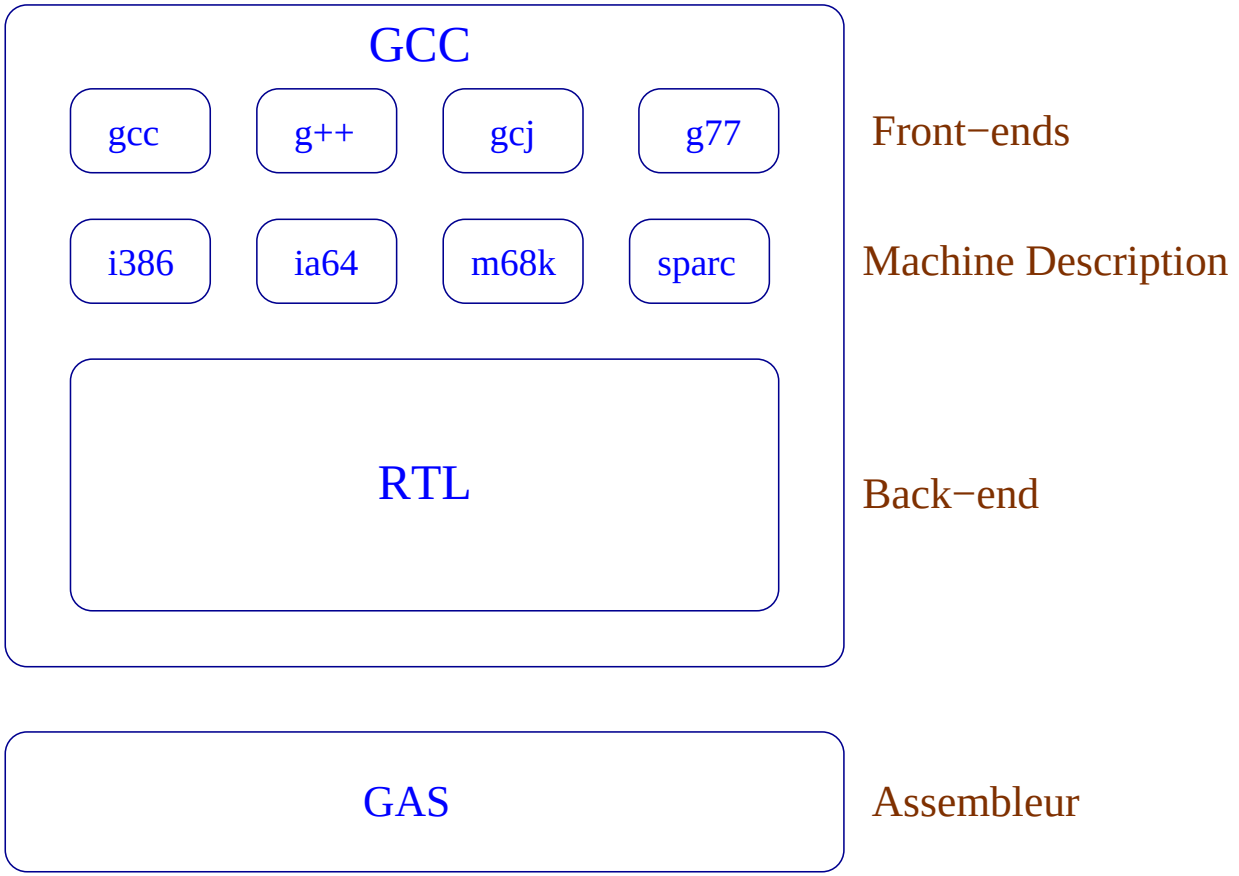
- Supposons qu'on veut générer du code Sparc
  `--target=sparc`
- on construit GCC sur mon portable :
  `--build=i586`
- et une fois construit, on fait tourner GCC sur mon portable : `--host=i586`

Exemple: cross-compilation

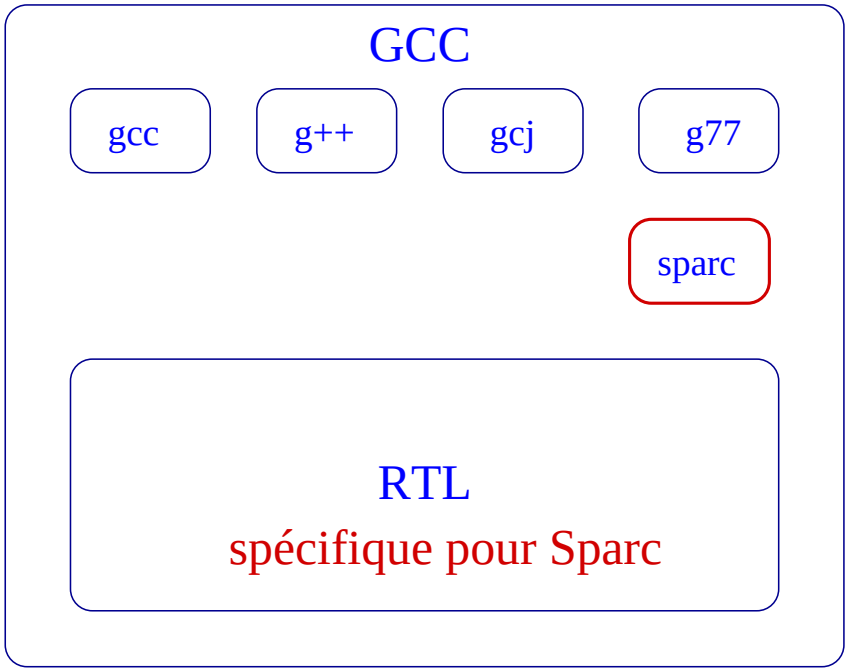
- Supposons qu'on veut générer du code Sparc
  `--target=sparc`
- on construit GCC sur mon portable :
  `--build=i586`
- et une fois construit, on fait tourner GCC sur mon portable : `--host=i586`

```
../gcc/configure --target=sparc --build=i586
--host=i586
```

Exemple: cross-compilation

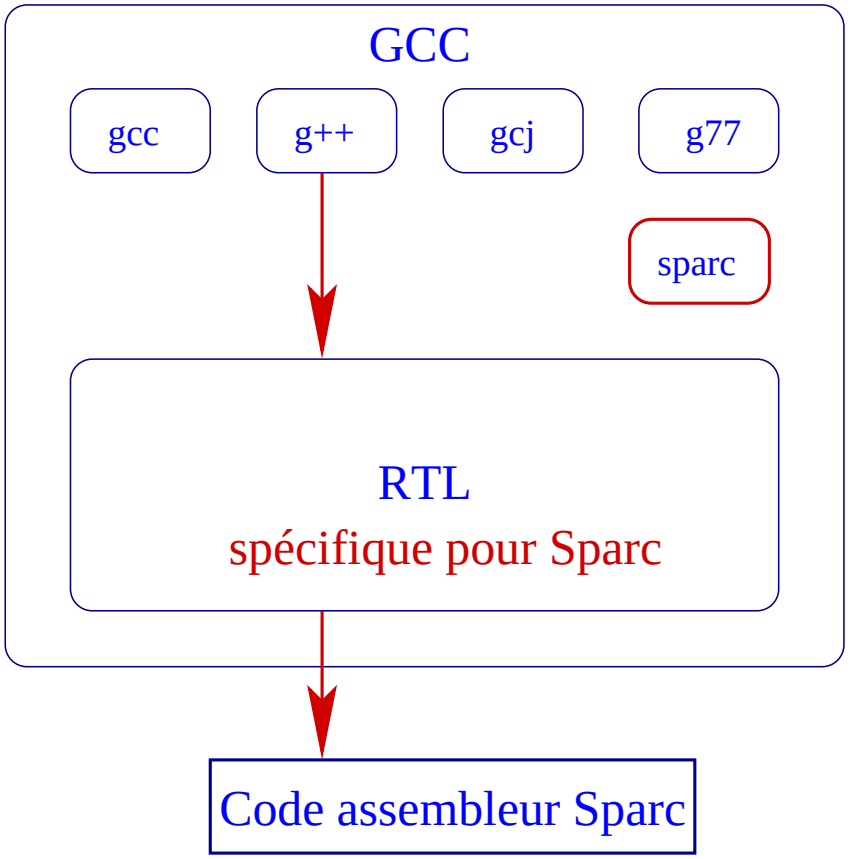


Exemple: cross-compilation



1. Configurer gcc pour produire du code sparc.

Exemple: cross-compilation



1. Configurer gcc pour produire du code sparc.

2. Compiler.

Optimisations RTL

- Une optimisation touche tous les fronts-ends.

Optimisations RTL

- Une optimisation touche tous les fronts-ends.
- Optimisations dépendantes des spécificités de la machine de destination.

Optimisations RTL

- Une optimisation touche tous les fronts-ends.
- Optimisations dépendantes des spécificités de la machine de destination.
- Types et accès aux structures mémoire perdus lors de la traduction vers le RTL.

  tous les accès mémoire se font sous la forme

  <adresse de début + offset>

Optimisations RTL

- Une optimisation touche tous les fronts-ends.
- Optimisations dépendantes des spécificités de la machine de destination.
- Types et accès aux structures mémoire perdus lors de la traduction vers le RTL.

  tous les accès mémoire se font sous la forme

  <adresse de début + offset>
- Idée : avoir des optimisations

Optimisations RTL

- Une optimisation touche tous les fronts-ends.
- Optimisations dépendantes des spécificités de la machine de destination.
- Types et accès aux structures mémoire perdus lors de la traduction vers le RTL.

  tous les accès mémoire se font sous la forme

  <adresse de début + offset>
- Idée : avoir des optimisations
  - indépendantes des architectures

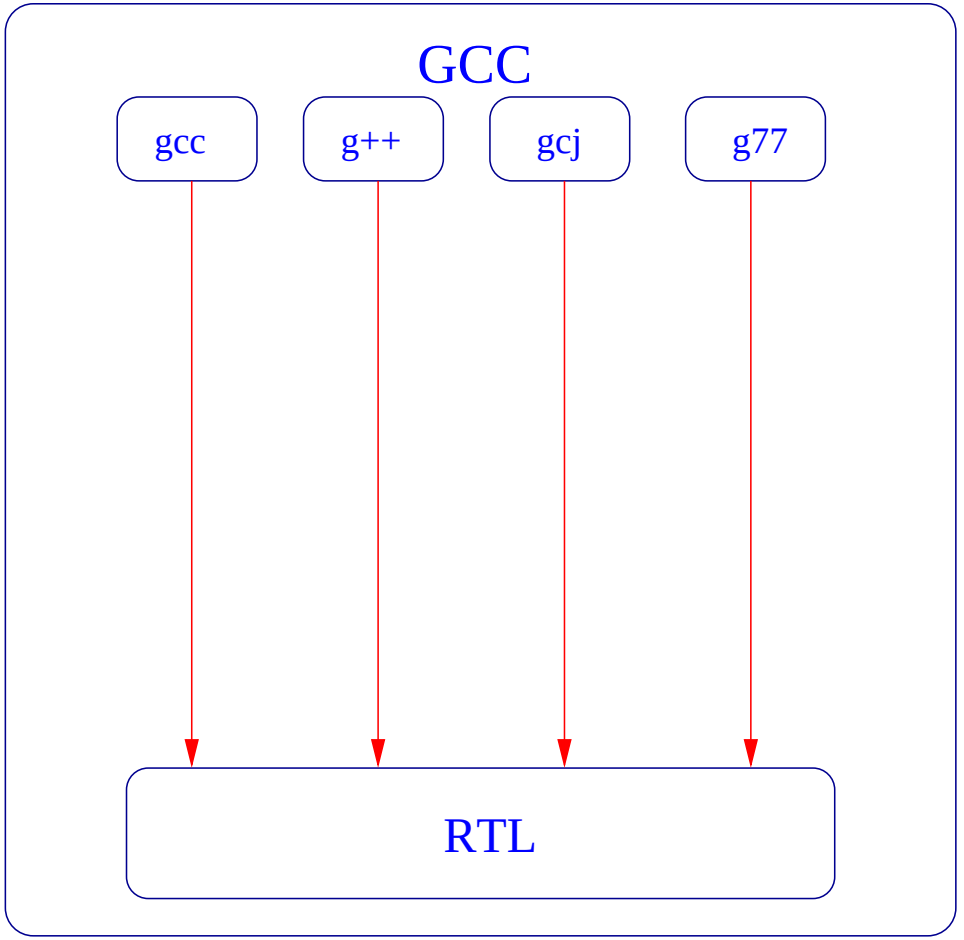
Optimisations RTL

- Une optimisation touche tous les fronts-ends.
- Optimisations dépendantes des spécificités de la machine de destination.
- Types et accès aux structures mémoire perdus lors de la traduction vers le RTL.

