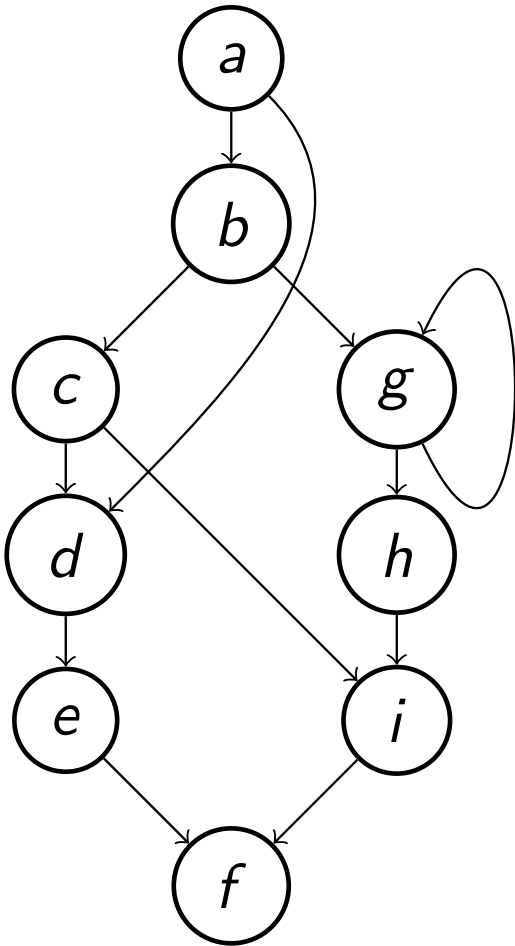


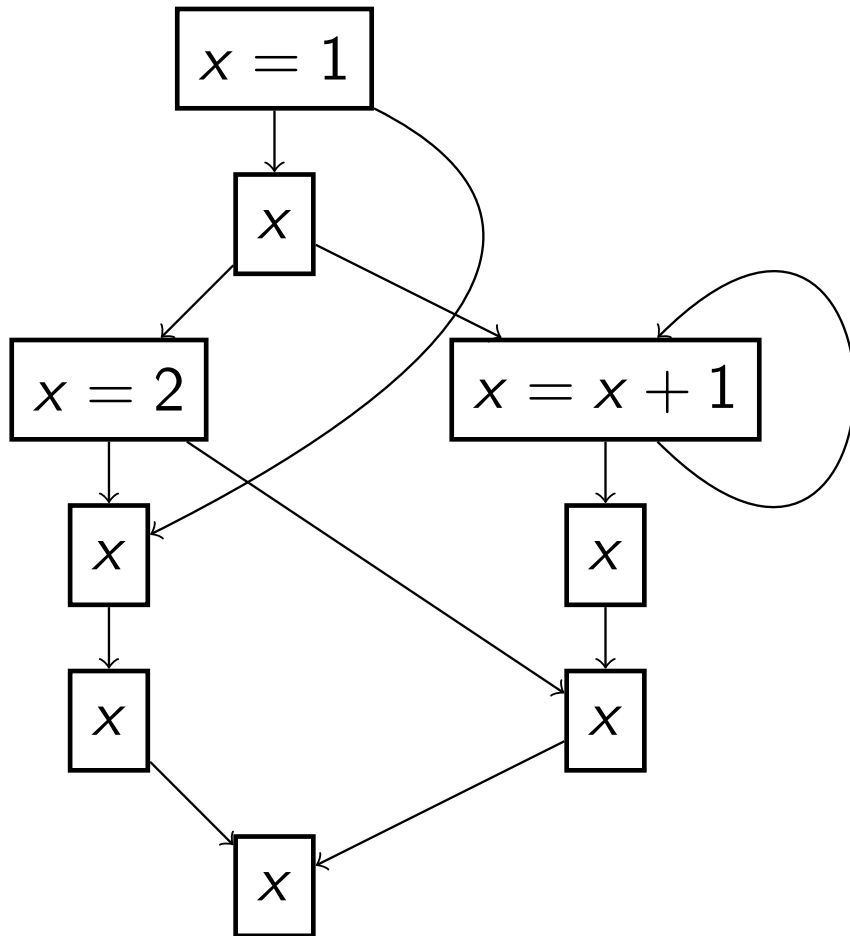
Repetition

- The Lengauer-Tarjan algorithm
- Translation to SSA Form
- Hash-Based Value Numbering
- Global Value Numbering
- Partial Redundancy Elimination
- Operator Strength Reduction

