CS 135 Winter 2017

Tutorial 2: Stepping and Structures
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: \(id \Rightarrow val\), where \(\text{(define id val)}\) occurs to the left.
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 \textit{sum-of-squares}

The following have been processed in the Beginning Student language:

\begin{verbatim}
(define (sum-of-squares x y)
  (+ (sqr x) (sqr y)))
\end{verbatim}

Step through the following:

\begin{verbatim}
(sum-of-squares 3 4)
\end{verbatim}
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} \ [\text{false exp}] \ldots) \Rightarrow (\text{cond} \ldots).

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

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

These suffice to simplify any \text{cond} expression. Here we are using an omission ellipsis to avoid specifying the remaining clauses in the \text{cond}.
Group Question - Stepping cond

The following have been processed in the Beginning Student language:

(define x 1)
(define y 1)

Step through the following:

(cond [(= x 0) 'one]
      [else (< (/ y x) c)])
**Review: Stepping Rules**

**Simplification Rules for and and or**

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

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

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

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

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

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

\[(\text{or}) \Rightarrow \text{false}.\]
Group Question - Stepping and

The following have been processed in the Beginning Student language:

(define x 0)
(define y (+ x 1))

Step through the following:
(and (not (= x 0)) (<= (/ y x) c))
Review: Posn structures

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

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

Example:

```scheme
(define mypoint (make-posn 8 1))
```

```
(posn-x mypoint) => 8
(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 `(make-posn 8 1)` is considered a value, which will not be rewritten by the Stepper or our semantic rules.
- The expression `(make-posn (+ 4 4) (− 3 2))` would be rewritten to (eventually) `(make-posn 8 1)`. 
Exploring Structures - Student Example

(define-struct student (questid name grade))
;; A Student is a (make-student Str Str Nat)
;; requires: grade \leq 100
;; first character in name should be capitalized

(define student1 (make-student "cpt6amrka" "Steve" 52))
(define student2 (make-student "ironman" "Tony" 100))

(student-questid student1)
=> "cpt6amrka"

(student-name student2)
=> "Tony"

(student-grade student2)
=> 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).