CS107: Dataflow Analysis

Guannan Wei
guannan.wei@tufts.edu

March 24, 2026

Spring 2026

Tufts University

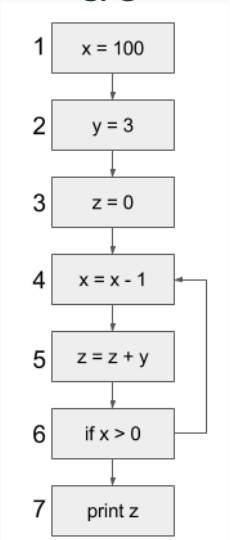
Recap

Dataflow analysis

- Available expressions
- Liveness analysis
- Reaching definitions

Reaching Definitions - Example

CFG



Equations

$$o_1 = \{(x,1)\}$$
$$o_2 = \{(y,2)\} \cup o_1 \downarrow y$$
$$o_3 = \{(z,3)\} \cup o_2 \downarrow z$$
$$o_4 = \{(x,4)\} \cup (o_3 \cup o_6) \downarrow x$$
$$o_5 = \{(z,5)\} \cup o_4 \downarrow z$$
$$o_6 = o_5$$
$$o_7 = o_6$$

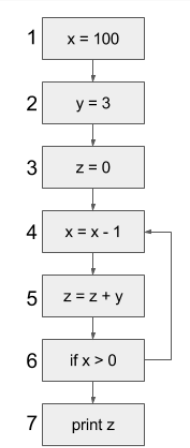
Solution

$$o1 = \{(x,1)\}$$
$$o2 = \{(x,1),(y,2)\}$$
$$o3 = \{(x,1),(y,2),(z,3)\}$$
$$o4 = \{(x,4),(y,2),(z,3),(z,5)\}$$
$$o5 = \{(x,4),(y,2),(z,5)\}$$
$$o6 = \{(x,4),(y,2),(z,5)\}$$
$$o7 = \{(x,4),(y,2),(z,5)\}$$

Using Reaching Definitions

The previous analysis shows that a single constant definition of y reaches node 5.
Therefore, y can be replaced by 3 in node 5.

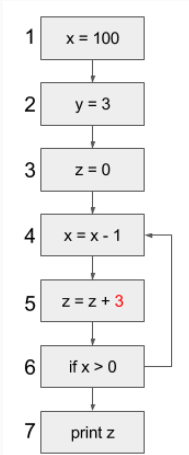
Original CFG



Analysis result

$$o1 = \{(x,1)\}$$
$$o2 = \{(x,1),(y,2)\}$$
$$o3 = \{(x,1),(y,2),(z,3)\}$$
$$o4 = \{(x,4),(y,2),(z,3),(z,5)\}$$
$$o5 = \{(x,4),(y,2),(z,5)\}$$
$$o6 = \{(x,4),(y,2),(z,5)\}$$
$$o7 = \{(x,4),(y,2),(z,5)\}$$

Optimized CFG



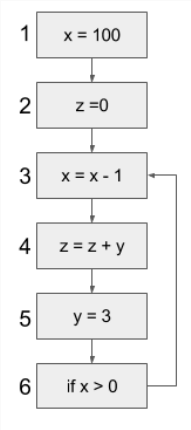
Uninitialized Variables

However, if the language being analyzed permits **uninitialized variables**, the above analysis can produce incorrect results.

Uninitialized Variables

However, if the language being analyzed permits **uninitialized variables**, the above analysis can produce incorrect results.

CFG



Equations

$$o_1 = \{(x,1)\}$$
$$o_2 = \{(z,2)\} \cup o_1 \downarrow z$$
$$o_3 = \{(x,3)\} \cup (o_2 \cup o_6) \downarrow x$$
$$o_4 = \{(z,4)\} \cup o_3 \downarrow z$$
$$o_5 = \{(y,5)\} \cup o_4 \downarrow y$$
$$o_6 = o_5$$

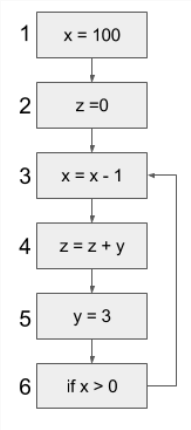
Solution

$$o1 = \{(x,1)\}$$
$$o2 = \{(x,1),(z,2)\}$$
$$o3 = \{(x,3),(y,5),(z,2),(z,4)\}$$
$$o4 = \{(x,3),(y,5),(z,4)\}$$
$$o5 = \{(x,3),(y,5),(z,4)\}$$
$$o6 = \{(x,3),(y,5),(z,4)\}$$

Uninitialized Variables

The solution suggests that **y can be safely replaced by 3 in node 4**, but this is clearly wrong!