The Lengauer-Tarjan Algorithm



Translation to SSA Form

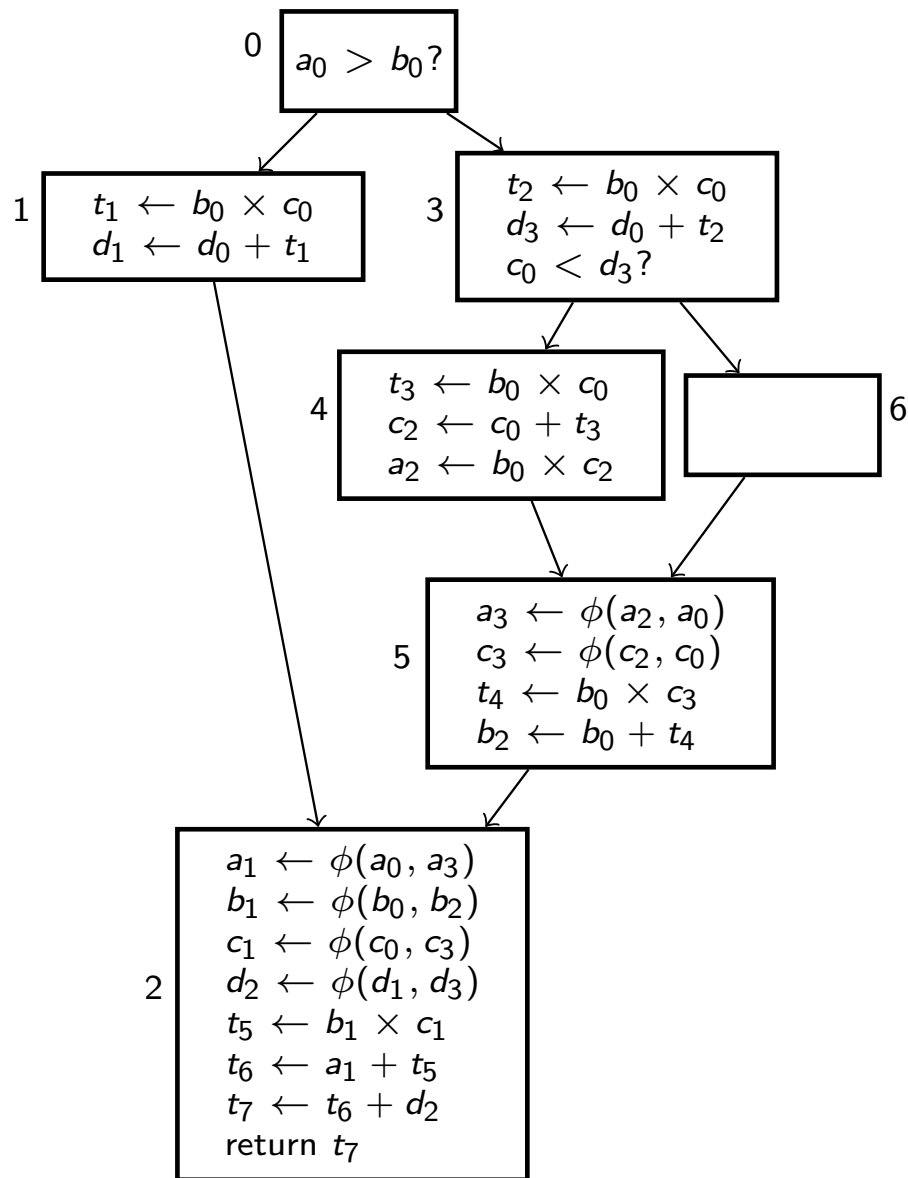


Redundancy elimination

```
int f(int a, int b, int c, int d)
{
    if (a > b) {
        d += b * c;
        if (c < d) {
            c += b * c;
            a = b * c;
        }
        b += b * c;
    } else
        d += b * c;

    return a + b * c + d;
}
```

