P423/P523 Compilers
Single Static Assignment

Based on material from Static Single Assignment Book

Deyaaeldeen Almahallawi[1]

[1]dalmahal@indiana.edu
Indiana University

April 23, 2015

History

- Developed by Wegman, Zadeck, Alpern, and Rosen in 1988.
- First used for for efficient computation of dataflow problems such as global value numbering, congruence of variables, aggressive deadcode removal, and constant propagation with conditional branches
- Currently used by GCC, Suns HotSpot JVM, IBMs RVM,Chromium V8, Mono, and LLVM

Introduction

Definition

A program is defined to be in SSA form if each variable is a target of exactly one assignment statement in the program text.

Introduction

Consider the following code:

```
x = 1;
y = x +1;
x = 2;
z = x +1;
```

Introduction

Consider the following code:

```
x = 1;
y = x +1;
x = 2;
z = x +1;
```

```
x1 = 1;
y = x1 +1;
x2 = 2;
z = x2 +1;
```

What about this code?

```
x = input();
if (x == 42)
then
y = 1;
else
y = x + 2;
end
print(y);
```

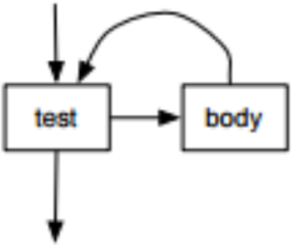
Introduction

```
x = input();
if (x == 42)
then
y1 = 1;
else
y2 = x +2;
end
y3 = φ (y1, y2);
print(y3);
```

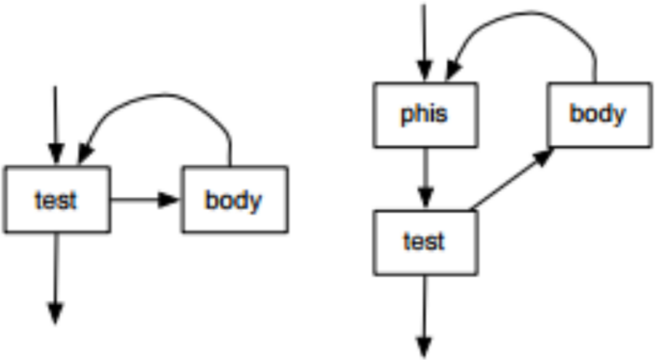
Introduction

```
x = 0;
y = 0;

while(x < 10){
  y = y + x;
  x = x + 1;
}

print(y)
```

Introduction



Introduction



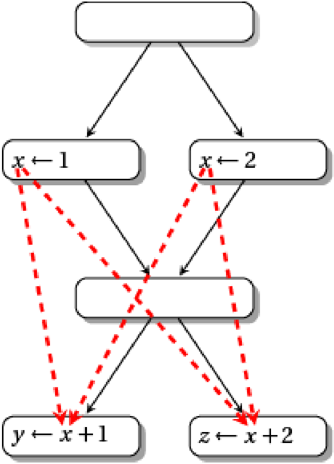
Introduction

```
x1 = 0;
y1 = 0;

x2 = φ(x1, x3)
y2 = φ(y1, y3)
while(x2 < 10){
  y3 = y2 + x2;
  x3 = x2 + 1;
}

print(y2)
```

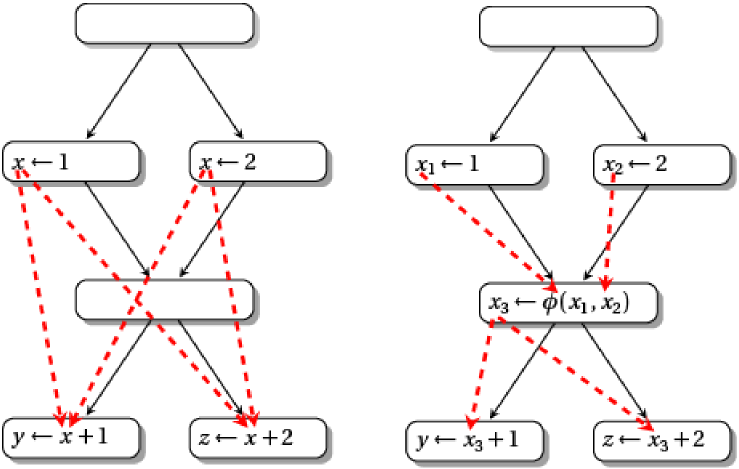
Properties

- Since there is only a single definition for each variable in the program text, a variables value is independent of its position in the program.
- Almost free use-def chains.
- Simplifies def-use chains.

Properties (Def-use)



Properties (Def-use)



Properties

- Since there is only a single definition for each variable in the program text, a variables value is independent of its position in the program.
- Almost free use-def chains.
- Simplifies def-use chains.
- No program point can be reached by two definitions of the same variable (First phase).