CFG



Equations

$$o_1 = \{(x,1)\}$$
$$o_2 = \{(z,2)\} \cup o_1 \downarrow z$$
$$o_3 = \{(x,3)\} \cup (o_2 \cup o_6) \downarrow x$$
$$o_4 = \{(z,4)\} \cup o_3 \downarrow z$$
$$o_5 = \{(y,5)\} \cup o_4 \downarrow y$$
$$o_6 = o_5$$

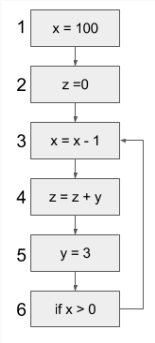
Solution

$$o1 = \{(x,1)\}$$
$$o2 = \{(x,1),(z,2)\}$$
$$o3 = \{(x,3),(y,5),(z,2),(z,4)\}$$
$$o4 = \{(x,3),(y,5),(z,4)\}$$
$$o5 = \{(x,3),(y,5),(z,4)\}$$
$$o6 = \{(x,3),(y,5),(z,4)\}$$

Uninitialized Variables

If the language being analyzed permits uninitialised variables, all variables should be recorded as "initialized in some unknown location" at the entry of the first node!

CFG



Equations

$$o_1 = \{(x,1),(y,?),(z,?)\}$$
$$o_2 = \{(z,2)\} \cup o_1 \downarrow z$$
$$o_3 = \{(x,3)\} \cup (o_2 \cup o_6) \downarrow x$$
$$o_4 = \{(z,4)\} \cup o_3 \downarrow z$$
$$o_5 = \{(y,5)\} \cup o_4 \downarrow y$$
$$o_6 = o_5$$

Solution

$$o1 = \{(x,1),(y,?),(z,?)\}$$
$$o2 = \{(x,1),(y,?),(z,2)\}$$
$$o3 = \{(x,3),(y,?),(y,5),\ (z,2),(z,4)\}$$
$$o4 = \{(x,3),(y,?),(y,5),(z,4)\}$$
$$o5 = \{(x,3),(y,5),(z,4)\}$$
$$o6 = \{(x,3),(y,5),(z,4)\}$$

Analysis #4: Very Busy Expression

An expression is **very busy** at some program point if it will definitely be evaluated before its value changes.

Dataflow analysis can approximate the set of very busy expressions for all program points.

The result of that analysis can then be used to perform code hoisting: a very busy expression can be performed at the earliest point where it is busy.

Example

```
x = a + b;
if (c > 0) {
    y = a + b;
} else {
    z = a + b;
}
```

Example

```
x = a + b;
if (c > 0) {
    y = a + b;
} else {
    z = a + b;
}
```

- Is it a backward or forward analysis?
- Is it a must or may analysis?

Intuitions

- an expression is very busy after a node if it is very busy in all of its successors.
- an expression is very busy before node n if it is either evaluated by n itself, or very busy after n and not killed by n.

  (A node kills an expression e if and only if it redefines a variable appearing in e).
- Finally, no expression is very busy after an exit node.

Equations

We associate to every node n a pair of variables (i_n,o_n) that give the set of very-busy expressions when the node is entered or exited, respectively.

- $i_n = gen_{VB}(n) \cup (o_n \setminus kill_{VB}(n))$

  where $gen_{VB}(n)$ is the set of expressions evaluated by n, and $kill_{VB}(n)$ is the set of expressions redefined by n,

- $o_n = i_{s_1} \cap i_{s_2} \cap \ldots \cap i_{s_k}$, where s_1, ..., s_k are the successors of n.

Simplification: Substituting o_n in i_n, we obtain the following equation for i_n:

$$i_n = gen_{VB}(n) \cup [(i_{s_1} \cap i_{s_2} \cap \ldots \cap i_{s_k}) \setminus kill_{VB}(n)]$$

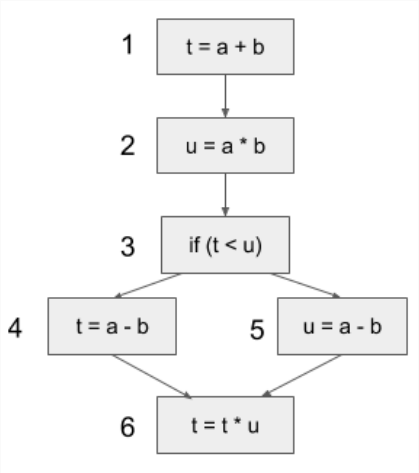
Equation Solving

We are interested in finding the **largest sets** of very busy expressions, as the information conveyed by a set increases with its size.

Therefore, to solve the equations by iteration, we initialize all sets with the set of all non-trivial expressions appearing in the program.