Hash-based value numbering

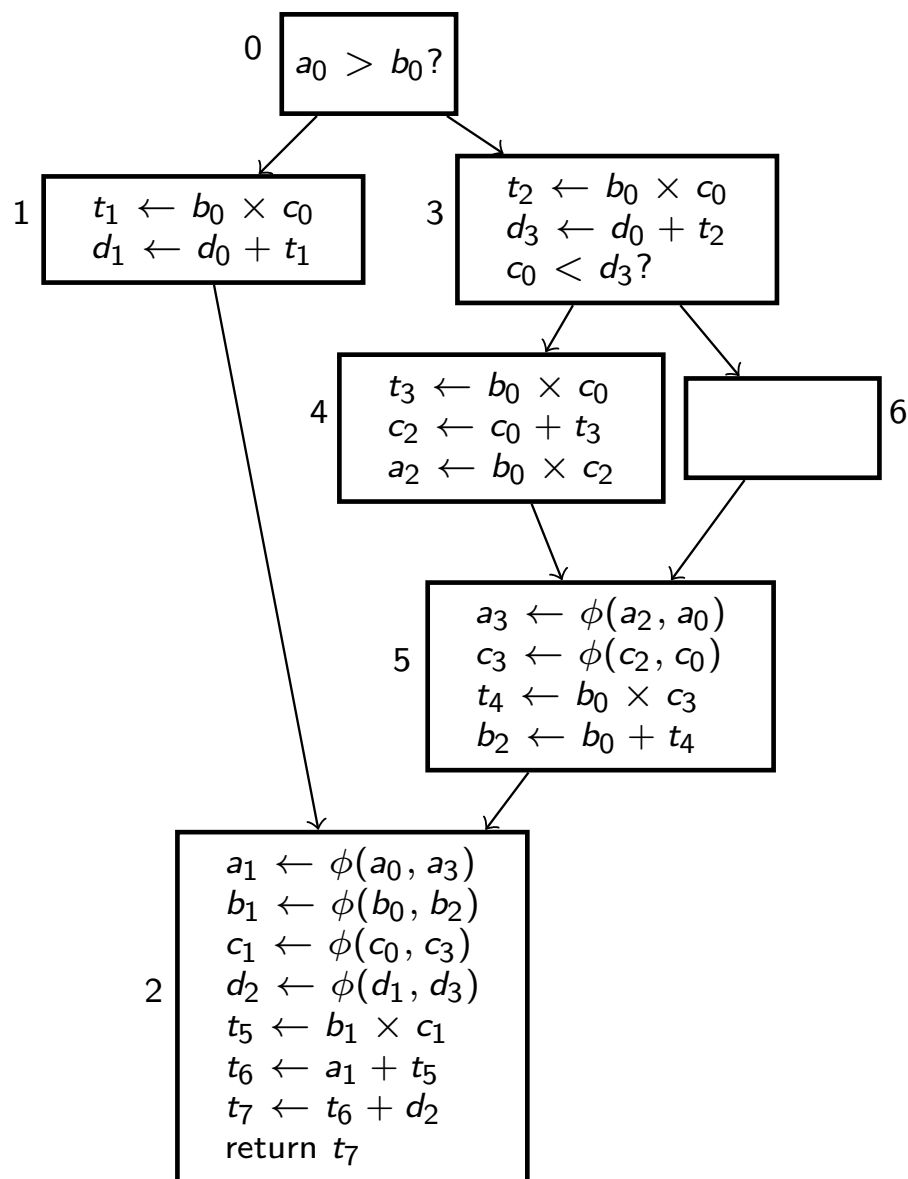


Dominator tree traversal during SSA renaming.

