Using SSA Form

Lecture 5: CPEN 400P

Karthik Pattabiraman, UBC

Outline

Constant Propagation

Traditional Data-flow: without SSA

Sparse Simple Constant Propagation (SCCP)

Constant Propagation

Lot of program statements result in constant values - these must be propagated along the DEF-USE chains

Leads to less computation, exposes optimization opportunities

Many branches may also become redundant or unidirectional due to constants, thereby simplifying control-flow, and further optimizations

Simple Examples: What's the Value of I ?

```
J = 1;
...
if (J > 0)
    I = 1;
else
    I = 2;
```

```
I = 1;
...
while (...) {
    J = I;
    I = f(...);
    …
    I = J;
}
```

Simple Examples: What's the Value of I ?

```
J = 1;
...
if (J > 0)
    I = 1;
else
    I = 2;
```

Needs both constant propagation and conditional branch evaluation to get I

(We'll not cover this - needs SCCP)

```
I = 1;
...
while (...) {
    J = I;
    I = f(...);
    …
    I = J;
}
```

Needs "Optimistic" Initial Assumption (Focus of our class - SSCP)

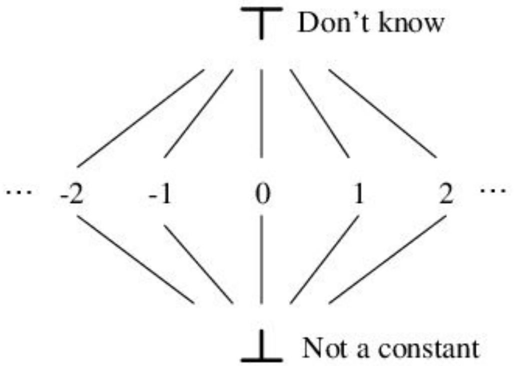
Outline

Constant Propagation

Traditional Data-flow: without SSA

Sparse Simple Constant Propagation (SSCP)

Constant Lattice



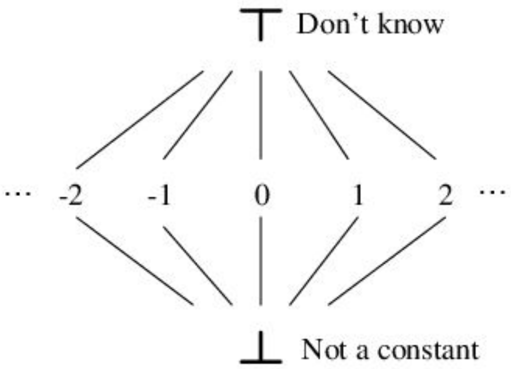
We represent the constants as a Lattice, with special elements

- T (Top): Represents as yet "unknown" values
- ⊥ (Bottom): Represents values that are not constants for sure

Meet Operator

Represented as ∧, and has the following rules:

- **$c_1 \wedge c_2 = c_1$ if $c_1 = c_2$, else ⊥**
- **$c_1 \wedge \top = c_1$**
- **$c_1 \wedge \bot = \bot$**
- **$\top \wedge \bot = \bot$**



Intuition: When you do a meet of two elements *x* and *y*, you descend the lattice to find a "greatest lower bound" of them

- **Represents facts about what is known and unknown in the program**

Constant Propagation as traditional Data-flow - 1

Domain is the set of pairs <v_i,c_i> where v_i is a variable and $c_i \in C$

$$CONSTANTS(b) = \bigwedge_{p \in preds(b)} f_p(CONSTANTS(p))$$

- **$\wedge$ performs a pairwise meet on two sets of pairs**
- **$f_p(x)$ is a block specific function that models the effects of block p on the <v_i,c_i> pairs in *x***

Constant propagation is a forward flow problem

Constant Propagation as traditional Data-flow - 2

- **If *p* has one statement then**

  $x \leftarrow y$ with $CONSTANTS(p) = \{\ldots<x,l_1>,\ldots<y,l_2>\ldots\}$
  then $f_p(CONSTANTS(p)) = CONSTANTS(p) - <x,l_1> + <x,l_2>$

  $X \leftarrow y\ op\ z$ with $CONSTANTS(p) = \{\ldots<x,l_1>,\ldots<y,l_2>\ldots>,\ldots<z,l_3>\ldots\}$
  then $f_p(CONSTANTS(p)) = CONSTANTS(p) - <x,l_1> + <x,l_2\ op\ l_3>$

- **If *p* has *n* statements then**

  $f_p(CONSTANTS(p)) = f_n(f_{n-1}(f_{n-2}(\ldots f_2(f_1(CONSTANTS(p)))\ldots)))$
  where f_i is the function generated by the i^{th} statement in *p*

