The Development of Static Single Assignment Form

Kenneth Zadeck
NaturalBridge, Inc.
zadeck@naturalbridge.com

Ken's Graduate Optimization Seminar

- We learned:
  1. what kinds of problems could be addressed by compiler optimization.
  2. how to formulate optimization problems as dataflow equations.
  3. how to solve dataflow equations.

Ken's Graduate Optimization Seminar

- We learned:
    1. what kinds of problems could be addressed by compiler optimization.
    2. how to formulate optimization problems as dataflow equations.
    3. how to solve dataflow equations.
- Because of my dyslexia, I am really bad at 2.

Ken's Graduate Optimization Seminar

- We learned:
  1. what kinds of problems could be addressed by compiler optimization.
  2. how to formulate optimization problems as dataflow equations.
  3. how to solve dataflow equations.
- Because of my dyslexia, I am really bad at 2.
- I was able to reason about dataflow problems geometrically.

Variable by Variable Analysis.

- Viewing the program variable by variable exposes structure that is obscured by the dataflow model:
  - A kill allows the cfg to be clipped.
  - The dataflow for a single variable can be solved without iteration.

The Dataflow Abstraction

Dataflow analysis is an abstraction:

- Get:
  - Use bit vectors for simple problems.
  - Use interval analysis to solve equations quickly.

The Dataflow Abstraction

Dataflow analysis is an abstraction:

- Get:
  - Use bit vectors for simple problems.
  - Use interval analysis to solve equations quickly.
- Give:
  - Cannot play games with kill sets.

The Dataflow Abstraction

Dataflow analysis is an abstraction:

- Get:
  - Use bit vectors for simple problems.
  - Use interval analysis to solve equations quickly.
- Give:
  - Cannot play games with kill sets.
  - Cannot do SSA form.

NaturalBridge INC

Constant Propagation

```
j = 0
k = 1
if (j > 0)
    then k = 4


k ?
```

NaturalBridge INC

Constant Propagation - Kildall

```
                                    j   k

   j = 0
   k = 1
   if (j > 0)
       then k = 4



   k ?
```

Constant Propagation - Kildall

```
j = 0
k = 1
if (j > 0)
    then k = 4


k ?
```

j	k
0	T

Constant Propagation - Kildall

```
j = 0
k = 1
if (j > 0)
     then k = 4



k ?
```

j	k
0	T
0	1

Constant Propagation - Kildall

```
                                      j  k
   j = 0                              0  T
   k = 1                              0  1
   if (j > 0)                         0  1
       then k = 4



   k ?
```

Constant Propagation - Kildall

	j	k
`j = 0`	0	T
`k = 1`	0	1
`if (j > 0)`	0	1
`    then k = 4`	0	4

```
k ?
```

Constant Propagation - Kildall

```
                                   j  k
j = 0                              0  T
k = 1                              0  1
if (j > 0)                         0  1
    then k = 4                     0  4


k ?                                0  °
```

Constant Propagation - Wegbreit

```
                                   j   k   (3À    3

j = 0            1
k = 1            2
if (j > 0)       3
   then k = 4    4



k ?              5
```

Constant Propagation - Wegbreit

```
j = 0              1
k = 1              2
if (j > 0)         3
   then k = 4      4



k ?                5
```

j	k	(3°	3
0	°		
0	1		
0	1		X

Constant Propagation - Wegbreit

		j	k	(3ð	3
`j = 0`	1	0	ð		
`k = 1`	2	0	1		
`if (j > 0)`	3	0	1		X
`   then k = 4`	4				
`k ?`	5	0	1		