- 1 In 0 nothing happens.
- 2 In 1 $b_0 \times c_0$ is pushed and then popped.
- 3 In 2 $b_1 \times c_1$ is pushed and then popped.
- 4 In 3 $b_0 \times c_0$ is pushed.
- 5 In 4 $b_0 \times c_0$ found on the stack and reused.
- 6 In 4 $b_0 \times c_2$ is pushed and then popped.
- 7 In 5 $b_0 \times c_3$ is pushed and then popped.
- 8 In 6 nothing happens.
- 9 In 3 $b_0 \times c_0$ is popped.

Only one expression was optimized.

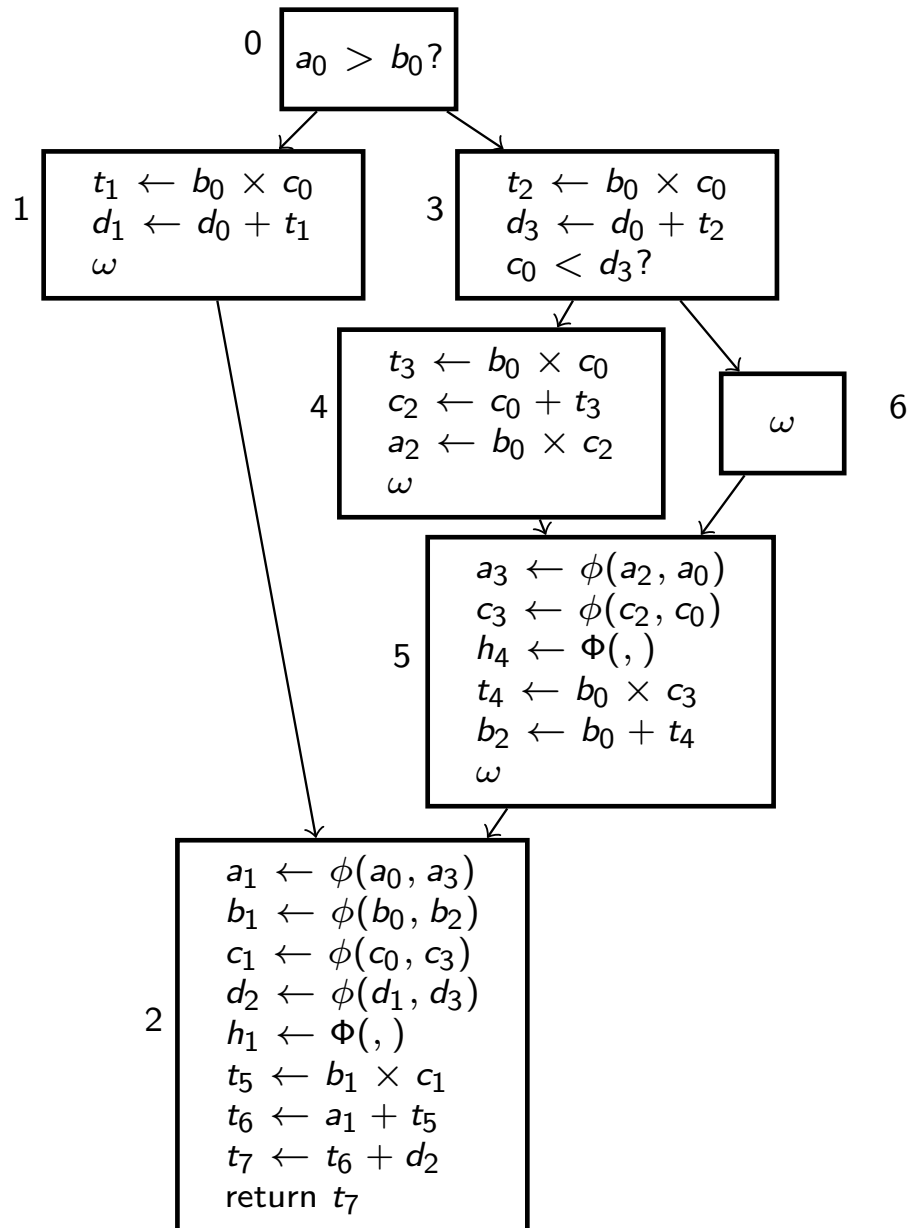
Global value numbering



N^2 -partitioning algorithm results in the following.

