Binary Trees

Binary trees are also called S-Expressions in Lisp and Scheme.

They are of the form

\((\text{item} . \text{item})\)

where item is any atomic value or any S-Expression. For example:

\((A . B)\)
\((1.2 . "xyz")\)
\(((A . B) . C)\)
\((A . (B . C))\)

S-Expressions are linearizations of binary trees:

\[
\begin{array}{c}
A \\
B \\
\hline
1.2 \\
"xyz"
\end{array}
\]

S-Expressions are built and accessed using the predefined functions \(\text{cons}\), \(\text{car}\) and \(\text{cdr}\).

\(\text{cons}\) builds a new S-Expression from two S-Expressions that represent the left and right children.

\(\text{cons}(E1,E2) = (E1 . E2)\)

\(\text{car}\) returns are left subtree of an S-Expression.

\(\text{car} (E1 . E2) = E1\)

\(\text{cdr}\) returns are right subtree of an S-Expression.

\(\text{cdr} (E1 . E2) = E2\)

Lists

In Lisp and Scheme lists are a special, widely-used form of S-Expressions.

() represents the empty or null list

(A) represents the list containing A.

By definition, \((A) \equiv (A . () )\)

(A B) represents the list containing A and B.

By definition,

\((A . B) \equiv (A . (B . () ) )\)

In general, \((A B C . . . Z) \equiv \)

\((A . (B . (C . . . (Z . () ) . . . ))))\)

\((A B C) \equiv \)

\[
\begin{array}{c}
A \\
B \\
C \\
\hline
()
\end{array}
\]

Function Calls

In List and Scheme, function calls are represented as lists.

\((A B C)\) means:

Evaluate A (to a function)

Evaluate B and C (as parameters)

Call A with B and C as its parameters

Then use the value returned by the call as the “meaning” of \((A B C)\).

\(\text{cons}, \text{car} \quad \text{and} \quad \text{cdr}\) are predefined symbols bound to built-in functions that build and access lists and S-Expressions.

Literals (of type integer, real, rational, complex, string, character and boolean) evaluate to themselves.
For example ($\Rightarrow$ means “evaluates to”)

\[(\text{cons} \ 1 \ 2) \Rightarrow (1 \ . \ 2)\]

\[(\text{cons} \ 1 \ ()\ ) \Rightarrow (1)\]

\[(\text{car} \ (\text{cons} \ 1 \ 2)) \Rightarrow 1\]

\[(\text{cdr} \ (\text{cons} \ 1 \ ())) \Rightarrow ()\]

But,

\[(\text{car} \ (1 \ 2)) \text{ fails during execution!}\]

Why?

The expression \((1 \ 2)\) looks like a call, but \(1\) isn’t a function! We need some way to “quote” symbols and lists we don’t want evaluated.

\[(\text{quote} \ \text{arg})\]

is a special function that returns its argument unevaluated.

Thus \((\text{quote} \ (1 \ 2))\) doesn’t try to evaluate the list \((1 \ 2)\); it just returns it.

Since quotation is so often used, it may be abbreviated using a single quote. That is

\[(\text{quote} \ \text{arg}) \equiv '\text{arg}\]

Thus

\[(\text{car} \ '(\text{a b c})) \Rightarrow \text{a}\]

\[(\text{cdr} \ '(\text{(A) (B) (C)})) \Rightarrow (\text{(B) (C)})\]

\[(\text{cons} \ 'a \ '1) \Rightarrow (a \ . \ 1)\]

But,

\[('\text{cdr} \ '(\text{A B})) \text{ fails!}\]

Why?

User-defined Functions

The list

\[(\text{lambda} \ (\text{args}) \ (\text{body}))\]

evaluates to a function with \((\text{args})\) as its argument list and \((\text{body})\) as the function body.

No quotes are needed for \((\text{args})\) or \((\text{body})\).

Thus

\[(\text{lambda} \ (x) \ (+ \ x \ 1))\]

evaluates to the increment function.

Similarly,

\[((\text{lambda} \ (x) \ (+ \ x \ 1)) \ 10) \Rightarrow 11\]

We can bind values and functions to global symbols using the \text{define} function.

The general form is

\[(\text{define} \ \text{id} \ \text{object})\]

\(\text{id}\) is not evaluated but \(\text{object}\) is. \(\text{id}\) is bound to the value \(\text{object}\) evaluates to.

For example,

\[(\text{define} \ \text{pi} \ 3.1415926535)\]

\[(\text{define} \ \text{plus1} \ (\text{lambda} \ (x) \ (+ \ x \ 1))\ )\]

\[(\text{define} \ \text{pi}\text{*2} \ (* \ \text{pi} \ 2))\]

Once a symbol is defined, it evaluates to the value it is bound to:

\[(\text{plus1} \ 12) \Rightarrow 13\]
Since functions are frequently defined, we may abbreviate

\[(\text{define } \text{id} \ (\lambda (\text{args}) (\text{body})))\]

as

\[(\text{define} \ (\text{id}\ \text{args}) \ (\text{body}))\]

Thus

\[(\text{define} \ (\text{plus1}\ x) \ (+\ x\ 1))\]