Stepping Scheme Via Logic

Jonathan Buss
July 4, 2017

We shall give a detailed example of the logical analysis of the execution of Scheme code. For
the code, we shall use a previous example: the computation of binary increment. The actual code
is the following. (It differs slightly from the earlier code, as explained below.)

(define incr
  (lambda (a)
    (cond ((equal? a '()) (list 'one))
          ((equal? (first a) 'zero) (cons 'one (rest a)))
          (#true (cons 'zero (incr (rest a)))))
  )
)

The corresponding term of FOL is

\[ \text{define, incr,} \]
\[ \lambda, (\text{cond, } (\text{equals?, a, e}, \text{one}),
    (\text{equals?, } (\text{first, a}), \text{zero}, \text{cons, one, (rest, a)}),
    \text{#true, cons, zero, (incr, (rest, a))}) \]

To use this definition, one calls the function; our example will evaluate the code

\[ \text{incr (list 'one 'zero 'one)} \]

The corresponding FOL term is \( \text{(incr, (one, zero, one)} \).

1 Preliminaries: Notation and Overview

DrRacket and its “stepper”

Our logical analysis will closely resemble the operation of the “stepper” in DrRacket. In order
to make this resemblance even closer, we have re-written the code for \text{incr} to use notation more
similar to what DrRacket produces. The differences are as follows.

- DrRacket distinguishes two kinds of names: functions and constant symbols. A name that
  starts with a quote mark is a symbol; one that does not is a function (or a keyword).
  In FOL, we have only one kind of name—lists of the form \text{cons(name, x)}. Names that appear
  in the dictionary behave as functions, while names not in the dictionary behave more like
  constants. (We could have defined several kinds of names. But we didn’t.)

- DrRacket writes ‘() for the empty list. For the list that we would notate as \( \langle a_1, a_2, \ldots, a_k \rangle \),
it uses “(list \( a_1 a_2 \ldots a_k \))”. Note, however, that you will see this only while using the
  stepper—outside of the stepper, DrRacket will immediately evaluate the \( a_i \), in order to call
  the function “list”, which creates an internal data structure.
(Intuitively, if we ignored the difference between constant symbols and function symbols in DrRacket, then “\((a_1 \ a_2 \ \ldots \ a_k)\)” would behave much like \(\langle a_1, a_2, \ldots, a_k \rangle\). DrRacket, however, won’t ignore the difference—it treats the two quite differently.)

- Our version of \textit{define} can only define a single symbol. To define a function, we must use an explicit \texttt{lambda}.

These points are reflected in the code above.

**Some notation**

In order to avoid having to write down some very large terms, we define the following abbreviations.

\[
\begin{align*}
\tau_e^\text{def} &= \langle \text{equals?}, a, e \rangle \\
\tau_{1st}^\text{def} &= \langle \text{equals?}, \langle \text{first}, a \rangle, \text{zero} \rangle \\
\tau_{\text{incr}}^\text{def} &= \text{cons}(\text{zero}, \langle \text{incr}, \langle \text{rest}, a \rangle \rangle) \\
\tau_{\text{body}}^\text{def} &= \langle \text{cond}, \gamma_e^\text{def}, \gamma_{1st}^\text{def}, \gamma_{\text{incr}}^\text{def} \rangle
\end{align*}
\]

Using these names, the definition term is \(\langle \text{define}, \text{incr}, \langle \lambda, \langle a \rangle, \tau_{\text{body}}^\text{def} \rangle \rangle\). The actual dictionary is a list containing only one pair—the pair that defines \texttt{incr}:

\[D = \langle \langle \text{incr}, \langle \lambda, \langle a \rangle, \tau_{\text{body}}^\text{def} \rangle \rangle \rangle \]