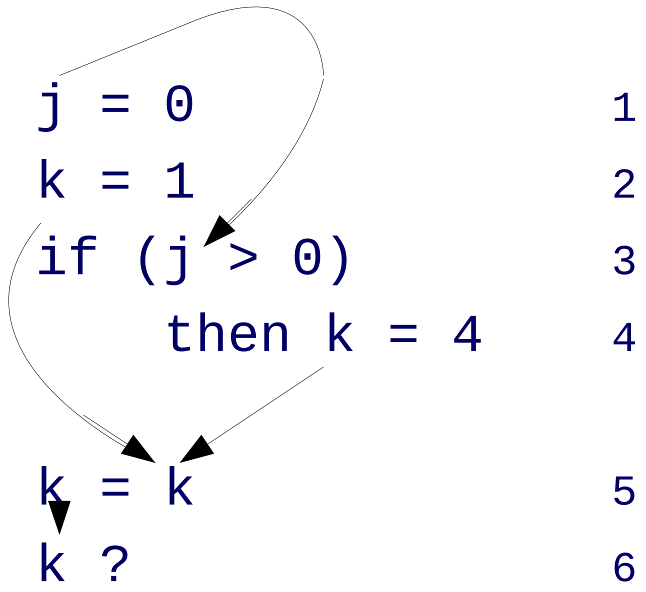
Constant Propagation – Reif & Lewis

```
j = 0
k = 1
if (j > 0)
    then k = 4

k = k
k ?
```

- Add Reif and Tarjan birthpoints.

Constant Propagation – Reif & Lewis

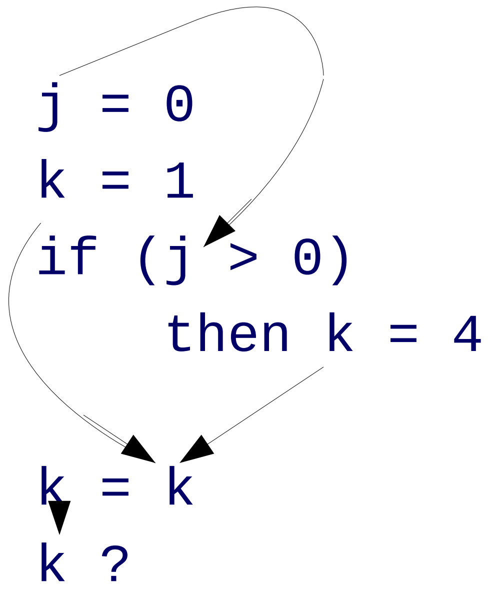


```
j = 0                   1
k = 1                   2
if (j > 0)              3
    then k = 4          4

k = k                   5
k ?                     6
```

- Add Reif and Tarjan birthpoints.
- Add def-use chains.

Constant Propagation – Reif & Lewis

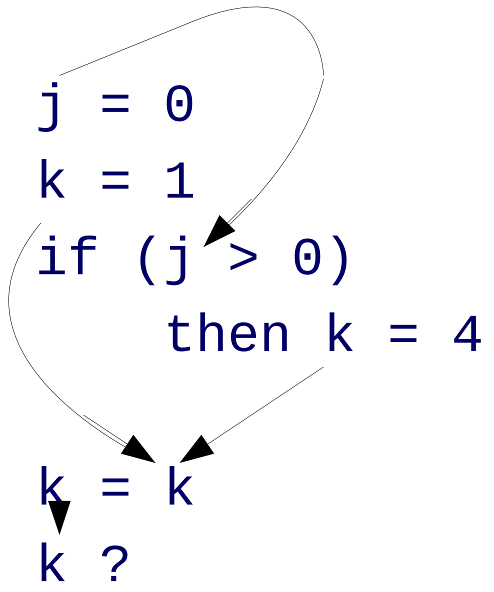


```
j = 0                 1
k = 1                 2
if (j > 0)            3
    then k = 4        4

k = k                 5
k ?                   6
```