- $B_1 = \{a_0\}$.
- $B_2 = \{b_0\}$.
- $B_3 = \{c_0\}$.
- $B_4 = \{d_0\}$.
- $B_5 = \{t_1, t_2, t_3, a_2, t_4, t_5\}$.
- $B_6 = \{d_1, t_6, t_7, d_3, c_2, b_2\}$.
- $B_7 = \{a_1, b_1, c_1, d_2\}$.
- $B_8 = \{a_3, c_3\}$.
- $B_5^1 = \{t_1, t_2, t_3\}$.
- $B_5^2 = \{a_2, t_4, t_5\}$.
- $B_6^1 = \{d_1\}$.
- $B_6^2 = \{t_6, t_7, d_3, c_2, b_2\}$.
- $B_7^1 = \{a_1\}$.
- $B_7^2 = \{b_1\}$.
- $B_7^3 = \{c_1\}$.
- $B_7^4 = \{d_2\}$.
- $B_8^1 = \{a_3\}$.
- $B_8^2 = \{c_3\}$.
- All except B_5^1 will be split until they are singleton blocks.
- In B_5^1 t_2 dominates t_3 and t_3 is optimized away.

SSAPRE for $b \times c$



Dominator tree traversal during SSAPRE renaming.

- 1 In 0 nothing happens.
- 2 In 1 $h_0 \leftarrow b_0 \times c_0$ is pushed.
- 3 In 1 $\text{has-real-use}(\omega) \leftarrow 1$.
- 4 In 1 the real occurrence is popped.
- 5 In 2 $h_1 \leftarrow \Phi$ is pushed.
- 6 In 2 the real occurrence points to the Φ and is pushed.
- 7 In 2 the Φ remains downsafe due to real at the top.
- 8 In 2 both occurrences are popped.
- 9 In 3 $h_2 \leftarrow b_0 \times c_0$ is pushed.
- 10 In 4 $b_0 \times c_0$ points to h_2 and is pushed.
- 11 In 4 $h_3 \leftarrow b_0 \times c_2$ is pushed.
- 12 In 4 $\text{has-real-use}(\omega) \leftarrow 1$.
- 13 In 4 both real occurrences are popped.
- 14 In 5 $h_4 \leftarrow \Phi$ is pushed.
- 15 In 5 $b_0 \times c_3$ points to the Φ and is pushed.
- 16 In 5 $\omega \leftarrow \perp$.
- 17 In 5 both the Φ and real occurrences are popped.
- 18 In 6 $\text{has-real-use}(\omega) \leftarrow 1$.
- 19 In 3 $b_0 \times c_0$ is popped.

SSAPRE continues on the next page.

