A (Lisp-Like) TOY Language [Version 1]

The language **TOY** is composed of terms (as opposed to instructions) built up from constants, variables and lesser terms by applying functions. The objects of computation (i.e., the data structures) are integers … -2 -1 0 1 2 3 ... . **Z** denotes the set of integers. **FUN** is a set of non-numeric words naming functions. **MINUS** and **IF** (the constants of **FUN**) name primitive functions from ZxZ to Z. Other names in **FUN** refer to functions defined by terms of the language **L**.

L is the set of all **terms** as defined by the following.

(t1) The variables v1 v2 v3 v4 . . . (i.e., identifiers) are terms.

(t2) The constants . . -3 -2 -1 0 1 2 3 . . are terms.

(t3) IF t1 and t2 are terms, then **(MINUS t1 t2)** and **(IF t1 t2)** are terms.

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

The function **VALUE**: TOY --> Z is defined as follows.

(v1) VALUE< u > = undefined if u is a variable.

(v2) VALUE< n > = n if n is an integer.

(v3) VALUE< (MINUS t1 t2) >

= < t1 – t2 > if t1 and t2 are integers,

= VALUE< (MINUS VALUE<t1> VALUE<t2>) > otherwise.

VALUE< (IF t1 t2) >

= VALUE<t2> if t1 is positive integer and t2 has a value,

= 0 if t1 is 0 or a negative integer,

= VALUE< (IF VALUE<t1> t2) > otherwise.

* Tip: The TOY language Interpreter consists of two parts:

The first part generates "Intermediate Codes".

The second part executes the "Intermediate Codes" and produces the result.