Constant Propagation over DEF-USE Chains

Complexity

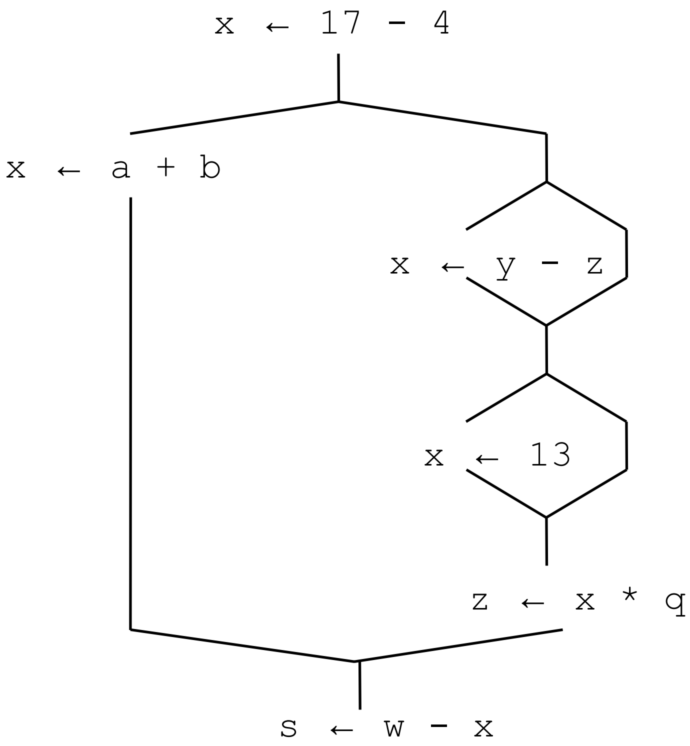
- Initial step takes O(1) time per operation
- Propagation takes
  - |*USES(v,i)*| for each *i* pulled from Worklist
  - Summing over all ops, becomes |edges in DEF-USE graph|
  - A definition can be on the worklist twice (lattice height)
  - O(|operations| + |edges in DU graph|)

Can we do better?

- Not on the def-use chains …
- Would like to compute $\wedge$ when new values are "born"
  - Where control flow brings chains together …

How does SSA help ?

Only update a variable when any of its operands change in the lattice



There are four birth points for `x`

Value is born: 17 - 4 ∧ y - z ∧ 13 ∧ a+b

- **Need to identify birth points**
- **SSA form allows easy identification of birth points and following their edges**

Outline

Constant Propagation

Traditional Data-flow: without SSA

Sparse Simple Constant Propagation (SSCP)

Main Idea

Only propagate constant information on edges of vars that have been modified

- Use SSA form to easily find all the uses of a variable
- Keep adding edges to a worklist until you run out of edges

Variables assigned to constants are initially marked to a constant value

When you come to a Phi-Node, perform a meet operation over the operands

For all other nodes, it depends on how complex are the semantics we implement

- Need to encode simple arithmetic rules (e.g., const + const = const)
- Can be quite complex to capture all possible permuations

Using SSA — Sparse Constant Propagation

∀ expression, e
    Value(e) ← { TOP if its value is unknown; c_i if its value is known; BOT if its value is known to vary (e.g., I/O ops) }
WorkList ← Ø

∀ SSA edge s = <u,v>
    if Value(u) ≠ TOP then
        add s to WorkList

while (WorkList ≠ Ø)
    remove s = <u,v> from WorkList
    let o be the operation that uses v

i.e., o is "a←b op v" or "a ←v op b"

    if Value(o) ≠ BOT then
        t ← result of evaluating o
        if t ≠ Value(o) then
            Value(0) ← t
            ∀ SSA edge <o,x>
                add <o,x> to WorkList

Example of SSCP Algorithm : Initial

$i_0 \leftarrow 12$

while (…)

$i_1 \leftarrow \varnothing(i_0, i_3)$

$x \leftarrow i_1 * 17$

$j \leftarrow i_1$

$i_2 \leftarrow \ldots$

…

$i_3 \leftarrow j$

Time Step	**i0**	**i1**	**x**	**j**	**i2**	**i3**
0						
1						
2						

WorkList: [ ]

Example of SSCP Algorithm: Time Step 0

$i_0 \leftarrow 12$

while (…)

$i_1 \leftarrow Ø(i_0, i_3)$
$x \leftarrow i_1 * 17$
$j \leftarrow i_1$
$i_2 \leftarrow …$
…
$i_3 \leftarrow j$

