CS443: Compiler Construction

Lecture 14: Dataflow Analysis

Stefan Muller

Based on material by Steve Zdancewic

Dataflow algorithm can be used for more than just liveness analysis

- Reaching definitions analysis
- Available expressions analysis
- Alias Analysis
- Constant Propagation

Generalized dataflow analysis: produce a set of “facts” in and out of each node

- Every statement (node):
  - Produces (**gen**erates) some set of facts
  - Eliminates (**kill**s) some set of facts
- Constraints at each node computed from other nodes based on constraints (somewhat) specific to the analysis

Dataflow analysis in 4 steps

1. Define facts, gen, kill
2. Define constraints
3. Convert constraints to equations
   - Sets should increase or decrease monotonically
4. Initialize facts for each node
   - Initial value should be consistent with whether sets are increasing or decreasing

Liveness analysis as a dataflow analysis (Steps 1-2)

- Facts: Live variables
- gen[n] = use[n]
- kill[n] = def[n]

- Constraints:
  - in[n] $\supseteq$ gen[n]
  - out[n] $\supseteq$ in[n’] if n’ $\in$ succ[n]
  - in[n] $\supseteq$ out[n] / kill[n]

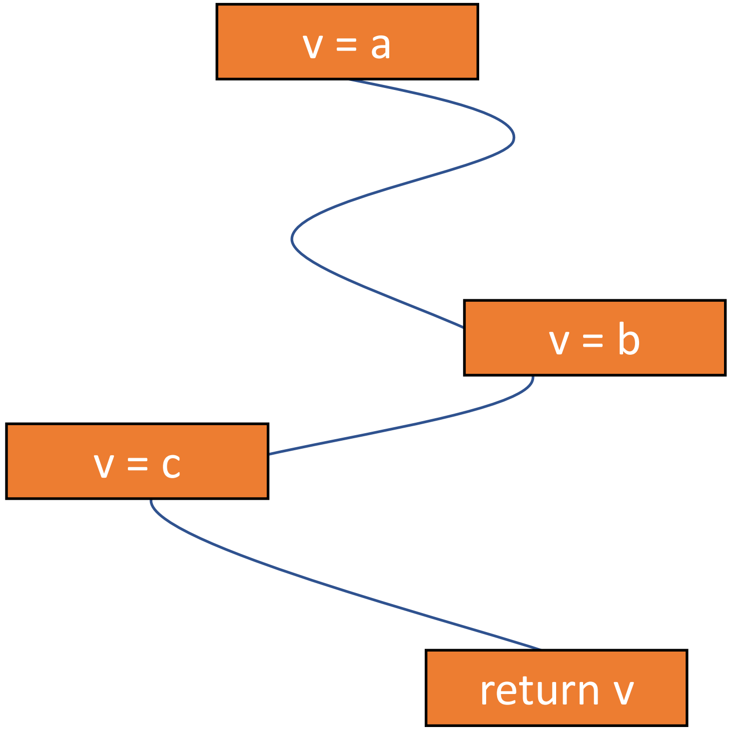
Liveness analysis as a dataflow analysis (Steps 3-4)

- Equations:
  - $out[n] := \cup_{n' \in succ[n]} in[n']$
  - $in[n] := gen[n] \cup (out[n] / kill[n])$
- Initial values:
  - $out[n] := \emptyset$
  - $in[n] := \emptyset$