Properties

Single reaching-definition property

A definition D of variable v reaches a point p in the CFG if there exists a path from D to p that does not pass through another definition of v

Minimal SSA Construction

Minimal SSA Construction

Minimality property

The minimality of the number of inserted ϕ-functions.

Minimal SSA Construction

Minimality property

The minimality of the number of inserted ϕ-functions.

- ϕ-function insertion:

Minimal SSA Construction

Minimality property

The minimality of the number of inserted ϕ-functions.

- ϕ-function insertion: performs live-range splitting

Minimal SSA Construction

Minimality property

The minimality of the number of inserted ϕ-functions.

- ϕ-function insertion: performs live-range splitting to ensures that any use of a given variable v is reached by exactly one definition of v.

Minimal SSA Construction

Minimality property

The minimality of the number of inserted ϕ-functions.

- ϕ-function insertion: performs live-range splitting to ensures that any use of a given variable v is reached by exactly one definition of v.
- Variable renaming:

Minimal SSA Construction

Minimality property

The minimality of the number of inserted ϕ-functions.

- ϕ-function insertion: performs live-range splitting to ensures that any use of a given variable v is reached by exactly one definition of v.
- Variable renaming: assigns a unique variable name to each live-range.

Minimal SSA Construction

Background

Minimal SSA Construction

Background

- Join sets: For a given set of nodes S in a CFG, the join set $\mathscr{J}(S)$ is the set of nodes in S that can be reached by two (or more) distinct elements of S using disjoint paths.

Minimal SSA Construction

Background

- Join sets: For a given set of nodes S in a CFG, the join set $\mathscr{J}(S)$ is the set of nodes in S that can be reached by two (or more) distinct elements of S using disjoint paths.
- Dominance: d dom i if all paths from entry to node i include d

Minimal SSA Construction

Background

- Join sets: For a given set of nodes S in a CFG, the join set $\mathscr{J}(S)$ is the set of nodes in S that can be reached by two (or more) distinct elements of S using disjoint paths.
- Dominance: d dom i if all paths from entry to node i include d
- Strict dominance: d sdom i if d dom i and d $\neq$ i

Minimal SSA Construction

Background

- Join sets: For a given set of nodes S in a CFG, the join set $\mathscr{J}(S)$ is the set of nodes in S that can be reached by two (or more) distinct elements of S using disjoint paths.
- Dominance: d dom i if all paths from entry to node i include d
- Strict dominance: d sdom i if d dom i and d $\neq$ i
- Dominance frontier: $DF(n)$ is the border of the CFG region that is dominated by n,

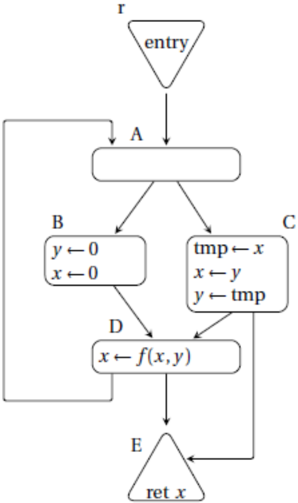
Minimal SSA Construction

Background

- Join sets: For a given set of nodes S in a CFG, the join set $\mathscr{J}(S)$ is the set of nodes in S that can be reached by two (or more) distinct elements of S using disjoint paths.
- Dominance: d dom i if all paths from entry to node i include d
- Strict dominance: d sdom i if d dom i and d $\neq$ i
- Dominance frontier: $DF(n)$ is the border of the CFG region that is dominated by n, i.e. it contains all nodes x such that n dominates a predecessor of x but n does not strictly dominate x.

Dominance Frontier

What is the border frontier of y in blocks B and C?



ϕ-function Insertion (First Phase)

Constructing minimal SSA form requires for each variable v, the insertion of ϕ-functions at $\mathscr{S}(\text{Defs}\ (v))$

ϕ-function Insertion (First Phase)

Constructing minimal SSA form requires for each variable v, the insertion of ϕ-functions at $\mathscr{S}(\text{Defs }(v))$, where Defs(v) is the set of nodes that have definitions of v.

ϕ-function Insertion

```
F ← {};                                          /* set of basic blocks where φ is added */
for v: variable names in original program do
    W ← {};                                      /* set of basic blocks */
    for d ∈ Defs(v) do
        let B be the basic block containing d;
        W ← W ∪ {B};
    end
    while W ≠ {} do
        remove a basic block X from W;
        for Y: basic block ∈ DF(X) do
            if Y ∉ F then
                add v ← φ(...) at entry of Y;
                F ← F ∪ {Y};
                if Y ∉ Defs(v) then
                    W ← W ∪ {Y};
                end
            end
        end
    end
end
```

ϕ-function Insertion

Notes

- Because a ϕ-function is itself a definition,

ϕ-function Insertion

Notes

- Because a ϕ-function is itself a definition, it may require further ϕ-functions to be inserted.

