Review: Stepping Rules

Application of built-in functions:  \((f \, v_1 \ldots \, v_n) \Rightarrow v\)
where \(f\) is a built-in function and \(v\) is the value of \(f(v_1, \ldots, v_n)\).

Substitution of Constants: \(id \Rightarrow \text{val}\), where \((\text{define} \ id \ \text{val})\) occurs to the left.

Review: Stepping Rules

Application of user-defined functions

The general substitution rule is:
\((f \, v_1 \ldots \, v_n) \Rightarrow \exp'\)
where \((\text{define} (f \, x_1 \ldots \, x_n) \ \exp)\) occurs to the left, and \(\exp'\) is obtained by substituting into the expression \(\exp\), with all occurrences of the formal parameter \(x_i\) replaced by the value \(v_i\) (for \(i\) from 1 to \(n\)).
Group Question - Stepping sum-of-squares
The following have been processed in the Beginning Student language:

\[
\text{(define (sum-of-squares x y) (+: (sqr x) (sqr y)))}
\]

Step through the following:
\[
\text{(sum-of-squares 3 4)}
\]

Review: Stepping Rules
Substitution in cond expressions
There are three rules: when the first expression is false, when it is true, and when it is else.

\[
\text{(cond [false exp] \ldots \Rightarrow (cond \ldots).)}
\]

\[
\text{(cond [true exp] \ldots \Rightarrow exp.)}
\]

\[
\text{(cond [else exp]) \Rightarrow exp.}
\]

These suffice to simplify any cond expression. Here we are using an omission ellipsis to avoid specifying the remaining clauses in the cond.

Group Question - Stepping cond
The following have been processed in the Beginning Student language:

\[
\text{(define x 1)}
\]

\[
\text{(define y 1)}
\]

Step through the following:

\[
\text{(cond [(= x 0) 'one]}
\]

\[
\text{[else (< (/ y x) c)])}
\]
Review: Stepping Rules

Simplification Rules for and and or

The simplification rules we use for Boolean expressions involving and and or differ from the ones the Stepper in DrRacket uses in the intermediate steps.

\[(\text{and} \ false \ldots) \Rightarrow false.\]

\[(\text{and} \ true \ldots) \Rightarrow (\text{and} \ldots).\]

\[(\text{and}) \Rightarrow true.\]

\[(\text{or} \ true \ldots) \Rightarrow true.\]

\[(\text{or} \ false \ldots) \Rightarrow (\text{or} \ldots).\]

\[(\text{or}) \Rightarrow false.\]

Group Question - Stepping and

The following have been processed in the Beginning Student language:

\[(\text{define} \ x \ 0)\]

\[(\text{define} \ y \ (+ \ x \ 1))\]

Step through the following:

\[(\text{and} \ (\text{not} \ (= \ x \ 0)) \ (<= \ (/ \ y \ x) \ c))\]

Review: Posn structures

- **constructor** function `make-posn`, with contract
  
  `;; \ make-posn: Num \ Num \ \rightarrow \ Posn`

- **selector** functions `posn-x` and `posn-y`, with contracts
  
  `;; \ posn-x: \ Posn \ \rightarrow \ Num`
  
  `;; \ posn-y: \ Posn \ \rightarrow \ Num`

Example:

\[(\text{define} \ mypoint \ (\text{make-posn} \ 8 \ 1))\]

\[(\text{posn-x} \ mypoint) = > 8\]

\[(\text{posn-y} \ mypoint) = > 1\]
Review: Posn structures
Possible uses:
• coordinates of a point on a two-dimensional plane
• positions on a screen or in a window
• a geographical position

Note:
• An expression such as \( \text{(make-posn 8 1)} \) is considered a value, which will not be rewritten by the Stepper or our semantic rules.
• The expression \( \text{(make-posn (+ 4 4) (- 3 2))} \) would be rewritten to (eventually) \( \text{(make-posn 8 1)} \).

Exploring Structures - Student Example
(\text{define-struct student (questid name grade)})
;; A Student is a (make-student Str Str Nat)
;; requires: grade \(<=\) 100
;; first character in name should be capitalized
(\text{define student1 (make-student \"cpt6amrka\" \"Steve\" 52)})
(\text{define student2 (make-student \"ironman\" \"Tony\" 100)})

(\text{student-questid student1})
\Rightarrow \"cpt6amrka\"

(\text{student-name student2})
\Rightarrow \"Tony\"

(\text{student-grade student2})
\Rightarrow 100

Group Problem - hand=?
Given the following data definitions, write a function \text{hand=}\? that consumes two Hands, produce true if two Hand have same Cards (order doesn’t matter)

(\text{define-struct card (rank suit)})
;; A Card is a (make-card Nat Sym).

(\text{define-struct hand(c1 c2)})
;; A Hand is a (make-hand Card Card).