Example

CFG

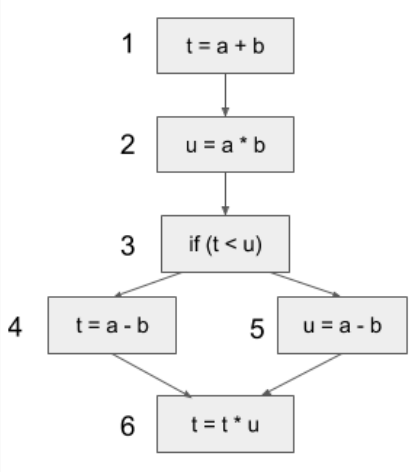


Equations

Solution

Example

CFG



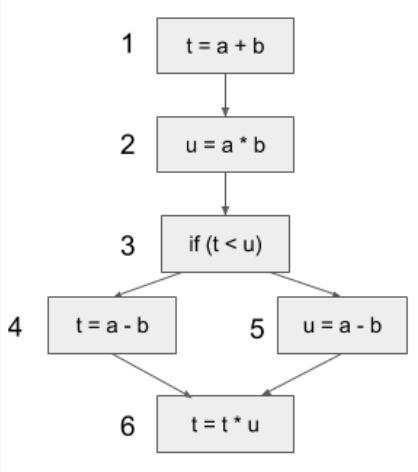
Equations

$$i_1 = \{a+b\} \cup i_2 \downarrow t$$

Solution

Example

CFG



Equations

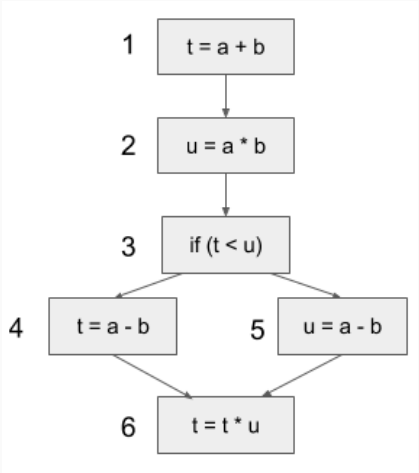
$i_1 = \{a+b\} \cup i_2 \downarrow t$

$i_2 = \{a*b\} \cup i_3 \downarrow u$

Solution

Example

CFG



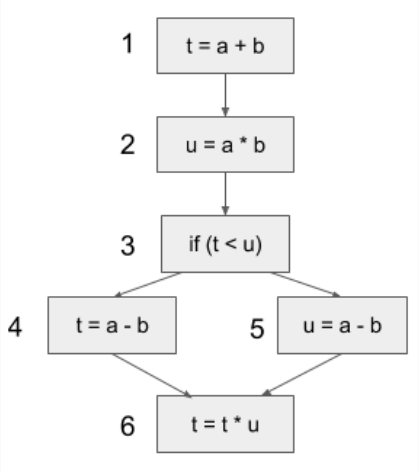
Equations

$$i_1 = \{a+b\} \cup i_2 \downarrow t$$
$$i_2 = \{a*b\} \cup i_3 \downarrow u$$
$$i_3 = i_4 \cap i_5$$

Solution

Example

CFG



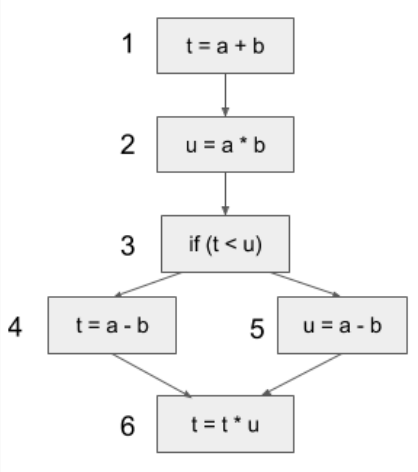
Equations

$$i_1 = \{a+b\} \cup i_2 \downarrow t$$
$$i_2 = \{a*b\} \cup i_3 \downarrow u$$
$$i_3 = i_4 \cap i_5$$
$$i_4 = \{a-b\} \cup i_6 \downarrow t$$

Solution

Example

CFG



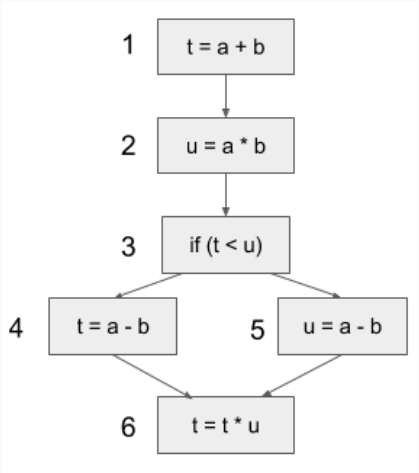
Equations

$$i_1 = \{a+b\} \cup i_2 \downarrow t$$
$$i_2 = \{a*b\} \cup i_3 \downarrow u$$
$$i_3 = i_4 \cap i_5$$
$$i_4 = \{a-b\} \cup i_6 \downarrow t$$
$$i_5 = \{a-b\} \cup i_6 \downarrow u$$