ϕ-function Insertion

Notes

- Because a ϕ-function is itself a definition, it may require further ϕ-functions to be inserted.
- Dominance frontiers of distinct nodes may intersect,

ϕ-function Insertion

Notes

- Because a ϕ-function is itself a definition, it may require further ϕ-functions to be inserted.
- Dominance frontiers of distinct nodes may intersect, but once a ϕ-function for a particular variable has been inserted at a node, there is no need to insert another.

Example: x

while loop #	X	DF(X)	F	W
-	-	-	{}	$\{B, C, D\}$

Example: x

while loop #	X	DF(X)	F	W
-	-	-	$\{\}$	$\{B, C, D\}$
1	B	$\{D\}$	$\{D\}$	$\{C, D\}$

Example: x

while loop #	X	DF(X)	F	W
-	-	-	$\{\}$	$\{B, C, D\}$
1	B	$\{D\}$	$\{D\}$	$\{C, D\}$
2	C	$\{D, E\}$	$\{D, E\}$	$\{D, E\}$

Example: x

while loop #	X	DF(X)	F	W
-	-	-	$\{\}$	$\{B, C, D\}$
1	B	$\{D\}$	$\{D\}$	$\{C, D\}$
2	C	$\{D, E\}$	$\{D, E\}$	$\{D, E\}$
3	D	$\{E, A\}$	$\{D, E, A\}$	$\{E, A\}$

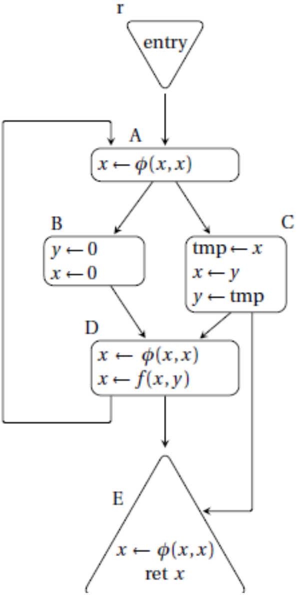
Example: x

while loop #	X	DF(X)	F	W
-	-	-	$\{\}$	$\{B, C, D\}$
1	B	$\{D\}$	$\{D\}$	$\{C, D\}$
2	C	$\{D, E\}$	$\{D, E\}$	$\{D, E\}$
3	D	$\{E, A\}$	$\{D, E, A\}$	$\{E, A\}$
4	E	$\{\}$	$\{D, E, A\}$	$\{A\}$

Example: x

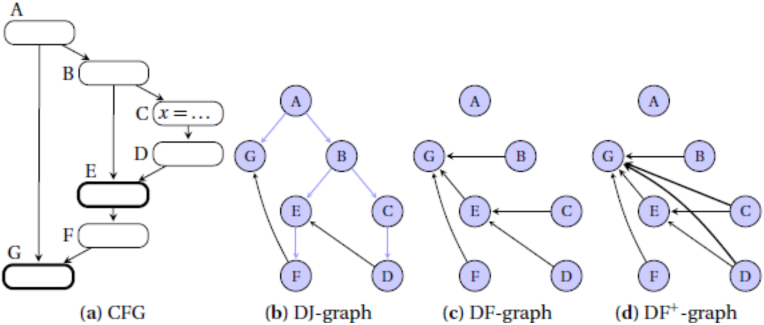
while loop #	X	DF(X)	F	W
-	-	-	$\{\}$	$\{B, C, D\}$
1	B	$\{D\}$	$\{D\}$	$\{C, D\}$
2	C	$\{D, E\}$	$\{D, E\}$	$\{D, E\}$
3	D	$\{E, A\}$	$\{D, E, A\}$	$\{E, A\}$
4	E	$\{\}$	$\{D, E, A\}$	$\{A\}$
5	A	$\{A\}$	$\{D, E, A\}$	$\{\}$

Example: x



r
entry
A
$x \leftarrow \phi(x,x)$
B
$y \leftarrow 0$
$x \leftarrow 0$
C
tmp $\leftarrow x$
$x \leftarrow y$
$y \leftarrow$ tmp
D
$x \leftarrow \phi(x,x)$
$x \leftarrow f(x,y)$
E
$x \leftarrow \phi(x,x)$
ret x

Computing Dominance Frontier



Computing Dominance Frontier

```
for (a, b) ∈ CFG edges do
    x ← a;
    while x does not strictly dominate b do
        DF(x) ← DF(x) ∪ b;
        x ← immediate dominator(x);
    end
end
```