Time Step	**i0**	**i1**	**x**	**j**	**i2**	**i3**
0	12					
1						
2						

WorkList: [i1]

Example of SSCP Algorithm: Time Step 0

$i_0 \leftarrow 12$

while (…)

$i_1 \leftarrow \emptyset(i_0, i_3)$

$x \leftarrow i_1 * 17$

$j \leftarrow i_1$

$i_2 \leftarrow \ldots$

…

$i_3 \leftarrow j$

Time Step	i0	i1	x	j	i2	i3
0	12	12				
1						
2						

WorkList: [x, j]

Example of SSCP Algorithm: Time Step 0

$i_0 \leftarrow 12$

while (…)

$i_1 \leftarrow \varnothing(i_0, i_3)$

$x \leftarrow i_1 * 17$

$j \leftarrow i_1$

$i_2 \leftarrow$ …

…

$i_3 \leftarrow j$

Time Step	**i0**	**i1**	**x**	**j**	**i2**	**i3**
0	12	12	204			
1						
2						

WorkList: [j]

Example of SSCP Algorithm: Time Step 0

$i_0 \leftarrow 12$

while (…)
$i_1 \leftarrow Ø(i_0, i_3)$
$x \leftarrow i_1 * 17$
$j \leftarrow i_1$
$i_2 \leftarrow …$
…
$i_3 \leftarrow j$

Time Step	**i0**	**i1**	**x**	**j**	**i2**	**i3**
0	12	12	204	12		
1						
2						

WorkList: [i3]

Example of SSCP Algorithm: Time Step 0

$i_0 \leftarrow 12$

while (…)

$i_1 \leftarrow \varnothing(i_0, i_3)$

$x \leftarrow i_1 * 17$

$j \leftarrow i_1$

$i_2 \leftarrow \ldots$

…

$i_3 \leftarrow j$

Time Step	i0	i1	x	j	i2	i3
0	12	12	204	12		12
1						
2						

WorkList: [i1]

Example of SSCP Algorithm: Time Step 1

$i_0 \leftarrow 12$

while (…)

$i_1 \leftarrow Ø(i_0, i_3)$

$x \leftarrow i_1 * 17$

$j \leftarrow i_1$

$i_2 \leftarrow …$

…

$i_3 \leftarrow j$

Time Step	i0	i1	x	j	i2	i3
0	12	12	204	12		12
1		12				
2						

WorkList: [ ]   → Empty worklist (converged !)

What’d happen if we had this code instead?

$i_0 \leftarrow 12$

while (…)

$i_1 \leftarrow Ø(i_0, i_3)$

$x \leftarrow i_1 * 17$

$j \leftarrow i_1$

$i_2 \leftarrow \ldots$

…

$i_3 \leftarrow j * 2$

Time Step	i0	i1	x	j	i2	i3
0	12	12	204	12		
1						
2						

WorkList: [i3]

Example of SSCP Algorithm: Time Step 0

$i_0 \leftarrow 12$

while (…)

$i_1 \leftarrow \varnothing(i_0, i_3)$

$x \leftarrow i_1 * 17$

$j \leftarrow i_1$

$i_2 \leftarrow \ldots$

…

$i_3 \leftarrow j * 2$

Time Step	i0	i1	x	j	i2	i3
0	12	12	204	12		24
1						
2						

WorkList: [i1]

Example of SSCP Algorithm: Time Step 1

$i_0 \leftarrow 12$

while (…)

$i_1 \leftarrow Ø(i_0, i_3)$
$x \leftarrow i_1 * 17$
$j \leftarrow i_1$
$i_2 \leftarrow \ldots$
…
$i_3 \leftarrow j * 2$

Time Step	i0	i1	x	j	i2	i3
0	12	12	204	12		24
1		BOT				
2						

WorkList: [x, j]

Example of SSCP Algorithm: Time Step 1

$i_0 \leftarrow 12$

while (…)

$i_1 \leftarrow Ø(i_0, i_3)$

$x \leftarrow i_1 * 17$

$j \leftarrow i_1$

$i_2 \leftarrow \ldots$

…

$i_3 \leftarrow j * 2$

Time Step	i0	i1	x	j	i2	i3
0	12	12	204	12		24
1		BOT	BOT			
2						

WorkList: [j]

Example of SSCP Algorithm: Time Step 1

$i_0 \leftarrow 12$

while (…)

    $i_1 \leftarrow \emptyset(i_0, i_3)$
    $x \leftarrow i_1 * 17$
    $j \leftarrow i_1$
    $i_2 \leftarrow \ldots$
    …
    $i_3 \leftarrow j * 2$