  tous les accès mémoire se font sous la forme

  <adresse de début + offset>
- Idée : avoir des optimisations
  - indépendantes des architectures
  - sur représentations de haut niveau (AST)

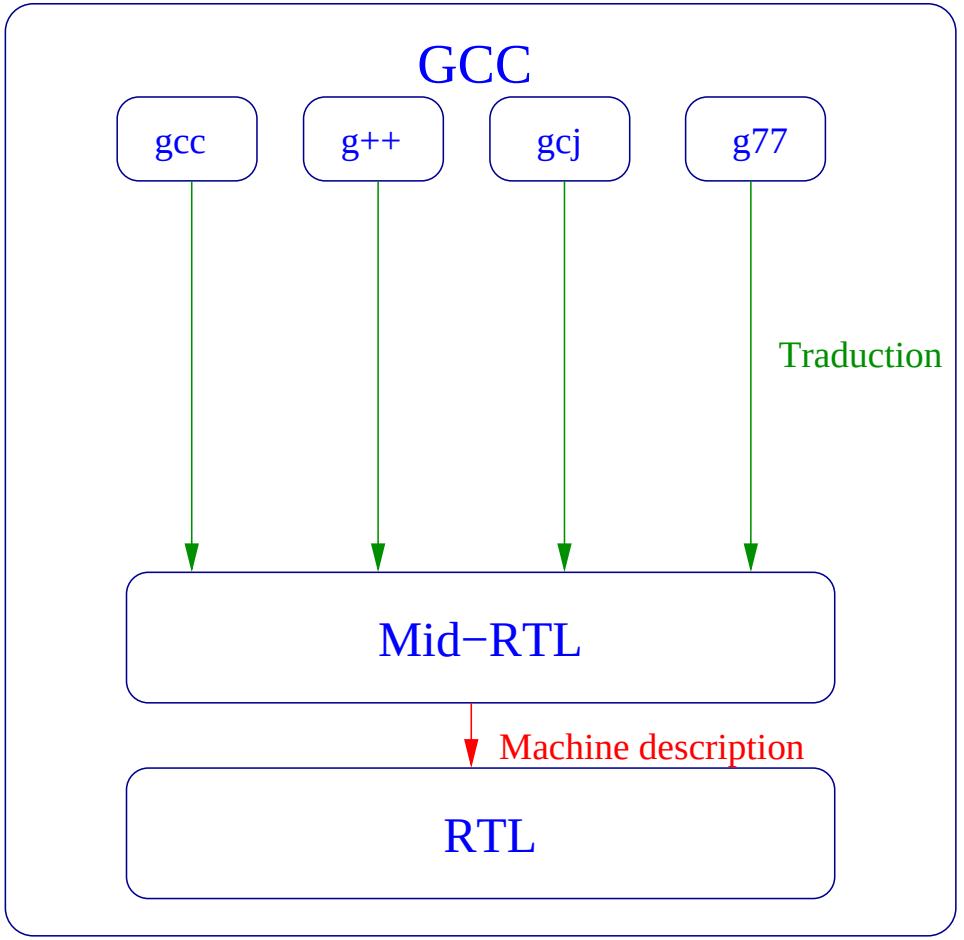
Représentations intermédiaires



Traduction selon les spécificités de la machine

Machine description

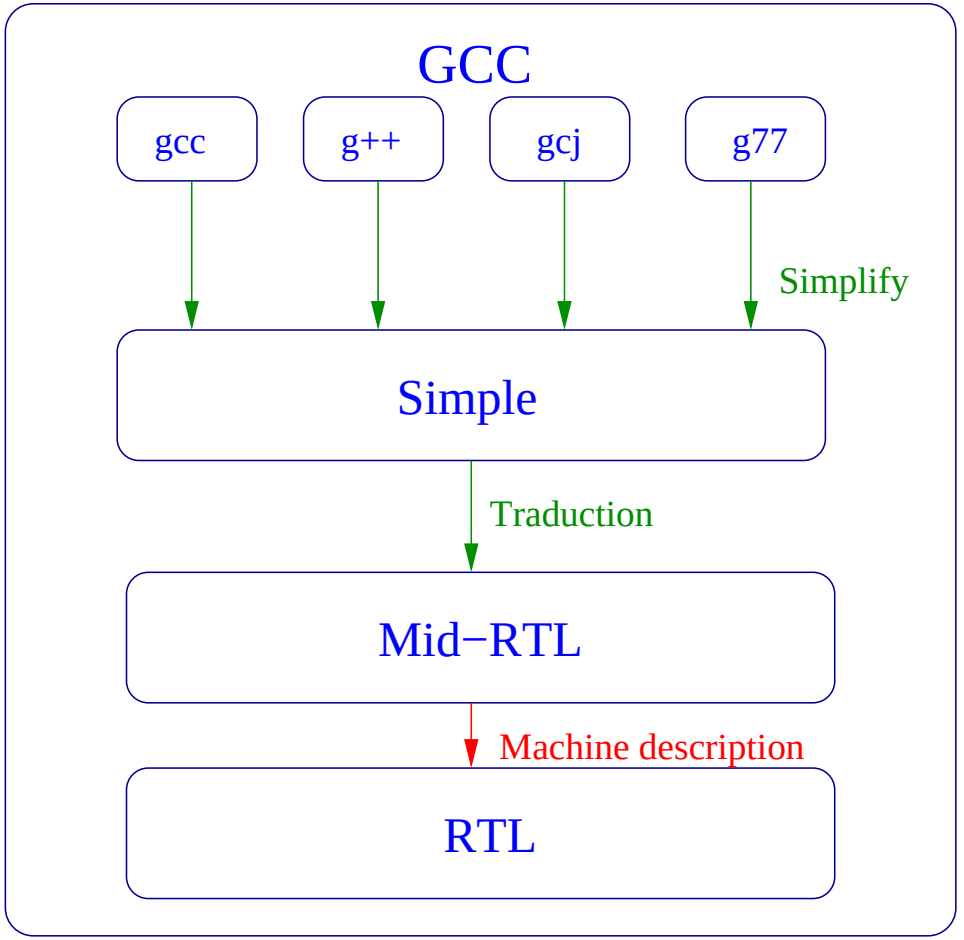
Représentations intermédiaires



Transition progressive des AST vers le RTL

Indépendant de l'architecture

Représentations intermédiaires



Représentation commune sous forme d'AST

Défini pour etre facilement analysé

Optimisations sur les AST factorisées

Transition progressive des AST vers le RTL

Indépendant de l'architecture

Arbres de Syntaxe

- Liste chaînée de noeuds.

Arbres de Syntaxe

- Liste chaînée de noeuds.
- Pour manipuler les noeuds, une interface :
  `TREE_CHAIN, TREE_OPERAND, TREE_CODE, ...`

Arbres de Syntaxe

- Liste chaînée de noeuds.
- Pour manipuler les noeuds, une interface : `TREE_CHAIN, TREE_OPERAND, TREE_CODE, ...`
- Structures de données invisibles : ces accesseurs sont des macros traduites par le précompilateur.

Arbres de Syntaxe

- Liste chaînée de noeuds.
- Pour manipuler les noeuds, une interface : `TREE_CHAIN, TREE_OPERAND, TREE_CODE, ...`
- Structures de données invisibles : ces accesseurs sont des macros traduites par le précompilateur.
- Possibilité d'effectuer du *Tree-checking* sur les types des noeuds.

Arbres de Syntaxe

- Liste chaînée de noeuds.
- Pour manipuler les noeuds, une interface : `TREE_CHAIN, TREE_OPERAND, TREE_CODE, ...`
- Structures de données invisibles : ces accesseurs sont des macros traduites par le précompilateur.
- Possibilité d'effectuer du *Tree-checking* sur les types des noeuds.
- C'est du C orienté objet.

