CS107: Dataflow Analysis

Guannan Wei
guannan.wei@tufts.edu

March 12, 2026

Spring 2026

Tufts University

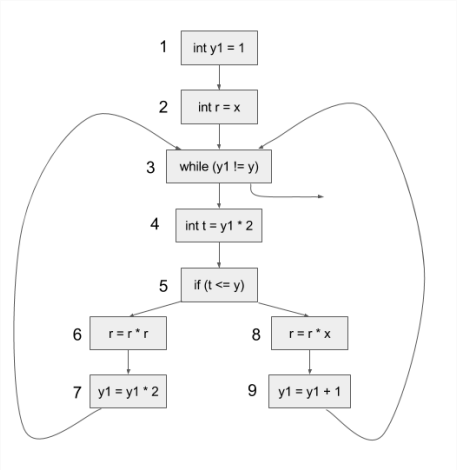
Recap: Available Expressions

- Available expressions: expressions that have already been computed and whose value is still available.
- How to define the analysis?
- How to solve the equations?

Recap: Available Expressions

- Available expressions: expressions that have already been computed and whose value is still available.
- How to define the analysis?
  - For each node n of the control-flow graph, we specify an equation describing the relation between AE before and after node n.
- How to solve the equations?
  - Fixpoint iteration: we initialize all sets with the set of all expressions, and we iteratively apply the equations until a fixed point is reached.

Equations



$$
\begin{aligned}
i_1 &= \{\} & o_1 &= i_1 \\
i_2 &= o_1 & o_2 &= i_2 \\
i_3 &= o_2 \cap o_7 \cap o_9 & o_3 &= i_3 \\
i_4 &= o_3 & o_4 &= \{\ \text{y1 * 2}\ \} \cup i_4 \\
i_5 &= o_4 & o_5 &= i_5 \\
i_6 &= o_5 & o_6 &= i_6 \downarrow \text{r} \\
i_7 &= o_6 & o_7 &= i_7 \downarrow \text{y1} \\
i_8 &= o_5 & o_8 &= i_8 \downarrow \text{r} \\
i_9 &= o_8 & o_9 &= i_9 \downarrow \text{y1}
\end{aligned}
$$

Notation: $S \downarrow x = S \setminus \{$ all expressions using x $\}$

Solving Equations

Simplifying the equations: replace all i_k variables by their values, obtaining a set of equations only involving o_k variables.

For our example, we obtain the following equations about the "out'' facts:

$$
\begin{aligned}
\mathsf{o}_1 &= \{\} & \mathsf{o}_6 &= \mathsf{o}_5 \downarrow \mathsf{r} \\
\mathsf{o}_2 &= \mathsf{o}_1 & \mathsf{o}_7 &= \mathsf{o}_6 \downarrow \mathsf{y1} \\
\mathsf{o}_3 &= \mathsf{o}_2 \cap \mathsf{o}_7 \cap \mathsf{o}_9 & \mathsf{o}_8 &= \mathsf{o}_5 \downarrow \mathsf{r} \\
\mathsf{o}_4 &= \mathsf{o}_3 \cup \{\ \mathsf{y1} * 2\ \} & \mathsf{o}_9 &= \mathsf{o}_8 \downarrow \mathsf{y1} \\
\mathsf{o}_5 &= \mathsf{o}_4
\end{aligned}
$$

Solving Equations

The simpler system can be solved by iterating until a fixed point is reached, which happens after 7 iterations.

it.	1	2	3	4	5	6	7
o_1	$\top$	{}	{}	{}	{}	{}	{}
o_2	$\top$	$\top$	{}	{}	{}	{}	{}
o_3	$\top$	$\top$	R	{}	{}	{}	{}
o_4	$\top$	$\top$	$\top$	{y1*2, r*r, r*x}	{y1*2}	{y1*2}	{y1*2}
o_5	$\top$	$\top$	$\top$	$\top$	{y1*2, r*r, r*x}	{y1*2}	{y1*2}
o_6	$\top$	Y	Y	Y	Y	{y1*2}	{y1*2}
o_7	$\top$	R	{}	{}	{}	{}	{}
o_8	$\top$	Y	Y	Y	Y	{y1*2}	{y1*2}
o_9	$\top$	R	{}	{}	{}	{}	{}