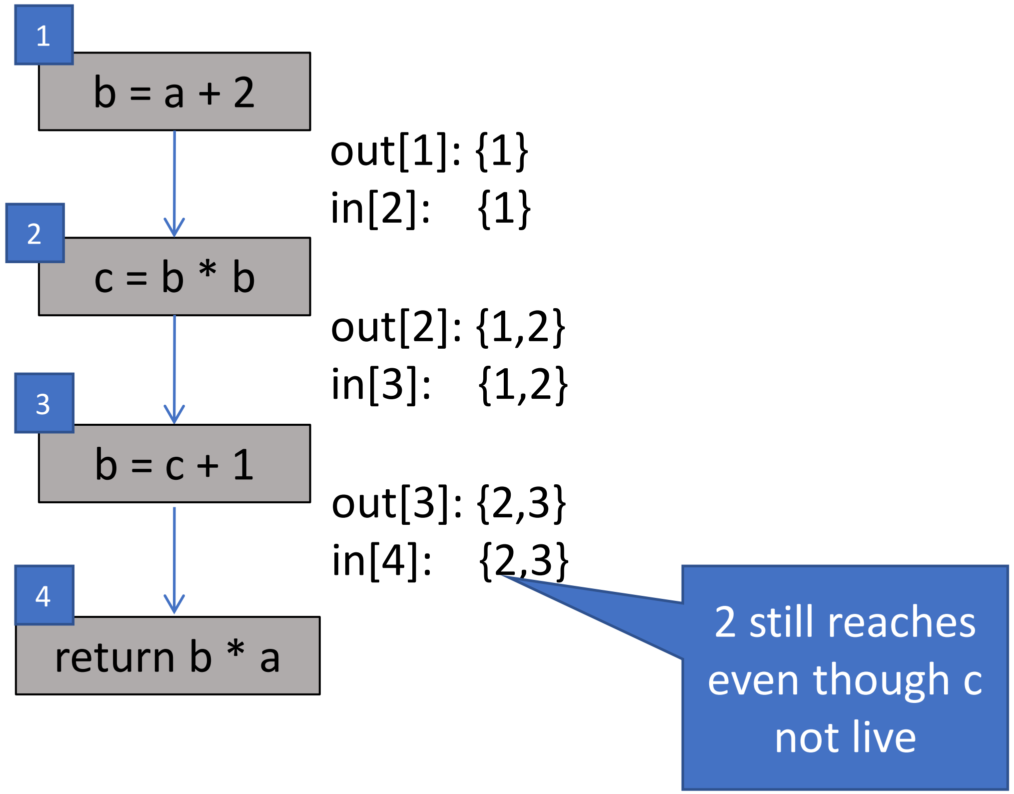
Dataflow algorithm can be used for more than just liveness analysis

- Reaching definitions analysis
- Available expressions analysis
- Alias Analysis
- Constant Propagation

Recall from last time: a variable might be live for a long time, but w/ different definitions

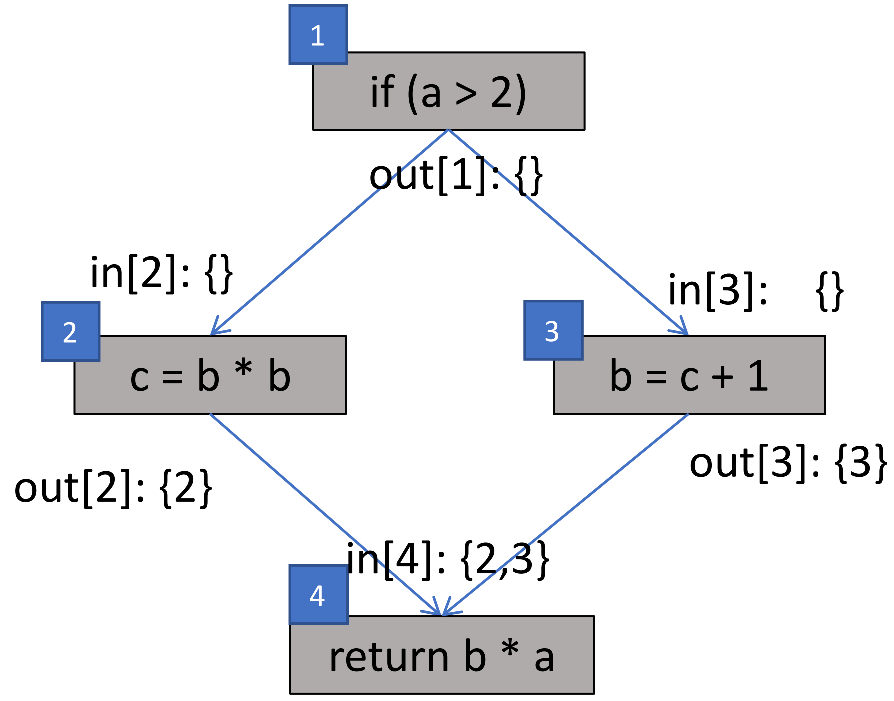


Reaching definitions:
What *definitions* of a var might reach a node?