AST: exemple

```
a = (--b) * 7;
x = y+z;
```

AST: exemple

AST: exemple

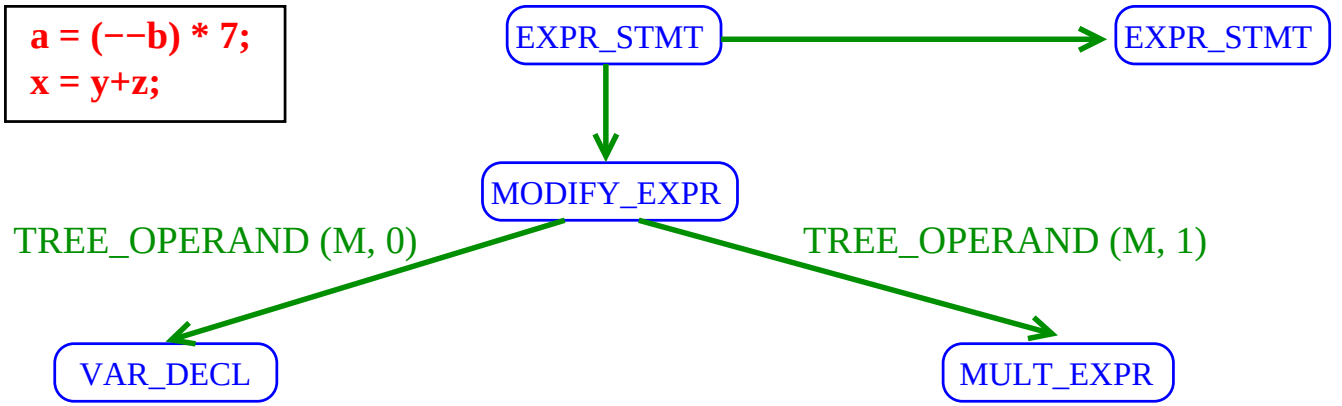
```
a = (--b) * 7;
x = y+z;
```

AST: exemple

```
a = (--b) * 7;
x = y+z;
```

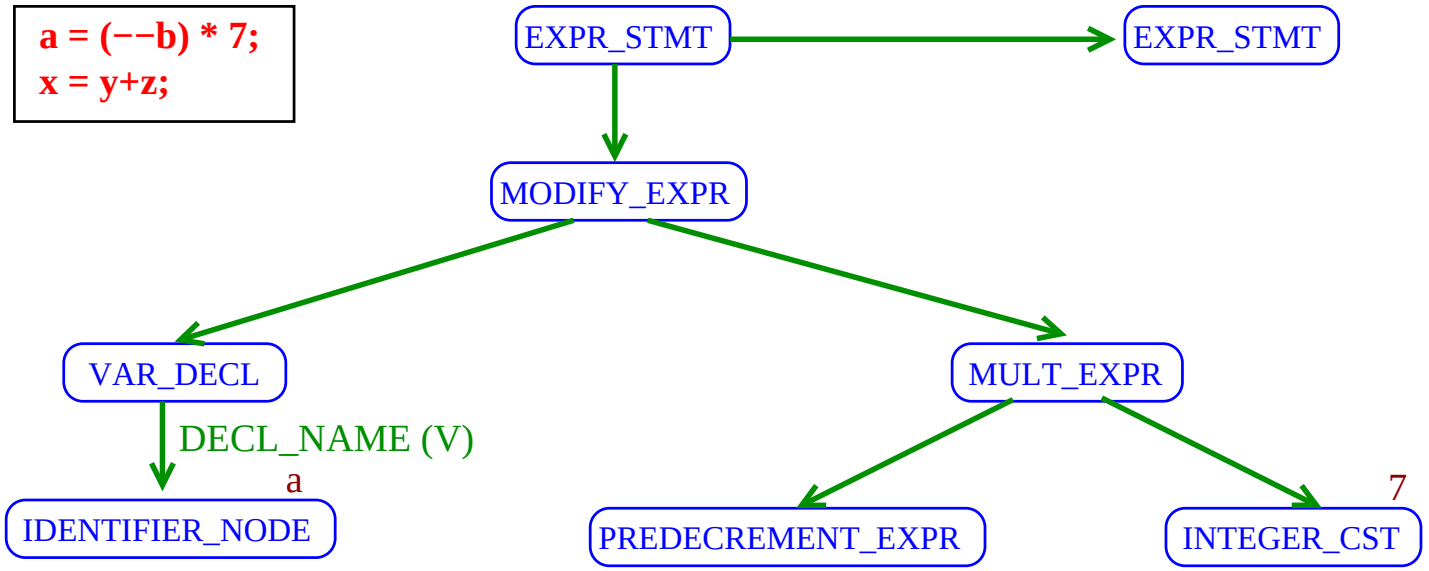
AST: exemple

```
a = (--b) * 7;
x = y+z;
```



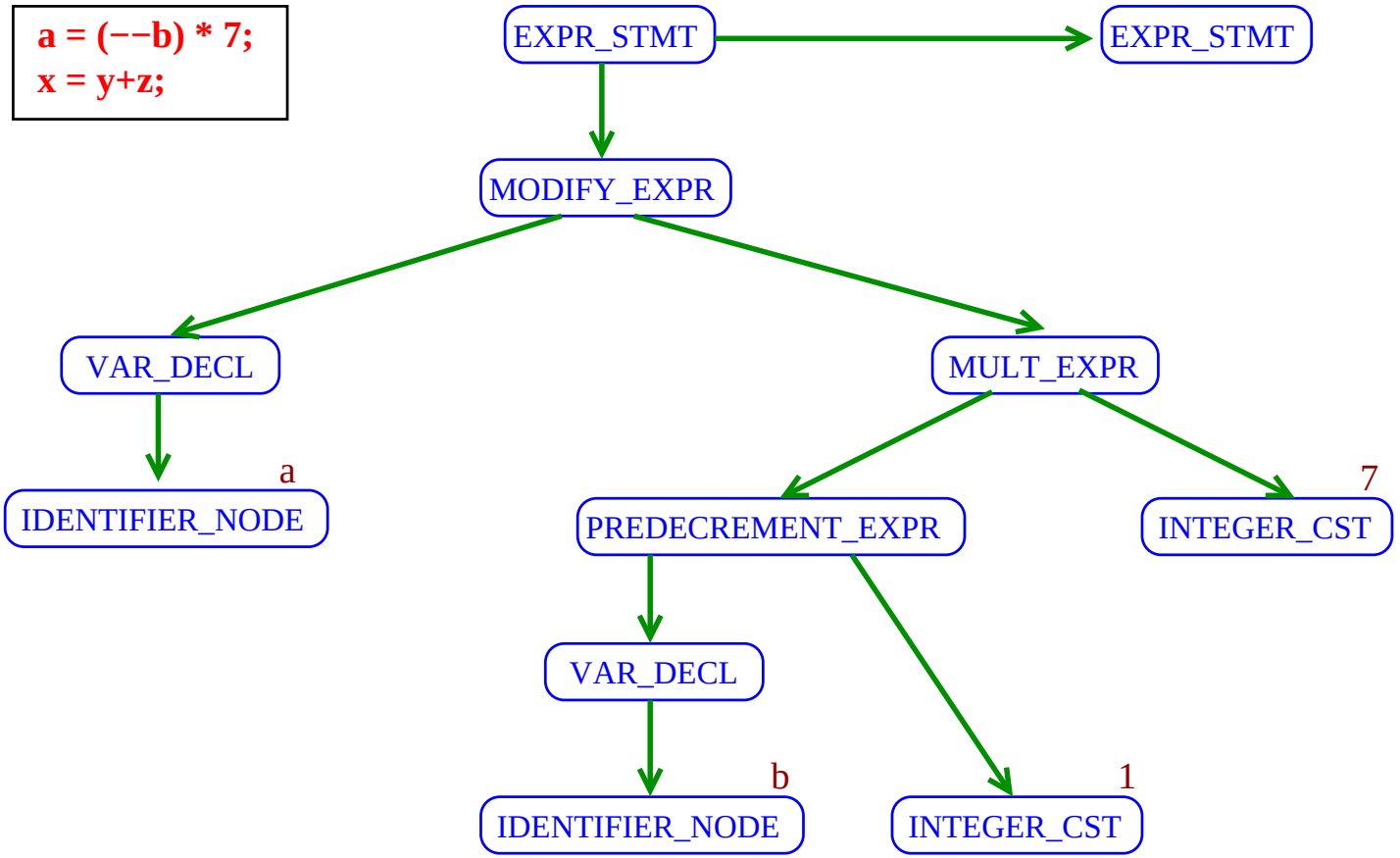
AST: exemple

```
a = (--b) * 7;
x = y+z;
```



AST: exemple

```
a = (--b) * 7;
x = y+z;
```



Simple: en général

- La grammaire de Simple définit des expressions faciles à analyser.

Simple: en général

- La grammaire de Simple définit des expressions faciles à analyser.
  - Nombre réduit d’expressions autorisées.

Simple: en général

- La grammaire de Simple définit des expressions faciles à analyser.
  - Nombre réduit d’expressions autorisées.
  - Nombre réduit de structures de contrôle.

Simple: en général

- La grammaire de Simple définit des expressions faciles à analyser.
  - Nombre réduit d'expressions autorisées.
  - Nombre réduit de structures de contrôle.
- L'AST a une structure régulière.

Simple: en général

- La grammaire de Simple définit des expressions faciles à analyser.
  - Nombre réduit d'expressions autorisées.
  - Nombre réduit de structures de contrôle.
- L'AST a une structure régulière.
- Permet une analyse systématique des AST.

Simple: en général

- La grammaire de Simple définit des expressions faciles à analyser.
  - Nombre réduit d'expressions autorisées.
  - Nombre réduit de structures de contrôle.
- L'AST a une structure régulière.
- Permet une analyse systématique des AST.
- Représentation intermédiaire commune à tous les front-ends.

Simple: exemple

```
a = --b*7;
```

Simple: exemple

`a = --b*7;`	`b=b-1;` `a=b*7;`

Simple: exemple

```
a = --b*7;
```

```
b=b-1;
a=b*7;
```

```
if (i++ && --k)
 {
   j=f(i+3*k);
 }
```

Simple: exemple

```
a = --b*7;
```

```
b=b-1;
a=b*7;
```

```
if (i++ && --k)
 {
  j=f(i+3*k);
 }
```

```
if (i)
 {
  k=k-1;
  if(k)
   {
    i=i+1;
    T1=3*k;
    T2=i+T1;
    j=f(T2);
   }
  else
   i=i+1;
 }
 else
  i=i+1;
```

Simple: exemple

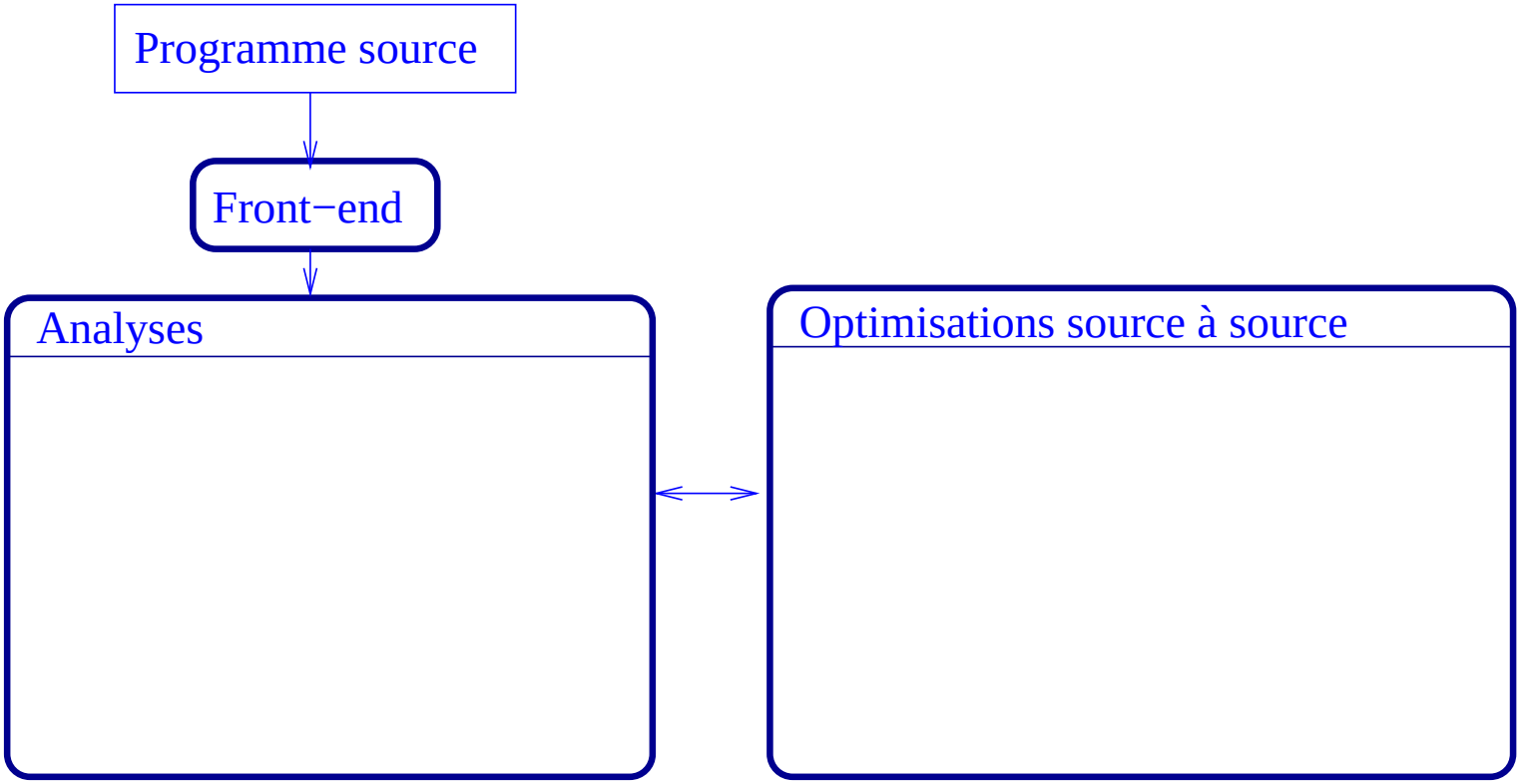
```
while(i++ && --k)
 {
  A[i]=A[i+3*k];
 }
```