Notation: $Y = \{\texttt{y1*2},\ \texttt{y1+1}\}$, $R = \{\texttt{r*r},\ \texttt{r*x}\}$, $\top = Y \cup R$

Formalization

For a node n of the control-flow graph, general form of equations:

- $\mathsf{i}_n = \mathsf{o}_{p_1} \cap \mathsf{o}_{p_2} \cap \ldots \cap \mathsf{o}_{p_k}$
- $\mathsf{o}_n = \mathsf{gen}_{AE}(\mathsf{n}) \cup (\mathsf{i}_n \setminus \mathsf{kill}_{AE}(\mathsf{n}))$

where

- p_1, ..., p_k are the predecessors of node n,
- $\mathsf{gen}_{AE}(\mathsf{n})$ the set of non-trivial expressions computed by n,
- $\mathsf{kill}_{AE}(\mathsf{n})$ the set of all non-trivial expressions that use a variable modified by n.

Formalization

For a node n of the control-flow graph, general form of equations:

- $\mathsf{i}_n = \mathsf{o}_{p_1} \cap \mathsf{o}_{p_2} \cap \ldots \cap \mathsf{o}_{p_k}$
- $\mathsf{o}_n = \mathsf{gen}_{AE}(\mathsf{n}) \cup (\mathsf{i}_n \setminus \mathsf{kill}_{AE}(\mathsf{n}))$

Intuition:

- input fact is gather from output facts of predecessors (must be available in all predecessors)
- output fact is generated by the node itself ($\mathsf{gen}_{AE}(\mathsf{n})$), or ($\cup$) is available from the input and not removed by the node itself ($\mathsf{i}_n \setminus \mathsf{kill}_{AE}(\mathsf{n})$).

Formalization

For a node n of the control-flow graph, general form of equations:

- $\mathsf{i}_n = \mathsf{o}_{p_1} \cap \mathsf{o}_{p_2} \cap \ldots \cap \mathsf{o}_{p_k}$
- $\mathsf{o}_n = \mathsf{gen}_{AE}(\mathsf{n}) \cup (\mathsf{i}_n \setminus \mathsf{kill}_{AE}(\mathsf{n}))$

Intuition:

- input fact is gather from output facts of predecessors (must be available in all predecessors)
- output fact is generated by the node itself ($\mathsf{gen}_{AE}(\mathsf{n})$), or ($\cup$) is available from the input and not removed by the node itself ($\mathsf{i}_n \setminus \mathsf{kill}_{AE}(\mathsf{n})$).

Further simplification: substituting i_n in o_n, we obtain the following equation for o_n:

$$\mathsf{o}_n = \mathsf{gen}_{AE}(\mathsf{n}) \cup [(\mathsf{o}_{p_1} \cap \mathsf{o}_{p_2} \cap \ldots \cap \mathsf{o}_{p_k}) \setminus \mathsf{kill}_{AE}(\mathsf{n})]$$

Generated Expressions

Simplified equations:

- $\mathsf{o}_n = \mathsf{gen}_{AE}(\mathsf{n}) \cup [(\mathsf{o}_{p_1} \cap \mathsf{o}_{p_2} \cap \ldots \cap \mathsf{o}_{p_k}) \setminus \mathsf{kill}_{AE}(\mathsf{n})]$,
- $\mathsf{gen}_{AE}(\mathsf{n})$ the set of non-trivial expressions computed by n,
- $\mathsf{kill}_{AE}(\mathsf{n})$ the set of all non-trivial expressions that use a variable modified by n.

