Lattice-Theoretic Data-Flow Framework and Intro to SSA

Last Time

- Started lattice theoretic frameworks for Data-flow analysis [Aho,Sethi,Ullman,Lam]

Today

- Complexity and correctness of IDFA
- Affect of program representation on data-flow analysis efficiency
- Static single assignment (SSA) form
  - Program representation for sparse data-flow

Next Time

- SSA complications

Bitwidth Analysis Paper

Why did we read this paper?

Can all dataflow analyses be defined in terms of Gen and Kill?

Do all dataflow analysis problems operate on sets?

What questions arise from reading this paper?

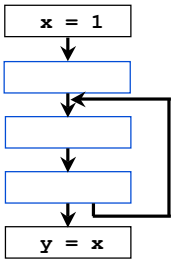
Implementing Simple Constant Propagation

Standard worklist algorithm

- Identifies simple constants
- For each program point, maintains one constant value for each variable
- O(EV) (E is the number of edges in the CFG; V is number of variables)

Problem

- Inefficient, since constants may have to be propagated through irrelevant nodes



Solution

- Exploit a sparse dependence representation (*e.g.,* du-chains, SSA)

Data Dependence

Definition

- Data dependences are constraints on the order in which statements may be executed

We say statement s_2 depends on s_1

- **Flow (true) dependence**: s_1 writes memory that s_2 later reads (RAW)
- **Anti-dependence**: s_1 reads memory that s_2 later writes (WAR)
- **Output dependences**: s_1 writes memory that s_2 later writes (WAW)
- **Input dependences**: s_1 reads memory that s_2 later reads (RAR)

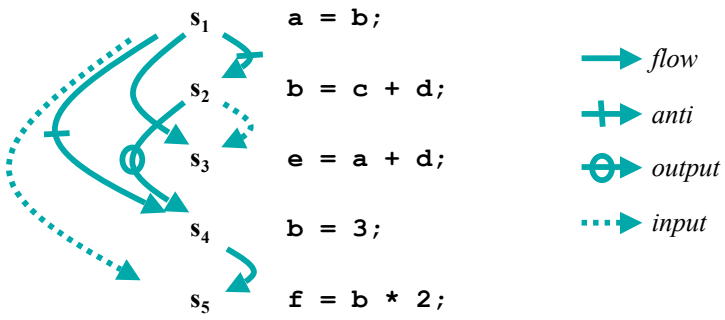
True dependences

- Flow dependences represent actual flow of data

False dependences

- Anti- and output dependences reflect reuse of memory, not actual data flow; can often be eliminated

Example



Representing Data Dependences

Implicitly

- Using variable defs and uses
- Pros: simple
- Cons: hides data dependence (analyses must find this info)

Def-use chains (du chains)

- Link each def to its uses
- Pros: explicit; therefore fast
- Cons: must be computed and updated, space consuming

Alternate representations

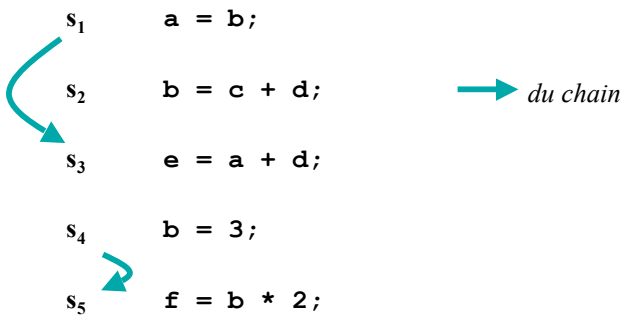
- *e.g.,* Static single assignment form (SSA), dependence flow graphs (DFG), value dependence graphs (VDG)

DU Chains

Definition

– du chains link each def to its uses

Example

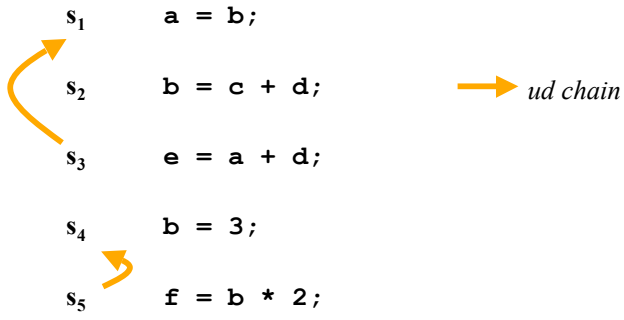


UD Chains

Definition

– ud chains link each use to its defs

Example



Static Single Assignment (SSA) Form

Idea

- Each variable has only one static definition
- Makes it easier to reason about values instead of variables
- Similar to the notion of functional programming