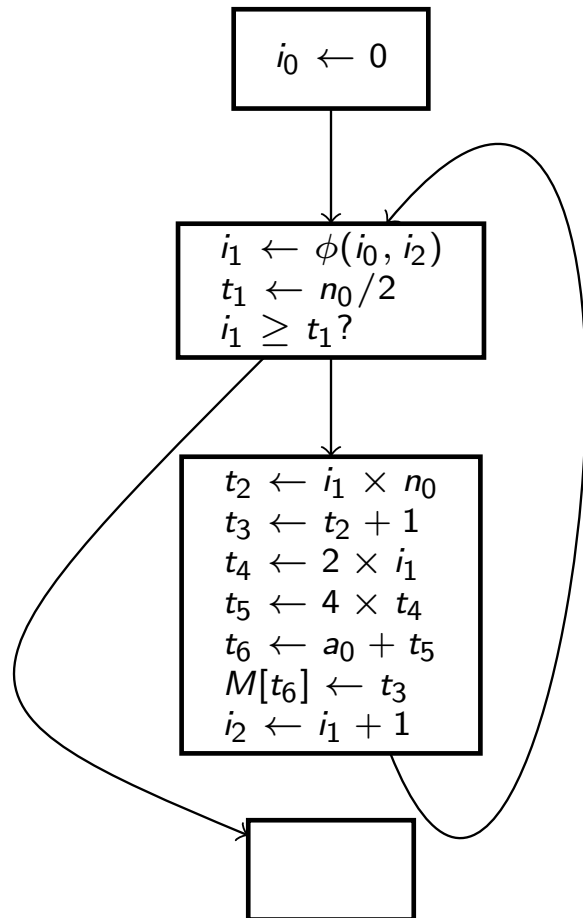
- Nothing happens when computing downsafe since both are downsafe.
- Both therefore are can-be-available.
- None is later since each has an operand with has-real-use.
- Both are will-be-available.
- During finalize1 the dominator tree is again traversed which will result in the \perp operand being replaced with the expression $b_2 \times c_3$, and t_3 , t_4 , and t_5 being removed by setting their reload attributes to true and the save attribute of their representative expressions (the one which produced the redundancy class) to true.

Operator strength reduction

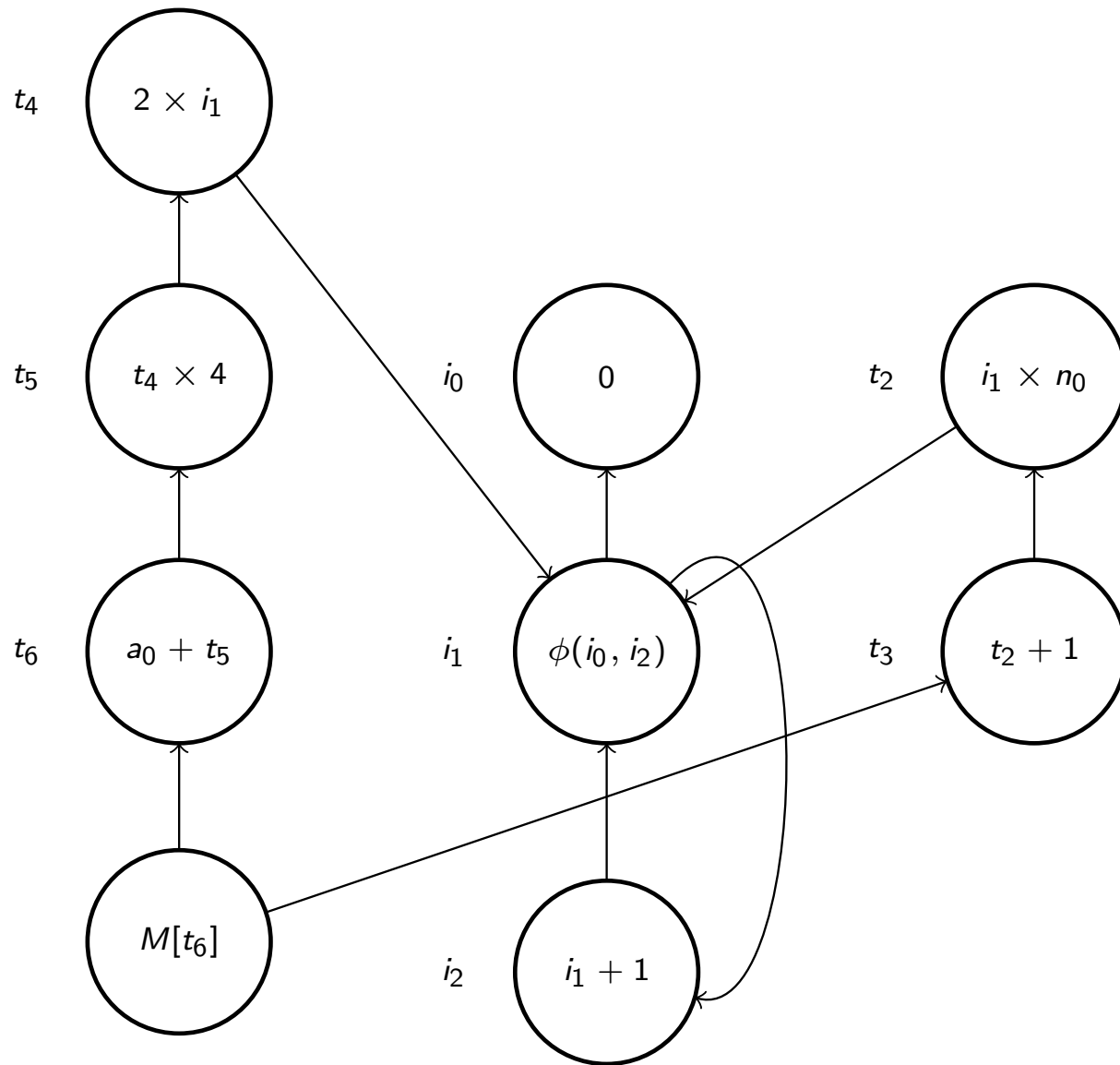
```
void work(float* a, size_t n)
{
    size_t    i;

    for (i = 0; i < n/2; i++)
        a[2*i] = i*n+1;
}
```