In order for this equation to be correct, expressions that are computed by n but which use a variable modified by n must not be part of $\mathsf{gen}_{AE}(\mathsf{n})$.

Generated Expressions

Simplified equations:

- $\mathsf{o}_n = \mathsf{gen}_{AE}(\mathsf{n}) \cup [(\mathsf{o}_{p_1} \cap \mathsf{o}_{p_2} \cap \ldots \cap \mathsf{o}_{p_k}) \setminus \mathsf{kill}_{AE}(\mathsf{n})]$,
- $\mathsf{gen}_{AE}(\mathsf{n})$ the set of non-trivial expressions computed by n,
- $\mathsf{kill}_{AE}(\mathsf{n})$ the set of all non-trivial expressions that use a variable modified by n.

In order for this equation to be correct, expressions that are computed by n but which use a variable modified by n must not be part of $\mathsf{gen}_{AE}(\mathsf{n})$.

For example:

$\mathsf{gen}_{AE}(\mathsf{x=y*y}) = \{\mathsf{y*y}\}$ but $\mathsf{gen}_{AE}(\mathsf{y=y*y}) = \{\}$

Generalization: Dataflow Analysis

Available expressions is one example of dataflow analyses. Dataflow analysis is a global analysis framework that can be used to approximate various properties of programs.

- In this lecture, we only consider intra-procedural analysis, that is, analyses that work on a single function at a time.
- The analysis works on a CFG representation of the function.

Generalization: Dataflow Analysis

We will see other examples of dataflow analyses, that can be used to perform several optimizations, for example:

- common sub-expression elimination, as we have seen,
- dead code elimination,
- constant propagation,
- register allocation,
- etc.

Generalization: Dataflow Analysis

Equations:

- $\mathsf{i}_n = \mathsf{o}_{p_1} \cap \mathsf{o}_{p_2} \cap \ldots \cap \mathsf{o}_{p_k}$
- $\mathsf{o}_n = \mathsf{gen}_{AE}(\mathsf{n}) \cup (\mathsf{i}_n \setminus \mathsf{kill}_{AE}(\mathsf{n}))$

Available expressions is a **forward must analysis**:

- Forward analysis: data flow facts are gather from predecessors in the CFG
- Must analysis: at join points, only keep facts that **must** hold for **all** paths

Analysis #2: Live Variable

A variable x is **live** at program point if x will be used on later before x is overwritten.

Intuitions: we need to look at downstream use of variables to determine whether a variable is live or not.

Analysis #2: Live Variable

Liveness is an undecidable property, but a conservative approximation can be computed using dataflow analysis.

Analysis #2: Live Variable

Liveness is an undecidable property, but a conservative approximation can be computed using dataflow analysis.

Rice's theorem: any non-trivial property of the behavior of a program is undecidable.

Example: halting problem.

Analysis #2: Live Variable

Liveness is an undecidable property, but a conservative approximation can be computed using dataflow analysis.

Rice's theorem: any non-trivial property of the behavior of a program is undecidable.

Example: halting problem.

This approximation can then be used, for example, to allocate registers: - a dead variable does not need to be allocated a register. - a set of variables that are never live at the same time can share a single register. - ...

Intuitions

- A variable is live before node `n` if it is read by `n`.

```
// before this x is live
val z = x + 1
```

Intuitions

- A variable is live before node `n` if it is read by `n`.

```
// before this x is live
val z = x + 1
```

- A variable can be live after node `n` if it is not written by `n`.

```
val z = x + 1
// after this x can be live, but z is not live
```

Intuitions

- A variable is live before node `n` if it is read by `n`.

```
// before this x is live
val z = x + 1
```

- A variable can be live after node `n` if it is not written by `n`.

```
val z = x + 1
// after this x can be live, but z is not live
```

- A variable is live after a node if it is live before any of its successors.

```
val z = x + 1
// x is live if any of its successors read x
if (...) { x } else { y }
```

Intuitions

- A variable is live before node `n` if it is read by `n`.

```
// before this x is live
val z = x + 1
```

- A variable can be live after node `n` if it is not written by `n`.

```
val z = x + 1
// after this x can be live, but z is not live
```

- A variable is live after a node if it is live before any of its successors.

```
val z = x + 1
// x is live if any of its successors read x
if (...) { x } else { y }
```

- Finally, no variable is live after an exit node.

