A more realistic language: IMP

$$
\begin{array}{l}
op \in \mathsf{Op} ::= + \mid * \mid <= \\
e \in \mathsf{Exp} ::= x \mid i \mid \mathsf{true} \mid \mathsf{false} \mid e_1\ op\ e_2 \\
c \in \mathsf{Com} ::= x := e \mid \mathsf{skip} \mid c_1; c_2 \mid \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ e\ \mathsf{do}\ c \\
s \in \mathsf{Store} : \mathsf{Var} \to \mathsf{Int}
\end{array}
$$

Two kinds of configuration: (e, s) and (c, s)
Two transition relations: $(e, s) \to (e', s)$ and $(c, s) \to (c', s')$

Expressions:

$$
\begin{array}{rl}
\mathsf{var} & \dfrac{}{(x, s) \to (s(x), s)} \\
\mathsf{plus} & \dfrac{}{(i_1 + i_2, s) \to (i, s)}\ (i = i_1 + i_2) \\
\mathsf{leq} & \dfrac{}{(i_1 <= i_2, s) \to (\mathsf{true}, s)}\ (i_1 \le i_2) \\
\mathsf{gt} & \dfrac{}{(i_1 <= i_2, s) \to (\mathsf{false}, s)}\ (i_1 > i_2) \\
\mathsf{times} & \dfrac{}{(i_1 * i_2, s) \to (i, s)}\ (i = i_1 * i_2) \\
\mathsf{left} & \dfrac{(e_1, s) \to (e_1', s')}{(e_1\ op\ e_2, s) \to (e_1'\ op\ e_2, s')} \\
\mathsf{right} & \dfrac{(e_2, s) \to (e_2', s')}{(i + e_2, s) \to (i\ op\ e_2, s')}
\end{array}
$$

Commands:

$$
\begin{array}{rl}
\mathsf{set} & \dfrac{}{(x := i, s) \to (\mathsf{skip}, s[x \mapsto i])} \\
\mathsf{assign} & \dfrac{(e, s) \to (e', s')}{(x := e, s) \to (x := e', s')} \\
\mathsf{seq\text{-}skip} & \dfrac{}{(\mathsf{skip}; c, s) \to (c, s)} \\
\mathsf{seq} & \dfrac{(c_1, s) \to (c_1', s')}{(c_1; c_2, s) \to (c_1'; c_2, s)} \\
\mathsf{if\text{-}true} & \dfrac{}{(\mathsf{if\ true\ then}\ c_1\ \mathsf{else}\ c_2, s) \to (c_1, s)} \\
\mathsf{if\text{-}false} & \dfrac{}{(\mathsf{if\ false\ then}\ c_1\ \mathsf{else}\ c_2, s) \to (c_2, s)} \\
\mathsf{if} & \dfrac{(e, s) \to (e', s')}{(\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, s) \to (\mathsf{if}\ e'\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, s')} \\
\mathsf{while} & \dfrac{}{(\mathsf{while}\ e\ \mathsf{do}\ c, s) \to (\mathsf{if}\ e\ \mathsf{then}\ (c; \mathsf{while}\ e\ \mathsf{do}\ c)\ \mathsf{else\ skip}, s)}
\end{array}
$$

Evaluation of a simple program in some arbitrary store s:

$(\mathsf{x} := 42; \mathsf{y} := 30; \mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}, s)$
$\rightarrow$ $(\mathsf{y} := 30; \mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}, s[\mathsf{x} \mapsto 42])$
$\rightarrow$ $(\mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 30])$
$\rightarrow$ $(\mathsf{if}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{then}\ \{\ \mathsf{y} := \mathsf{y}+13; \mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}\ \}\ \mathsf{else}\ \mathsf{skip}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 30])$
$\rightarrow$ $(\mathsf{if}\ (30 <= \mathsf{x})\ \mathsf{then}\ \{\ \mathsf{y} := \mathsf{y}+13; \mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}\ \}\ \mathsf{else}\ \mathsf{skip}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 30])$
$\rightarrow$ $(\mathsf{if}\ (30 <= 42)\ \mathsf{then}\ \{\ \mathsf{y} := \mathsf{y}+13; \mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}\ \}\ \mathsf{else}\ \mathsf{skip}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 30])$
$\rightarrow$ $(\mathsf{if}\ (\mathsf{true})\ \mathsf{then}\ \{\ \mathsf{y} := \mathsf{y}+13; \mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}\ \}\ \mathsf{else}\ \mathsf{skip}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 30])$
$\rightarrow$ $(\mathsf{y} := \mathsf{y}+13; \mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 30])$
$\rightarrow$ $(\mathsf{y} := 30+13; \mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 30])$
$\rightarrow$ $(\mathsf{y} := 43; \mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 30])$
$\rightarrow$ $(\mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 43])$
$\rightarrow$ $(\mathsf{if}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{then}\ \{\ \mathsf{y} := \mathsf{y}+13; \mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}\ \}\ \mathsf{else}\ \mathsf{skip}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 43])$
$\rightarrow$ $(\mathsf{if}\ (43 <= \mathsf{x})\ \mathsf{then}\ \{\ \mathsf{y} := \mathsf{y}+13; \mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}\ \}\ \mathsf{else}\ \mathsf{skip}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 43])$
$\rightarrow$ $(\mathsf{if}\ (43 <= 42)\ \mathsf{then}\ \{\ \mathsf{y} := \mathsf{y}+13; \mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}\ \}\ \mathsf{else}\ \mathsf{skip}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 43])$
$\rightarrow$ $(\mathsf{if}\ (\mathsf{false})\ \mathsf{then}\ \{\ \mathsf{y} := \mathsf{y}+13; \mathsf{while}\ (\mathsf{y} <= \mathsf{x})\ \mathsf{do}\ \{\ \mathsf{y} := \mathsf{y}+13\ \}\ \}\ \mathsf{else}\ \mathsf{skip}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 43])$
$\rightarrow$ $(\mathsf{skip}, s[\mathsf{x} \mapsto 42, \mathsf{y} \mapsto 43])$