Solution

Example

CFG



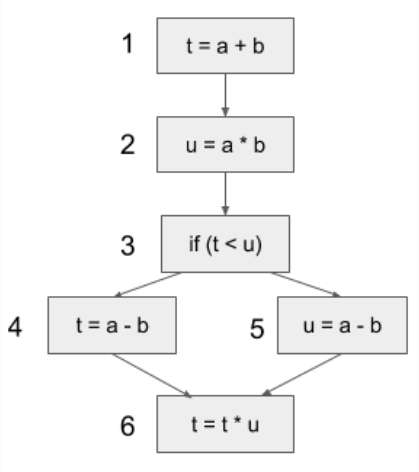
Equations

$$i_1 = \{a+b\} \cup i_2 \downarrow t$$
$$i_2 = \{a*b\} \cup i_3 \downarrow u$$
$$i_3 = i_4 \cap i_5$$
$$i_4 = \{a-b\} \cup i_6 \downarrow t$$
$$i_5 = \{a-b\} \cup i_6 \downarrow u$$
$$i_6 = \{t*u\}$$

Solution

Example

CFG



Equations

$$i_1 = \{a+b\} \cup i_2 \downarrow t$$
$$i_2 = \{a*b\} \cup i_3 \downarrow u$$
$$i_3 = i_4 \cap i_5$$
$$i_4 = \{a-b\} \cup i_6 \downarrow t$$
$$i_5 = \{a-b\} \cup i_6 \downarrow u$$
$$i_6 = \{t*u\}$$

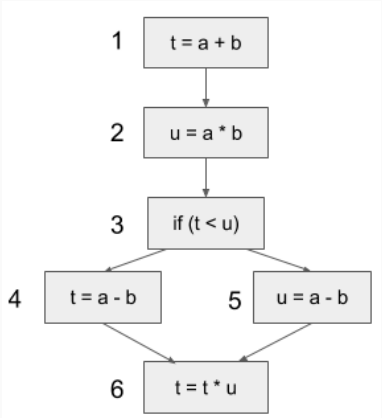
Solution

$$i_1 = \{a+b, a-b, a*b\}$$
$$i_2 = \{a-b, a*b\}$$
$$i_3 = \{a-b\}$$
$$i_4 = \{a-b\}$$
$$i_5 = \{a-b\}$$
$$i_6 = \{t*u\}$$

Using Very Busy Expressions

The previous analysis shows that `a - b` is very busy before the conditional. It can therefore be evaluated earlier.

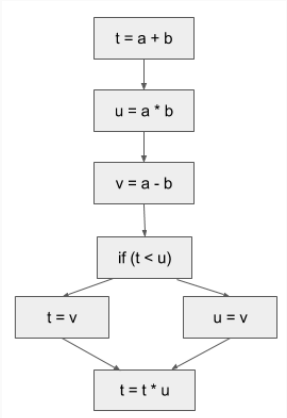
Original CFG



Analysis result

$$i_1 = \{a+b, a-b, a*b\}$$
$$i_2 = \{a-b, a*b\}$$
$$i_3 = \{a-b\}$$
$$i_4 = \{a-b\}$$
$$i_5 = \{a-b\}$$
$$i_6 = \{t*u\}$$

Optimized CFG



Classification Of Dataflow Analyses

Analysis	Input Equation	Output Equation
Available Expressions	$\mathsf{i}_n = \mathsf{o}_{p_1} \cap \mathsf{o}_{p_2} \cap ... \cap \mathsf{o}_{p_k}$	$\mathsf{o}_n = \mathsf{gen}_{AE}(\mathsf{n}) \cup (\mathsf{i}_n \setminus \mathsf{kill}_{AE}(\mathsf{n}))$
Live Vars	$\mathsf{i}_n = \mathsf{gen}_{LV}(\mathsf{n}) \cup (\mathsf{o}_n \setminus \mathsf{kill}_{LV}(\mathsf{n}))$	$\mathsf{o}_n = \mathsf{i}_{s_1} \cup \mathsf{i}_{s_2} \cup ... \cup \mathsf{i}_{s_k}$
Reaching Definitions	$\mathsf{i}_n = \mathsf{o}_{p_1} \cup \mathsf{o}_{p_2} \cup ... \cup \mathsf{o}_{p_k}$	$\mathsf{o}_n = \mathsf{gen}_{RD}(\mathsf{n}) \cup (\mathsf{i}_n \setminus \mathsf{kill}_{RD}(\mathsf{n}))$
Very Busy Expressions	$\mathsf{i}_n = \mathsf{gen}_{VB}(\mathsf{n}) \cup (\mathsf{o}_n \setminus \mathsf{kill}_{VB}(\mathsf{n}))$	$\mathsf{o}_n = \mathsf{i}_{s_1} \cap \mathsf{i}_{s_2} \cap ... \cap \mathsf{i}_{s_k}$