Time Step	i0	i1	x	j	i2	i3
0	12	12	204	12		24
1		BOT	BOT	BOT		
2						

WorkList: [i3]

Example of SSCP Algorithm: Time Step 1

$i_0 \leftarrow 12$

while (…)

$i_1 \leftarrow Ø(i_0, i_3)$

$x \leftarrow i_1 * 17$

$j \leftarrow i_1$

$i_2 \leftarrow …$

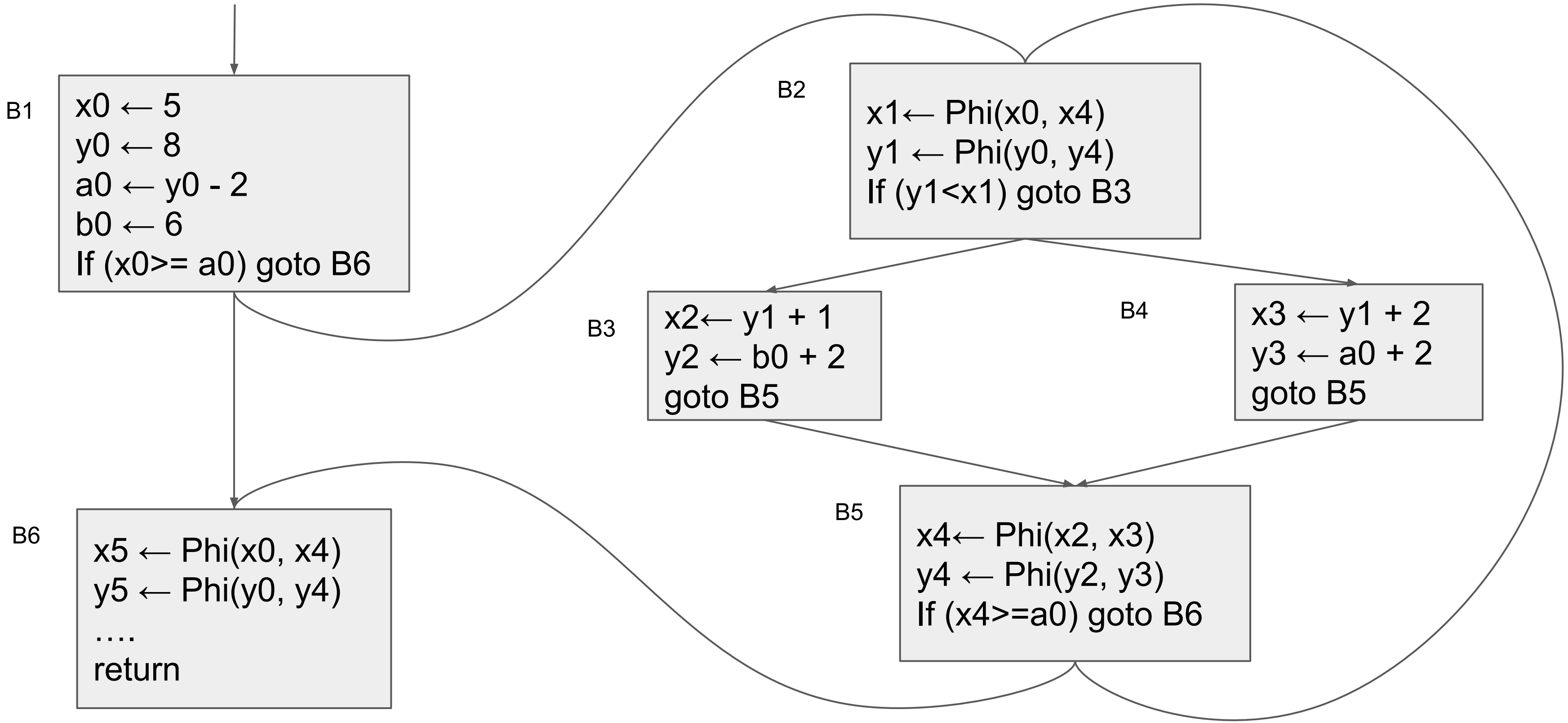
…

$i_3 \leftarrow j * 2$

Time Step	i0	i1	x	j	i2	i3
0	12	12	204	12		24
1		BOT	BOT	BOT		BOT
2						

WorkList: [ ] → i1 is already BOT, so stop

Class Activity



Outline

Constant Propagation

Traditional Data-flow: without SSA

Sparse Simple Constant Propagation (SSCP)