Simple: exemple

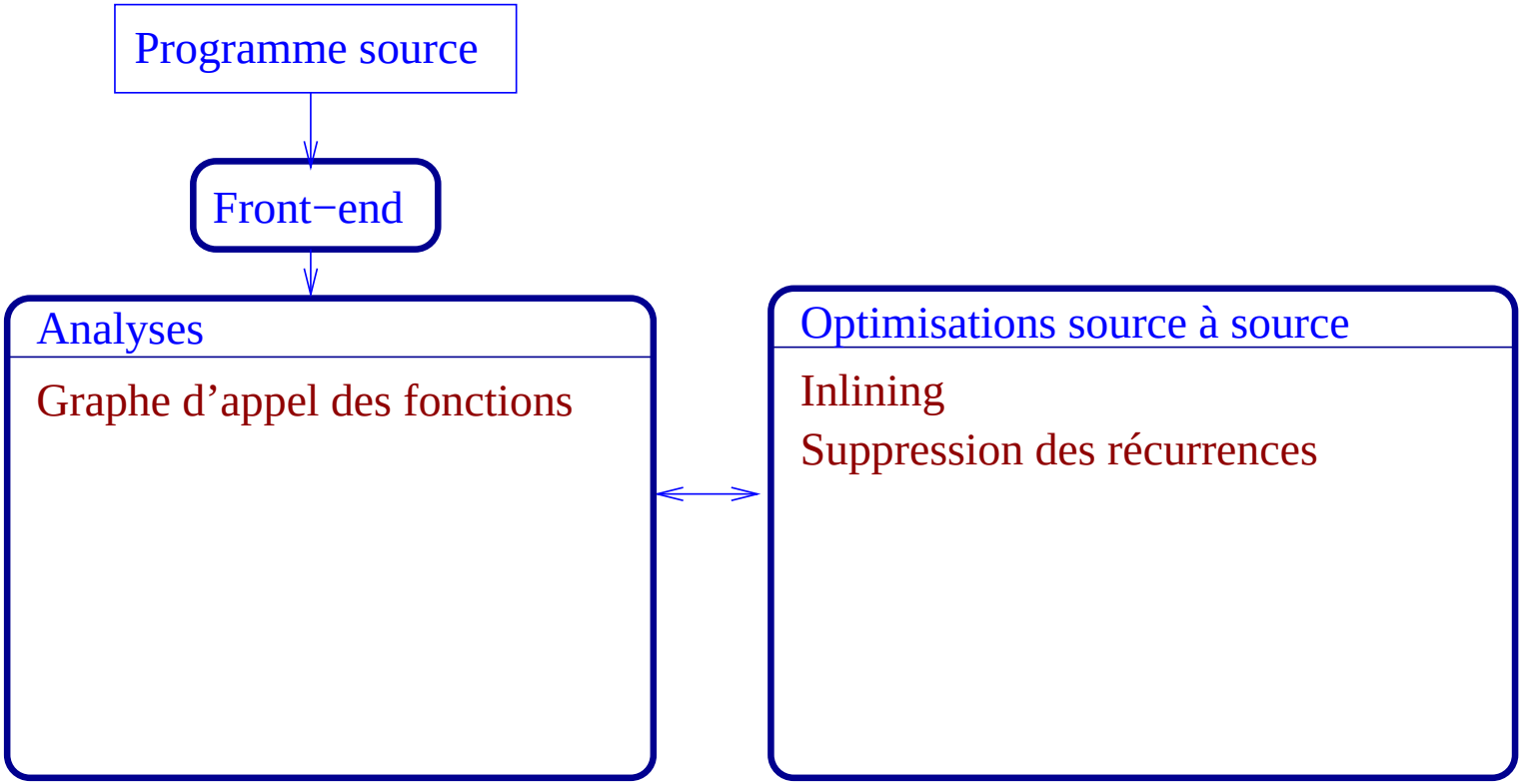
```
while(i++ && --k)
 {
  A[i]=A[i+3*k];
 }
```

```
if(i)
 {
  k=k-1;
  if (k)
   while(1)
    {
     i=i+1;
     T1=3*k;
     T2=i+T1;
     A[i]=A[T2];
     if(i)
      {
       k=k-1;
       if(k)
        i=i+1;
       else
        break;
      }
     else
      break;
    }
 }
i=i+1;
```

Un compilateur optimiseur



Un compilateur optimiseur



Call Graph

- (noeud, arc) => (déclaration, appel)

Call Graph

- (noeud, arc) => (déclaration, appel)
- Représenter un graphe :

Call Graph

- (noeud, arc) => (déclaration, appel)
- Représenter un graphe :
  - en mémoire : pointeurs, (P-Space).

Call Graph

- (noeud, arc) => (déclaration, appel)
- Représenter un graphe :
  - en mémoire : pointeurs, (P-Space).
  - dans un fichier : XML, (EXP-Space).

Call Graph

- (noeud, arc) => (déclaration, appel)
- Représenter un graphe :
  - en mémoire : pointeurs, (P-Space).
  - dans un fichier : XML, (EXP-Space).
- Utiliser des métriques pour déterminer si une fonction est candidate pour l'inlining.

Call Graph

- (noeud, arc) => (déclaration, appel)
- Représenter un graphe :
  - en mémoire : pointeurs, (P-Space).
  - dans un fichier : XML, (EXP-Space).
- Utiliser des métriques pour déterminer si une fonction est candidate pour l'inlining.
- Gcc est limité au traitement d'un seul fichier à la fois, ce qui limite l'extraction de connaissances à son "translation unit".

Call Graph : solution

- Une solution serait d'effectuer les optimisations du graphe d'appel à l'extérieur de GCC.

Call Graph : solution

- Une solution serait d'effectuer les optimisations du graphe d'appel à l'extérieur de GCC.
- Problèmes :

Call Graph : solution

- Une solution serait d'effectuer les optimisations du graphe d'appel à l'extérieur de GCC.
- Problèmes :
  - Extraire l'information, décider, puis appliquer les optimisations : 3 passes.

Call Graph : solution

- Une solution serait d'effectuer les optimisations du graphe d'appel à l'extérieur de GCC.
- Problèmes :
  - Extraire l'information, décider, puis appliquer les optimisations : 3 passes.
  - Taille de la base de connaissances (KB).

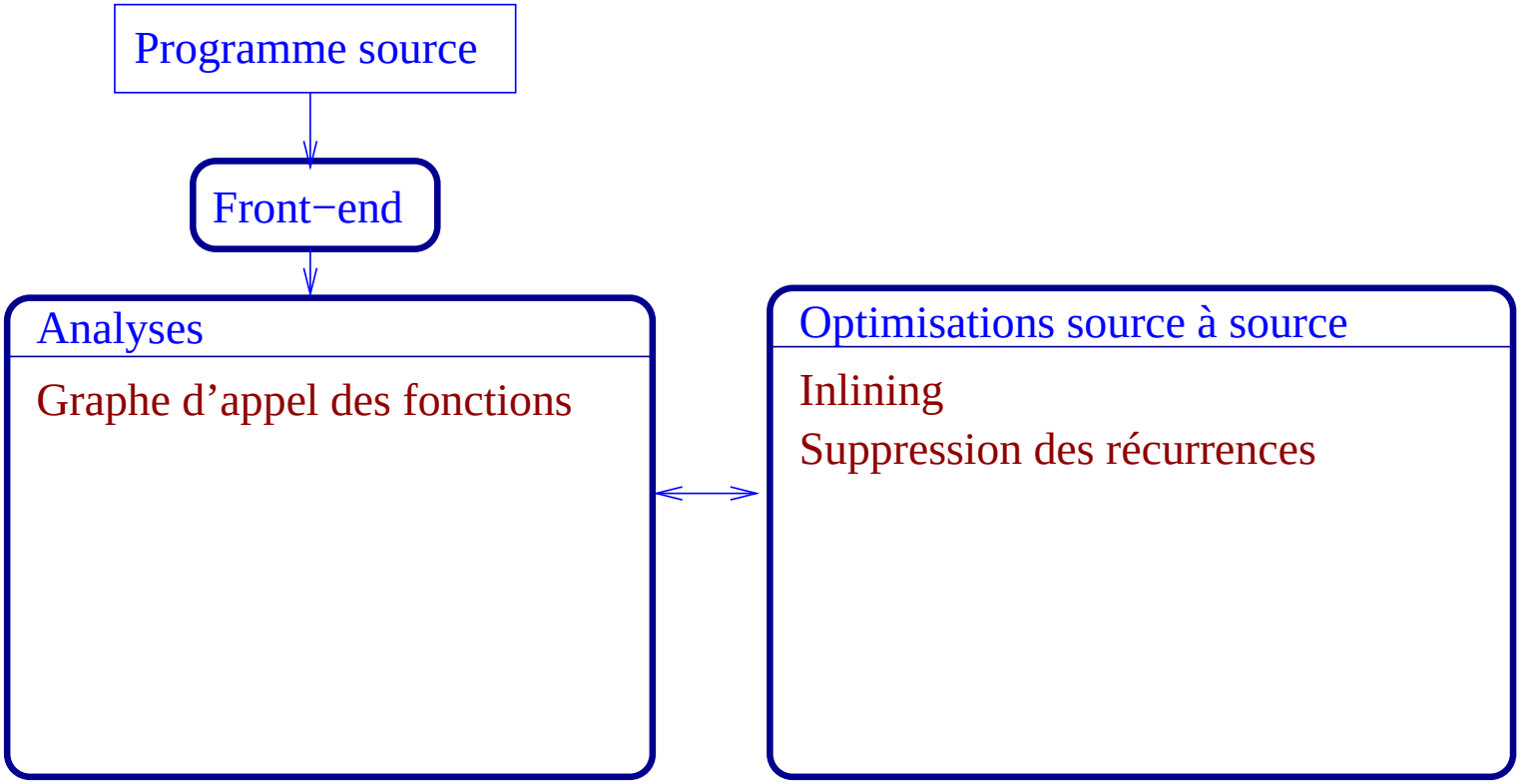
Call Graph : solution

- Une solution serait d'effectuer les optimisations du graphe d'appel à l'extérieur de GCC.
- Problèmes :
  - Extraire l'information, décider, puis appliquer les optimisations : 3 passes.
  - Taille de la base de connaissances (KB).
  - Quelles informations stocker dans KB?