Taxonomy

- **Forward Analysis**: property of a node depends on those of its predecessors - e.g. available expressions, reaching definitions.
- **Backward Analysis**: property of a node depends on those of its successors - e.g. very busy expressions, live variables.
- **Must analysis**: property must be true in all successors/predecessors of a node to be true - e.g. available, very busy expressions.
- **May analysis**: property must be true in at least one successor/predecessor of a node to be true - e.g. reaching definitions, live variables.

Speeding-up Dataflow Analyses

We have seen the specification of dataflow analyses and a simple iterative technique to solve the equations that define them.

Speeding-up Dataflow Analyses

We have seen the specification of dataflow analyses and a simple iterative technique to solve the equations that define them.

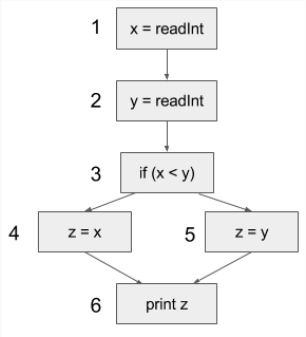
Several techniques can be used to speed up the various dataflow analyses:

- work-list algorithms can avoid useless computations,
- the equations can be ordered in order to propagate information faster,
- the analyses can be performed on smaller control-flow graphs, where nodes are basic blocks instead of individual instructions,
- bit-vectors can be used to represent sets.

Running Example

We will reuse the live variable analysis (backward/may) example to illustrate the techniques used to speed up dataflow analyses.

CFG



Equations

$$
\begin{aligned}
i_1 &= i_2 \setminus \{x\} \\
i_2 &= i_3 \setminus \{y\} \\
i_3 &= \{x,y\} \cup (i_4 \cup i_5) \\
i_4 &= \{x\} \cup (i_6 \setminus \{z\}) \\
i_5 &= \{y\} \cup (i_6 \setminus \{z\}) \\
i_6 &= \{z\}
\end{aligned}
$$

Solution

$$
\begin{aligned}
i_1 &= \{\} \\
i_2 &= \{x\} \\
i_3 &= \{x,y\} \\
i_4 &= \{x\} \\
i_5 &= \{y\} \\
i_6 &= \{z\}
\end{aligned}
$$

Base Case: Iteration

How many iterations and computations needed to reach the fixed point?

$i_1 = i_2 \setminus \{x\}$, $i_2 = i_3 \setminus \{y\}$, $i_3 = \{x,y\} \cup (i_4 \cup i_5)$

$i_4 = \{x\} \cup (i_6 \setminus \{z\})$, $i_5 = \{y\} \cup (i_6 \setminus \{z\})$, $i_6 = \{z\}$

Base Case: Iteration

How many iterations and computations needed to reach the fixed point?

$i_1 = i_2 \setminus \{x\}$, $i_2 = i_3 \setminus \{y\}$, $i_3 = \{x,y\} \cup (i_4 \cup i_5)$

$i_4 = \{x\} \cup (i_6 \setminus \{z\})$, $i_5 = \{y\} \cup (i_6 \setminus \{z\})$, $i_6 = \{z\}$

it.	i_1	i_2	i_3	i_4	i_5	i_6
0	{}	{}	{}	{}	{}	{}

Base Case: Iteration

How many iterations and computations needed to reach the fixed point?

$i_1 = i_2 \setminus \{x\}$, $i_2 = i_3 \setminus \{y\}$, $i_3 = \{x,y\} \cup (i_4 \cup i_5)$

$i_4 = \{x\} \cup (i_6 \setminus \{z\})$, $i_5 = \{y\} \cup (i_6 \setminus \{z\})$, $i_6 = \{z\}$

it.	i_1	i_2	i_3	i_4	i_5	i_6
0	{}	{}	{}	{}	{}	{}
1	{}	{}	{x,y}	{x}	{y}	{z}

Base Case: Iteration

How many iterations and computations needed to reach the fixed point?

