

A (Lisp-Like) TOY Language [Version2]

The language **TOY** is composed of terms (as opposed to instructions/statements) built up from constants, variables and lesser terms by applying functions. The object of computation (ie. the data structures) are integers $\dots -2 -1 0 1 2 3 \dots$. \mathbb{Z} denotes the set of integers. **FUN** is a set non-numeric words naming functions. **MINUS** and **IF** (the constants of **FUN**) name primitive functions from $\mathbb{Z} \times \mathbb{Z}$ to \mathbb{Z} . Other names in **FUN** refer to functions defined by terms of the language **L**.

\mathbb{L} is the set of all **terms** as defined by the following.

- (t1) The variables $v_1 v_2 v_3 v_4 \dots$ are terms.
- (t2) The constants $\dots -3 -2 -1 0 1 2 3 \dots$ are terms.
- (t3) IF t_1 and t_2 are terms, then **(MINUS $t_1 t_2$)** and **(IF $t_1 t_2$)** are terms.
- (t4) If fn names a function defined by a term t_1 of k variables and $s_1 \dots s_k$ are k terms, then **(fn $s_1 \dots s_k$)** is a term.

→ **사용자 정의 함수** → **define function**
DEFUN

DEFUN is used to create functions in the TOY language. It has the following syntax:

(DEFUN <function name> (<parameter 1> <parameter 2> \dots <parameter n >)
<process description>)

Example: (DEFUN ADD ($x y$) (MINUS x (MINUS 0 y)))

DEFUN does not evaluate its arguments. It just looks at them and establishes a function definition, which can later be referred to by having the function name appear as the first element of a list to be evaluated. The function name must be a symbolic atom. When DEFUN is used, like any function, it gives back a value.

The value DEFUN gives back is the function name, but this is of little consequence since the main purpose of DEFUN is to establish a definition, not to return some useful value.

INTERPRETER. A higher-level expression of interpreting term is built on top of the terms of TOY. These interpreting terms use the functions **VALUE**<>, **SUBST**<>, and **APPLY**<>. The interpreter itself consists of a set of equations (rewrite rules) in these terms used to direct computations in TOY.

(ADD 1 2) → (MINUS 1 (MINUS 0 2))

SUBST is a function whose domain includes certain t4-terms. Given the t4-term (fn $n_1 \dots n_k$), let t_1 be the term defining the function fn and $u_1 \dots u_k$ be the variables of t_1 listed in order of

increasing index. Define **SUBST** $\langle \text{fn } (n1 \dots nk) \rangle$

to be the term formed from $t1$ by replacing each uj by the corresponding nj .

The functions **VALUE**: $\text{TOY} \rightarrow Z$ and **APPLY**: $\text{FUN} \times Z^* \rightarrow \text{TOY}$ are defined as follows.

- (v1) **VALUE** $\langle u \rangle = \text{undefined}$ if u is a variable,
 (v2) **VALUE** $\langle n \rangle = n$ if n is an integer,
 (v3) **VALUE** $\langle (\text{MINUS } t1 \ t2) \rangle$
 $= \langle t1 - t2 \rangle$ if $t1$ and $t2$ are integers,
 $= \text{VALUE}\langle \text{MINUS VALUE}\langle t1 \rangle \text{VALUE}\langle t2 \rangle \rangle$ otherwise.
VALUE $\langle (\text{IF } t1 \ t2) \rangle$
 $= \text{VALUE}\langle t2 \rangle$ if $t1$ is positive integer and $t2$ has a value.
 $= 0$ if $t1$ is 0 or a negative integer,
 $= \text{VALUE}\langle (\text{IF VALUE}\langle t1 \rangle \ t2) \rangle$ otherwise

- (v4) **VALUE** $\langle (\text{fn } (s1 \dots sk) \rangle$
 $= \text{APPLY}\langle \text{fn } (\text{VALUE}\langle s1 \rangle \dots \text{VALUE}\langle sk \rangle) \rangle$ if fn is neither IF nor MINUS
 (a) **APPLY** $\langle \text{fn } (n1 \dots nk) \rangle$
 $= \text{VALUE}\langle \text{SUBST}\langle \text{fn } (n1 \dots nk) \rangle \rangle$.

Where the $n1 \dots nk$ are integers corresponding to the values of
 $\text{VALUE}\langle s1 \rangle \dots \text{VALUE}\langle sk \rangle$.

The evaluation of **APPLY** $\langle \text{fn } (\text{VALUE}\langle s1 \rangle \dots \text{VALUE}\langle sk \rangle) \rangle$

must begin with the $\dots \text{VALUE}\langle si \rangle \dots$ expressions. Viewing interpreter terms in tree-form, in which successive levels correspond to successive nestings of interpreter functions we have

APPLY $\langle \dots \rangle$

$\text{VALUE}\langle s1 \rangle \dots \text{VALUE}\langle sk \rangle$

Such a tree may achieve arbitrary finite depth. But regardless of depth evaluation can occur only at the terminal nodes.

