Static single assignment form (1)

Static single assignment form (SSA) is an intermediate representation where

- Each definition defines a unique name
- Each use refers to a single definition

Static single assignment form (2)

Example (1)

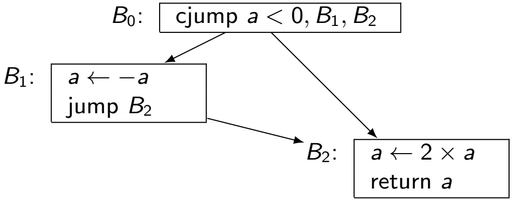
TACL code

```
if (a < 0)
  a = -a;
a = 2 * a;
^ a
```

Alternate IR

$$
\begin{array}{rl}
 & \text{cjump } a < 0, l_1, l_2 \\
l_1: & a \leftarrow -a \\
l_2: & a \leftarrow 2 \times a \\
 & \text{return } a
\end{array}
$$

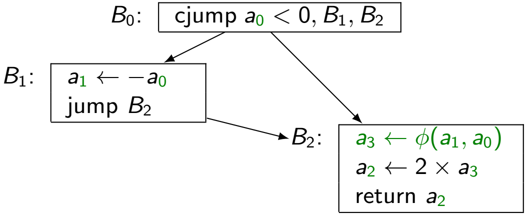
CFG



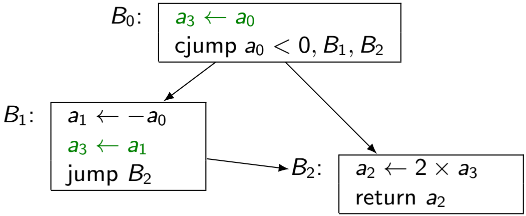
Static single assignment form (3)

Example (1, cont.)

SSA form



Removing the ϕ-function



Static single assignment form (4)

ϕ-functions reconcile different values for a name coming along different edges

A ϕ-function $a = \phi(a, \ldots, a)$ in node m selects the correct value for a according to the CFG edge traversed to reach m

ϕ-functions are a device for encoding data-flow information and are not meant to be implemented

All ϕ-functions in a block are considered as being evaluated simultaneously

ϕ-functions are only needed for global names (i.e., names that are not local to a basic block)

Dominance

A control flow graph (CFG) node m dominates node n if every path from the root to n passes through m

- m is a dominator of n

Node m strictly dominates n if m is a dominator of n and $m \neq n$

- m is a strict dominator of n

Node n is in the dominance frontier (DF) of m if

- m dominates a predecessor of n, and
- m does not strictly dominate n

Static single assignment form (5)

A join point is a CFG node that has multiple predecessors

ϕ-functions are only needed at join points

A node $n \in DF(m)$ is a join point in the CFG since

a. There is a path in the graph from m to n
b. There is a path from the root of the graph to n that does not go through m (otherwise, m would strictly dominate n)
c. In the path from m to n, n is the first node not strictly dominated by m

Inserting ϕ-functions

If node m defines a, every node in $\mathrm{DF}(m)$ needs a ϕ-function for a

Static single assignment form (6)

Translating into SSA

1. Building the CFG
2. Computation of the dominance frontiers
3. Insertion of ϕ-functions
   - Adding a ϕ-function for a to a node makes that node define a
4. Numbering the definitions
5. Renaming uses

Static single assignment form (7)

Example (2)

TACL code

```
r = 1;
while (n > 0)
[
  r = r * n;
  n = n - 1;
]
^ r
```

Alternate IR

	$r \leftarrow 1$	B_0
l_0 :	cjump $n > 0, l_1, l_2$	B_1
l_1 :	$r \leftarrow r \times n$ $n \leftarrow n - 1$ jump l_0	B_2
l_2 :	return r	B_3

CFG

$$\begin{array}{c} B_0 \to B_1,\quad B_1 \to B_2,\quad B_1 \to B_3,\quad B_2 \to B_1 \end{array}$$

Dominance

	Dominates	DF
B_0	B_0, B_1, B_2, B_3	—
B_1	B_1, B_2, B_3	B_1
B_2	B_2	B_1
B_3	B_3	—

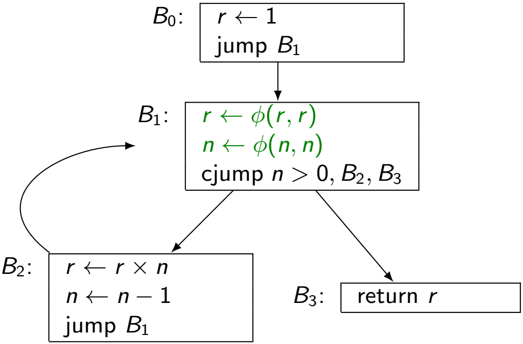
Static single assignment form (8)

Example (2, cont.)

