Conventions

Parentheses only appear to fill their traditional mathematical role – to override associativity. Other than this, they are never required as syntactic constructs.  
  
The right arrow ⇒ is used to denote a replacement operation. The language processor transforms the expression template on the left into the expression template on the right.

The traditional equal sign = is only used in definitions to indicate that the expression to the left is being defined as the expression on the right.  
  
The equivalence (or identical) sign ≡ is used to indicate that the expressions to the left and to the right are operationally equivalent to each other. The behavior of the two expressions is the same.

Definitions

Symbols: σ are alphanumeric strings that begin with a letter.

Terms: τ are either a symbol or a pair.

Pairs: π are formed via an infix “dot operator”. The pair formed from terms τ1 and τ2 is written:

τ1 . τ2

Unlike terms, the dot operator is *right associative*:

τ1 . τ2 . τ3 ≡ τ1 . (τ2 . τ3)

Note: Turner preferred use of the more legible “colon” to the “dot”. We keep with dot here, following its more conventional use by the Lisp family of languages.  
  
Lists are either the symbol nil, which designates an empty list, or they are constructed from one or more pairs.  
  
In the case of a pair, the first term in the pair is referred to as the head of a list represented by that pair where the remaining sub-list is represented by the second term and is referred to as the tail of that list.

Each head element can be any term and therefore may itself represent either a symbol or a list.

Such proper lists are also referred to as nil-terminated lists. If the second element of the final pair is terminated by some symbol other than nil, the entire sequence of pairs is referred to as a “dotted list”.

Printed Representation of a Pair

To write any sequence of pairs (whether they may turn out to represent a proper, nil-terminated list or a dotted list) one begins by establishing a reference to the “remaining sub-list” of elements. Then the following iteration is performed:  
  
First, If the remaining sub-list is the symbol nil, writing of the list is complete. (If the remaining sub-list is some other non-nil symbol a dot is written followed by writing the non-nil symbol to complete writing of a dotted list.)

Otherwise, the remaining sub-list is a pair; and the head of that sub-list is written. If this head itself represents a list, an open parenthesis is written, followed by writing all of the elements of that list (using a recursive application of this method), followed by a close parenthesis.   
  
Second, the “remaining sub-list” of elements is replaced with the second element of the pair; and the process of writing the sub-list returns to the first step.  
  
Application

When two terms appear next to each other, the first term is said to be applied to the second. Applications are left associative:

τ1 τ2 τ3 ≡ (τ1 τ2) τ3

Abstraction

Abstraction of a symbol from a term, produces a term:

λx x ⇒ I

λx y ⇒ K y

λx (τ1 τ2) ⇒ S (λx τ1) (λx τ2)

Currying

[head . tail] expr = λ(head . tail) expr  
  
λ(head . tail) τ = U (λhead (λtail τ))  
  
λnil τ ⇒ Knil τ

Uncurrying

U func (head . tail) ⇒ func head tail

Reduction

Reduction of terms:

S τ1 τ2 τ3 ⇒ τ1 τ3 (τ2 τ3)

K τ1 τ2 ⇒ τ1

A similar combinator, introduced by Currying, only reduces where τ2 is nil:

Knil τ1 nil ⇒ τ1

I τ ⇒ τ

Y h ⇒ h (Y h)

Expressions

An expression: expr can be any term, or abstraction

**Note:** An expression can be represented either by a dotted pair; or by a proper list.

Function Definition

(Recursive) Functions: func are defined in terms of expressions.

def func = expr ≡ func = Y (λfunc expr)

def func x expr ≡ def func = λx expr