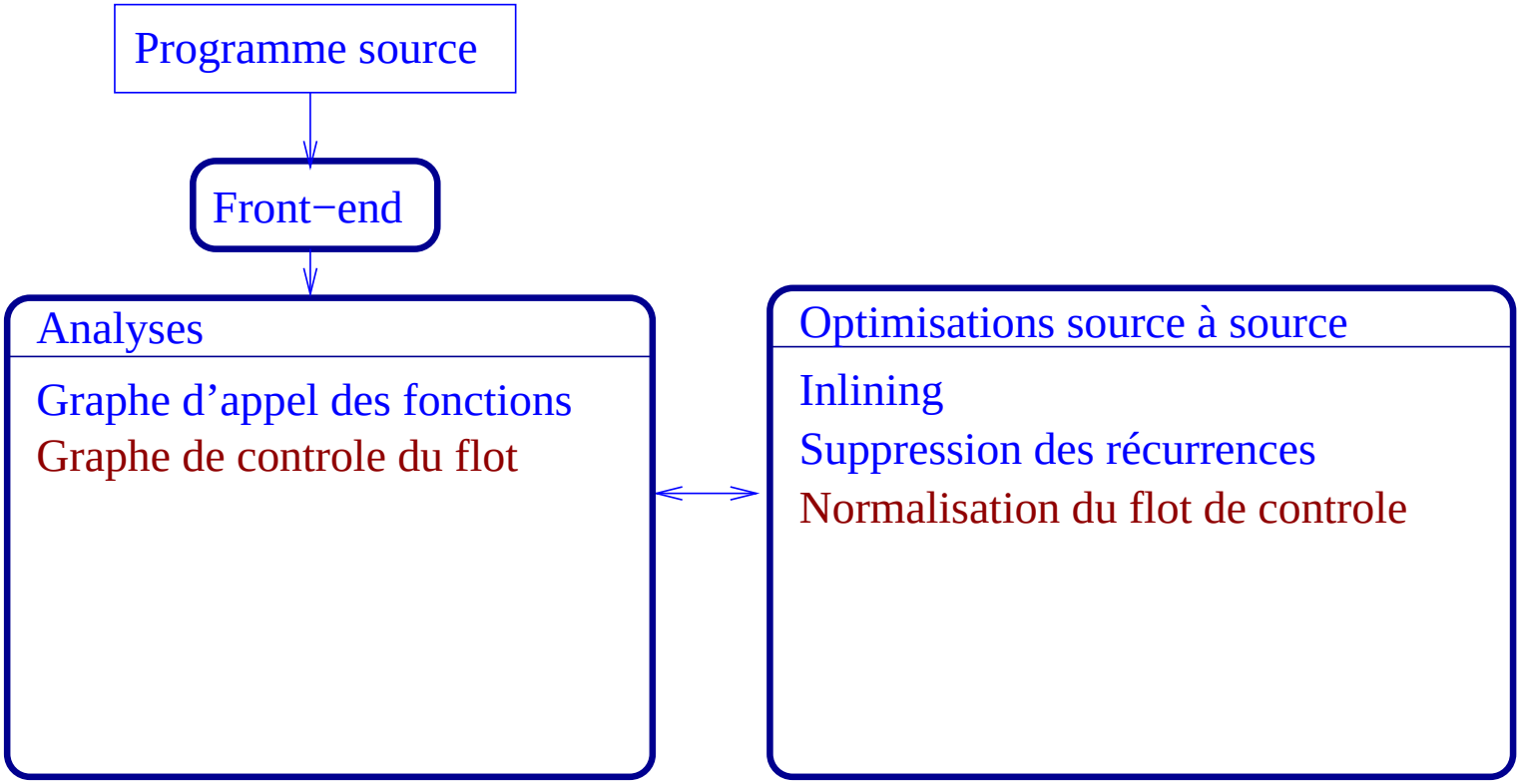
Call Graph : solution

- Une solution serait d'effectuer les optimisations du graphe d'appel à l'extérieur de GCC.
- Problèmes :
  - Extraire l'information, décider, puis appliquer les optimisations : 3 passes.
  - Taille de la base de connaissances (KB).
  - Quelles informations stocker dans KB?
- Petite démo . . .

Un compilateur optimiseur



Un compilateur optimiseur



CFG Normalization

- Supprimer les contrôles irréguliers : goto, break, continue.

CFG Normalization

- Supprimer les contrôles irréguliers : goto, break, continue.
- La normalisation se base sur Simple.

CFG Normalization

- Supprimer les contrôles irréguliers : goto, break, continue.
- La normalisation se base sur Simple.
- Pourquoi normaliser le graphe de flot?

CFG Normalization

- Supprimer les contrôles irréguliers : goto, break, continue.
- La normalisation se base sur Simple.
- Pourquoi normaliser le graphe de flot?
  - GCC n’optimise pas les programmes contenant des goto.

CFG Normalization

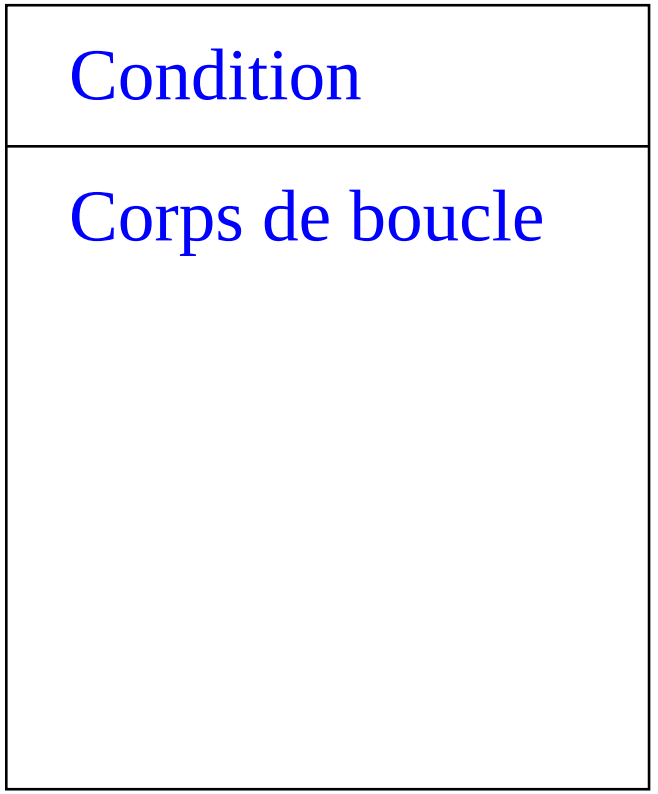
- Supprimer les contrôles irréguliers : goto, break, continue.
- La normalisation se base sur Simple.
- Pourquoi normaliser le graphe de flot?
  - GCC n’optimise pas les programmes contenant des goto.
  - La traduction des breaks et des continues vers le RTL génère des goto.

CFG Normalization

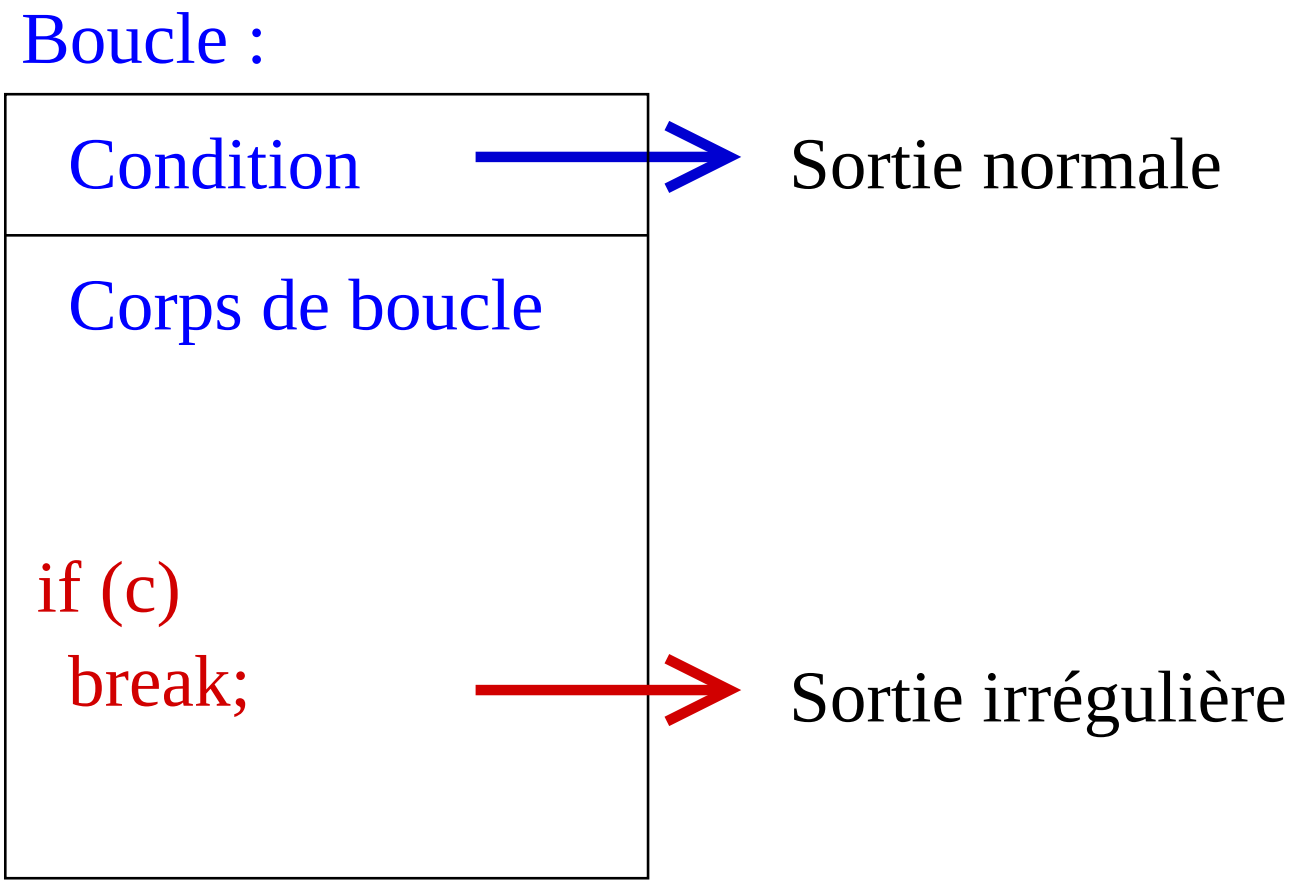
- Supprimer les contrôles irréguliers : goto, break, continue.
- La normalisation se base sur Simple.
- Pourquoi normaliser le graphe de flot?
  - GCC n'optimise pas les programmes contenant des goto.
  - La traduction des breaks et des continues vers le RTL génère des goto.
  - La simplification des expressions (simplify) peut générer du code irrégulier.