During the analysis, we shall need to consider versions to which substitutions have been applied. Thus we define

\[
\begin{align*}
\tau_e^\text{call} &= \tau_e^\text{def}[\langle \text{one}, \text{zero}, \text{one} \rangle / a] \\
\tau_{1st}^\text{call} &= \tau_{1st}^\text{def}[\langle \text{one}, \text{zero}, \text{one} \rangle / a] \\
\tau_{\text{incr}}^\text{call} &= \tau_{\text{incr}}^\text{def}[\langle \text{one}, \text{zero}, \text{one} \rangle / a] \\
\tau_{\text{body}}^\text{call} &= \tau_{\text{body}}^\text{def}[\langle \text{one}, \text{zero}, \text{one} \rangle / a]
\end{align*}
\]

and

\[
\begin{align*}
\tau_e^\text{recur} &= \tau_e^\text{def}[\langle \text{zero}, \text{one} \rangle / a] \\
\tau_{1st}^\text{recur} &= \tau_{1st}^\text{def}[\langle \text{zero}, \text{one} \rangle / a] \\
\tau_{\text{incr}}^\text{recur} &= \tau_{\text{incr}}^\text{def}[\langle \text{zero}, \text{one} \rangle / a] \\
\tau_{\text{body}}^\text{recur} &= \tau_{\text{body}}^\text{def}[\langle \text{zero}, \text{one} \rangle / a]
\end{align*}
\]
Overall approach

"Begin at the beginning. Go on until the end. Then stop." –Lewis Carroll, Through the Looking Glass

Execution of the code proceeds step by step:

- Replace the name \textit{incr} with its definition.
- Apply the function (substitute for the formal argument).
- Evaluate the \textit{cond}.
  - Evaluate the first test.
    * Apply the built-in function \texttt{[equals?]}.
  - Skip the false condition.
  - Evaluate the second test.
    * Apply the built-in function \texttt{[first]}.
    * ...
  - ...

- ...

For our analysis, however, we will start from the simplest steps, and proceed “outwards” to larger and larger pieces. The pieces then assemble into the overall whole.

Our goal is to find a term \( \omega \) such that \( \text{Eval}(\langle \text{incr}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle, D, \omega) \), with \( D \) as above.

In order to achieve this goal, we shall actually determine a sequence of terms, \( \alpha_0, \ldots, \alpha_k \), where \( \alpha_0 = \langle \text{incr}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle \), \( \alpha_k = \omega \), and \( \text{Step}(\alpha_i, D, \alpha_{i+1}) \) for each \( 1 \leq i < k \). Since the axioms for \text{Step} (see Figure 1) include recursion, some of the steps from \( \alpha_i \) to \( \alpha_{i+1} \) actually require several sub-steps, which use \text{Step} for terms which are sub-terms of \( \alpha_i \). We shall begin our analysis with some of these sub-steps, and then put them together to form the entire derivation.

In order to do this, we must refer to intermediate stages, or “partial evaluations”. For this, we characterize a relation \text{PartEval} by the following two formulas:

\[
\text{PartEval}(x, D, x) \quad (1a)
\]
\[
x \neq z \rightarrow \left( \text{PartEval}(x, D, z) \leftrightarrow (\exists y \ (\text{Step}(x, D, y) \wedge \text{PartEval}(y, D, z))) \right) \quad (1b)
\]

Thus \text{PartEval} is like \text{Eval}, except that the result need not be fully evaluated.

1. \textbf{Lemma.} For every \( x, y, z \) and \( E \),
   \begin{itemize}
   \item (a) \( \vdash \text{Eval}(x, E, y) \leftrightarrow (\text{PartEval}(x, E, y) \wedge \text{IsValue}(y)) \)
   \item (b) \( \{ \text{PartEval}(x, E, y), \text{PartEval}(y, E, z) \} \vdash \text{PartEval}(x, E, z) \).
   \end{itemize}

2. \textbf{Exercise.} (a) Prove part (a) of Lemma 1.
   (b) (Advanced!) Part (b) of the theorem cannot be proven easily from the List axioms, using what we have covered. However, it can be proven as a “meta-theorem” using ordinary mathematical induction—an induction of the number of applications of \text{Step} required to demonstrate \text{PartEval}(x, E, y). Explain the difference between the preceding two statements, and explain how to do to the proof of the second.
Step1: $\rho_{\text{fun}}(\vec{x}, z) \rightarrow \text{Step}(\text{cons}(\text{fun}, \vec{x}), D, z)$, for each built-in function \text{fun}.

