# Reaching definition

In compiler theory, a **reaching definition** for a given instruction is an earlier instruction whose target variable can reach (be assigned to) the given one without an intervening assignment. For example, in the following code:

```
d1 : y := 3
d2 : x := y
```

`d1` is a reaching definition for `d2`. In the following, example, however:

```
d1 : y := 3
d2 : y := 4
d3 : x := y
```

`d1` is no longer a reaching definition for `d3`, because `d2` kills its reach: the value defined in `d1` is no longer available and cannot reach `d3`.

## As analysis

The similarly named **reaching definitions** is a data-flow analysis which statically determines which definitions may reach a given point in the code. Because of its simplicity, it is often used as the canonical example of a data-flow analysis in textbooks. The data-flow confluence operator used is set union, and the analysis is forward flow. Reaching definitions are used to compute use-def chains.

The data-flow equations used for a given basic block $S$ in reaching definitions are:

- $\mathrm{REACH}_{\mathrm{in}}[S] = \bigcup_{p \in pred[S]} \mathrm{REACH}_{\mathrm{out}}[p]$
- $\mathrm{REACH}_{\mathrm{out}}[S] = \mathrm{GEN}[S] \cup (\mathrm{REACH}_{\mathrm{in}}[S] - \mathrm{KILL}[S])$

In other words, the set of reaching definitions going into $S$ are all of the reaching definitions from $S$'s predecessors, $pred[S]$. $pred[S]$ consists of all of the basic blocks that come before $S$ in the control-flow graph. The reaching definitions coming out of $S$ are all reaching definitions of its predecessors minus those reaching definitions whose variable is killed by $S$ plus any new definitions generated within $S$.

For a generic instruction, we define the $\mathrm{GEN}$ and $\mathrm{KILL}$ sets as follows:

- $\mathrm{GEN}[d : y \leftarrow f(x_1, \cdots, x_n)] = \{d\}$ , a set of locally available definitions in a basic block
- $\mathrm{KILL}[d : y \leftarrow f(x_1, \cdots, x_n)] = \mathrm{DEFS}[y] - \{d\}$, a set of definitions (not locally available, but in the rest of the program) killed by definitions in the basic block.

where $\mathrm{DEFS}[y]$ is the set of all definitions that assign to the variable $y$. Here $d$ is a unique label attached to the assigning instruction; thus, the domain of values in reaching definitions are these instruction labels.

## Worklist algorithm

Reaching definition is usually calculated using an iterative worklist algorithm.

Input: control-flow graph CFG = (Nodes, Edges, Entry, Exit)

```
// Initialize
for all CFG nodes n in N,
    OUT[n] = emptyset; // can optimize by OUT[n] = GEN[n];

// put all nodes into the changed set
// N is all nodes in graph,
Changed = N;

// Iterate
while (Changed != emptyset)
{
    choose a node n in Changed;
    // remove it from the changed set
    Changed = Changed -{ n };

    // init IN[n] to be empty
    IN[n] = emptyset;

    // calculate IN[n] from predecessors' OUT[p]
    for all nodes p in predecessors(n)
         IN[n] = IN[n] Union OUT[p];

    oldout = OUT[n]; // save old OUT[n]

    // update OUT[n] using transfer function f_n ()
    OUT[n] = GEN[n] Union (IN[n] -KILL[n]);

    // any change to OUT[n] compared to previous value?
    if (OUT[n] changed) // compare oldout vs. OUT[n]
    {
        // if yes, put all successors of n into the changed set
        for all nodes s in successors(n)
             Changed = Changed U { s };
    }
}
```

# See also

- Dead-code elimination
- Loop-invariant code motion
- Reachable uses
- Static single assignment form

# Further reading


- Aho, Alfred V.; Sethi, Ravi & Ullman, Jeffrey D. (1986). *Compilers: Principles, Techniques, and Tools*. Addison Wesley. ISBN 0-201-10088-6.
- Appel, Andrew W. (1999). *Modern Compiler Implementation in ML*. Cambridge University Press. ISBN 0-521-58274-1.
- Cooper, Keith D. & Torczon, Linda. (2005). *Engineering a Compiler*. Morgan Kaufmann. ISBN 1-55860-698-X.
- Muchnick, Steven S. (1997). *Advanced Compiler Design and Implementation* (https://archive.org/details/advancedcompiler00much). Morgan Kaufmann. ISBN 1-55860-320-4.
- Nielson F., H.R. Nielson; , C. Hankin (2005). *Principles of Program Analysis*. Springer. ISBN 3-540-65410-0.




-