“Compiling” a program

- Instead of expressing the denotation as a result (from \(\text{Nat}\)) we can express it as a function of \(n \in \text{Nat}\).

\[
P[Z := 1; \text{if } A = 0 \text{ then diverge; } Z := 3] \overset{\lambda n.\text{let } s = \text{update } [A] n \text{ newstore in}}{=} \text{let } s' = E \text{ in}
\]

Where \(E = C[Z := 3; C[\text{if } A = 0 \text{ then diverge}]](C[Z := 1])\).

- i.e.

\[
(\lambda s.\text{update } [Z] \text{ three } s)(\lambda s.\text{access } [A] s) \text{ equals zero} \rightarrow (\lambda s.\bot) s
\]

Simplify to:

\[
\lambda n.\text{let } s' = \text{let } s_3 = (\text{access } [Z] s') \text{ in}
\]

\[
\text{update } [Z] \text{ three } s_3 \text{ in }
\]

A more readable version

- Convert strict lambdas to lets (same property of strictness)

\[
P[Z := 1; \text{if } A = 0 \text{ then diverge; } Z := 3] \overset{\lambda n.\text{let } s = \text{update } [A] n \text{ newstore in}}{=} \text{let } s' = \text{let } s_3 = \text{let } s_2 = \text{let } s_1 = s \text{ in}
\]

\[
\text{update } [Z] \text{ three } s_3 \text{ in }
\]

\[
\text{access } [Z] s
\]

We can use (by extensionality of functions):

\[
\text{let } s = (e_1 \rightarrow \bot) s_2 \text{ in } s_3 \text{ is the same as } e_2 \rightarrow \bot s_2
\]

to simplify further
Final simplification

- \( \lambda n. \text{let } s' = (n \text{ equals zero } \rightarrow \bot) \) in access \([Z] s'\)
- We can apply the same simplification again:
- \( \lambda n. n \text{ equals zero } \rightarrow \bot \) in access \([Z] (\text{update } [Z] \text{ three } s_2)\)
- Finally:
- \( \lambda n. n \text{ equals zero } \rightarrow \bot \) in update \([Z] \text{ three } s_2\)
- Here all identifiers have been optimized away; stores have been used where they are proper (not bottom); the final result is very intuitive as to the core meaning of the program

Proving program properties

- If two denotations are equal, then the two programs with those denotations are equivalent
- Since denotations can be functions, we need to show that two functions are the same
- Use extensionality principle:
  - If \( f \ x = g \ x \) for all \( x \), then \( f \) and \( g \) are the same function

Program equivalence example

- Prove:
- \( X := 0; Y := X + 1 \) and \( Y := 1; X := 0 \) are equivalent
- We can do this if:
- \( [P][ X := 0; Y := X + 1] \) and \( [P][ Y := 1; X := 0] \) are the same function

These two are different functions, but are extensionally equivalent because they produce the same result for \( X \) (zero) and for \( Y \) (one) and for any other argument \( I \) (all)