One thing that compilers have to do is transform code. Sometimes, these transformations are needed to eliminate high-level language features, and sometimes just do to optimizations.

How can we prove that an optimization is valid? In general, we want the compiler to only replace commands with other commands that are "equivalent" (and faster.) But what do we mean by "equivalent"? They shouldn't be exactly the same, since we want to eliminate unnecessary computational steps.

What we usually do is define some notion of external observations and optimize relative to those optimizations. For instance, we might say that commands c_1 and c_2 are equivalent if, given any input state s, $(c_1, s) \rightarrow^* (\mathsf{skip}, s')$ iff $(c_2, s) \rightarrow^* (\mathsf{skip}, s')$. That is, the two commands take equal states to equal states.

What are some optimizations we might consider? A simple one is to get rid of unnecessary skip statements. For instance: $\mathsf{skip}; c = c$, and this is really easy to prove. But it also doesn't show up that much in practice (unless introduced by some other simplifying optimization.)

Another optimization we might want to support is to replace

$$\mathsf{if}\ e\ \mathsf{then}\ c\ \mathsf{else}\ c$$

with just c. Again, it's very easy to prove that these are equivalent commands, but this situation occurs very rarely.

Here's a more interesting situation: suppose c is a command that does not assign to the variable x, but only reads from it. Then we can rewrite:

$$x := i; c$$

to

$$x := i; c[i/x]$$

(i.e., the command c with i substituted for x.) This is called constant propagation. Similarly, we can rewrite

$$x := y; c \qquad \text{to} \qquad x := y; c[y/x]$$

(as long as neither x nor y are assigned within c.) This is called copy propagation. Finally, we can eliminate an assignment to a variable if the variable is never read. In particular, if we have:

$$x := e; c$$

and c never reads the value out of x, then it should be safe to replace this with just c.

Another example is called loop-invariant removal. Consider a loop:

$$\mathsf{while}\ e_1\ \mathsf{do}\ \{c_1; x := e_2; c_2\}$$

If e_1, c_1, and c_2 never read or write x, then we can change this to:

$$x := e_2; \mathsf{while}\ e_1\ \mathsf{do}\ \{c_1; c_2\}$$

under what circumstances? Can you prove this is valid?

Note: here is the definition of substituting an expression e for a variable x within a command c (written $c[e/x]$) or within another expression (written $e'[e/x]$):

$\mathsf{skip}[e/x] = \mathsf{skip}$
$(x := e')[e/x] = x := (e'[e/x])$
$(c_1; c_2)[e/x] = (c_1[e/x]); (c_2[e/x])$
$(\mathsf{if}\ e'\ \mathsf{then}\ c1\ \mathsf{else}\ c2)[e/x] = \mathsf{if}\ e'[e/x]\ \mathsf{then}\ c_1[e/x]\ \mathsf{else}\ c_2[e/x]$
$(\mathsf{while}\ e'\ \mathsf{do}\ c)[e/x] = \mathsf{while}\ e'[e/x]\ \mathsf{do}\ c[e/x]$

$x[e/x] = e$
$y[e/x] = y\ (y! = x)$
$i[e/x] = i$
$\mathsf{true}[e/x] = \mathsf{true}$
$\mathsf{false}[e/x] = \mathsf{false}$
$(e_1\ op\ e_2)[e/x] = (e_1[e/x])\ op\ (e_2[e/x])$