

Le package fg

Flow Graph

Alexis Nasr
Carlos Ramisch
Manon Scholivet
Franck Dary

Compilation – L3 Informatique
Département Informatique et Interactions
Aix Marseille Université

Graphe d'analyse

- La construction du graphe d'analyse repose sur trois packages
 - Le package `graph`, représentation générique de graphes.
 - Le package `intset`, représentation d'ensembles d'entiers
 - Le package `fg`, représentation de graphe d'analyse.

Le package graph

- Représentation générique de graphes orientés.
- Chaque nœud du graphe est identifié par un entier.
- Les informations associées à chaque nœud du graphe sont représentées “à l’extérieur” du graphe.

La classe Graph

```
public class Graph {  
    int nodecount=0;  
    NodeList mynodes;  
    NodeList mylast;  
    public Node newNode() {...}  
    public void addEdge(Node from, Node to) {...}  
    public void show(java.io.PrintStream out) {...}  
}
```

- Les sommets du graphe sont représentés sous la forme d'une liste chaînée (NodeList)
- mynodes pointe sur le premier sommet de la liste.
- mylast sur le dernier.
- La méthode newNode ajoute un sommet au graphe.
- La méthode addEdge ajoute un arc.

La classe Node

```
public class Node {  
    Graph mygraph;  
    int mykey;  
    NodeList succs;  
    NodeList preds;  
  
    public Node(Graph g) {...}  
}
```

Un nœud :

- est identifié par un entier mykey;
- est associé à un graphe mygraph;
- définit un ensemble d'arcs entrants et un ensemble d'arcs sortants.
- Ces deux ensembles sont représentés par des listes de sommets (NodeList).

La classe NodeList

```
public class NodeList {  
    public Node head;  
    public NodeList tail;  
    public NodeList(Node h, NodeList t) {head=h; tail=t;}
```

- Une liste de sommets est représentée comme une simple liste chaînée.

Le package intset

- Permet de représenter des ensembles d'entiers consécutifs à partir de zéro.
- Structure statique : la cardinalité maximale doit être spécifiée à la création.
- Implémente les opérations ensemblistes standard.

Le package intset

```
public class IntSet{
private int size;
private boolean array[];

public          IntSet(int size){ ...}
public void     empty(){ ...}
public boolean  isMember(int elt){ ...}
public boolean  isEmpty(){ ...}
public void     add(int elt){ ...}
public void     remove(int elt){ ...}
public IntSet   minus(IntSet s){ ...}
public IntSet   copy(){ ...}
public IntSet   inter(IntSet s){ ...}
public IntSet   union(IntSet s){ ...}
public boolean  equal(IntSet s){ ...}
public String   toString(){ ...}
}
```

Le package intset

- Attention, la majorité des méthodes du package modifient l'objet à partir duquel la méthode a été invoquée.
- Exemple :

```
IntSet e1 = new Intset (10);  
e1.add(0);  
IntSet e2 = new Intset (10);  
e2.add(1);  
e1.union(e2);
```

- A l'issue de l'opération d'union (ligne 5), l'ensemble e1 est modifié ($e_1 = e_1 \cup e_2$).
- Pour ne pas modifier e1, il faut d'abord faire une copie à l'aide de la méthode copy.

Le package fg

- Contient deux classes :
 - `Fg`, permet de représenter le graphe d'analyse d'un programme `nasm`.
 - `FgSolution`, calcule une solution pour un graphe d'analyse donné.

La classe Fg

```
public class Fg implements NasmVisitor <Void> {  
    public Nasm nasm;  
    public Graph graph;  
    Map< NasmInst, Node>    inst2Node;  
    Map< Node, NasmInst>    node2Inst;  
    Map< String, NasmInst> label2Inst;  
}
```

- Définit un graphe d'analyse graph.
- Associe à toute instruction nasm un sommet du graphe grâce à deux tables de hash :
 - inst2Node
 - node2inst
- Associe à toute étiquette l'instruction à laquelle elle correspond grâce à la table de hash label2Inst.

La classe FgSolution

```
public class FgSolution{
    int iterNum = 0;
    public Nasm nasm;
    Fg fg;
    public Map< NasmInst, IntSet> use;
    public Map< NasmInst, IntSet> def;
    public Map< NasmInst, IntSet> in;
    public Map< NasmInst, IntSet> out;

    public FgSolution(Nasm nasm, Fg fg) { ... }
}
```

La classe FgSolution

- La classe FgSolution associe à chaque instruction nasm quatre ensembles :
 - use : comporte les numéros des registres qui sont utilisés par l'instruction.
 - def : comporte les numéros des registres qui sont modifiés par l'instruction.
 - in : comporte les numéros des registres qui sont vivants à l'entrée de l'instruction.
 - out : comporte les numéros des registres qui sont vivants à la sortie de l'instruction.
- use et def sont initialisés grâce aux valeurs de use et def des opérandes de l'instruction.

```
public abstract class NasmOperand{  
    public boolean use = false;  
    public boolean def = false;  
}
```

- Les ensembles in et out sont calculés à l'aide de l'algorithme de point fixe.