Specifying Equations

We associate to every node n a pair of variables:

- i_n the set of live variables before node n,
- o_n the set of live variables after node n.

Specifying Equations

We associate to every node n a pair of variables:

- i_n the set of live variables before node n,
- o_n the set of live variables after node n.

Definition:

- $\mathsf{i}_n = \mathsf{gen}_{LV}(\mathsf{n}) \cup (\mathsf{o}_n \setminus \mathsf{kill}_{LV}(\mathsf{n}))$

  where $\mathsf{gen}_{LV}(\mathsf{n})$ is the set of variables read by n, and $\mathsf{kill}_{LV}(\mathsf{n})$ is the set of variables written by n.

- $\mathsf{o}_n = \mathsf{i}_{s_1} \cup \mathsf{i}_{s_2} \cup \ldots \cup \mathsf{i}_{s_k}$, where s_1, ..., s_k are the successors of n.

Specifying Equations

We associate to every node n a pair of variables:

- i_n the set of live variables before node n,
- o_n the set of live variables after node n.

Definition:

- $\mathsf{i}_n = \mathsf{gen}_{LV}(\mathsf{n}) \cup (\mathsf{o}_n \setminus \mathsf{kill}_{LV}(\mathsf{n}))$

  where $\mathsf{gen}_{LV}(\mathsf{n})$ is the set of variables read by n, and $\mathsf{kill}_{LV}(\mathsf{n})$ is the set of variables written by n.

- $\mathsf{o}_n = \mathsf{i}_{s_1} \cup \mathsf{i}_{s_2} \cup \ldots \cup \mathsf{i}_{s_k}$, 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:

$$\mathsf{i}_n = \mathsf{gen}_{LV}(\mathsf{n}) \cup [(\mathsf{i}_{s_1} \cup \mathsf{i}_{s_2} \cup \ldots \cup \mathsf{i}_{s_k}) \setminus \mathsf{kill}_{LV}(\mathsf{n})]$$

Solving Equations

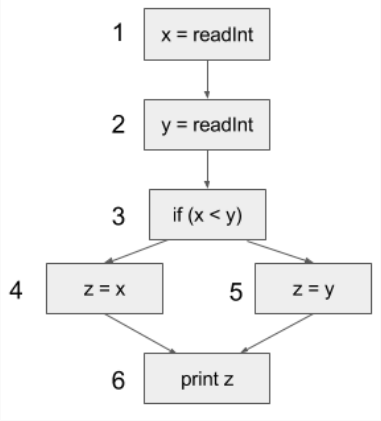
We are interested in finding the **smallest** sets of variables live at a given point.

Saying all possible variables are live is sound, but it does not convey any useful information for compiler to optimize.

Therefore, to solve the equations by iteration, we initialize all sets with the empty set.