$i_1 = i_2 \setminus \{x\}$, $i_2 = i_3 \setminus \{y\}$, $i_3 = \{x,y\} \cup (i_4 \cup i_5)$

$i_4 = \{x\} \cup (i_6 \setminus \{z\})$, $i_5 = \{y\} \cup (i_6 \setminus \{z\})$, $i_6 = \{z\}$

it.	i_1	i_2	i_3	i_4	i_5	i_6
0	{}	{}	{}	{}	{}	{}
1	{}	{}	{x,y}	{x}	{y}	{z}
2	{}	{x}	{x,y}	{x}	{y}	{z}

Base Case: Iteration

How many iterations and computations needed to reach the fixed point?

$i_1 = i_2 \setminus \{x\}$, $i_2 = i_3 \setminus \{y\}$, $i_3 = \{x,y\} \cup (i_4 \cup i_5)$

$i_4 = \{x\} \cup (i_6 \setminus \{z\})$, $i_5 = \{y\} \cup (i_6 \setminus \{z\})$, $i_6 = \{z\}$

it.	i_1	i_2	i_3	i_4	i_5	i_6
0	{}	{}	{}	{}	{}	{}
1	{}	{}	{x,y}	{x}	{y}	{z}
2	{}	{x}	{x,y}	{x}	{y}	{z}
3	{}	{x}	{x,y}	{x}	{y}	{z}

Base Case: Iteration

How many iterations and computations needed to reach the fixed point?

$i_1 = i_2 \setminus \{x\}$, $i_2 = i_3 \setminus \{y\}$, $i_3 = \{x,y\} \cup (i_4 \cup i_5)$

$i_4 = \{x\} \cup (i_6 \setminus \{z\})$, $i_5 = \{y\} \cup (i_6 \setminus \{z\})$, $i_6 = \{z\}$

it.	i_1	i_2	i_3	i_4	i_5	i_6
0	{}	{}	{}	{}	{}	{}
1	{}	{}	{x,y}	{x}	{y}	{z}
2	{}	{x}	{x,y}	{x}	{y}	{z}
3	{}	{x}	{x,y}	{x}	{y}	{z}

Computing the solution to the equations using the standard iterative technique requires 3 iterations, each of which requires 6 computations, for a total of 18 computations:

Work-List Algorithm

Computing the fixed point by simple iteration as we did works, but is wasteful as the information for all nodes is recomputed at every iteration.

It is possible to do better by remembering, for every variable v, the set `dep(v)` of the variables whose value depends on the value of v itself.

Example: $i_1 = i_2 \setminus \{x\}$, $i_2 = i_3 \setminus \{y\}$, then i_1 depends on i_2.

Then, whenever the value of some variable vchanges, we only re-compute the value of the variables that belong to `dep(v)`.

Work-List

$i_1 = i_2 \setminus \{x\}$, $i_2 = i_3 \setminus \{y\}$, $i_3 = \{x,y\} \cup (i_4 \cup i_5)$, $i_4 = \{x\} \cup (i_6 \setminus \{z\})$, $i_5 = \{y\} \cup (i_6 \setminus \{z\})$, $i_6 = \{z\}$

it.	q	i_1	i_2	i_3	i_4	i_5	i_6
0	[i_1,i_2,i_3,i_4,i_5,i_6]	{}	{}	{}	{}	{}	{}
1	[i_2,i_3,i_4,i_5,i_6]	{}	{}	{}	{}	{}	{}
2	[i_3,i_4,i_5,i_6]	{}	{}	{}	{}	{}	{}
3	[i_4,i_5,i_6,i_2]	{}	{}	{x,y}	{}	{}	{}
4	[i_5,i_6,i_2,i_3]	{}	{}	{x,y}	{x}	{}	{}
5	[i_6,i_2,i_3]	{}	{}	{x,y}	{x}	{y}	{}
6	[i_2,i_3,i_4,i_5]	{}	{}	{x,y}	{x}	{y}	{z}
7	[i_3,i_4,i_5,i_1]	{}	{x}	{x,y}	{x}	{y}	{z}
8	[i_4,i_5,i_1]	{}	{x}	{x,y}	{x}	{y}	{z}
9	[i_5,i_1]	{}	{x}	{x,y}	{x}	{y}	{z}
10	[i_1]	{}	{x}	{x,y}	{x}	{y}	{z}
11	[]	{}	{x}	{x,y}	{x}	{y}	{z}

Work-List Algorithm

Using the work-list, “only” 11 computations were required to compute the result.

Work-List Algorithm

Using the work-list, “only” 11 computations were required to compute the result.