Step2: \text{LookUp}(x, \text{cons}((x, y), z), y).

Step3: $x \neq u \rightarrow \left( \text{LookUp}(x, D, y) \rightarrow \text{LookUp}(x, \text{cons}((u, v), D), y) \right)$.

Step4: \text{IsName}(x) \rightarrow \left( \text{LookUp}(x, D, y) \rightarrow \text{Step}(x, D, y) \right)$

Step5: \text{IsValue}(c)$, for each defined constant $c$, and $(\forall y \text{IsValue}(\text{cons}(\text{list}, y)))$.

Step6: \text{IsValue}(\langle x, y \rangle)$.

Step7: $(\text{IsValue}(x) \land \text{IsValue}(y) \land \neg R_{\text{first}}(x, \lambda)) \rightarrow \text{IsValue}(\text{cons}(x, y))$.

Step8: $\text{Step}(x, D, y) \rightarrow \text{Step}(\text{cons}(x, z), D, \text{cons}(y, z))$.

Step9: $\text{Step}(\text{cons}(\text{cond}, \langle \#\text{true}, x \rangle), y, D, x)$.

Step10: $\text{Step}(\text{cons}(\text{cond}, \langle \#\text{false}, x \rangle), y, D, \text{cons}(\text{cond}, y))$.

Step11: $\text{Step}(z, D, w) \rightarrow \text{Step}(\text{cons}(\text{cond}, \langle z, x \rangle), y, D, \text{cons}(\text{cond}, \langle w, x \rangle), y))$.

Step12: $\text{IsValue}(u) \rightarrow \left( \text{Subst}(\langle x, u \rangle, y, v) \rightarrow \text{Step}(\langle \langle \lambda, \langle x \rangle, y \rangle, u \rangle, D, v) \right)$.

Step13: $\text{Step}(u, D, v) \rightarrow \text{Step}(\langle \langle \lambda, x, y \rangle, u \rangle, D, \langle \langle \lambda, x, y \rangle, v \rangle)$.

Step14: $\text{Subst}(\langle x, u \rangle, e, e)$, $\text{Subst}(\langle x, u \rangle, x, u)$, and $\left( (\exists y\ (y = \text{cons}(\text{name}, y))) \land (y \neq x) \right) \rightarrow \text{Subst}(\langle x, u \rangle, y, y)$.

Step15: $(\text{Subst}(\langle x, u \rangle, y, v) \land (y \neq \text{name})) \rightarrow \text{Subst}(\langle x, u \rangle, z, w) \rightarrow \text{Subst}(\langle x, u \rangle, \text{cons}(y, z), \text{cons}(v, w))$.

Figure 1: The Stepping Axioms

2 Building up terms

Some of the axioms refer to the predicate \text{IsValue}. We start there, and proceed outwards.

Values, and \text{IsValue}

To determine whether something is a value, we use axioms Step5–Step7. By Step5, we get \text{IsValue}(\langle \text{one} \rangle), \text{IsValue}(\text{zero}), etc. Step7 then implies \text{IsValue}(\langle \text{one}, \text{zero}, \text{one} \rangle) and likewise for other lists of constants.