	j_1	k_2	k_4	k_5
1	0			
2				
3				
4				
5				
6				

Constant Propagation – Reif & Lewis

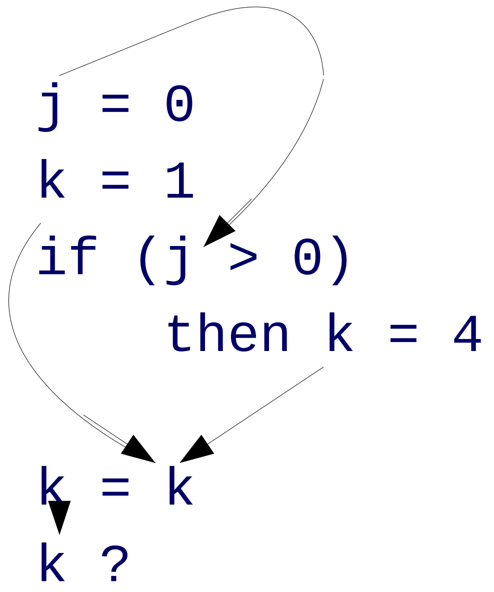


```
j = 0
k = 1
if (j > 0)
     then k = 4

k = k
k ?
```

	j1	k2	k4	k5
1	0			
2		1		
3				
4				
5				
6				

Constant Propagation – Reif & Lewis

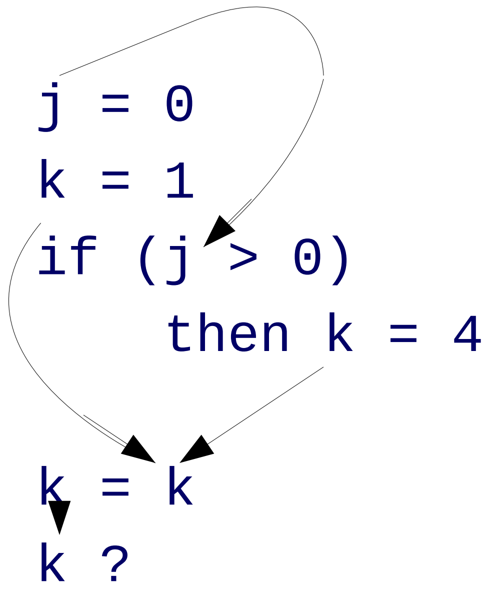


```
j = 0
k = 1
if (j > 0)
     then k = 4

k = k
k ?
```

	j_1	k_2	k_4	k_5
1	0			
2		1		
3				
4			4	
5				
6				

Constant Propagation – Reif & Lewis

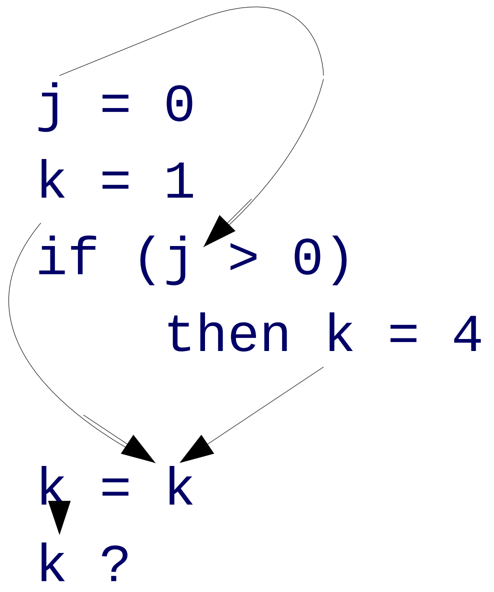


```
j = 0
k = 1
if (j > 0)
     then k = 4

k = k
k ?
```