Inserting ϕ-functions

$DF(B_0) = \emptyset$, so B_0 does not cause the insertion of any ϕ-function

Since B_1 does not define any name, it will not imply the insertion of any ϕ-function



B_2 defines names r and n and ϕ-functions for both names are needed in all nodes in $DF(B_2) = \{B_1\}$

B_1 now defines r and n, but $DF(B_1) = \{B_1\}$ and, since B_1 already has the corresponding ϕ-functions, no further ϕ-function is needed

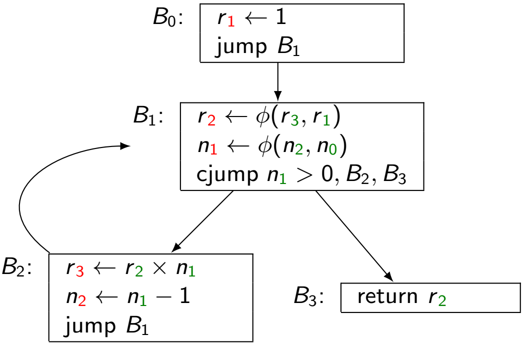
Static single assignment form (9)

Example (2, cont. 2)

Numbering definitions

Once ϕ-functions have been inserted, all name definitions are numbered so all define a different name

Each name's definition 0 is available at the start node



Every use of a name is then renamed to reflect the definition that reaches it

Static single assignment form (10)

SSA form incorporates both control-flow and data-flow information

The fact that each use refers to a single definition, makes it straightforward to implement copy-propagation and constant-propagation, and to recognise duplicate expressions

Translating out of SSA form

After program transformation and optimisation, the remaining ϕ-functions must be removed for code generation

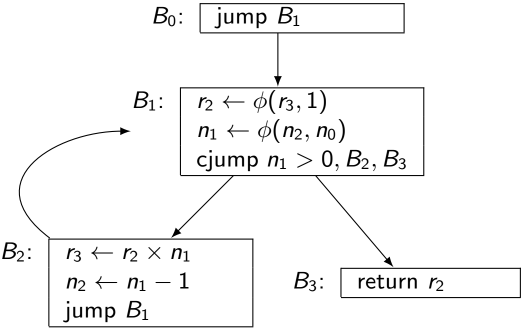
To remove each $a_i \leftarrow \phi(e_1, \ldots, e_k)$, a definition $a_i \leftarrow e_j$ is inserted at the end of the block where edge j starts, for every $j \in \{1, \ldots, k\}$

Static single assignment form (11)

Example (2, cont. 3)

Code after constant-
-propagation and
useless-code elimination

Once r_1's definition is propagated, $r_1 \leftarrow 1$ becomes useless code

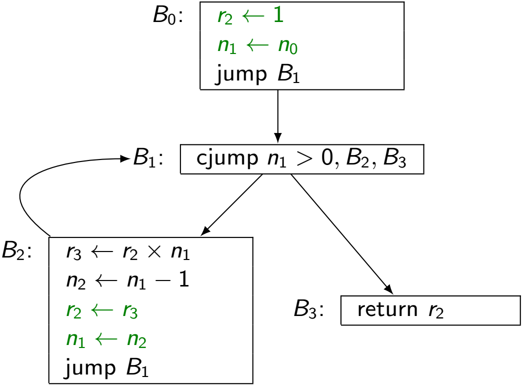


Static single assignment form (12)

Example (2, cont. 4)

Removing ϕ-functions

Since B_1 is the only block with a ϕ-function, definitions for r_2 and n_1 are inserted in its predecessors B_0 and B_2



Static single assignment form (13)

The lost-copy problem (1)

Following the previous rule when translating out of SSA form may sometimes lead to incorrect code

Original IR

B_0: $i \leftarrow 1$
jump B_1

B_1: $y \leftarrow i$
$i \leftarrow i + 1$
cjump $\ldots, B_1, B_2$

B_2: $z \leftarrow y + \ldots$

SSA form

B_0: $i_1 \leftarrow 1$
jump B_1

B_1: $i_2 \leftarrow \phi(i_1, i_3)$
$y_1 \leftarrow i_2$
$i_3 \leftarrow i_2 + 1$
cjump $\ldots, B_1, B_2$

B_2: $z_1 \leftarrow y_1 + \ldots$

After copy-propagation

B_0: $i_1 \leftarrow 1$
jump B_1

B_1: $i_2 \leftarrow \phi(i_1, i_3)$
$i_3 \leftarrow i_2 + 1$
cjump $\ldots, B_1, B_2$

B_2: $z_1 \leftarrow i_2 + \ldots$

and useless-code elimination

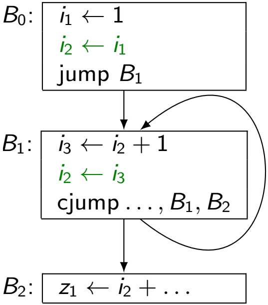
Static single assignment form (14)

The lost-copy problem (2)

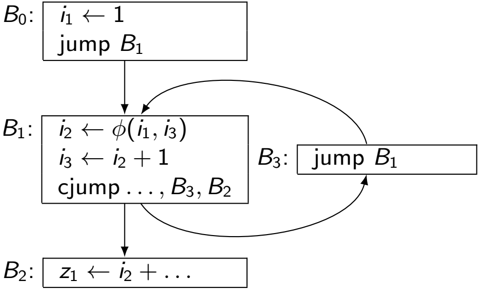
This may be avoided by inserting a dummy node in edges that start from a node from where at least one other edge starts *and* arrive at a node where at least one other edge arrives

This is called edge splitting

After ϕ-function removal



Edge splitting

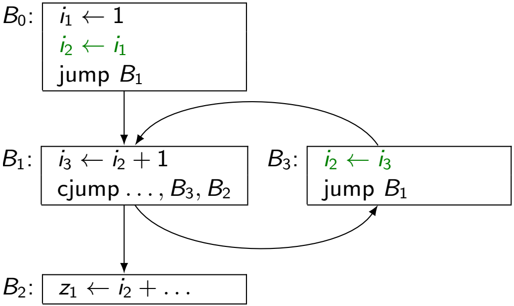


Now z_1 gets the wrong value

Static single assignment form (15)

The lost-copy problem (3)

After ϕ-function removal again



Now z_1 gets the correct value