Example

CFG

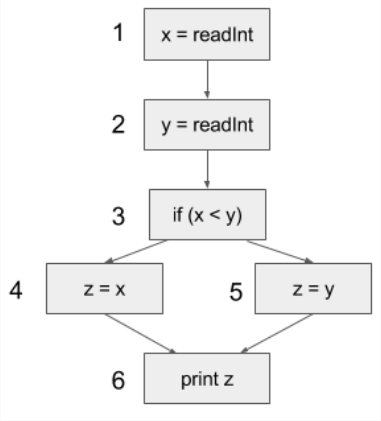


Equations

Solution

Example

CFG



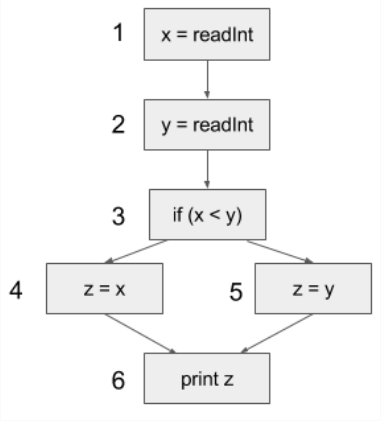
Equations

$$i_1 = i_2 \setminus \{x\}$$

Solution

Example

CFG



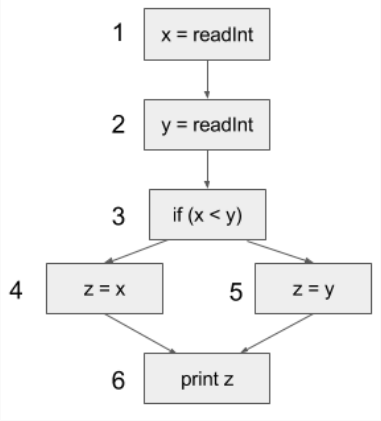
Equations

$$i_1 = i_2 \setminus \{x\}$$
$$i_2 = i_3 \setminus \{y\}$$

Solution

Example

CFG



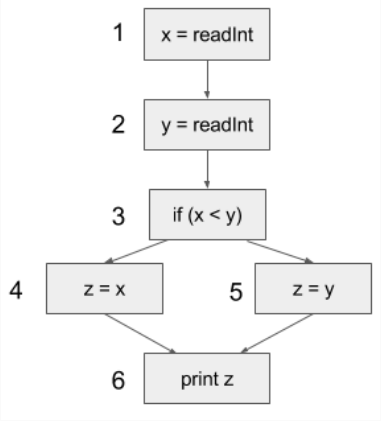
Equations

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

Solution

Example

CFG



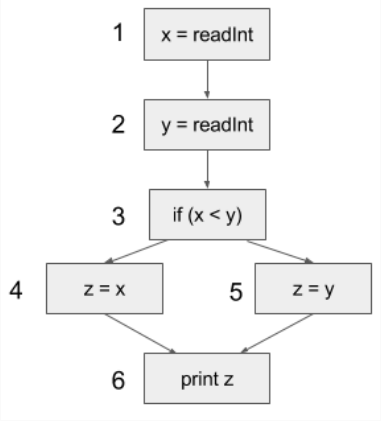
Equations

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

Solution

Example

CFG



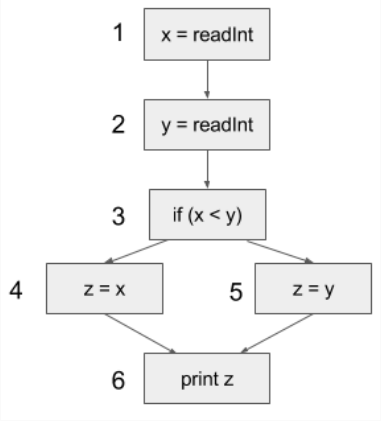
Equations

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

Solution

Example

CFG



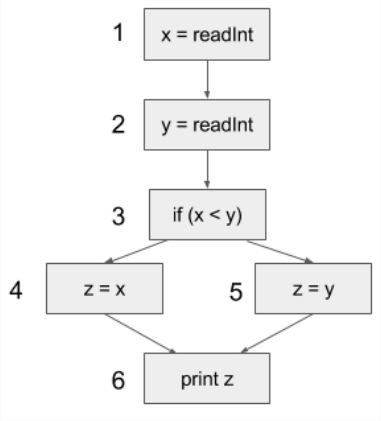
Equations

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

Solution

Example

CFG



Equations

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

Solution

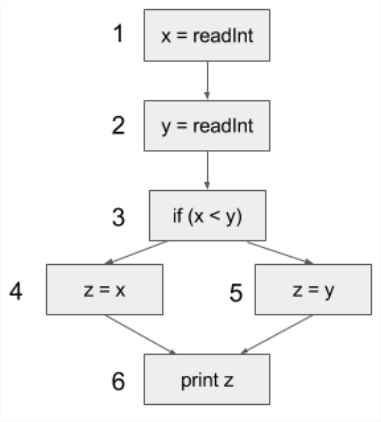
$$i_1 = \{\}$$
$$i_2 = \{x\}$$
$$i_3 = \{x,y\}$$
$$i_4 = \{x\}$$
$$i_5 = \{y\}$$
$$i_6 = \{z\}$$

Using Live Variables

The previous analysis shows that neither x nor y are live at the same time as z.

Therefore, z can be replaced by x or y, thereby removing one assignment.

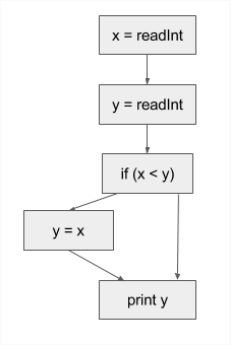
Original CFG



analysis result

$$i_1 = \{\}$$
$$i_2 = \{x\}$$
$$i_3 = \{x,y\}$$
$$i_4 = \{x\}$$
$$i_5 = \{y\}$$
$$i_6 = \{z\}$$

Optimized CFG



Live Variable Analysis

Equations:

- $\mathsf{i}_n = \mathsf{gen}_{LV}(\mathsf{n}) \cup (\mathsf{o}_n \setminus \mathsf{kill}_{LV}(\mathsf{n}))$
- $\mathsf{o}_n = \mathsf{i}_{s_1} \cup \mathsf{i}_{s_2} \cup \ldots \cup \mathsf{i}_{s_k}$