Transformation to SSA

- Rename each definition
- Rename all uses reached by that assignment

Example

```
    v := ...                    v0 := ...
... := ... v ...      ⇒     ... := ... v0 ...
    v := ...                    v1 := ...
... := ... v ...            ... := ... v1 ...
```

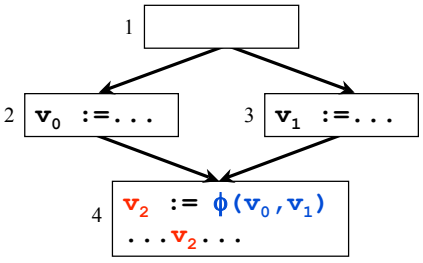
What do we do when there's control flow?

SSA and Control Flow (cont)

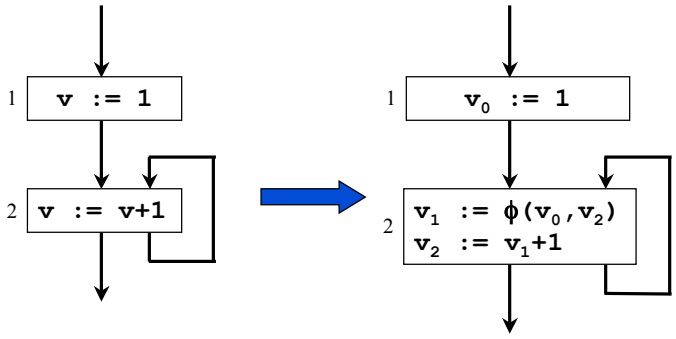
Merging Definitions

- ϕ-functions merge multiple reaching definitions

Example



Another Example



SSA vs. ud/du Chains

SSA form is more constrained

Advantages of SSA

- More compact
- Some analyses become simpler when each use has only one def
- Value merging is explicit
- Easier to update and manipulate?

Furthermore

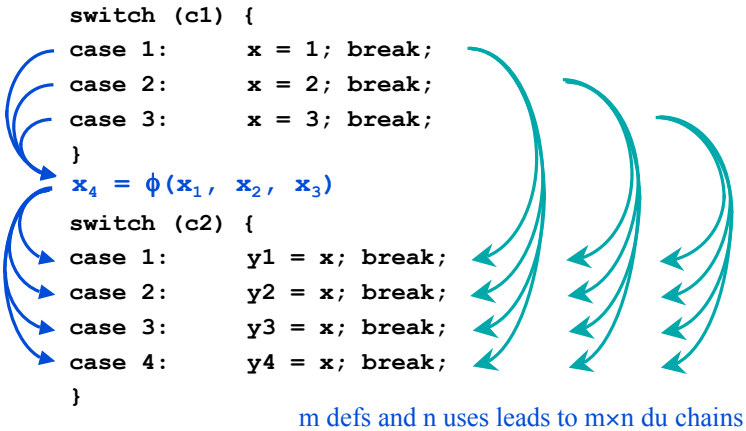
- Eliminates false dependences (simplifying context)

```
for (i=0; i<n; i++)
     A[i] = i;
for (i=0; i<n; i++)
     print(foo(i));
```

Unrelated uses of `i` are given different variable names

SSA vs. ud/du Chains (cont)

Worst case du-chains?



m defs and n uses leads to m×n du chains

Reality Check

How can we handle aliasing in SSA?

What about backward data-flow analysis problems?

How do we transform SSA and generate code from SSA?

Concepts

Lattice-theoretic framework used to prove
- Correctness
- Complexity
- Accuracy

Data dependences
- Three kinds of data dependences
- du-chains

Alternate representations
- du-chains
- SSA

Reality check
- aliasing?
- backward data-flow?
- transforming from SSA to code?

Next Time

Assignments
- Schedule for project 2 due Wednesday

Lecture
- Discuss those SSA reality checks