	j_1	k_2	k_4	k_5
1	0			
2		1		
3				
4			4	
5				1
6				

Constant Propagation – Reif & Lewis

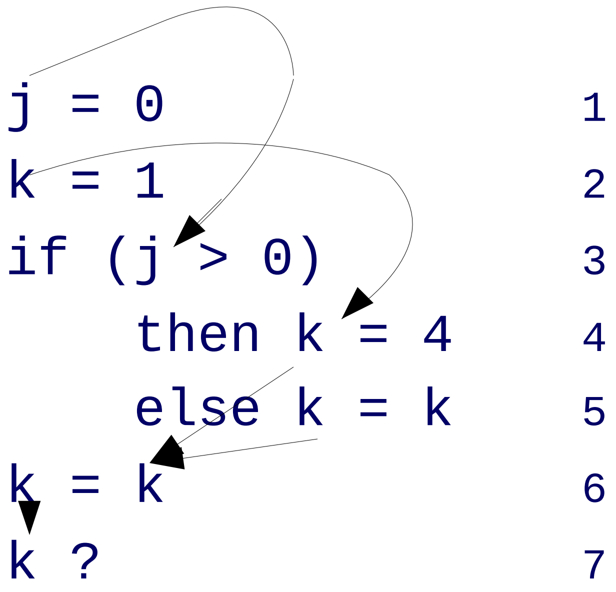


```
j = 0
k = 1
if (j > 0)
     then k = 4

k = k
k ?
```

	j1	k2	k4	k5
1	0			
2		1		
3				
4			4	
5				°
6				

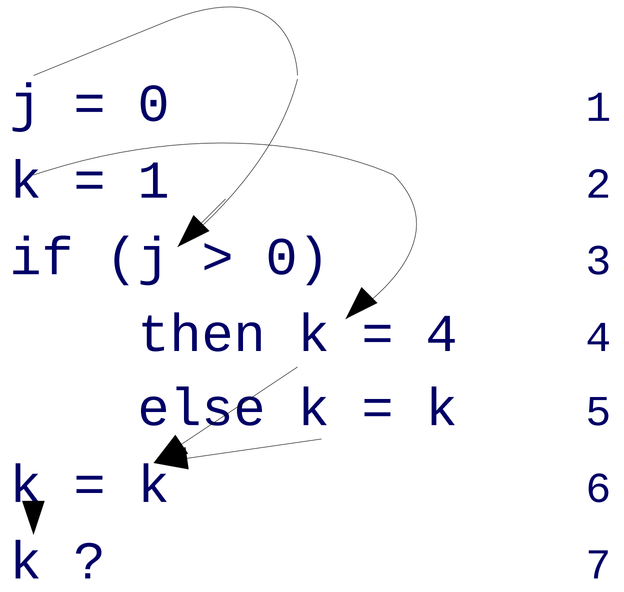
Constant Propagation – Wegman & Zadeck



```
j = 0                      1
k = 1                      2
if (j > 0)                 3
    then k = 4             4
    else k = k             5
k = k                      6
k ?                        7
```

- Add more identity assignments.

Constant Propagation – Wegman & Zadeck



```
j = 0                   1
k = 1                   2
if (j > 0)              3
    then k = 4          4
    else k = k          5
k = k                   6
k ?                     7
```

- Add more identity assignments.
- Propagate values along def-use edges iff statement is executable.

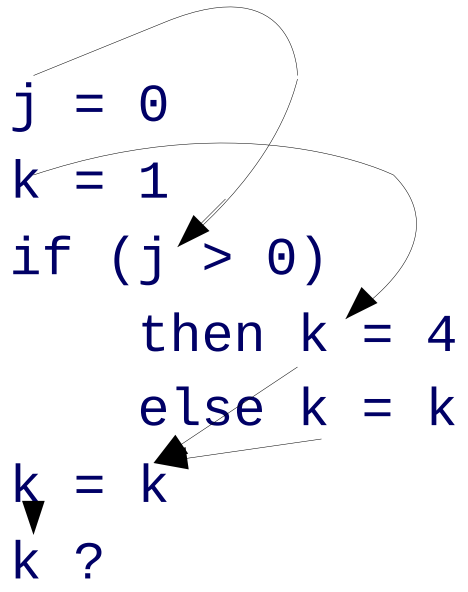
Constant Propagation – Wegman & Zadeck

```
j = 0                    1*
k = 1                    2
if (j > 0)               3
     then k = 4          4
     else k = k          5
k = k                    6
k ?                      7
```