Flow Out

Boucle :

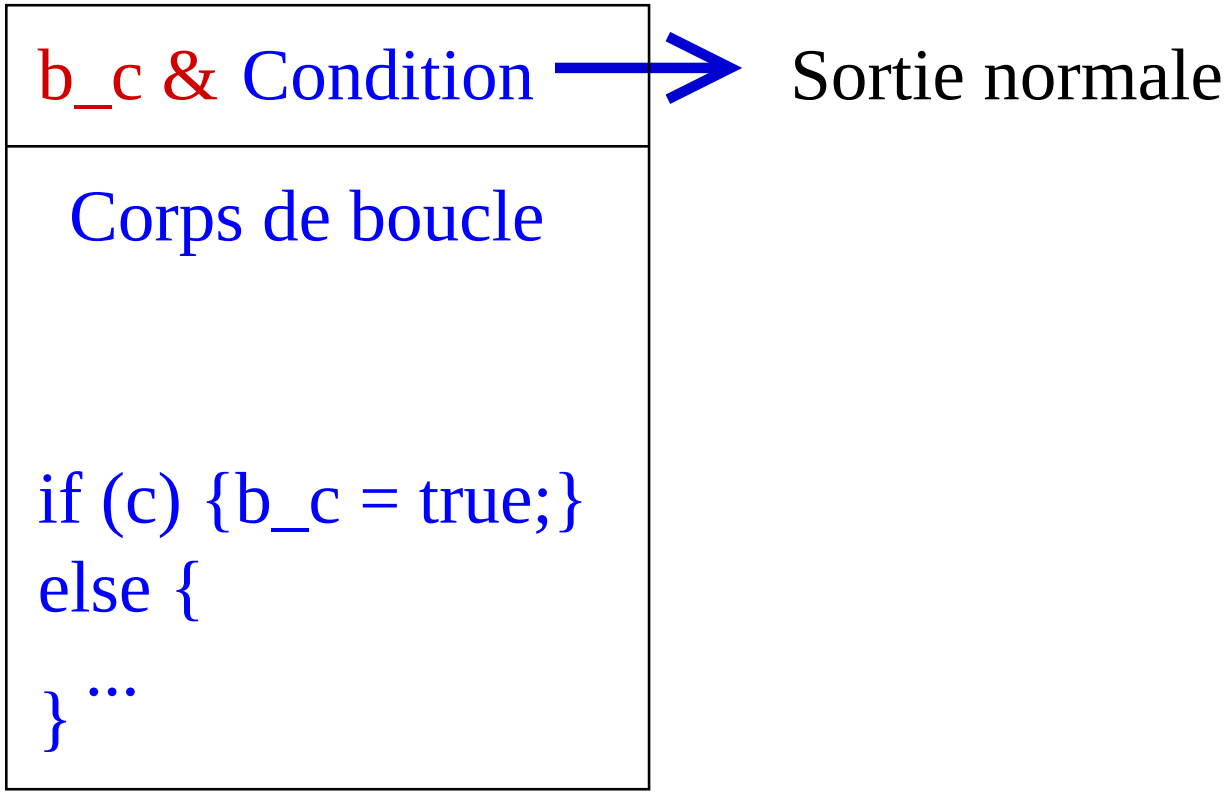


Flow Out


Boucle :
Condition
Sortie normale
Corps de boucle
if (c)
break;
Sortie irrégulière

Flow Out

Boucle :



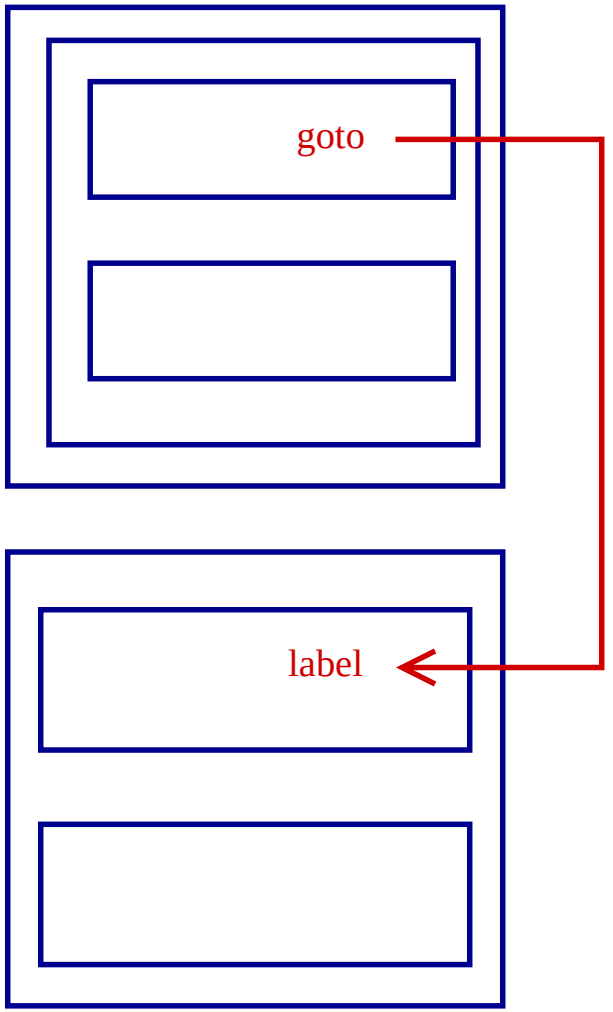
Break Elimination

```
while (a)
  {
    stmt1;
    if (b)
      break;
    stmt2;
  }
```

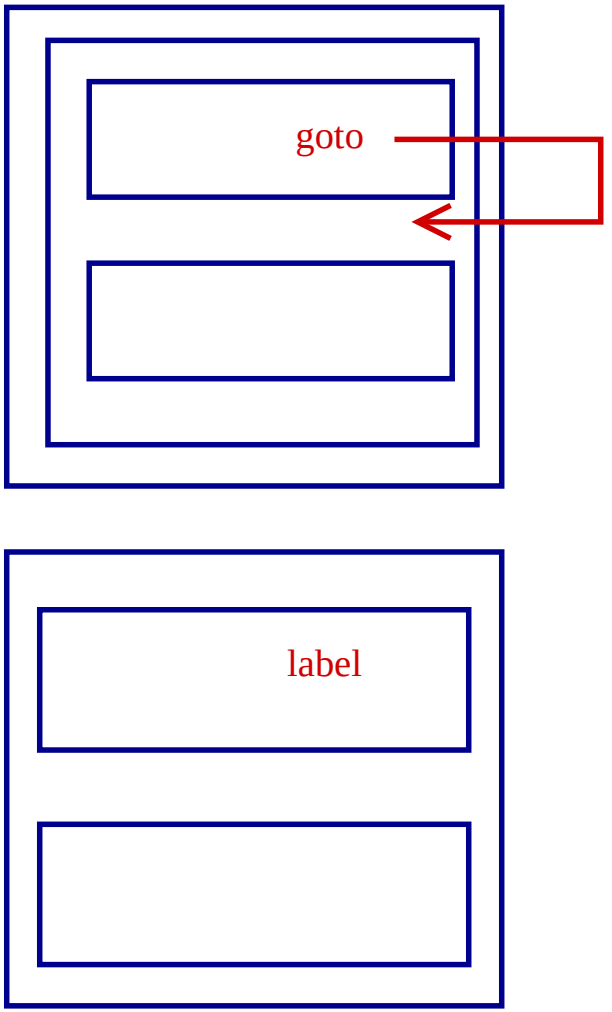
Break Elimination

```
int c_b = 0;
while (c_b == 0 && a)
  {
    stmt1;
    if (b)
      {c_b = 1;}
      else
        {
          stmt2;
        }
  }
```