Note, however, that “\text{IsValue}(\langle a \rangle)” is false; a name, such as $\langle a \rangle$, is not a value.
Steps for \textit{first} and \textit{rest}

The built-in functions for \textit{first} and \textit{rest} have formulas \( \rho_{\text{first}} = \mathcal{R}_{\text{first}} \) and \( \rho_{\text{rest}} = \mathcal{R}_{\text{rest}} \). Since we easily prove formulas such as \( \mathcal{R}_{\text{first}}(⟨\text{one, zero, one}⟩, \text{one}) \), we get

\begin{align*}
\text{Step}(⟨\text{first}, ⟨\text{one, zero, one}⟩⟩, D, \text{one}) , \quad (2a) \\
\text{Step}(⟨\text{rest}, ⟨\text{one, zero, one}⟩⟩, D, ⟨\text{zero, one}⟩) , \quad (2b) \\
\text{Step}(⟨\text{first}, ⟨\text{zero, one}⟩⟩, D, \text{zero}) , \quad (2c) \\
\text{Step}(⟨\text{rest}, ⟨\text{zero, one}⟩⟩, D, ⟨\text{one}⟩) , \quad (2d)
\end{align*}

and similarly for other lists.

Equality steps

Consider a term \( ⟨\text{equals?}, ⟨\text{one, zero, one}⟩, e⟩ \). What rule for the relation \text{Step} will allow an evaluation step? Looking through the list, we find only one—\text{Step1}.

Using \text{Step1} with \( ⟨⟨\text{one, zero, one}⟩, e⟩ \) as \( \vec{x} \) and \( \#\text{false} \) as \( z \) yields

\( \rho_{\text{equals?}}(⟨\text{one, zero, one}⟩, e, \#\text{false}) \rightarrow \text{Step}(⟨\text{equals?}, ⟨\text{one, zero, one}⟩, e⟩, D, \#\text{false}) \).

Since we easily prove \( \rho_{\text{equals?}}(⟨\text{one, zero, one}⟩, e, \#\text{false}) \), we get

\( \text{Step}(⟨\text{equals?}, ⟨\text{one, zero, one}⟩, e⟩, D, \#\text{false}) \). (3a)

Other comparisons behave similarly; we can prove, for example,

\begin{align*}
\text{Step}(⟨\text{equals?}, ⟨\text{zero, one}⟩, e⟩, D, \#\text{false}) , \quad (3b) \\
\text{Step}(⟨\text{equals?}, ⟨\text{zero, zero}⟩, D, \#\text{true}) , \quad (3c)
\end{align*}

Note, however, that neither relation \( \rho_{\text{equals?}}(⟨\text{equals?}, ⟨\text{first, one, zero, one}⟩⟩, \#\text{false}) \) nor \( \rho_{\text{equals?}}(⟨\text{equals?}, ⟨\text{first, one, zero, one}⟩⟩, \#\text{true}) \) holds. Evaluation of \( \tau_{\text{call}} \) requires reducing it to a value first. That comes from formula (2a) above; combining it with axiom \text{Step8} produces what we need:

\begin{align*}
\text{Step}(⟨\text{equals?}, ⟨\text{first, one, zero, one}⟩⟩, \text{zero}) , D, ⟨\text{equals?}, \text{one, zero}⟩) \quad (4a) \\
\text{Step}(⟨\text{equals?}, ⟨\text{first, zero, one}⟩⟩, \text{zero}) , D, ⟨\text{equals?}, \text{zero, zero}⟩) \quad (4b)
\end{align*}

and thus, by formula (3a),

\begin{align*}
\text{PartEval}(⟨\text{equals?}, ⟨\text{first, one, zero, one}⟩⟩, \text{zero}), D, \#\text{false} \quad (5a) \\
\text{PartEval}(⟨\text{equals?}, ⟨\text{first, one, zero, one}⟩⟩, \text{zero}), D, \#\text{true} \quad (5b)
\end{align*}

\textsuperscript{1}An actual proof of course requires an actual formula. We can take \( \rho_{\text{equals?}}(x, y, z) \) to be the formula \( ((\text{IsValue}(x) \land \text{IsValue}(y)) \land ((x = y) \land (z = \#\text{true})) \lor ((x \neq y) \land (z = \#\text{false}))) \), or any equivalent formula.
Stepping with cond

