Static single assignment form (1)

The static single assignment form (SSA form) is an intermediate representation where

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

Static single assignment form (2)

Example (22)

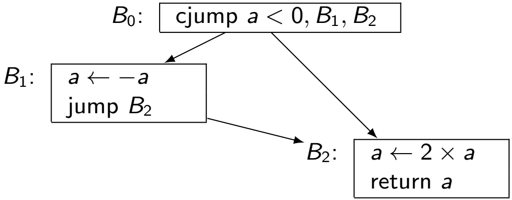
TACL code

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

Alternate IR

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

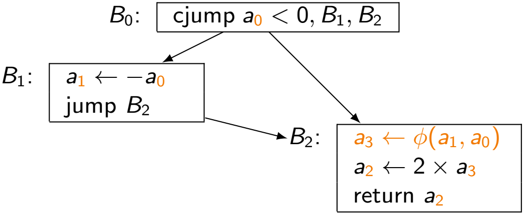
CFG



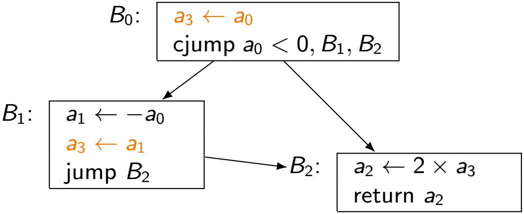
Static single assignment form (3)

Example (22, 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
  (either m does not dominate n or $m = n$)

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, a node in $\mathrm{DF}(m)$ needs a ϕ-function for a if a is live on entry to the corresponding block

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 (23)