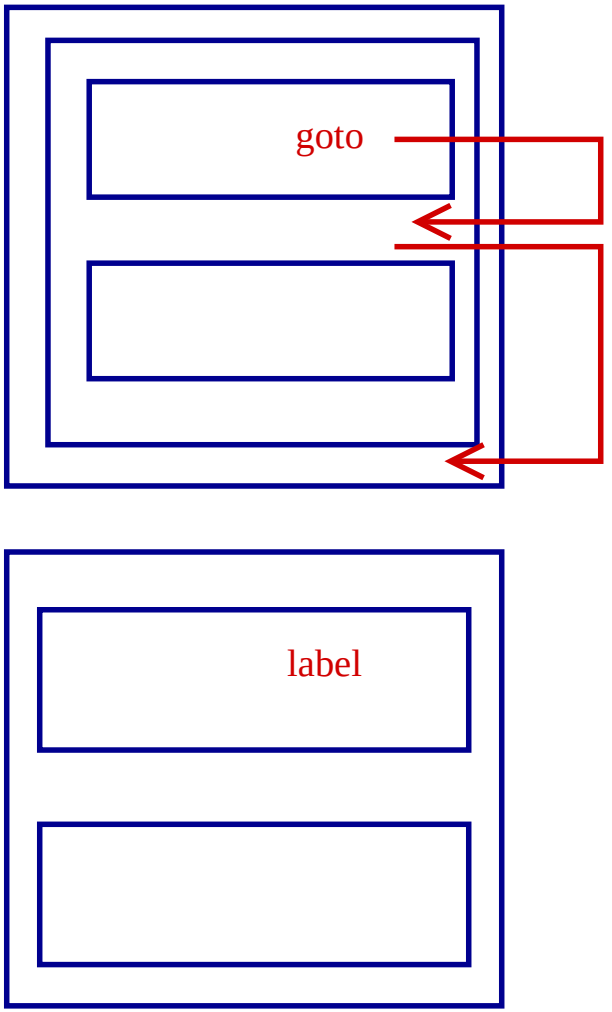
Goto Elimination



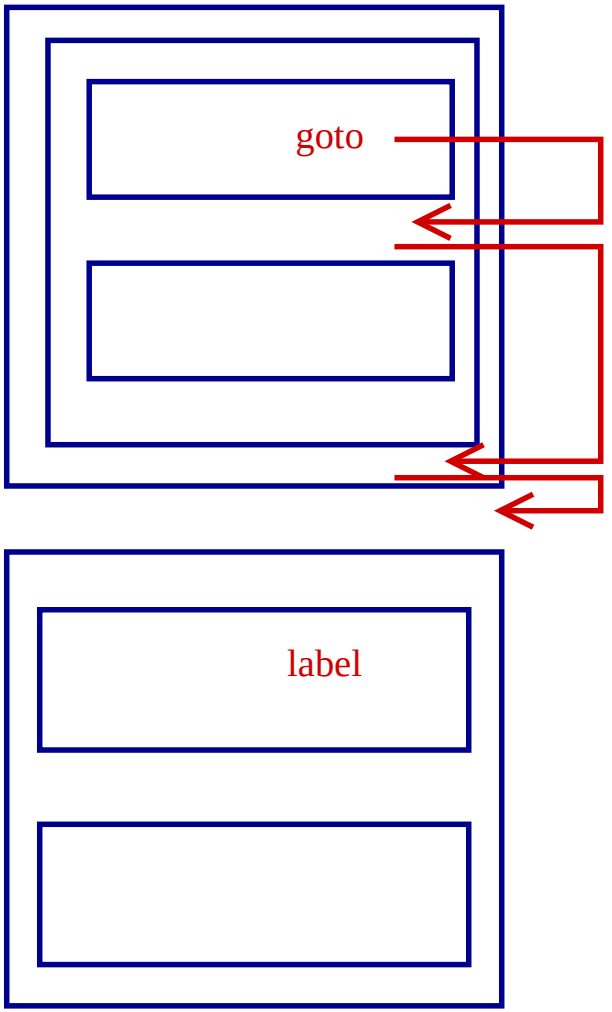
Goto Elimination



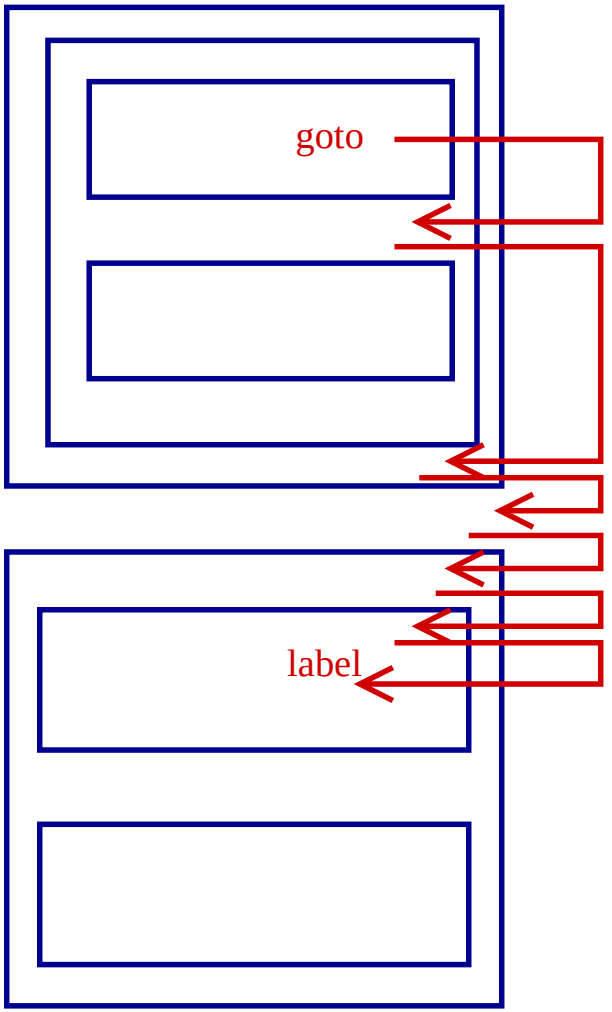
Goto Elimination



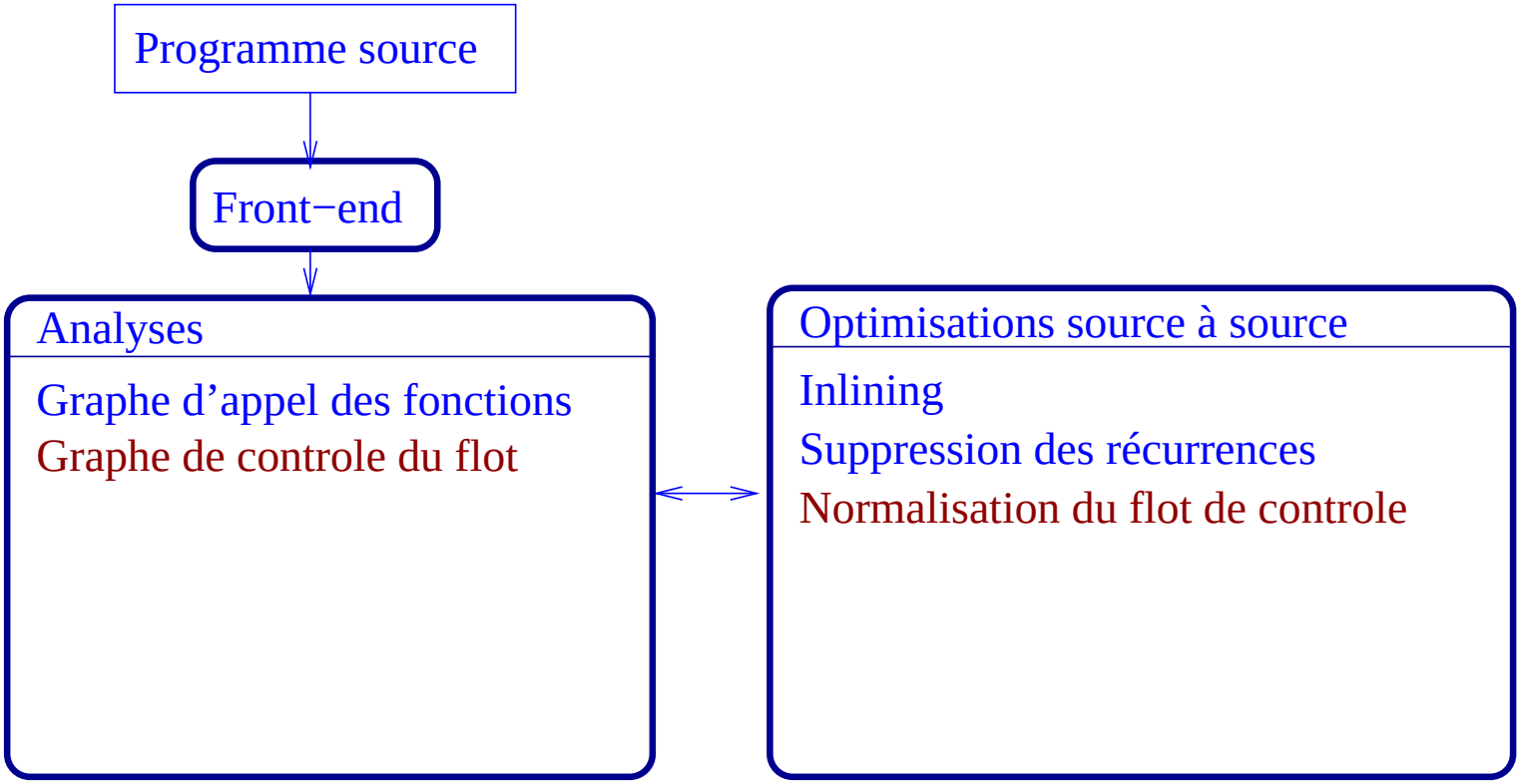
Goto Elimination



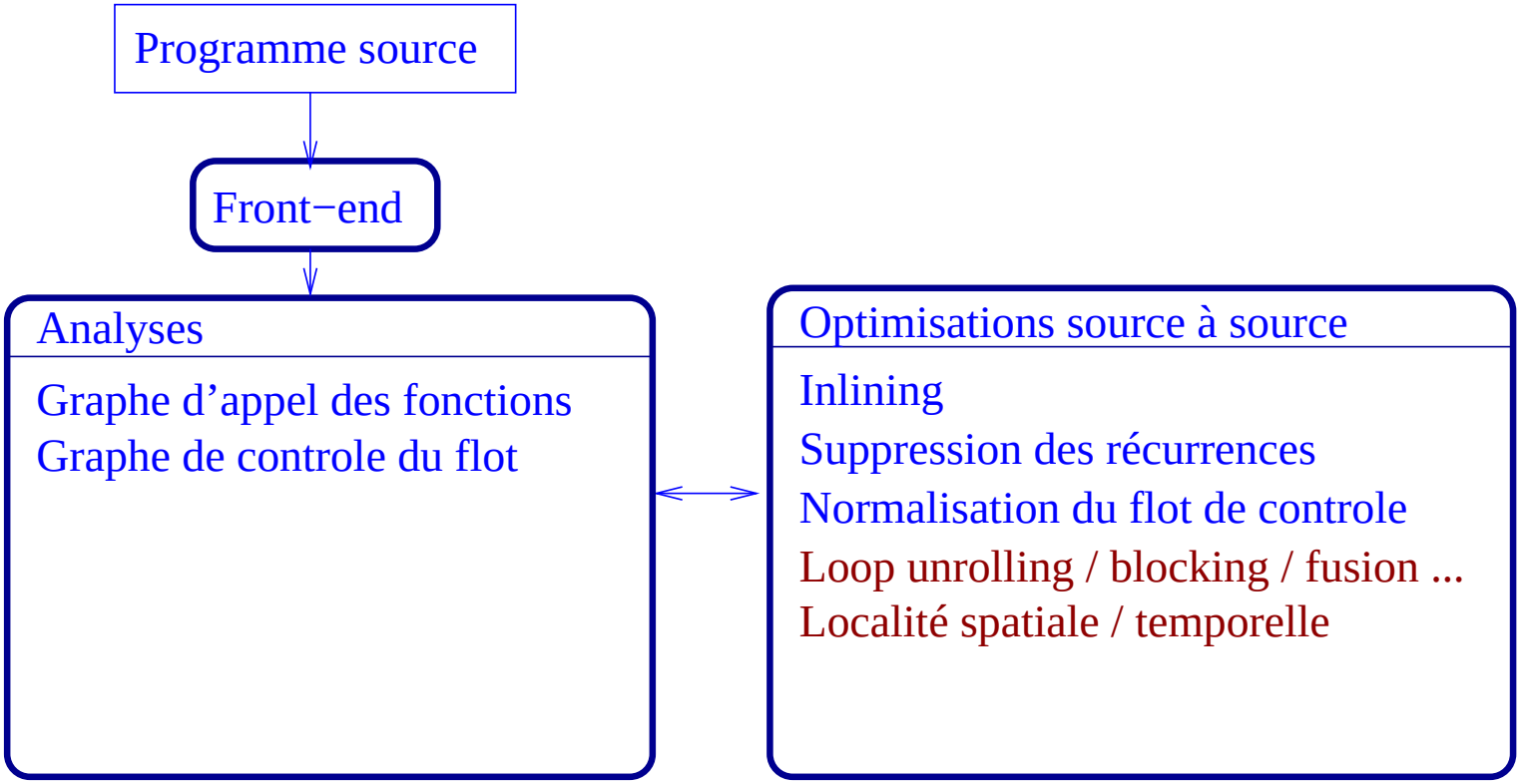
Goto Elimination



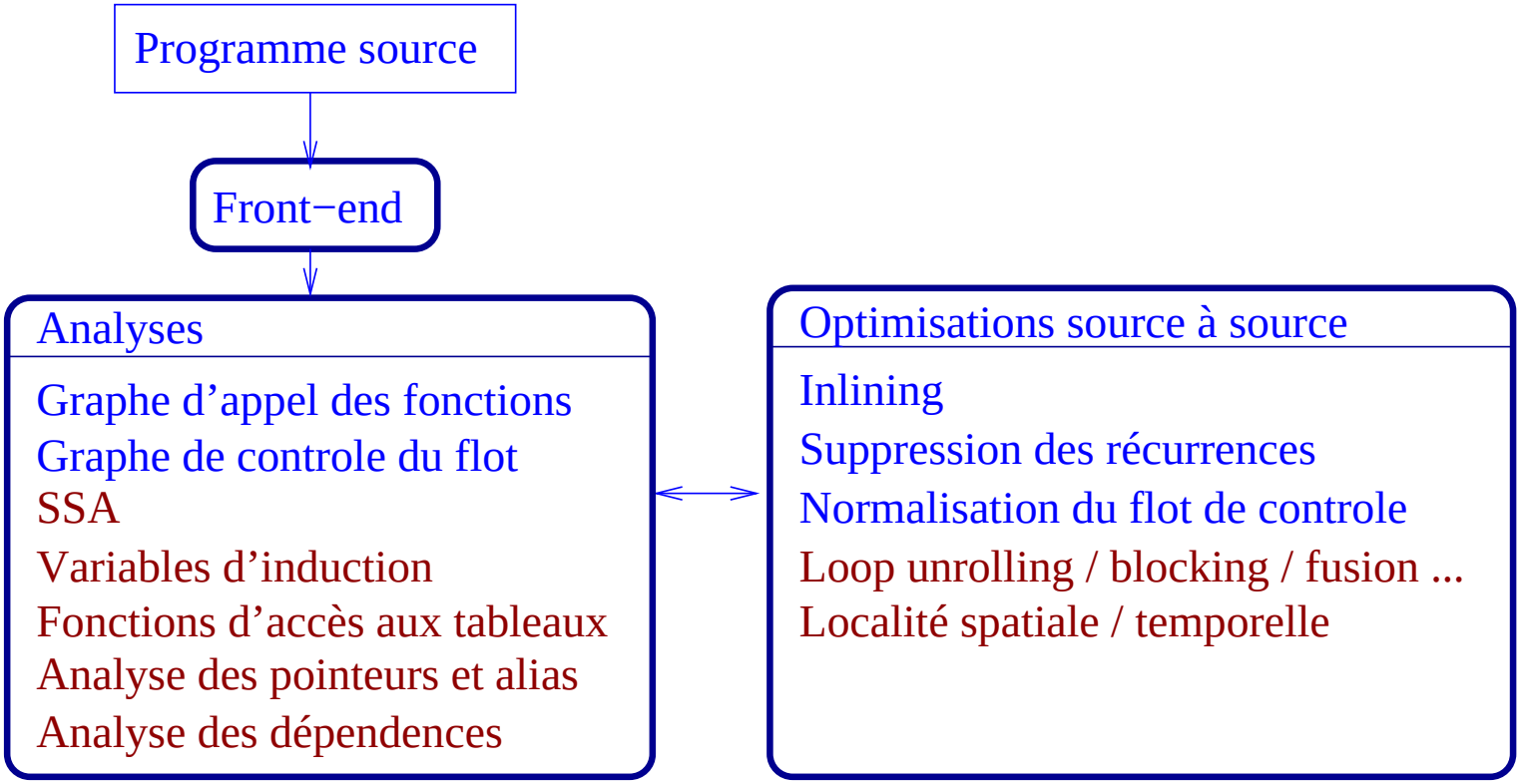
Un compilateur optimiseur



Un compilateur optimiseur



Un compilateur optimiseur



Loop Optimizations

- Après avoir déterminé les variables d’induction,

Loop Optimizations

- Après avoir déterminé les variables d’induction,
- il est possible de construire la représentation géométrique du domaine d’itération.

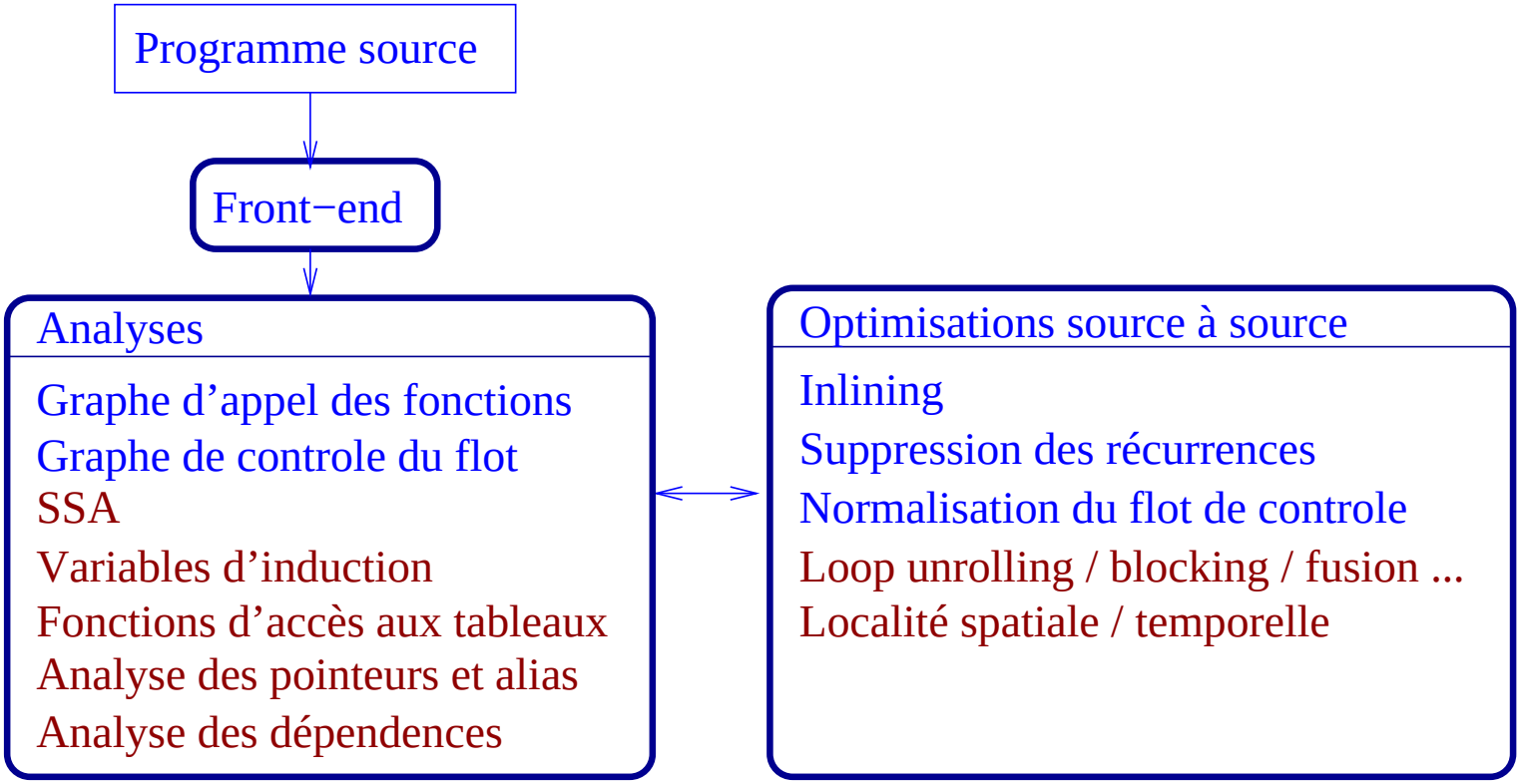
Loop Optimizations

- Après avoir déterminé les variables d'induction,
- il est possible de construire la représentation géométrique du domaine d'itération.
- Pour valider les transformations des boucles, l'analyse des dépendences est nécessaire.