To evaluate a term of the form \( \langle \text{cond}, \langle x, y \rangle, \ldots \rangle \), we look at \( x \); if it is either \#true or \#false, we use Step9 or Step10, as appropriate. These give, for example,

\[
\text{Step}\left( \langle \text{cond}, \langle \#false \rangle, \langle \text{rest}, \langle \text{zero}, \text{one} \rangle \rangle \rangle, \gamma_{\text{recur}}^{\text{incr}}, D, \langle \text{cond}, \langle \text{rest}, \langle \text{zero}, \text{one} \rangle \rangle \rangle \right).
\]

(6a)

Step\left( \langle \text{cond}, \langle \#true, \text{cons}(\text{one}, \langle \text{rest}, \langle \text{zero}, \text{one} \rangle \rangle) \rangle, \gamma_{\text{recur}}^{\text{incr}}, D, \langle \text{cons}(\text{one}, \langle \text{rest}, \langle \text{zero}, \text{one} \rangle \rangle) \rangle \right).

(6b)

In other cases, we will use Step11, which requires an evaluation of \( x \). For example, combining Step11 with formula (3a) above yields

\[
\text{Step}\left( \langle \text{cond}, \langle \tau_{\text{lst}}^{\text{recur}}, \text{cons}(\text{one}, \langle \text{rest}, \langle \text{zero}, \text{one} \rangle \rangle) \rangle, \gamma_{\text{recur}}^{\text{incr}}, D, \langle \text{cons}(\text{one}, \langle \text{rest}, \langle \text{zero}, \text{one} \rangle \rangle) \rangle \right).
\]

(7)

and hence

\[
\text{PartEval}(\langle \text{cond}, \langle \tau_{\text{lst}}^{\text{recur}}, \ldots \rangle, D, \langle \text{cons}(\text{one}, \langle \text{rest}, \langle \text{zero}, \text{one} \rangle \rangle) \rangle).
\]

(8)

The final term above, \( \text{cons}(\text{one}, \langle \text{rest}, \langle \text{zero}, \text{one} \rangle \rangle) \), is easily evaluated, yielding

\[
\text{PartEval}(\langle \text{cond}, \langle \tau_{\text{lst}}^{\text{recur}}, \ldots \rangle, D, \langle \text{one}, \text{one} \rangle).
\]

(9)

3 Superstructure: function invocations and sequencing steps

We now return to the start of the computation. At the same time, however, we shall also follow the eventual recursive call, since the two are quite similar.

To take a step from the initial configuration \( \langle \text{incr}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle \), the only applicable axiom for Step is Step8: evaluation of the first element on the list. Thus we want \( t_1 \) s.t. \( \text{Step}(\text{incr}, D, t_1) \); this itself comes from axiom Step4—dictionary lookup. It yields \( \text{Step}(\text{incr}, D, \langle \lambda, \langle a \rangle, \tau_{\text{def}}^{\text{body}} \rangle) \). This implies (together with Step8) the step we want:

\[
\text{Step}(\langle \text{incr}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle, D, \langle \lambda, \langle a \rangle, \tau_{\text{def}}^{\text{body}} \rangle, \langle \text{one}, \text{zero}, \text{one} \rangle).
\]

(10a)

Similarly, we also have

\[
\text{Step}(\langle \text{incr}, \langle \text{zero}, \text{one} \rangle \rangle, D, \langle \lambda, \langle a \rangle, \tau_{\text{def}}^{\text{body}} \rangle, \langle \text{zero}, \text{one} \rangle).
\]

(10b)

