Basic idea: like call-by-value but pass arguments unevaluated. The big-step operational semantics is the same for arithmetic, booleans, and if-then-else. For applications the big-step rule becomes:

\[ e_1 \Rightarrow \lambda x.e'_1 \quad [e_2/x]e'_1 \Rightarrow z \]
\[ (e_1 e_2) \Rightarrow z \]

Notice that this rule says that we proceed without evaluating \( e_2 \). Also, we leave tuples unevaluated:

\[ v ::= \ldots | (e_1, e_2) \]

So \( (2+2, 3+3) \) is a value. The big-step rules for projections are then:

\[ e \Rightarrow (e_1, e_2) \quad e_1 \Rightarrow z \]
\[ e.1 \Rightarrow z \]
\[ e \Rightarrow (e_1, e_2) \quad e_2 \Rightarrow z \]
\[ e.2 \Rightarrow z \]

Infinite data structures are now implementable. For example, define \( \text{zeros} \) to be the following term:

\[ \text{rec zeros} := (0, \text{zeros}) \]

Now we have the following reductions:

\[ \text{zeros}.1 \Rightarrow 0 \]
\[ \text{zeros}.2 \Rightarrow (0, \text{zeros}) \]
\[ \text{zeros}.2.1 \Rightarrow 0 \]