A **PROGRAM** in TOY is an expression of the type

(fn n1 .. nk)

Where the term named by fn has k variables, and $n1 \dots nk$ are integers. Such a program's computation is the evaluation of

APPLY $\langle \text{fn } (n1 \dots nk) \rangle$

by the equations of the previous paragraph, and the output of the computation (if it terminates) is the value of **APPLY** $\langle \text{fn } (n1 \dots nk) \rangle$.

Examples.

- (e1) Let "ADD" name the function defined as follows:

$$(\text{DEFUN ADD } (x \ y) \ (\text{MINUS } x \ (\text{MINUS } 0 \ y)) \)$$

Then (ADD 4 76) is evaluated by substituting arguments in to the term, using the equations v1, . . v4, a. The program is computed as follows.

APPLY< ADD (4 76) >
 VALUE< SUBST<ADD (4 76) >>
 VALUE< (MINUS 4 (MINUS 0 76)) >
 VALUE< (MINUS VALUE<4> VALUE<(MINUS 0 76)>) >
 VALUE< (MINUS 4 <0 - 76>) >
 VALUE< (MINUS 4 -76) >
 <4 - -76>
 80.

- (e2) Let "EQUAL" name the function determined by the term

$$(\text{MINUS } (\text{MINUS } 1 \ (\text{IF } (\text{MINUS } x \ y) \ 1)) \ (\text{IF } (\text{MINUS } y \ x) \ 1))$$

for integers m and n the program

$$(\text{EQUAL } m \ n)$$

returns 1 if $m = n$, otherwise 0 is returned.

- (e3) Let POS, ZERO, and NEG be names for

$$(\text{IF } x \ 1)$$

$$(\text{EQUAL } x \ 0)$$

$$(\text{IF } (\text{MINUS } 0 \ x) \ 1)$$

respectively. These functions are the characteristic functions for the indicated sets of integers.

- (e4) Let IF/THEN/ELSE denote

$$(\text{ADD } (\text{IF } x \ y) \ (\text{IF } (\text{MINUS } 1 \ x) \ z))$$

The value of the term

$$(\text{IF/THEN/ELSE } x \ y \ z)$$

is VALUE<y> if VALUE<x> is positive,
 is VALUE<z> if VALUE<x> is zero or negative,
 is undefined otherwise.

Handwritten notes:
 이거 함수형 (next to IF/THEN/ELSE)
 조건 (above IF/THEN/ELSE)
 if then (left of IF/THEN/ELSE)
 else (right of IF/THEN/ELSE)

재귀 함수로 곱하기 정의

Recursive terms are either (1) named terms which contain their own names, or (2) terms containing recursive subterms. Recursive terms can be used to program recursive computations. For an example, multiplication in this TOY language corresponds to a recursive term.

(e5) Let TIMES name the term

$(\text{IF/THEN/ELSE } x \text{ (ADD } y \text{ (TIMES (MINUS } x \text{ 1) } y)) \text{ 0})$

We will now prove, for non-negative n and all k , that

$(\text{TIMES } n \text{ } k)$

yields the product of n with k .

When $n = 0$

```
1  TIMES (0 k)
2  APPLY< TIMES (0 k) >
3  VALUE< SUBST< TIMES (0 k) >>
4  VALUE< (IF/THEN/ELSE 0 (ADD k (TIMES (MINUS 0 1) k)) 0) >
.....
5  VALUE<0>
6  0
```

Assume that $\text{TIMES } (m \text{ } k)$ returns $m*k$ and that $n=m+1>0$.

```
1'  TIMES (n k)
2'  APPLY< TIMES (n k) >
3'  VALUE< SUBST< TIMES (n k) >>
4'  VALUE< (IF/THEN/ELSE n (ADD k (TIMES (MINUS n 1) k)) 0) >
5'  APPLY< IF/THEN/ELSE (n VALUE<(ADD k (TIMES (MINUS n 1) k)) >
0)>
.....
6'  APPLY< IF/THEN/ELSE (n VALUE<k>+ VALUE<(TIMES m k)> 0) >
.....
7'  APPLY< IF/THEN/ELSE (n k+m*k 0) >
8'  VALUE< (IF/THEN/ELSE n n*k 0) >
9'  n*k
```

Notice that steps have been left out after lines 4, 5' and 6'. However, by induction on n , we have verified that for non-negative n and all m the program $(\text{TIMES } n \text{ } k)$ return the product $n*k$.