In Scala:

```
def solve[T](eqs: Seq[(Int => T) => T],
             dep: Int => List[Int],
             init: T): (Int => T) = {
  def loop(q: List[Int], sol: Map[Int, T]): (Int => T) = q match {
    case i :: rest =>
      val y = eqs(i)(sol)
      if (y == sol(i))
        loop(rest, sol)
      else
        loop(rest ::: (dep(i) diff q), sol + i->y)
    case Nil =>
      sol
  }
  loop(List.range(0, eqs.length), Map.empty withDefaultValue init)
}
```

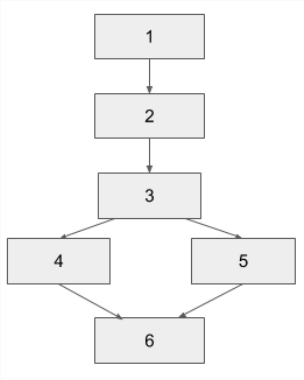
Node Ordering

It is however clear that the process could be even faster if the elements of the work-list are ordered in the reverse order. This is because live variables analysis is a backward analysis.

The goal of node ordering is to order the elements of the work-list in such a way that the solution is computed as fast as possible.

(Reverse) Post-Order

For backward analyses, ordering the variables in the worklist according to a post-order traversal of the CFG nodes speeds up convergence. For forward analyses, reverse post-order has the same characteristic.



Post-order:

6 5 4 3 2 1 or 6 4 5 3 2 1

Reverse post-order:

1 2 3 4 5 6 or 1 2 3 5 4 6

Note: reverse post-order is not the same as pre-order!

Pre-order:

1 2 3 4 6 5 or 1 2 3 5 6 4

Post-Order Work-List

Let's see how post-ordering the work-list speeds up convergence for our example:

$i_1 = i_2 \setminus \{x\}$, $i_2 = i_3 \setminus \{y\}$, $i_3 = \{x,y\} \cup (i_4 \cup i_5)$

$i_4 = \{x\} \cup (i_6 \setminus \{z\})$, $i_5 = \{y\} \cup (i_6 \setminus \{z\})$, $i_6 = \{z\}$

it.	q	i_1	i_2	i_3	i_4	i_5	i_6
0	[**i_6**,i_5,i_4,i_3,i_2,i_1]	{}	{}	{}	{}	{}	{}
1	[**i_5**,i_4,i_3,i_2,i_1]	{}	{}	{}	{}	{}	{z}
2	[**i_4**,i_3,i_2,i_1]	{}	{}	{}	{}	{y}	{z}
3	[**i_3**,i_2,i_1]	{}	{}	{}	{x}	{y}	{z}
4	[**i_2**,i_1]	{}	{}	{x,y}	{x}	{y}	{z}
5	[**i_1**]	{}	{x}	{x,y}	{x}	{y}	{z}
6	[]	{}	{x}	{x,y}	{x}	{y}	{z}

Post-Order Work-List

Let's see how post-ordering the work-list speeds up convergence for our example:

it.	q	i_1	i_2	i_3	i_4	i_5	i_6
0	[**i_6**,i_5,i_4,i_3,i_2,i_1]	{}	{}	{}	{}	{}	{}
1	[**i_5**,i_4,i_3,i_2,i_1]	{}	{}	{}	{}	{}	{z}
2	[**i_4**,i_3,i_2,i_1]	{}	{}	{}	{}	{y}	{z}
3	[**i_3**,i_2,i_1]	{}	{}	{}	{x}	{y}	{z}
4	[**i_2**,i_1]	{}	{}	{x,y}	{x}	{y}	{z}
5	[**i_1**]	{}	{x}	{x,y}	{x}	{y}	{z}
6	[]	{}	{x}	{x,y}	{x}	{y}	{z}

Now only 6 computations are required to obtain the solution!

Post-Order Work-List

Let's see how post-ordering the work-list speeds up convergence for our example:

it.	q	i_1	i_2	i_3	i_4	i_5	i_6
0	[$\mathbf{i}_6$,i_5,i_4,i_3,i_2,i_1]	{}	{}	{}	{}	{}	{}
1	[$\mathbf{i}_5$,i_4,i_3,i_2,i_1]	{}	{}	{}	{}	{}	{z}
2	[$\mathbf{i}_4$,i_3,i_2,i_1]	{}	{}	{}	{}	{y}	{z}
3	[$\mathbf{i}_3$,i_2,i_1]	{}	{}	{}	{x}	{y}	{z}
4	[$\mathbf{i}_2$,i_1]	{}	{}	{x,y}	{x}	{y}	{z}
5	[$\mathbf{i}_1$]	{}	{x}	{x,y}	{x}	{y}	{z}
6	[]	{}	{x}	{x,y}	{x}	{y}	{z}

Now only 6 computations are required to obtain the solution!

What if the CFG contains loops? In that case, post-order is not defined. Find strongly connected components first and reduce to DAG.