Live variable analysis is a **backward may analysis**:

- Backward analysis: data flow facts are gather from successors in the CFG
- May analysis: at join points, keep facts that **may** hold for **some** path

Comparing with Available Expressions

Available expressions:

- Forward must analysis
- $\mathsf{i}_n = \mathsf{o}_{p_1} \cap \mathsf{o}_{p_2} \cap \ldots \cap \mathsf{o}_{p_k}$
- $\mathsf{o}_n = \mathsf{gen}_{AE}(\mathsf{n}) \cup (\mathsf{i}_n \setminus \mathsf{kill}_{AE}(\mathsf{n}))$

Live variable analysis:

- Backward may analysis
- $\mathsf{i}_n = \mathsf{gen}_{LV}(\mathsf{n}) \cup (\mathsf{o}_n \setminus \mathsf{kill}_{LV}(\mathsf{n}))$
- $\mathsf{o}_n = \mathsf{i}_{s_1} \cup \mathsf{i}_{s_2} \cup \ldots \cup \mathsf{i}_{s_k}$

Analysis #3: Reaching Definitions

The reaching definitions for a program point are the **assignments** that may have defined the values of variables at that point.

Dataflow analysis can approximate the set of reaching definitions for all program points. These sets can then be used to perform constant propagation, for example.

Example

```
x = 100
y = 3
z = 0
do {
  x = x - 1
  // at this point, what are the reaching definitions?
  z = z + y
} while (x > 0)
print(z)
```

Example

```
x = 100
y = 3
z = 0
do {
  x = x - 1
  // x = x-1, y=3, z=0, and z=z+y
  z = z + y
} while (x > 0)
print(z)
```

Intuitions

- A definition reaches the beginning of a node if it reaches the exit of any of its predecessors.
- Moreover, a definition contained in a node n always reaches the end of n itself.
- Finally, a definition reaches the end of a node n if it reaches the beginning of n and is not killed by n itself.

