universitätbonn ROOTS

Intraprocedural Data Flow Analysis

Sönke Gluch

gluch@cs.uni-bonn.de

Overview

- Motivation
- Analysis
  - Reaching Definitions
  - Alias Analysis
  - Derived Data Flow Information
    - ⇨ Use-Definition-Chains
    - ⇨ Definition-Use-Chains
- Transformations
  - Constant Propagation
  - Copy Propagation
- Efficient Implementation

Motivation

- Question
  - Where do my values come from?
  - Where do my values flow to?
- Application
  - Dead Code Elimination
  - Constant Propagation
  - Debugging Help
  - Compiler Optimization

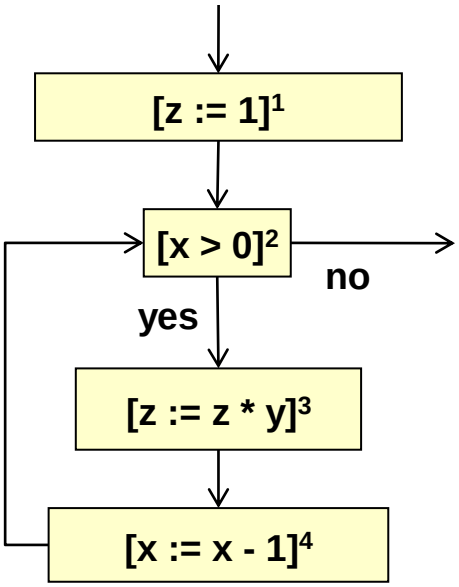
```
public class Sample {

    int m(int x){
      int y = x;
      int z = 1;
      while (y > 1) {
        z = z*y;
        y = y-1;
      }
      y = 0;
      return z;
    }

    public static void main(String[] args) {
      Sample s = new Sample();
      System.out.println("m(4) = " + s.m(4) );
    }

}
```

Introduction

- Data Flow Analysis
  - Based on control flow graph
    - ⇨ Nodes are statements (blocks)
  - Intraprocedural
  - 2 Equation Classes
    - ⇨ Entry Information
      - – Entry of starting block always well defined
    - ⇨ Exit Information
  - Analysis Direction
    - ⇨ Forward
      - – From exit of a block to entry of its successor
      - – Examining the future of a statement
    - ⇨ Backward
      - – From entry of a block to exit of its predecessor
      - – Examining the past of a statement

Auxiliary functions

- Example
  - power
    - ⇨ $[z := 1]^1$; while $[x > 0]^2$ do ($[z := z * y]^3$; $[x := x - 1]^4$)
  - init(power)
    - ⇨ 1
  - final(power)
    - ⇨ {2}
  - labels(power)
    - ⇨ {1, 2, 3, 4}
  - blocks(power)
    - ⇨ {$[z := 1]^1$} ∪ blocks(while $[b]^2$ do (S^3; S^4))
    - ⇨ {$[z := 1]^1$} ∪ {$[x > 0]^2$} ∪ blocks(S^3; S^4)
    - ⇨ {$[z := 1]^1$, $[x > 0]^2$, $[z := z * y]^3$, $[x := x - 1]^4$}
  - flow(power)
    - ⇨ {(1,2),(2,3),(3,4),(4,2)}



Reaching Definitions

- Reaching Definitions
  - Is an assignment to t able to affect the value of t at a another point in the program?
    - ⇨ Does a definition d reach a statement s through the control flow graph?
  - Forward Analysis
  - Goal
    - ⇨ Which assignments might already exist at every point in the program and have not yet been overwritten?
  - Applications
    - ⇨ Basis for many transformations and further analysis
  - Example
    - ⇨ $[x := 5]^1$; $[y := 1]^2$; while $[x > 1]^3$ do ($[y := x * y]^4$; $[x := x - 1]^5$)
      - $Rd_{entry}(4) = \{(x,1),(y,2),(y,4),(x,5)\}$
      - $Rd_{exit}(4) = \{(x,1),(y,4),(x,5)\}$
      - $Rd_{entry}(5) = \{(x,1),(y,4),(x,5)\}$
      - $Rd_{exit}(5) = \{(y,4),(x,5)\}$

Reaching Definitions

- kill and gen Functions

$$kill_{RD}, gen_{RD} : Blocks_* \rightarrow P(Var_* \times Lab_*^?)$$

$$kill_{RD}([x := a]^l) = \{(x, ?)\} \cup \{(x, l') | B^{l'}\}$$
$$kill_{RD}([skip]^l) = \{\}$$
$$kill_{RD}([b]^l) = \{\}$$

$$gen_{RD}([x := a]^l) = \{(x, l)\}$$
$$gen_{RD}([skip]^l) = \{\}$$
$$gen_{RD}([b]^l) = \{\}$$

- Entry and Exit Functions

$$RD_{entry}, RD_{exit} : Blocks_* \rightarrow P(Var_* \times Lab_*^?)$$

$$RD_{entry}(l) = \left\{ \begin{array}{c} \{(x, ?) | x \in FV(S_*)\} \quad falls \quad l = init(S_*) \\ \bigcup \{RD_{exit}(l') | (l', l) \in flow(S_*)\} \end{array} \right\}$$

$$RD_{exit}(l) = \begin{array}{c} (RD_{entry}(l) \backslash kill_{RD}(B^l)) \cup gen_{RD}(B^l) \\ wobei \quad B^l \in blocks(S_*) \end{array}$$

Reaching Definitions

- Entry and Exit Functions

$$RD_{entry}, RD_{exit} : Blocks_* \rightarrow P(Var_* \times Lab_*^?)$$

$$RD_{entry}(l) = \left\{ \begin{array}{c} \{(x,?) | x \in FV(S_*)\} \quad falls \quad l = init(S_*) \\ \bigcup \{RD_{exit}(l') | (l',l) \in flow(S_*)\} \end{array} \right\}$$

$$RD_{exit}(l) = \begin{array}{c} (RD_{entry}(l) \backslash kill_{RD}(B^l)) \cup gen_{RD}(B^l) \\ wobei \quad B^l \in blocks(S_*) \end{array}$$

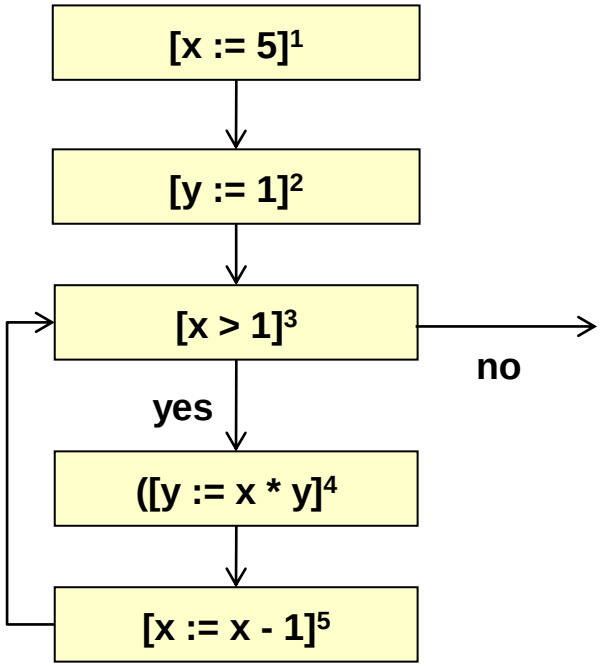
- Algorithm

```
OUT[ENTRY] = Ø;
for (each basic block B other than ENTRY) OUT[B] = Ø;
while (changes to any OUT occur)
    for (each basic block B other than ENTRY) {
        IN[B] = U_{p∈pred[B]} OUT[p];
        OUT[B] = gen[B] U (IN[B] - kill[B]);
    }
```

Reaching Definitions

- $[x := 5]^1; [y := 1]^2;$ while $[x > 1]^3$ do $([y := x * y]^4; [x := x - 1]^5)$

l	$kill_{RD}(l)$	$gen_{RD}(l)$
1	{(x,?),(x,1),(x,5)}	{(x,1)}
2	{(y,?),(y,2),(y,4)}	{(y,2)}
3	{}	{}
4	{(y,?),(y,2),(y,4)}	{(y,4)}
5	{(x,?),(x,1),(x,5)}	{(x,5)}



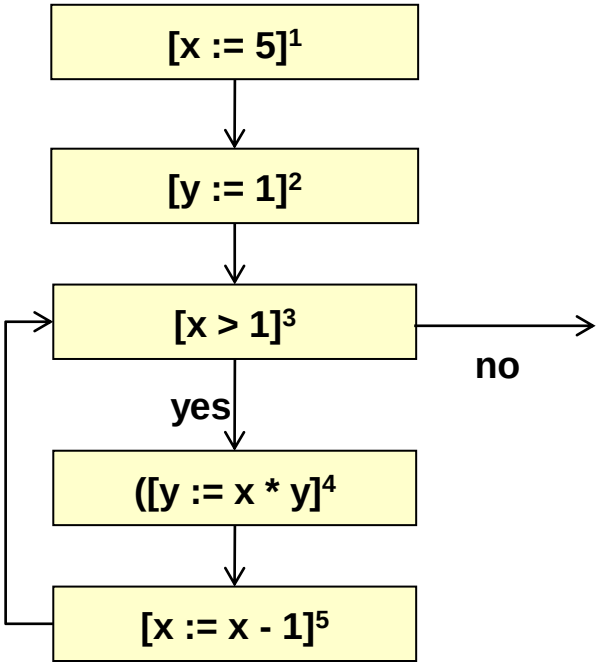
$$RD_{entry}(1) = \{(x, ?), (y, ?)\}$$
$$RD_{entry}(2) = RD_{exit}(1) RD_{entry}(3) = RD_{exit}(2) \cup RD_{exit}(5)$$
$$RD_{entry}(4) = RD_{exit}(3)$$
$$RD_{entry}(5) = RD_{exit}(4)$$

$$RD_{exit}(1) = (RD_{entry}(1) \setminus \{(x, ?), (x, 1), (x, 5)\}) \cup \{(x, 1)\}$$
$$RD_{exit}(2) = (RD_{entry}(2) \setminus \{(y, ?), (y, 2), (y, 4)\}) \cup \{(y, 2)\}$$
$$RD_{exit}(3) = RD_{entry}(3)$$
$$RD_{exit}(4) = (RD_{entry}(4) \setminus \{(y, ?), (y, 2), (y, 4)\}) \cup \{(y, 4)\}$$
$$RD_{exit}(5) = (RD_{entry}(5) \setminus \{(x, ?), (x, 1), (x, 5)\}) \cup \{(x, 5)\}$$

Reaching Definitions

- $[x := 5]^1; [y := 1]^2;$ while $[x > 1]^3$ do $([y := x * y]^4; [x := x - 1]^5)$

l	$RD_{entry}(l)$	$RD_{exit}(l)$
1	{(x,?),(y,?)}	{(y,?),(x,1)}
2	{(y,?),(x,1)}	{(x,1),(y,2)}
3	{(x,1),(y,2),(y,4),(x,5)}	{(x,1),(y,2),(y,4),(x,5)}
4	{(x,1),(y,2),(y,4),(x,5)}	{(x,1),(y,4),(x,5)}
5	{(x,1),(y,4),(x,5)}	{(y,4),(x,5)}



- Result
  - Reachability of Assignments

Alias Analysis

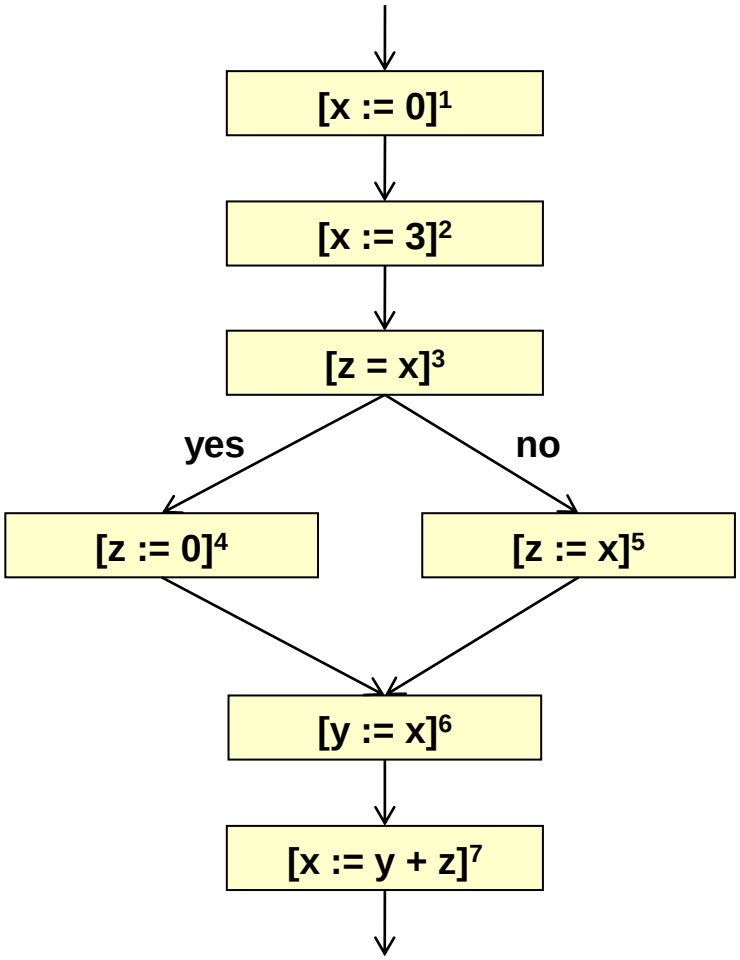
- Alias Analysis
  - Many values not included so far
    - ⇨ Call-by-reference Parameters
    - ⇨ Pointers
    - ⇨ Arrays
  - may-alias relation
    - ⇨ p may-alias q, if both may point to the same memory location
  - type-based
    - ⇨ divide relevant memory locations of the same type into alias classes
    - ⇨ Let i, j be alias classes and $M_i[x]$, $M_j[y]$ memory locations
      - then $M_i[x]$ may-alias $M_j[y]$, if i = j
  - flow-based
    - ⇨ new alias-class for each point of creation
      - C: `malloc`
      - Java, Pascal: `new`
  - Application
    - ⇨ Increasing quality and accuray of data flow analysis

Derived Data Flow Information

- Connection between variable-defining and variable-using program parts
- Two Directions
  - Use-Definition-Chains (ud-chains)
    - ⇨ returns the label of the block defining the used variable
  - Definition-Use-Chains (du-chains)
    - ⇨ returns the labels of blocks using the defined variable
- Applications
  - Dead Code Elimination
  - Code Motion (Hoisting)
  - Common Subexpression Elimination
  - Constant Propagation
  - Copy Propagation

Derived Data Flow Information

- $[x := 0]^1; [x := 3]^2; (\text{if } [z = x]^3 \text{ then } [z := 0]^4 \text{ else } [z := x]^5); [y := x]^6; [x := y + z]^7$



du(x,l)	x	y	z
1	{}	{}	{}
2	{3, 5, 6}	{}	{}
3	{}	{}	{}
4	{}	{}	{7}
5	{}	{}	{7}
6	{}	{7}	{}
7	{}	{}	{}
?	{}	{}	{3}

ud(x,l)	x	y	z
1	{}	{}	{}
2	{}	{}	{}
3	{2}	{}	{?}
4	{}	{}	{}
5	{2}	{}	{}
6	{2}	{}	{}
7	{}	{6}	{4,5}

Data Flow

Transformations

Constant Propagation

- Constant Propagation
  - Replaces variables with constants

Let $[t = c]^d$ and $[y \leftarrow t \otimes x]^n$ be statements and c constant. Then t is constant in n if d and no other Definition of t reaches n. In this case we can rewrite n as $[y \leftarrow c \otimes x]^n$.

Copy Propagation

- Copy Propagation
  - Replaces unchanged variables

Let $[t \leftarrow z]^d$ and $[y \leftarrow t \otimes x]^n$ be statements and z a variable. If d and no other definition of t reaches n, and there is no definition of z on any path from d to n, then we can rewrite n as $[y \leftarrow z \otimes x]^n$.

Data Flow

Efficient Implementation

Efficient Implementation

- Strategies to avoid repeated calculation of data flow information
  - Cutoff
    - ⇨ Constrain the number of analysis rounds
  - Cascading Analysis
    - ⇨ Predict transformation effects
    - ⇨ Inkremental analysis
    - ⇨ Patch data flow information

- ud-/du-chains
  - allow efficient implementation of transformation algorithms
  - SSA
    - ⇨ Static Single-Assignment Form
    - ⇨ Provides more informationen
    - ⇨ Very efficient computation

Efficient Implementation

- Basic Blocks
  - Combination of gen/kill-effects of consecutive statements
  - gen and kill functions
    - ⇨ Block B with only one predecessor A
    - ⇨ Block A with only one successor B
    - ⇨ Outgoing definitions of block B
      - `OUT[B] = gen[B]∪(IN[B]−kill[B])`
    - ⇨ `IN[B] = OUT[A]`, so
      - `OUT[B] = gen[B]∪((gen[A]∪(IN[A]−kill[A]))−kill[B])`
    - ⇨ Thus
      - `OUT[B] = gen[B]∪(gen[A]−kill[B])∪(IN[A]−(kill[A]∪kill[B]))`
    - ⇨ Block AB now combines the gen/kill-effects of A and B
      - `gen[AB] = gen[B] ∪ (gen[A] − kill[B])`
      - `kill[AB] = kill[A] ∪ kill[B]`

Efficient Implementation

- Ordering the Nodes of the control flow graph
  - Optimal, if each block is being processed after its predecessor
    - ⇨ Cykles
      - – Sort quasi-topologically per DFS

```
repeat
    for (i = 1 to N) {
          B = sorted[i];
          IN[B] = U_{p∈pred[B]} OUT[p];
          OUT[B] = gen[B] U (IN[B] – kill[B]);
    }
until (no changes to any OUT occur);
```

```
N = Number of Blocks;
for (each Block B) mark[B] = false;
DFS(ENTRY);
```

```
DFS(B) {
    if (mark[B] = false) {
          mark[B] = true;
          for (each Successor S of B) DFS(S);
          sorted[N] = B;
          N = N – 1;
    }
}
```

Efficient Implementation

- Work-List Algorithms
  - Many equations do not get changed during iterations
  - Manage set of outgoing definitions OUT
    - ⇨ Work-List contains blocks, that have to be recalculated
    - ⇨ On change of the set OUT of a block, all the successors are put into the list
  - Faster with sorted control flow graph

```
W = all Blocks;
while (W is not empty) {
   remove a block B from W;
   old = OUT[B];
   IN[B] = U_{p∈pred[B]} OUT[p];
   OUT[B] = gen[B] U (IN[B] – kill[B]);
   if (old ≠ OUT[B]) {
      for (each successor S of B)
         if (S ∉ W) put S into W;
   }
}
```

Overview

- Motivation
- Analysis
  - Reaching Definitions
  - Alias Analysis
  - Derived Data Flow Information
    - ⇨ Use-Definition-Chains
    - ⇨ Definition-Use-Chains
- Transformations
  - Constant Propagation
  - Copy Propagation
- Efficient Implementation

Sources

- [NNH05]
  - Flemming Nielson, Hanne Riis Nielson, Chris Hankin: Principles of Program Analysis (Springer, 2005)
- [A02]
  - Andrew W. Appel: Modern compiler implementation in Java (Cambridge University Press, 2002)
- [ALSU06]
  - Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman: Compilers: Principles, Techniques, and Tools (Addison Wesley, 2006)