Optimization: common subexpression elimination

If there is a quadruple

```
d:  v = x op y
```

and a (later) quadruple

```
u:  t = x op y
```

can we delete u?

Common subexpression elimination

If there is a quadruple

```
d:  v = x op y
```

and a (later) quadruple

```
u:  t = x op y
```

d can be replaced by

```
d:  w = x op y
d′: v = w
```

and *u* can be replaced by

```
u:  t = w
```

Conditions:

1. There is such a definition *d* on every path to *u*

2. No definitions of x or y between (each) *d* and *u*

Example:

```
a = b - c
d = a + b
b = b - c
c = a + b
```

```
a = b - c
d = a + b
b = a
c = a + b
```

Available expressions

An expression

```
x op y
```

is ***available*** at *u* if `(x op y)` is computed on *every* path (of control flow) to *u* and there are no definitions of `x` or `y` after the latest computation on each path to *u*.

in(*s*) = set of expressions available at beginning of (statement) *s*

out(*s*) = set of expressions available at end of (statement) *s*

In a program, each statement

- **generates** some available expressions
- **kills** some available expressions

gen(*s*) = set of expressions made available by statement *s*

kill(*s*) = set of expressions made unavailable by statement *s*

For any assignment to a temporary `s: t = x op y`:

gen(*s*) = {`x op y`} – *kill*(*s*)
kill(*s*) = expressions containing `t`

For any assignment to memory `s: M[a] = b`:

gen(*s*) = {} *kill*(*s*) = expressions of form `M[...]`

For any other quadruple:

gen(*s*) = {} *kill*(*s*) = {}

Algorithm:

1. For each statement n:

   $out(n) = in(n) = full$; $in(1) = \{\}$

2. Repeat

   For each statement n:

   $in'(n) = in(n)$

   $out'(n) = out(n)$

   $in(n) = \bigcap_{p \in pred(n)} out(p)$

   $out(n) = gen(n) \cup (in(n) - kill(n))$

   until $in'(n) == in(n)$ && $out'(n) == out(n)$ for all n

Example:

s	*gen*(*s*)	*kill*(*s*)
1: sum = 0	{}	exp(sum)
2: i = 0	{}	exp(i)
3: t = i * 4	{i*4}	exp(t)
4: if (i > 10) goto 11	{}	{}
5: t = i * 4	{i*4}	exp(t)
6: v = M[t]	{M[t]}	exp(v)
7: sum = sum + v	{}	exp(sum)
8: i = i + 1	{}	exp(i)
9: t = i * 4	{i*4}	exp(t)
10: goto 4	{}	{}
11: write(sum)	{}	{}

1st iteration:

```
in(1) = {}
out(1) = {}
in(2) = {}
out(2) = {}
in(3) = {}
out(3) = {i*4}
in(4) = {i*4} ∩ full = {i*4}
out(4) = {i*4}
in(5) = {i*4}
out(5) = {i*4}
in(6) = {i*4}
out(6) = {i*4, M[t]}
in(7) = {i*4, M[t]}
out(7) = {i*4, M[t]} = {i*4, M[t]}
in(8) = {i*4, M[t]}
out(8) = {i*4, M[t]} – exp(i) = {M[t]}
in(9) = {M[t]}
out(9) = {M[t], i*4} – exp(t) = {i*4}
in(10) = {i*4}
out(10) = {i*4}
in(11) = {i*4}
```

2nd iteration:

```
in(1) = {}
out(1) = {}
in(2) = {}
out(2) = {}
in(3) = {}
out(3) = {i*4}
in(4) = {i*4} ∩ {i*4} = {i*4}
out(4) = {i*4}
in(5) = {i*4}
out(5) = {i*4}
in(6) = {i*4}
out(6) = {i*4, M[t]}
in(7) = {i*4, M[t]}
out(7) = {i*4, M[t]} = {i*4, M[t]}
in(8) = {i*4, M[t]}
out(8) = {i*4, M[t]} – exp(i) = {M[t]}
in(9) = {M[t]}
out(9) = {M[t], i*4} – exp(t) = {i*4}
in(10) = {i*4}
out(10) = {i*4}
in(11) = {i*4}
```

Common subexpression elimination (contd.)

The expression `i*4` is available at *5*.

So `i*4` in *5* can be eliminated.

To eliminate `(x op y)` in

```
u:  t = x op y
```

given that `(x op y)` is available:

- replace *u* by

```
u:  t = w
```

- find occurrences of expression `(x op y)` that reach *u*, e.g.:

```
d:  v = x op y
```

- replace *d* by

```
d:  w = x op y
d′: v = w
```

Reaching expressions

An expression

```
x op y
```

in

d: `v = x op y`

reaches *u* if there is a path from *d* to *u* that does not include any computation of `(x op y)` or any definition of `x` or `y`.