Redundancy Optimizations

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

Constant Folding

```
a = 1;
b = 2;
c = a + b;
```

```
c = 3;
```

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

Copy Propagation

```
x = y;
a = x + 1;
b = x * 5;
c = 7 - x;
```

```
a = y + 1;
b = y * 5;
c = 7 - y;
```

Copy Propagation

Reaching copies dataflow analysis

1. Forwards

At each use of x where ℓ:x = y reaches, replace x with y.

Copy Propagation

Reaching copies dataflow analysis

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

At each use of `x` where ℓ:`x = y` reaches, replace `x` with `y`.

Copy Propagation

Reaching copies dataflow analysis

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

At each use of `x` where ℓ:`x = y` reaches, replace `x` with `y`.

Copy Propagation

Reaching copies dataflow analysis

1. Forwards
2. Lattice is $(\mathcal{P}(\mathit{Stmts}), \supseteq)$
3. $\cap$
4. ℓ:`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 ℓ.

At each use of `x` where ℓ:`x = y` reaches, replace `x` with `y`.

Copy Propagation

Reaching copies dataflow analysis

1. Forwards
2. Lattice is $(\mathcal{P}(\mathit{Stmts}), \supseteq)$
3. $\bigcap$
4. ℓ:`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 ℓ.

   ℓ:`x = y` $\quad \mathsf{out}_\ell = \{\ell\} \cup (\mathsf{in}_\ell \setminus \{\texttt{x = *, * = x}\})$

   ℓ:`x = ...` $\quad \mathsf{out}_\ell = \mathsf{in}_\ell \setminus \{\texttt{x = *, * = x}\}$

At each use of `x` where ℓ:`x = y` reaches, replace `x` with `y`.

Copy Propagation

Reaching copies dataflow analysis

1. Forwards
2. Lattice is $(\mathcal{P}(\mathit{Stmts}), \supseteq)$
3. $\cap$
4. ℓ:`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 ℓ.

   ℓ:`x = y` $\quad \mathsf{out}_\ell = \{\ell\} \cup (\mathsf{in}_\ell \setminus \{\texttt{x = *, * = x}\})$

   ℓ:`x = ...` $\quad \mathsf{out}_\ell = \mathsf{in}_\ell \setminus \{\texttt{x = *, * = x}\}$
5. start node value is $\{\}$

At each use of `x` where ℓ:`x = y` reaches, replace `x` with `y`.

Copy Propagation

Reaching copies dataflow analysis

1. Forwards
2. Lattice is $(\mathcal{P}(\mathit{Stmts}), \supseteq)$
3. $\cap$
4. ℓ:`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 ℓ.

   ℓ:`x = y` $\quad \mathsf{out}_\ell = \{\ell\} \cup (\mathsf{in}_\ell \setminus \{\texttt{x = *, * = x}\})$

   ℓ:`x = ...` $\quad \mathsf{out}_\ell = \mathsf{in}_\ell \setminus \{\texttt{x = *, * = x}\}$
5. start node value is $\{\}$
6. $\bot = \{\text{all copies}\}$

At each use of `x` where ℓ:`x = y` reaches, replace `x` with `y`.

Dead Code Elimination

```
z = x + y;
z never used
```

```
don't compute x + y
```

Dead Code Elimination

```
z = x + y;
z never used
```

```
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:

```
if(false) { ...}
```

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;
```

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

- Partial redundancies:

```
if() {
    a = x + y;
}
b = x + y;
```

Local Value Numbering

Idea: Each expression gets a number.
Same number $\Longrightarrow$ same runtime value.

Maintain two tables:

- variable $\rightarrow$ number
- number op number $\rightarrow$ number

Example [Appel]

```
g = x + y
h = u - v
i = x + y
x = u - v
u = g + h
v = i + x
w = u + v
```