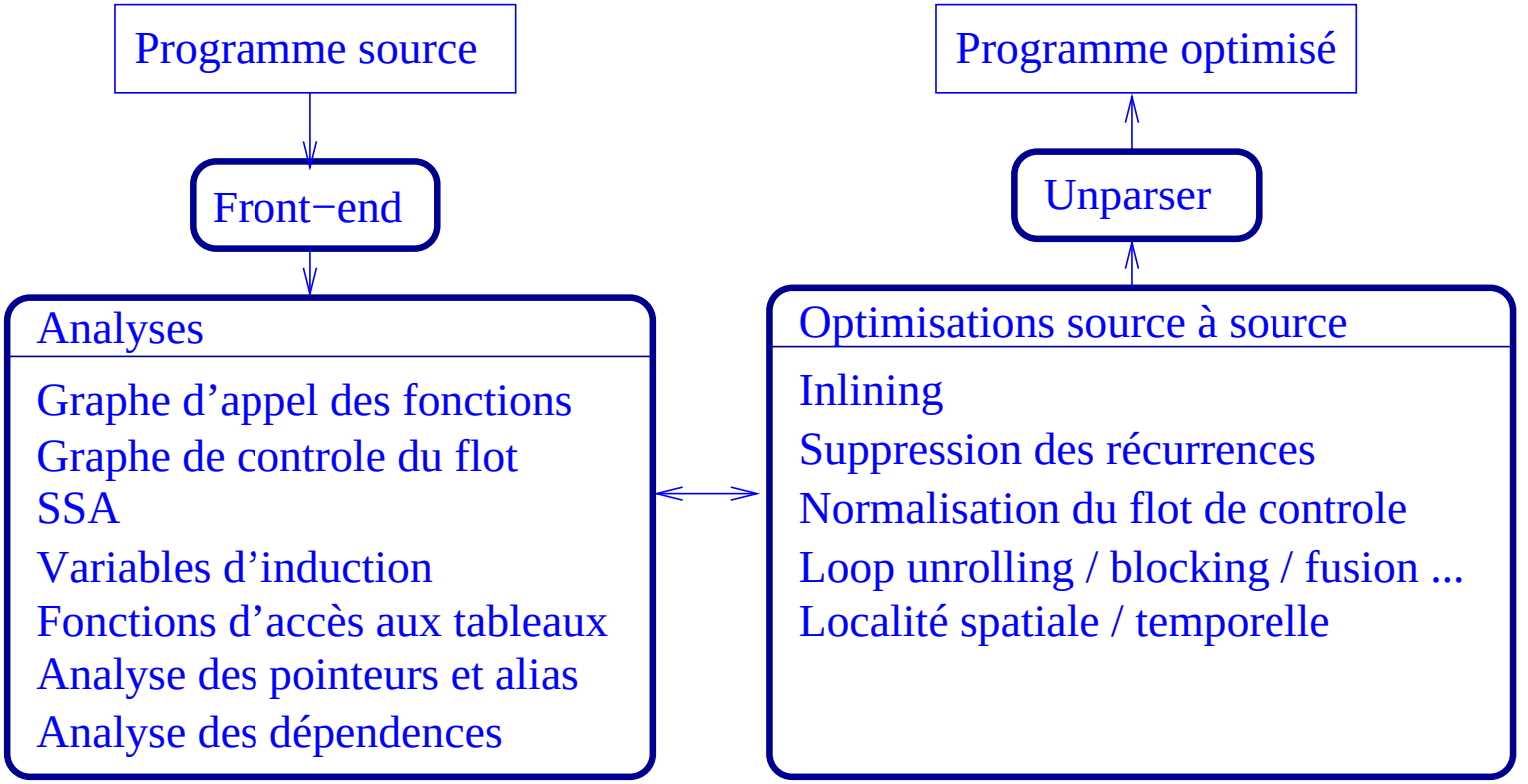
Loop Optimizations

- Après avoir déterminé les variables d'induction,
- il est possible de construire la représentation géométrique du domaine d'itération.
- Pour valider les transformations des boucles, l'analyse des dépendences est nécessaire.
- Ces points restent en développement.

Un compilateur optimiseur



Un compilateur optimiseur



Remerciements

Merci à tous ceux qui ont contribués à la réussite de ce projet :

Remerciements

Merci à tous ceux qui ont contribués à la réussite de ce projet :

- l'équipe ICPS pour l'excellente ambiance,
- Philippe Clauss, Vincent Loechner et Benoît Meister pour leur travail de recherche,
- Catherine Mongenet pour le cours de compil,
- Frédéric Wagner et Diego Novillo pour m'avoir accompagné dans ce projet,
- the FSF for GCC, Debian, Linux and Prosper
- et ma famille.