Intuitions

- A definition reaches the beginning of a node if it reaches the exit of any of its predecessors.
- Moreover, a definition contained in a node `n` always reaches the end of `n` itself.
- Finally, a definition reaches the end of a node `n` if it reaches the beginning of `n` and is not killed by `n` itself.

(A node `n` kills a definition `d` if and only if `n` is a definition and defines the same variable as `d`.)

As a first approximation, we consider that no definition reaches the beginning of the entry node.

Equations

We associate to every node n a pair of variables (i_n,o_n) that give the set of definitions reaching the entry and exit of n, respectively.

- $\mathsf{i}_n = \mathsf{o}_{p_1} \cup \mathsf{o}_{p_2} \cup \ldots \cup \mathsf{o}_{p_k}$, where p_1, ..., p_k are the predecessors of n.
- $\mathsf{o}_n = \mathsf{gen}_{RD}(\mathsf{n}) \cup (\mathsf{i}_n \setminus \mathsf{kill}_{RD}(\mathsf{n}))$

  where $\mathsf{gen}_{RD}(\mathsf{n})$ is $\{\mathsf{n}\}$ if n is a definition, $\{\}$ otherwise, and $\mathsf{kill}_{RD}(\mathsf{n})$ is the set of definitions defining the same variable as n itself.

Equations

We associate to every node n a pair of variables (i_n,o_n) that give the set of definitions reaching the entry and exit of n, respectively.

- $i_n = o_{p_1} \cup o_{p_2} \cup \ldots \cup o_{p_k}$, where p_1, ..., p_k are the predecessors of n.
- $o_n = gen_{RD}(n) \cup (i_n \setminus kill_{RD}(n))$

  where $gen_{RD}(n)$ is $\{n\}$ if n is a definition, $\{\}$ otherwise, and $kill_{RD}(n)$ is the set of definitions defining the same variable as n itself.

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

$$o_n = gen_{RD}(n) \cup [(o_{p_1} \cup o_{p_2} \cup \ldots \cup o_{p_k}) \setminus kill_{RD}(n)]$$

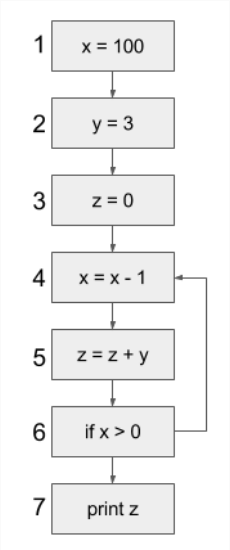
Equation Solving

We are interested in finding the **smallest** sets of definitions reaching a point.

Therefore, to solve the equations by iteration, we initialize all sets with the empty set.