Next, the function call, via Step12, makes a replacement of \( \tau_{\text{body}}^{\text{def}} \) by \( \tau_{\text{body}}^{\text{call}} \), i.e.,

\[
\text{Step}(\langle \lambda, \langle a \rangle, \tau_{\text{body}}^{\text{def}} \rangle, \langle \text{one}, \text{zero}, \text{one} \rangle, D, \langle \text{cond}, \gamma_{\text{e}}^{\text{call}}, \gamma_{\text{e}}^{\text{call}}, \gamma_{\text{e}}^{\text{call}} \rangle).
\]

(11a)

and

\[
\text{Step}(\langle \lambda, \langle a \rangle, \tau_{\text{body}}^{\text{def}} \rangle, \langle \text{zero}, \text{one} \rangle, D, \langle \text{cond}, \gamma_{\text{e}}^{\text{recur}}, \gamma_{\text{e}}^{\text{recur}}, \gamma_{\text{e}}^{\text{recur}} \rangle).
\]

(11b)

\(^2\)If \( x \) is a value, but neither \#true nor \#false, then the result depends on which version of Scheme/Racket one is using. This case does not occur for \text{incr}.
The entire computation

\[
\langle \text{incr}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle \quad \text{[initial term]}
\]

\[
\text{incr}
\]

\[
\langle \lambda, \langle a \rangle, \tau_{\text{def}} \text{ body} \rangle
\]

\[
\langle \langle \lambda, \langle a \rangle, \tau_{\text{def}} \text{ body} \rangle, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle
\]

Step8

\[
\text{cond}, \langle \langle \text{equals?}, \langle \text{one}, \text{zero}, \text{one} \rangle, e \rangle, \text{one} \rangle, \gamma_{\text{\text{call} \ \text{call}}}, \gamma_{\text{\text{incr}}} \rangle
\]

Step12

\[
\text{equals?}, \langle \text{one}, \text{zero}, \text{one} \rangle, e
\]

\[
\#\text{false}
\]

Step8

\[
\text{cond}, \langle \text{\#false}, \text{one} \rangle, \gamma_{\text{\text{call} \ \text{call}}}, \gamma_{\text{\text{incr}}} \rangle
\]

\[
\text{cond}, \langle \text{equals?}, \langle \text{first}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle, \text{zero} \rangle, \gamma_{\text{\text{call}}} \rangle
\]

Step12

\[
\text{equals?}, \langle \text{first}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle, \text{zero}
\]

\[
\#\text{false}
\]

Step8

\[
\text{cond}, \langle \text{\#true}, \gamma_{\text{\text{call}}} \rangle
\]

Step11

\[
\text{cons}\langle \text{zero}, \langle \text{incr}, \langle \text{rest}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle \rangle \rangle
\]

\[
\text{cons}\langle \text{zero}, \langle \text{incr}, \langle \text{rest}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle \rangle \rangle
\]

\[
\text{cons}\langle \text{zero}, \langle \text{incr}, \langle \text{rest}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle \rangle \rangle
\]

\[
\text{cons}\langle \text{zero}, \langle \text{incr}, \langle \text{rest}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle \rangle \rangle
\]

Step9

\[
\text{cons}\langle \text{zero}, \langle \text{incr}, \langle \text{rest}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle \rangle \rangle
\]

\[
\text{cons}\langle \text{zero}, \langle \text{incr}, \langle \text{rest}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle \rangle \rangle
\]

Step11

\[
\text{cons}\langle \text{zero}, \langle \text{incr}, \langle \text{rest}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle \rangle \rangle
\]

\[
\text{cons}\langle \text{zero}, \langle \text{incr}, \langle \text{rest}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle \rangle \rangle
\]

Step9

\[
\text{cons}\langle \text{zero}, \langle \text{incr}, \langle \text{rest}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle \rangle \rangle
\]

\[
\text{cons}\langle \text{zero}, \langle \text{incr}, \langle \text{rest}, \langle \text{one}, \text{zero}, \text{one} \rangle \rangle \rangle \rangle
\]

Step13