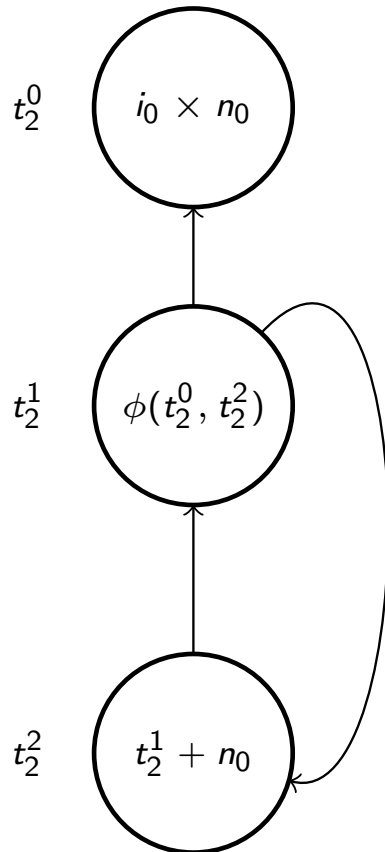
Operator strength reduction



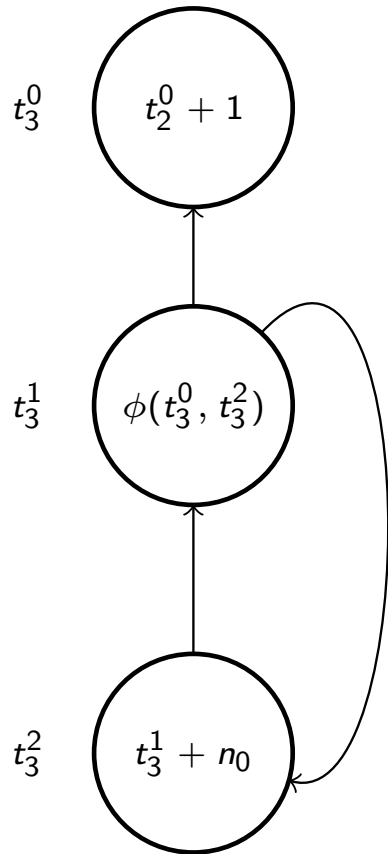
SSA-graph



During the execution of Tarjan's algorithm, i is classified as an induction variable, which leads to its strongly connected component is copied and modified for t_2 as follows:

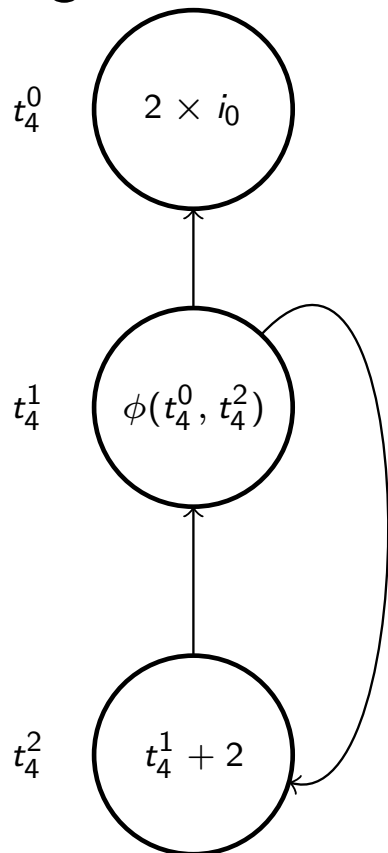


The use of t_2 is changed to instead use t_2^1 . The computation of t_3 now is an add of an induction variable and a region constant, and the SCC of t_2 is copied and modified for t_3 (see next page).

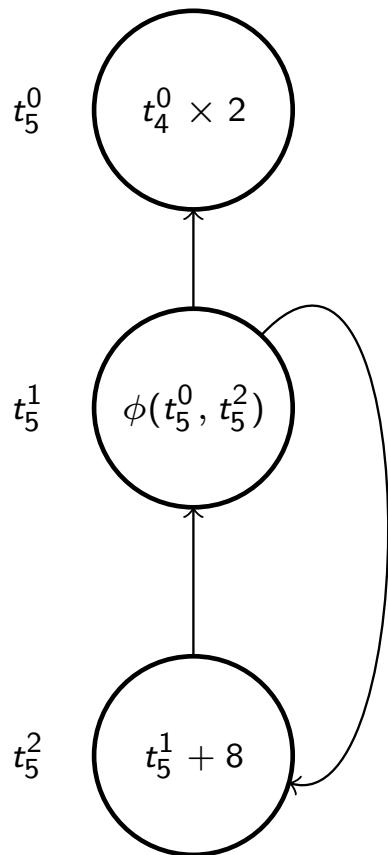


Then the use of t_3 is changed to instead use t_3^1 .

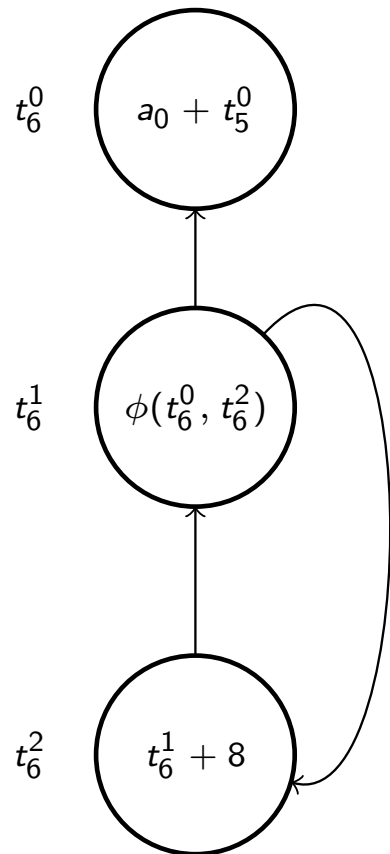
The computation of t_4 also is the product of an induction variable and a region constant and the SCC of i is again copied and now modified for t_4 :



The use of t_4 is changed to instead use t_4^1 . The computation of t_5 now also is a product of an induction variable and a region constant, and the SCC of t_4 is copied and modified for t_5 (see next page).



The use of t_5 is changed to instead use t_5^1 . The computation of t_6 now is the sum of an induction variable and a region constant, and the SCC of t_5 is copied and modified for t_6 (see next page).



The resulting program

The resulting program — after DCE — will look as follows:

