Optimizing Compilers
5th Lecture

Bernhard Scholz
Institut f. Computersprachen
Argentinierstr. 8
scholz@complang.tuwien.ac.at

Outline

- Introduction
- Basic Concepts
- Data Flow Equations
- Solver

Introduction

- Data flow analysis determines ***static*** properties of programs
- Data flow analysis is a unified theory
- Provides information for global analysis
- Examples of DFA Problems:
  - Register Allocation: Keep two non-overlapping temporaries in the same register.
  - Common-Subexpression-Elimination: Eliminate expressions which are computed more than once.
  - Constant Folding: Compute constant expressions at compile-time.
  - Dead-Code Elimination: Delete a useless computation
- "DFA solutions are pessimistic"
- DFA based on CFG and node properties

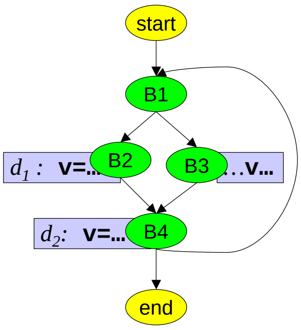
Reaching Definition

- Assignment of variable can directly affect the value at another point
- Unambiguous Definition *d* of variable *v*

```
d: v = <expression>;
```

- Definition reaches a statement *u* if all paths from *d* to *u* does not contain any unambiguous statements of *v*
- Functions can have side-effects to variables (not in miniC!)

Example: Reaching Definition

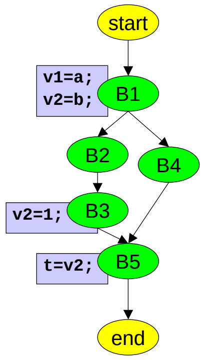


- Unambiguous definitions d_1 and d_2 of variable *v*
- Might reach d_1 node B3?
- Might reach d_2 node B3?
- Paths and effects of basic blocks influence solution
- Forward problem

Liveness Analysis

- Any use of variable *v* makes *v* ***alive***, and any definition kills *v*.
- Register allocation:
  Liveness for determining live ranges.
- Dead Code Elimination:
  Definitions of *v* can be eliminated if variable v is not alive on the path between definition and exit node.

Example: Liveness Analysis



- Use of variable $v2$ in B5 makes $v2$ alive.
- $v2$ is killed in B3
- $v2$ is not alive in B2
- $v2$ is alive in B4
- Liveness information is propagated backwards
- $v1$ is defined but never alive
- $v1$ might be eliminated!

Constant Propagation

- Assignment of a variable v with a constant value c

```
v = c;
```

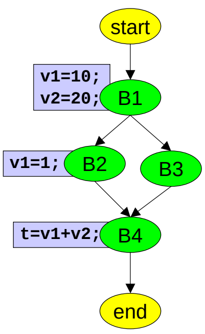
- Variable v can be replaced in a statement u if there is no other definition of v that reaches u.
- Replacement:

```
u: t = …v…;
```

⇩

```
u: t = …c…;
```

Example: Constant Propagation



- Assignments with constants in B1
- Assignment with constant in B2
- Has variable t a constant value?
- What is if t would become a constant?

Basic Concepts

- Data flow information represented as semi-lattice
- Elements of lattice abstract properties of program
- Various types of lattices (bit-vector, constants,...)
- Lattice induces partial ordered set(POR)
- Data flow functions model effect of basic blocks
- Data flow equations
- relations of control flow and effects of basic blocks
- Data flow solutions

Semilattices

- Semi-lattice L for representing DFA information
- L is an algebraic structure $L\langle \wedge, \perp, ? \rangle$
- L consists of a set of values: $L=\{x_1, x_2, ...\}$
- L has a meet operator $z=x \wedge y$, where $x,y,z \in L$
- Two unique elements of L: $\perp$, ? (bottom,top)
- L might have infinite number of elements
- Height of semilattice is finite
- L can be an algebraic product:
  $L= L_1 \times L_2 \times .... \times L_k$

Properties of Meet Operator

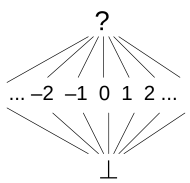
- For all $x,y \in L$ there exists a unique $z \in L$
  $z=x \wedge y$ (closure)
- For all $x,y \in L$:
  $x \wedge y = y \wedge x$ (commutativity)
- For all $x,y,z \in L$:
  $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ (associativity)
- For all $x \in L$:
  $(x \wedge \perp) = \perp$
- For all $x \in L$:
  $(x \wedge ?) = x$

Partial Order

- Meet operator induces a partial order ($\leq$) on values in L:
  $x \leq y \Leftrightarrow x \wedge y = x$
- Interpretation: If $x \leq y$ then value x has less information than value y.
- Partial order has following properties:
  - Transitivity (if $x \leq y$ *and* $y \leq z$, then $x \leq z$)
  - Antisymmetry (if $x \leq y$ *and* $y \leq x$, then $x = y$)
  - Reflexivity (for all x, $x \leq x$)
- Strict partial order: $x<y \Leftrightarrow x \wedge y=x$ and $x \neq y$

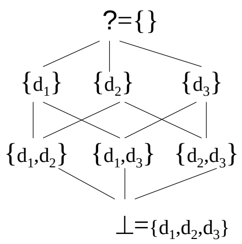
Examples of Semilattices

Constant Propagation



- Infinite number of elements
- Top: Any constant
- Bottom: not a constant

Reaching Definition



- Meet operator: set union
- Top: no RD
- Bottom: all RD

Data Flow Functions

- Effect of basic blocks is represented as function $f{:}L \rightarrow L$
- Useful properties for f
  - Distributivity: $f(x \wedge y) = f(x) \wedge f(y)$
  - Monotoncity: $f(x \wedge y) \leq f(x) \wedge f(y)$
  - Closure: $f(g(x)) = f\,g(x)$
- Mapping between nodes in CFG and functions
  $M(n)[x] = f_n(x)$
  where f is function of node n
- Basic blocks with no effects: identity function
- Mapping extension for paths:
  $M(\langle n_1,n_2,\ldots, n_k\rangle)[x] = f_k \ldots f_2\, f_1\,(x)$

Meet Over All Path Solution

- Forward/Backward Problem
- E.g., reaching definition ⇒ forward problem
  liveness analysis ⇒ backward problem
- MOP solution for a forward problem

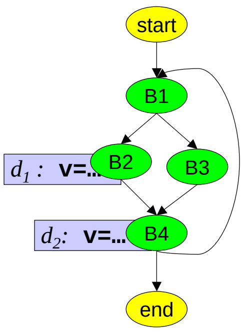
$$MOP(n) = \bigwedge_{\pi \in Path(start,n)} M(\pi)[c]$$

- MOP solution for a backward problem

$$MOP(n) = \bigwedge_{\pi \in ReversePath(end,n)} M(\pi)[c]$$

- c is prescribed information for start node
- MOP is the composition of data flow functions along all possible paths by propagating c and applying the meet operator.

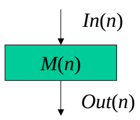
Example: MOP



- Reaching Definition
- Data flow information: $L=2^D$ *and* $D=\{d_1,d_2\}$
- Meet Operator: set-union
- MOP describe solution of infinite number of paths
- MOP for B1
  $M(\pi)$[<start,B1>](c) ∧
  $M(\pi)$[<start,B1,B3,B4,B1>](c) ∧
  $M(\pi)$[<start,B1,B2,B4,B1>](c) ∧
  ....

Data Flow Equations

- How can MOP be computed in finite steps?
- Data flow equations describe control flow and effect of basic blocks



Forward problems	Backward problems
$Out(start)$ = c	$Out(end)$ = c
$In(n) = \bigwedge_{p \in preds(n)} Out(p)$	$In(n) = \bigwedge_{s \in succs(n)} Out(s)$
$Out(n) = M(n)[In(n)]$	$Out(n) = M(n)[In(n)]$

<u>Lemma:</u> If monotone problem is distributive $[f(x \wedge y) = f(x) \wedge f(y)]$, then ***maximal fixpoint*** of data flow equations is equal to MOP

Gen/Kill Functions

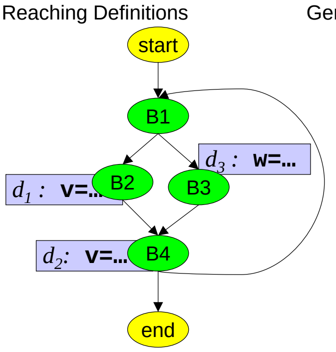
- Most problems have a power set of a data flow fact set D as semilattice ($L=2^D$)
- Meet operator: set-union, set-intersection
- DFA computes which facts hold at a program point
- Functions represented by two constant sets
  - gen-set $gen(n)$ of a node n (generated data facts)
  - kill-set $kill(n)$ of a node n (killed data facts)
- Algebraic relation: $M(n)[x] = (x\text{-}kill(n)) \cup gen(n)$
- Bit-vectors

Dataflow facts $d_1\, d_2\, d_3\, d_4$

Bitvector: 0110 $\Longleftrightarrow$ $\{d_2, d_3\}$

Example: Data Flow Equations

Reaching Definitions

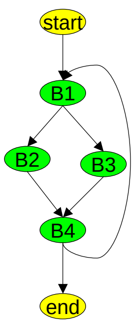


Gen/Kill Sets

Node	Gen	Kill
Start	{}	{}
B1	{}	{}
B2	$\{d_1\}$	$\{d_2\}$
B3	$\{d_3\}$	{}
B4	$\{d_2\}$	$\{d_1\}$
end	{}	{}

Example cont'd

CFG



Gen/Kill sets and data flow equations

Node	Gen	Kill	In	Out
Start	{}	{}		{}
B1	{}	{}	Out(start) ∪ Out(B4)	In(B1)
B2	$\{d_1\}$	$\{d_2\}$	Out(B1)	[In(B2)- $\{d_2\}$] ∪ $\{d_1\}$
B3	$\{d_3\}$	{}	Out(B1)	In(B3) ∪ $\{d_3\}$
B4	$\{d_2\}$	$\{d_1\}$	Out(B2) ∪ Out(B3)	[In(B4)- $\{d_1\}$] ∪ $\{d_2\}$
end	{}	{}	Out(B4)	In(B4)

Solvers

- How to solve data flow equations?
  - Iterative Approaches
  - Algebraic Approaches
- Which approach is better?
  - Algebraic properties of DFA
  - Complexity of solver
  - Implementation effort
  - Interprocedural/Intraprocedural Analysis

Iterative Solvers

- Solution Criteria
  - Semilattice with finite height
  - DFA problem must be monotone
  - For obtaining MOP problem must be distributive
- Iterative approach (Fix-Point-Algorithm)
  - start with a non-solution for each node

    $Out(n)=?$
  - Iterate computation of Out-values in arbitrary order
  - Stop if Out-values are stable for all nodes

Algorithm for Gen/Kill-Problems

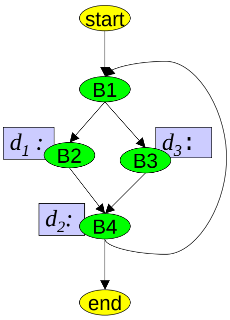
- Meet Operator $\wedge$
  - union($\cup$) or intersection($\cap$)
- Initialization for $\cup$
  - $Out(start)=c$
  - $In(u)=\{\}$
  - $Out(u)=Gen(u)$
- Initialization for $\cap$
  - $Out(start)=c$
  - $In(u)=L$
  - $Out(u)=(L-Kill(u))\cup Gen(u)$
- Backward problems
  - preds $\Rightarrow$ succs
- Complexity: $O(n^3)$

Algorithm

```
Repeat
 s = true;
 for all u∈N-{start} do
  for all v∈preds(u) do
   In(u)=In(u)∧Out(v)
  endfor
  X=(In(u)-Kill(u))
      ∪Gen(u)
  s = s and [Out(u) = X]
  Out(u) = X
 endfor
until s;
```

Example Reaching Definitions

CFG



Iteration Steps

Node	In_0	Out_0	In_k	Out_k
Start	{}	{}	{}	{}
B1	{}	{}	$\{d_2,d_3\}$	$\{d_2,d_3\}$
B2	{}	$\{d_1\}$	$\{d_2,d_3\}$	$\{d_1,d_3\}$
B3	{}	$\{d_3\}$	$\{d_2,d_3\}$	$\{d_2,d_3\}$
B4	{}	$\{d_2\}$	$\{d_1,d_2,d_3\}$	$\{d_2,d_3\}$
end	{}	{}	$\{d_2,d_3\}$	$\{d_2,d_3\}$

- Stable after two iterations
- Solution equal to MOP

Stop

- Next lecture: 15.5.2003, 13:45 – 14:45