	k_1	k_2	k_4	k_5	k_6
1*	0				

NaturalBridge INC

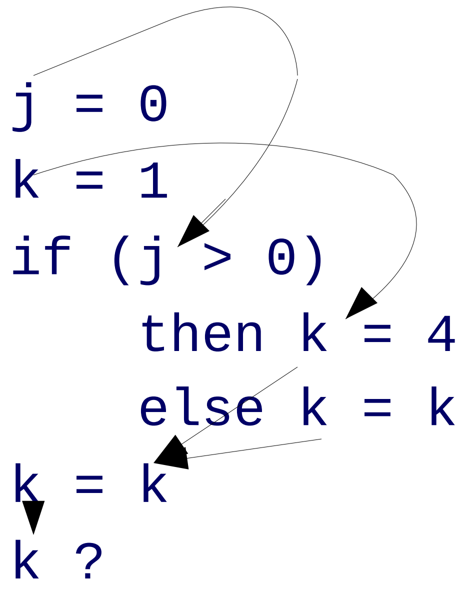
Constant Propagation – Wegman & Zadeck



```
j = 0
k = 1
if (j > 0)
     then k = 4
     else k = k
k = k
k ?
```

	k1	k2	k4	k5	k6
1*	0				
2*		1			
3					
4					
5					
6					
7					

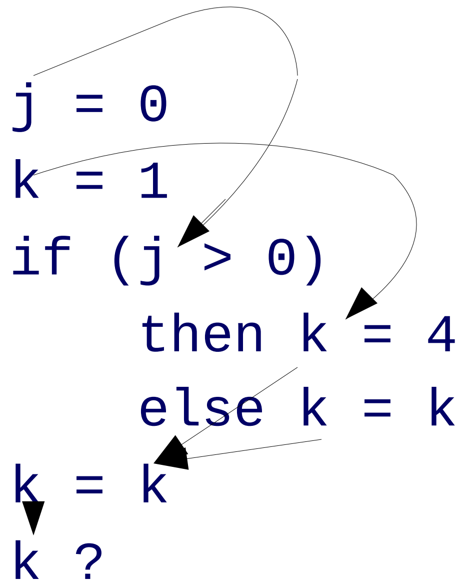
Constant Propagation – Wegman & Zadeck



```
j = 0                   1*
k = 1                   2*
if (j > 0)              3*
     then k = 4         4
     else k = k         5
k = k                   6
k ?                     7
```

k1	k2	k4	k5	k6
0				
	1			

Constant Propagation – Wegman & Zadeck



```
j = 0                      1*
k = 1                      2*
if (j > 0)                 3*
     then k = 4            4
     else k = k            5*
k = k                      6
k ?                        7
```

	k1	k2	k4	k5	k6
1*	0				
2*		1			
3*					
4					
5*				1	
6					
7					

Constant Propagation – Wegman & Zadeck

```
j = 0                      1*
k = 1                      2*
if (j > 0)                 3*
     then k = 4            4
     else k = k            5*
k = k                      6*
k ?                        7
```

	k1	k2	k4	k5	k6
1*	0				
2*		1			
3*					
4					
5*				1	
6*					1
7					

Constant Propagation – Time and Power

- Kildall and Wegbreit use a conventional dataflow framework.
- The time to run these is between O(NlogNV) and $O(N^2 V)$ depending on the type of control flow graph processing.
- Reif & Lewis and Wegman & Zadeck are O(N) for the propagation + NV to compute the birthpoints.
- Kildall ≈ Reif & Lewis
- Wegbreit ≈ Wegman & Zadeck

SSA
Looking Forwards at Wegman & Zadeck

- We had no “vision” of SSA form.
- Wegman & Zadeck is yet another fast technique to perform some transformation that uses a one off data structure.

NaturalBridge INC

SSA
Looking Backwards at Wegman & Zadeck

- This is the first SSA optimization algorithm.
- The extra identity assignments change the birthpoints into Φ-functions.
- The algorithm preserves its form while being transformed.

Removal of Invariant Code from Loops

- Ron Cytron
- Andy Lowry
- Kenneth Zadeck

POPL13 - 1986

Removal of Invariant Code from Loops

```
j = 0



while (...)

    j = j + 1
    x = y + 3
    z = x + 1
    ... = z + j
```

- Both of these statements can be removed from the loop.
- The second can be removed only after the first one is out.