Basic Blocks

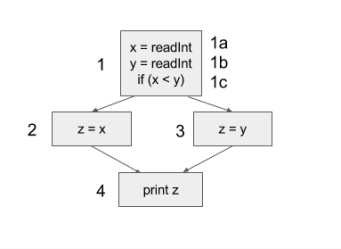
Until now, CFG nodes were single instructions. In practice, **basic blocks** tend to be used as nodes, to reduce the size of the CFG.

When dataflow analysis is performed on a CFG composed of basic blocks, a pair of variables is attached to every block, not to every instruction.

Once the solution is known for basic blocks, computing the solution for individual instructions is easy.

CFG With Basic Blocks

CFG



Equations

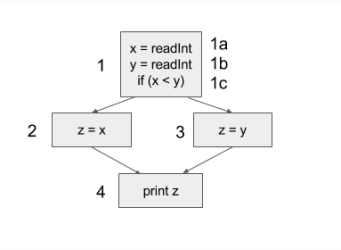
$$i_1 = \{i_2 \cup i_3\} \setminus \{x,y\}$$
$$i_2 = \{x\} \cup (i_4 \setminus \{z\})$$
$$i_3 = \{y\} \cup (i_4 \setminus \{z\})$$
$$i_4 = \{z\}$$

Solution

$$i_1 = \{\}$$
$$i_2 = \{x\}$$
$$i_3 = \{y\}$$
$$i_4 = \{z\}$$

CFG With Basic Blocks

CFG



Equations

$$i_1 = \{i_2 \cup i_3\} \setminus \{x,y\}$$
$$i_2 = \{x\} \cup (i_4 \setminus \{z\})$$
$$i_3 = \{y\} \cup (i_4 \setminus \{z\})$$
$$i_4 = \{z\}$$

Solution

$$i_1 = \{\}$$
$$i_2 = \{x\}$$
$$i_3 = \{y\}$$
$$i_4 = \{z\}$$

The solution for individual instructions is computed from the basic-block solution, in a single pass - here backwards:

$$i_{1_c} = \{x,y\} \cup (i_2 \cup i_3) = \{x,y\}$$

$$i_{1_b} = i_{1_c} \setminus \{y\} = \{x\}$$

$$i_{1_a} = i_{1_b} \setminus \{x\} = \{\}$$

Improving Representation: Bit Vectors

All dataflow analyses we have seen work on sets of values. If these sets are dense, a good way to represent them is to use bit vectors:

- a bit is associated to every possible element of the set,
- and its value is 1 if and only if the corresponding element belongs to the set.

On such a representation, set union is bitwise-or, set intersection is bitwise-and, set difference is bitwise-and composed with bitwise-negation.

Bit Vectors Example

Original equations

$i1 = i2 \setminus \{x\}$

$i2 = i3 \setminus \{y\}$

$i3 = \{x,y\} \cup (i4 \cup i5)$

$i4 = \{x\} \cup (i6 \setminus \{z\})$

$i5 = \{y\} \cup (i6 \setminus \{z\})$

$i6 = \{z\}$

Bit vector equations

$i1 = i2 \;\&\; \sim 100$

$i2 = i3 \;\&\; \sim 010$

$i3 = 110 \mid (i4 \mid i5)$

$i4 = 100 \mid (i6 \;\&\; \sim 001)$

$i5 = 010 \mid (i6 \;\&\; \sim 001)$

$i6 = 001$

Bit Vectors Example

Original equations

$i1 = i2 \setminus \{x\}$

$i2 = i3 \setminus \{y\}$

$i3 = \{x,y\} \cup (i4 \cup i5)$

$i4 = \{x\} \cup (i6 \setminus \{z\})$

$i5 = \{y\} \cup (i6 \setminus \{z\})$

$i6 = \{z\}$

Bit vector equations

$i1 = i2 \;\&\; \sim 100$

$i2 = i3 \;\&\; \sim 010$

$i3 = 110 \mid (i4 \mid i5)$

$i4 = 100 \mid (i6 \;\&\; \sim 001)$

$i5 = 010 \mid (i6 \;\&\; \sim 001)$

$i6 = 001$

Original solution

$i1 = \{\}$ $\quad$ $i4 = \{x\}$

$i2 = \{x\}$ $\quad$ $i5 = \{y\}$

$i3 = \{x,y\}$ $\quad$ $i6 = \{z\}$

Bit vector solution

$i1 = 000$ $\quad$ $i4 = 100$

$i2 = 100$ $\quad$ $i5 = 010$

$i3 = 110$ $\quad$ $i6 = 001$

Summary

Dataflow analysis is a framework to approximate various properties about programs.

- Liveness
- Available expressions
- Very busy expressions
- Reaching definitions

The result of those analysis can be used to perform various optimizations like dead-code elimination, constant propagation, etc.