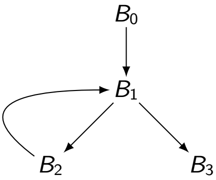
TACL code

```
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



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	—

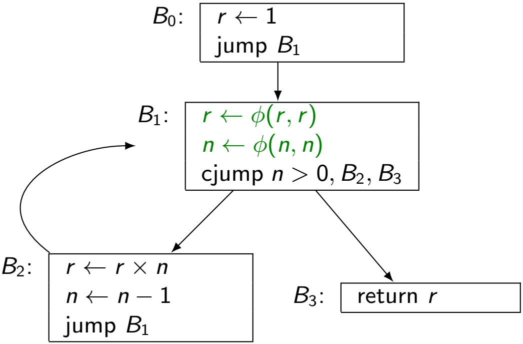
Static single assignment form (8)

Example (23, 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

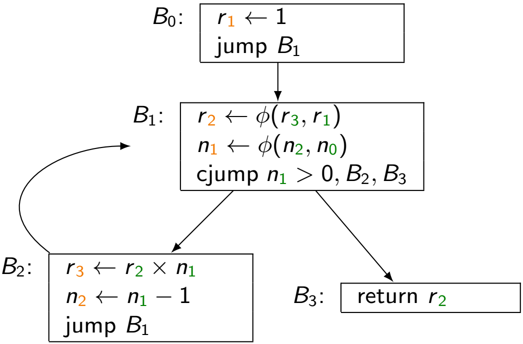
Static single assignment form (9)

Example (23, 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

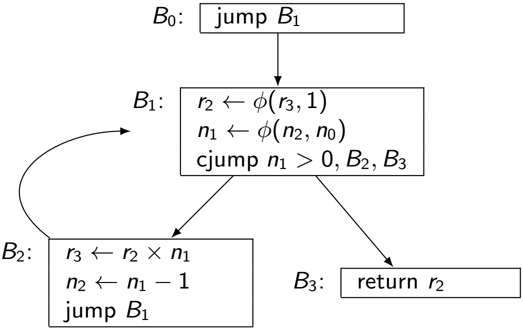
To remove each $a_i \leftarrow \phi(e_1, \ldots, e_k)$, a definition $a_i \leftarrow e_j$ is inserted at the end of the block where edge j starts, for every $j \in \{1, \ldots, k\}$

Static single assignment form (11)

Example (23, cont. 3)

Code after constant propagation and useless-code elimination

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

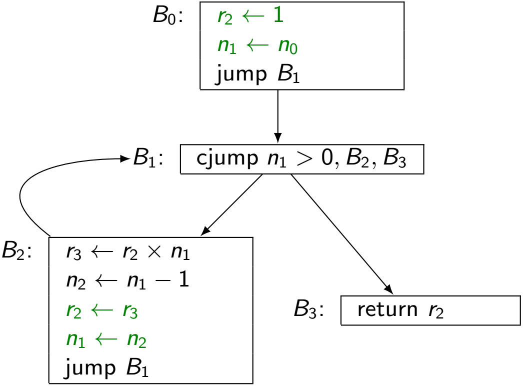


Static single assignment form (12)

Example (23, 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



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 $\ldots, B_1, B_2$

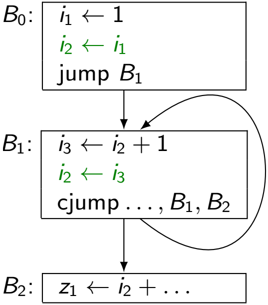
B_2: $z_1 \leftarrow i_2 + \ldots$

and useless-code elimination

The lost-copy problem (2)

In this case, the optimisation performed on the SSA form of the code leads to incorrect code

After ϕ-function removal



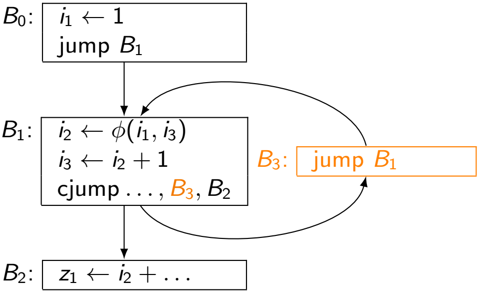
Now z_1 gets the wrong value

The lost-copy problem (3)

A critical edge is an edge that starts from a node from where at least one other edge starts *and* arrives at a node where at least one other edge arrives

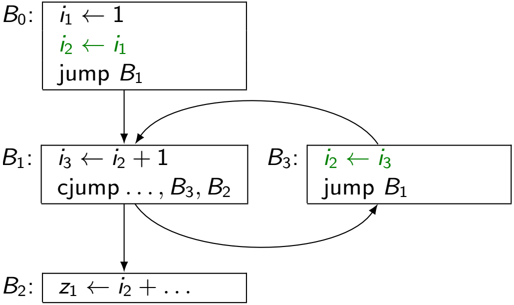
This error may be avoided by splitting critical edges, by inserting a dummy node into the edge

Edge splitting



The lost-copy problem (4)

After ϕ-function removal again

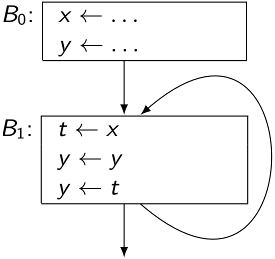


Now z_1 gets the correct value

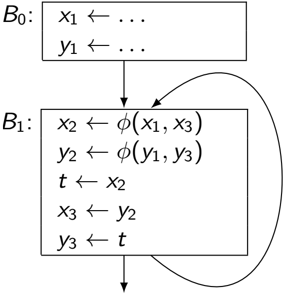
The swap problem (1)

The swap problem is another problem that may happen when translating out of SSA form

Original IR



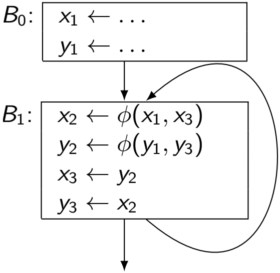
SSA form



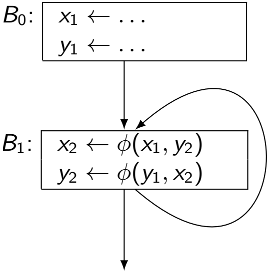
(Jump and cjump instructions have been omitted in this example)

The swap problem (2)

After copy propagation (t)

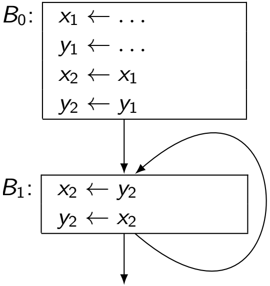


After further copy propagation



The swap problem (3)

ϕ-function removal



y_2 gets the wrong value

This problem is due to the interdependence between the ϕ-functions in B_1

To avoid it, the compiler must identify these dependences and insert code to save a value to a temporary before it is overwritten