Removal of Invariant Code from Loops

```
j = 0
j = j


while (...)
   birthpoint j
   j = j + 1
   x = y + 3
   z = x + 1
   ... = z + j
   j = j
```

- Add birthpoints and identity assignments.

Removal of Invariant Code from Loops

```
j₁= 0
j₂= j₁

while (...)
    birthpoint j₂
    j₃= j₂+ 1
    x₁= y₁+ 3
    z₁= x₁+ 1
    ... = z₁+ j₃
    j₂= j₃
```

- Add birthpoints and identity assignments.
- Rename variables.

NaturalBridge INC

Removal of Invariant Code from Loops

```
j₁= 0
j₂= j₁
x₁= y₁+ 3

while (...)
   birthpoint j₂
   j₃= j₂+ 1

   z₁= x₁+ 1
   ... = z₁+ j₃
   j₂= j₃
```

Any insn can be moved outside the loop if:

- the birthpoints of the rhs are outside the loop.
- the statement is not control dependent on a test inside the loop.

Removal of Invariant Code from Loops

```
j₁= 0
j₂= j₁
x₁= y₁+ 3
z₁= x₁+ 1
while (...)
   birthpoint j₂
   j₃= j₂+ 1


   ... = z₁+ j₃
   j₂= j₃
```

Any insn can be moved outside the loop if:

- the birthpoints of the rhs are outside the loop.
- the statement is not control dependent on a test inside the loop.

What is in a Name? or The Value of Renaming for Parallelism and Storage Allocation

- Ron Cytron
- Jeanne Ferrante

ICPP87

Proves that the renaming done in the prev paper removes all false dependencies for scalars.

The Origin of Φ-Functions and the Name

- Barry Rosen did not like the identity assignments.
  - He decided to replace them with “phony functions” that were able to see which control flow reached them.
  - A Φ-function was a more publishable name.
- The name Static Single Assignment Form came from the fact that Single Assignment languages were popular then.

Global Value Numbers and Redundant Computations

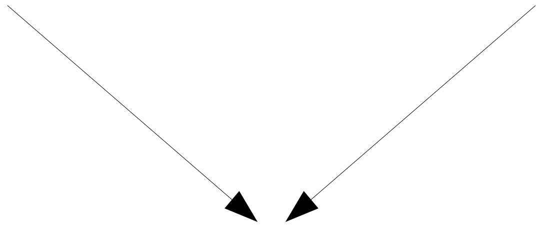
- Barry Rosen
- Mark Wegman
- Kenneth Zadeck

POPL15 - 1988

Global Value Numbers and Redundant Computations

- Classical value numbering algorithms are restricted to programs with no joins.
- With Φ-functions, it is possible to extend value numbering to acyclic regions.

Global Value Numbers and Redundant Computations



$x_3 = \Phi(x_1, x_2)$

$... = x_3 + y_1$

Global Value Numbers and Redundant Computations

$$\begin{tikzcd} \ldots=x_1+y_1 \arrow[dr] & & \ldots=x_2+y_1 \arrow[dl] \\ & x_3=\Phi(x_1,x_2) \\ & \ldots=x_3+y_1 \end{tikzcd}$$

Detecting Equality of Values in Programs

- Bowen Alpern
- Mark Wegman
- Kenneth Zadeck

POPL15 - 1988

Detecting Equality of Values in Programs

- Convert the program to SSA form.

Detecting Equality of Values in Programs

- Convert the program to SSA form.
- Use Hopcrofts finite state minimization algorithm to partition the program.
  - The dataflow edges are the edges in the graph.
  - Label each Φ-function at join point n to Φ_n.
  - The operators are labels on the nodes. Place all the operations with a given label in the same partition to start.

NaturalBridge INC

Detecting Equality of Values in Programs

- Convert the program to SSA form.
- Use Hopcrofts finite state minimization algorithm to partition the program.
  - The dataflow edges are the edges in the graph.
  - Label each Φ-function at join point n to Φ_n.
  - The operators are labels on the nodes. Place all the operations with a given label in the same partition to start.
- After partitioning, any operations in the same partition compute the same value.

Detecting Equality of Values in Programs

- All of us thought this was a very neat trick.
- It is not useful because many people add other tricks to their value numbering.
- We tried for two years to extend this along the lines of those tricks and we failed.

An Efficient Method of Computing Static Single Assignment Form

- Ron Cytron
- Jeanne Ferrante
- Barry Rosen
- Mark Wegman
- Kenneth Zadeck

POPL16 - 1989

An Efficient Method of Computing Static Single Assignment Form

- There should have been two papers in that POPL:
  - An Efficient Method of Computing Static Single Assignment Form by Rosen, Wegman and Zadeck
  - An Efficient Method of Computing the Program Dependence Graph by Cytron and Ferrante.

An Efficient Method of Computing Static Single Assignment Form

- There should have been two papers in that POPL:
  - An Efficient Method of Computing Static Single Assignment Form by Wegman and Zadeck
  - An Efficient Method of Computing the Program Dependence Graph by Cytron and Ferrante.
- We figured out that the algorithms were the same a couple of days before the submission deadline.
  - We barely had time to merge the abstracts.
  - We missed fixing the title.

An Efficient Method of Computing Static Single Assignment Form

- The algorithm presented here is generally linear.
  - It is a big improvement over Reif & Tarjan which is generally quadratic.
- It has been bettered by:
  - Sreedhar &Gao in POPL22.
  - Bilardi & Pingali in JACM 2003.

NaturalBridge INC

An Efficient Method of Computing Static Single Assignment Form

- The algorithm presented here is generally linear.
  - It is a big improvement over Reif & Tarjan which is generally quadratic.
- It has been bettered by:
  - Sreedhar &Gao in POPL22.
  - Bilardi & Pingali in JACM 2003.
- The journal version has a dead code elimination algorithm.

Analysis of Pointers and Structures

- David Chase
- Mark Wegman
- Kenneth Zadeck

Sigplan 90

Analysis of Pointers and Structures

- One of the first computationally efficient techniques to analyze pointers.
- Makes on minimal use of SSA.
  - Use of the ssa names gives a small amount of flow sensitivity to a problem that otherwise must be solved in a flow insensitive way.
  - This trick is used in other new algorithms.
- Many new and much better techniques have followed.

What Happened Next

- We stopped working on SSA.
  - None of us actually worked on a compiler project.
  - I was at Brown University.
  - We were blocked from transfering SSA to the IBM product compilers.
- People outside of IBM were picking it up.
  - Apollo, DEC, HP, SGI, and SUN were all using it to some extent.
  - We had built a good foundation.
  - It was easy to play the game.

Why Did SSA Win?

- All things being equal, SSA form only accounts for a few percent code quality over the comparable data flow techniques.
  - SSA techniques run much faster.
  - Scanning the program, building the transfer functions, and solving the equations is slow.
  - Incremental data flow never really worked.
- The high gain, parallel extraction techniques need SSA to keep things clean.
- SSA is easier to understand than dataflow.
  - I have no standing to say this.

References

There is a good bibliography online that contains most of the SSA papers:

- http://www.cs.man.ac.uk/~jsinger/ssa.html
- It is accessable from the wikipedia article on SSA.

Postscript

- For the last year I have been working to bring the analysis in the GCC back ends up to date.
  - It is infeasible to use SSA for the back ends.
  - Must maintain compatibility with the existing machine descriptions.
  - The back end is currently state of the art as of about 1986.
- The middle machine independent parts are now all SSA.

Postscript

- For the last year I have been working to bring the analysis in the GCC back ends up to date.
  - It is infeasible to use SSA for the back ends.
  - Must maintain compatibility with the existing machine descriptions.
  - The back end is currently state of the art as of about 1986.
- The middle machine independent parts are now all SSA.
- I still do not speak dataflow equations.