Variable Renaming (Second Phase)

```
foreach v : Variable do
    v.reachingDef ← ⊥;
end
foreach BB : basic Block in depth-first search preorder traversal of the dominance tree
do
    foreach i : instruction in linear code sequence of BB do
        foreach v : variable used by non-φ-function i do
            updateReachingDef(v, i);
            replace this use of v by v.reachingDef in i;
        end
        foreach v : variable defined by i (may be a φ-function) do
            updateReachingDef(v, i);
            create fresh variable v′;
            replace this definition of v by v′ in i;
            v′.reachingDef ← v.reachingDef ;
            v.reachingDef ← v′;
        end
    end
    foreach φ: φ-function in a successor of BB do
        foreach v : variable used by φ do
            updateReachingDef(v, φ);
            replace this use of v by v.reachingDef in φ;
        end
    end
end
```

Variable Renaming

Procedure updateReachingDef(v,i) Utility function for SSA renaming

```
Data: v : variable from program
Data: i : instruction from program
/* search through chain of definitions for v until we find the closest
   definition that dominates i, then update v.reachingDef in-place with
   this definition                                                    */
r ← v.reachingDef;
while not (r == ⊥ or definition(r) dominates i) do
    r ← r.reachingDef;
end
v.reachingDef ← r;
```

SSA Destruction

What's next?

- No we have a nice code in SSA form, but

SSA Destruction

What's next?

- No we have a nice code in SSA form, but it has ϕ-functions all over the place that we do not know how to implement them.

SSA Destruction

What's next?

- No we have a nice code in SSA form, but it has ϕ-functions all over the place that we do not know how to implement them.
- Let's remove them!

SSA Destruction

What's next?

- No we have a nice code in SSA form, but it has ϕ-functions all over the place that we do not know how to implement them.
- Let's remove them!
- How?

SSA Destruction

What's next?

- No we have a nice code in SSA form, but it has ϕ-functions all over the place that we do not know how to implement them.
- Let's remove them!
- How? rename all ϕ-related variables (ϕ-web) to one unique name.

SSA Destruction

Finding ϕ-webs

```
begin
    for each variable v do
        phiweb(v) ← {v};
    end
    for each instruction of the form a_dest = φ(a_1, ..., a_n) do
        for each source operand a_i in instruction do
            union(phiweb(a_dest), phiweb(a_i))
        end
    end
end
```

Closer Look at Minimal SSA

Is it really minimal?

Closer Look at Minimal SSA

Is it really minimal?

It can insert a ϕ-function to merge two values that are never used after the merge.

Closer Look at Minimal SSA

Is it really minimal?

It can insert a ϕ-function to merge two values that are never used after the merge.

Solution

Closer Look at Minimal SSA

Is it really minimal?

It can insert a ϕ-function to merge two values that are never used after the merge.

Solution

Construct pruned SSA that uses global data-flow analysis to decide where values are live, so it only inserts ϕ-function at those merge points where the analysis indicates that the value is potentially live.

Pruned SSA

Is it really the solution?

Pruned SSA

Is it really the solution?

- Time-consuming

Pruned SSA

Is it really the solution?

- Time-consuming, since computing the live ranges is not trivial.

Pruned SSA

Is it really the solution?

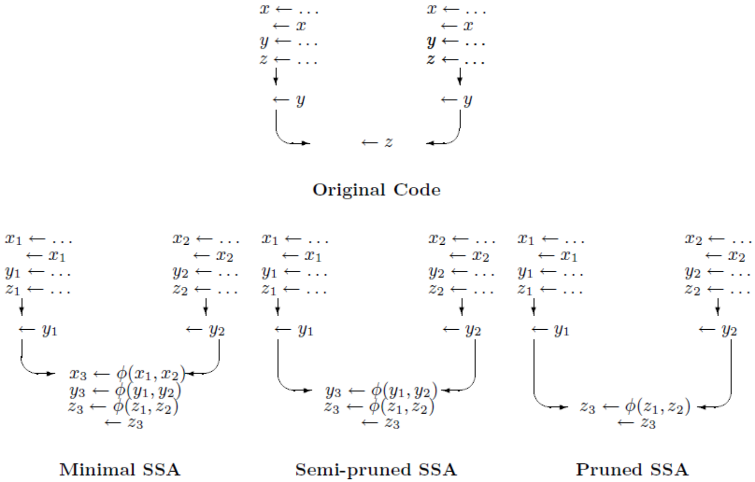
- Time-consuming, since computing the live ranges is not trivial.
- Space-consuming

Pruned SSA

Is it really the solution?

- Time-consuming, since computing the live ranges is not trivial.
- Space-consuming, it increases the space requirements for the build process.

SSA Flavors Example



Thank you!