Example

CFG

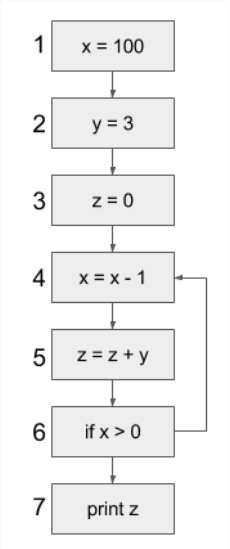


Equations

Solution

Example

CFG



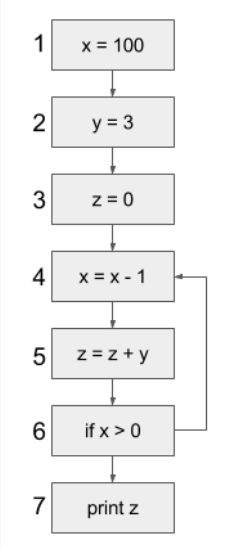
Equations

$o_1 = \{(x,1)\}$

Solution

Example

CFG



Equations

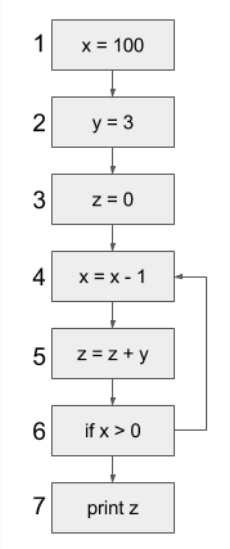
$$o_1 = \{(x,1)\}$$

$$o_2 = \{(y,2)\} \cup o_1 \downarrow y$$

Solution

Example

CFG



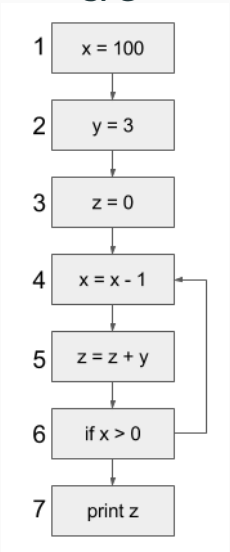
Equations

$$o_1 = \{(x,1)\}$$
$$o_2 = \{(y,2)\} \cup o_1 \downarrow y$$
$$o_3 = \{(z,3)\} \cup o_2 \downarrow z$$

Solution

Example

CFG



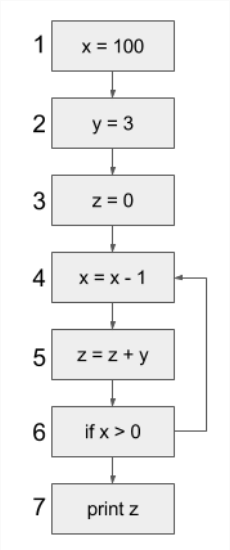
Equations

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

Solution

Example

CFG



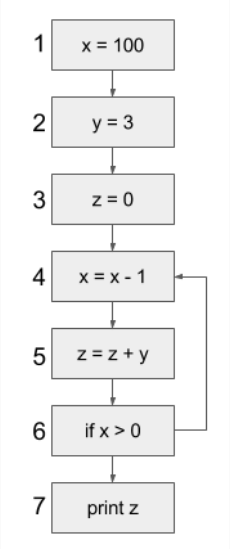
Equations

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

Solution

Example

CFG



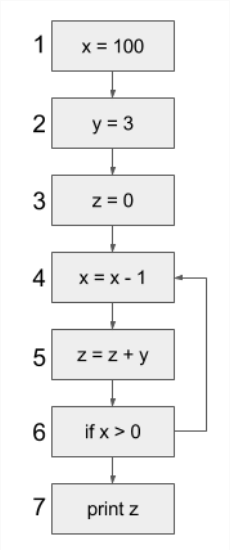
Equations

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

Solution

Example

CFG



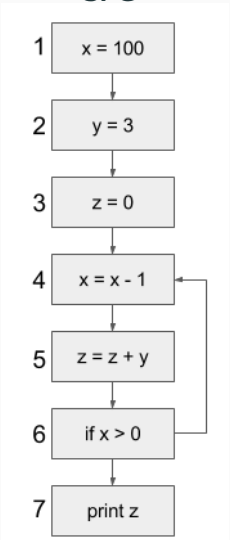
Equations

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

Solution

Example

CFG



Equations

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

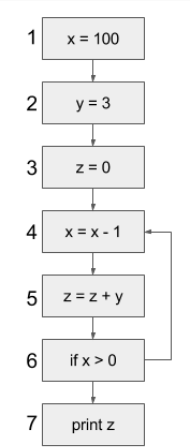
Solution

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

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.

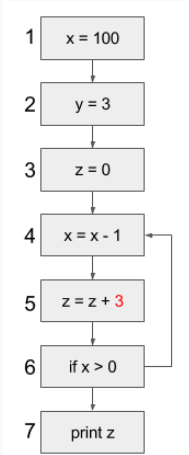
Original CFG



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



Questions?