Reaching definitions:

What *definitions* of a var might reach a node?



Reaching definitions as a dataflow analysis (Step 1)

- Facts: set of nodes whose definition of a variable reaches n
- Let defs[a] be the set of *nodes* that define the variable a

n	gen[n]	kill[n]
a = b op c	{n}	defs[a] - {n}
a = load b	{n}	defs[a] - {n}
store b, a	Ø	Ø
a = f(b_1,…,b_n)	{n}	defs[a] - {n}
f(b_1,…,b_n)	Ø	Ø
br L	Ø	Ø
br a L1 L2	Ø	Ø
return a	Ø	Ø

Reaching definitions as a dataflow analysis (Step 2)

- $out[n] \supseteq gen[n]$
- $in[n] \supseteq out[n']$ if n' is in pred[n]
- $out[n] \cup kill[n] \supseteq in[n]$
  - Equivalently: $out[n] \supseteq in[n] / kill[n]$

Reaching definitions as a dataflow analysis (Steps 3-4)

- $in[n] := \bigcup_{n' \in pred[n]} out[n']$
- $out[n] := gen[n] \cup (in[n] / kill[n])$

- Algorithm: initialize in[n] and out[n] to ∅

Dataflow algorithm can be used for more than just liveness analysis

- Reaching definitions analysis
- Available expressions analysis
- Alias Analysis
- Constant Propagation

When is this optimization safe?

- ```
  a = x + 1
  …
  b = x + 1
  ```
  ⇒
  ```
  a = x + 1
  …
  b = a
  ```
  As long as *a* isn't redefined here

- Available expressions: nodes whose definitions are "available"
  ```

Available =/= Live

```
a = x + 1
c = a
b = x + 1
d = b * 2
return d - c
```

> a = x + 1:
> Live? No
> Available? Yes

Available expressions as a dataflow analysis (Step 1)

n:	gen[n]	kill[n]
a = b op c	{n}	uses[a]
a = load b	{n}	uses[a]
store b, a	Ø	uses[*x] (for all x that may equal a)
br L	Ø	Ø
br a L1 L2	Ø	Ø
a = $f(b_1,…,b_n)$	Ø	uses[a]∪ uses[*x] (for all x)
$f(b_1,…,b_n)$	Ø	uses[*x] (for all x)
return a	Ø	Ø

Memory at loc. x

Alias analysis!

(assuming impure functions)

Available expressions as a dataflow analysis (Steps 2-3)

- $out[n] \supseteq gen[n]$
- $in[n] \subseteq out[n']$ if n' is in pred[n]
- $out[n] \cup kill[n] \supseteq in[n]$
  - Equivalently: $out[n] \supseteq in[n] / kill[n]$

- $in[n] := \bigcap_{n' \in pred[n]} out[n']$
- $out[n] := gen[n] \cup (in[n] / kill[n])$

Available expressions as a dataflow analysis (Steps 3-4)

- $in[n] := \bigcap_{n' \in pred[n]} out[n']$
- $out[n] := gen[n] \cup (in[n] / kill[n])$

- Initialize in[n] and out[n] to {set of all nodes}
  - Iterate the update equations until a fixed point is reached

- The algorithm terminates because in[n] and out[n] *decrease monotonically*
  - At most to a minimum of the empty set
- The algorithm is precise because it finds the *largest* sets that satisfy the constraints.

Contrasting RD/AE

Reaching Defs	Available Expressions
$in[n] := \cup_{n' \in pred[n]} out[n']$ $out[n] := gen[n] \cup (in[n] / kill[n])$	$in[n] := \cap_{n' \in pred[n]} out[n']$ $out[n] := gen[n] \cup (in[n] / kill[n])$
Which definitions *may* reach n?	Which expressions *must* reach n?
Initialize to Ø	Initialize to all expressions
"May" analysis	"Must" analysis

Contrasting RD/Liveness

Reaching Defs	Liveness
$in[n] := \cup_{n' \in pred[n]} out[n']$ $out[n] := gen[n] \cup (in[n] / kill[n])$	$out[n] := \cup_{n' \in succ[n]} in[n']$ $in[n] := gen[n] \cup (out[n] / kill[n])$
Propagate information *forward*	Propagate information *backward*
Forward analysis	Backward analysis