Redundancy Optimizations

- Constant Folding
- Copy Propagation
- Dead Code Elimination
- Common Subexpression Elimination
Constant Folding

- Do constant propagation dataflow analysis
- Whenever a variable is constant, replace it with the constant value.

```plaintext
a = 1;
b = 2;
c = a + b;
c = 3;
```
Copy Propagation

\[
x = y; \\
a = x + 1; \\
b = x \times 5; \\
c = 7 - x;
\]

\[
a = y + 1; \\
b = y \times 5; \\
c = 7 - y;
\]
At each use of $x$ where $\ell : x = y$ reaches, replace $x$ with $y$. 
Copy Propagation

Reaching copies dataflow analysis

1. Forwards
2. Lattice is $(\mathcal{P}(Stmts), \supseteq)$

At each use of $x$ where $\ell : x = y$ reaches, replace $x$ with $y$. 
At each use of $x$ where $\ell : x = y$ reaches, replace $x$ with $y$. 
Copy Propagation

Reaching copies dataflow analysis

1. Forwards
2. Lattice is \((\mathcal{P}(Stmts), \supseteq)\)
3. \(\cap\)
4. \(\ell : x = y\) is a reaching copy if it is the last assignment to \(x\) along the path, and there are no assignments to \(y\) after \(\ell\).

At each use of \(x\) where \(\ell : x = y\) reaches, replace \(x\) with \(y\).
Copy Propagation

Reaching copies dataflow analysis

1. Forwards
2. Lattice is \((\mathcal{P}(Stmts), \supseteq)\)
3. \(\cap\)
4. \(\ell : x = y\) is a reaching copy if it is the last assignment to \(x\) along the path, and there are no assignments to \(y\) after \(\ell\).
   \[
   \ell : x = y \quad \text{out}_\ell = \{\ell\} \cup (\text{in}_\ell \setminus \{x = *, * = x\})
   \]
   \[
   \ell : x = \ldots \quad \text{out}_\ell = \text{in}_\ell \setminus \{x = *, * = x\}
   \]

At each use of \(x\) where \(\ell : x = y\) reaches, replace \(x\) with \(y\).
Copy Propagation

Reaching copies dataflow analysis

1. Forwards
2. Lattice is $(\mathcal{P}(Stmts), \supseteq)$
3. $\cap$
4. $\ell : x = y$ is a reaching copy if it is the last assignment to $x$ along the path, and there are no assignments to $y$ after $\ell$.
   \[
   \ell : x = y \quad \text{out}_\ell = \{\ell\} \cup (\text{in}_\ell \setminus \{x = *, * = x\})
   \]
   \[
   \ell : x = \ldots \quad \text{out}_\ell = \text{in}_\ell \setminus \{x = *, * = x\}
   \]
5. Start node value is $\{\}$

At each use of $x$ where $\ell : x = y$ reaches, replace $x$ with $y$. 
Forwards

Lattice is \((\mathcal{P}(\text{Stmts}), \supseteq)\)

\(\cap\)

\(\ell: x = y\) is a reaching copy if it is the last assignment to \(x\) along the path, and there are no assignments to \(y\) after \(\ell\).

\[
\ell: x = y \quad \text{out}_\ell = \{\ell\} \cup (\text{in}_\ell \setminus \{x = *, *, = x\}) \\
\ell: x = \ldots \quad \text{out}_\ell = \text{in}_\ell \setminus \{x = *, *, = x\}
\]

5. Start node value is \(\{\}\)

6. \(\bot = \{\text{all copies}\}\)

At each use of \(x\) where \(\ell: x = y\) reaches, replace \(x\) with \(y\).
Dead Code Elimination

\[ z = x + y; \]
\[ z \text{ never used} \]

\[ \text{don't compute } x + y \]
Dead Code Elimination

\[ z = x + y; \]
\[ z \text{ never used} \]

\[ \text{don't compute } x + y \]

NOTE: Watch out for side effects. \((z = x/y)\)

NOTE: Eliminating dead code may make other code dead.

NOTE: DCE is not Unreachable Code Elimination:

\[ \text{if}(\text{false}) \{ \ldots \} \]
Common Subexpression Elimination

a = x + y;
b = x + y;
c = x + y;
t = x + y;
a = t;
b = t;
c = t;

NOTE: Often useful to do copy propagation afterwards.
Problems with CSE

- Syntactic substitution only:
  
  ```
  a = x + y;
b = y + x;
  ```

- Partial redundancies:
  
  ```
  if() {
    a = x + y;
  }
b = x + y;
  ```