Tour of common optimizations

Simple example

```plaintext
foo(z) {
    x := 3 + 6;
    y := x - 5
    return z * y
}
```

Another example

```plaintext
x := a + b;
...
\textcolor{red}{y := a + b;} \\
\textcolor{red}{\text{only if } x, a, b \text{ not defined!}}
```

Another example

```plaintext
if (...) {
    a := read();
    x := a + b;
    print(x);
} \& \{ y := a + b \}
    \quad \text{print(y)}
    \quad y := a + b;
```
Another example

if {...} {
    a := read();
    x := x + b;
    print(x);
}

Another example

Another example

Another example

Another example

Another example

Another example

Another example

Another example

Another example

Another example
Another example

\[ x := y^{z} \]

\[ \ldots \]

\[ x := \ldots \]

• Often used as a clean-up pass

\[ x := y \quad \text{Copy prop} \quad x := y \quad \text{DAE} \quad x := y \]

\[ z := z + x \rightarrow z := z + y \rightarrow z := z + y \]

Another example

\[
\text{if (false) }
\]

\[
\begin{array}{c}
\text{Another common clean up opt}
\end{array}
\]

Another example

• In Java:

```java
a = new int[10];
for (index = 0; index < 10; index++) {
    a[index] = 100;
}
```

Another example

• In "lowered" Java:

```java
a = new int[10];
for (index = 0; index < 10; index++) {
    if (index < 0 || index >= a.length()) {
        throw OutOfBoundsException;
    }
    a[index] = 0;
}
```
Another example

```plaintext
p := &x;
*p := 5
y := x + 1;
```

Another example

```plaintext
x := 5;
*p := 3
y := x + 1;
```

Another example

```plaintext
for j := 1 to N
  for i := 1 to M
    a[i] := a[i] + b[j]
```

Another example

```plaintext
area(h,w) { return h * w }
```

Another example

```plaintext
h := ...
w := 4;
a := area(h,w)
```

Another example

```plaintext
area(h,w) { return h * w }
```

```plaintext
h := ...
w := 4;
a := area(h,w)
```

Many "bill" dug beach

\[ h < 2 \]
Optimization themes

- Don’t compute if you don’t have to
  - unused assignment elimination
- Compute at compile-time if possible
  - constant folding, loop unrolling, inlining
- Compute it as few times as possible
  - CSE, PRE, PDE, loop invariant code motion
- Compute it as cheaply as possible
  - strength reduction
- Enable other optimizations
  - constant and copy prop, pointer analysis
- Compute it with as little code space as possible
  - unreachable code elimination