Programming Languages Design
Memory

Authors:
Daniel García Ruíz
Pedro Pazos Curra

User Manual

Formal exposition with execution samples of new interpreter functionalities.

1.1: Multi-line expressions recognition

Terms are read in different lines, as opposed to the former implementation, in which any term would've been read in only one line.

Now, it's used a double semicolon ";;" to indicate the end of an expression and the start of a new evaluation.

```
X = true;;
   ↑
```

2.1: Internal fixed point combinator implementation

In lambda calculus, the fixed point combinator is a function that returns a fixed point for a given function.
This means that we can have a function F get another function g as parameter and returning an input function parameter l

```
fix f = f(fix f)
fix f = f(f(fix f))
...
```

This function allows us to develop recursive functions in a lambda calculus framework, in which recursion is not natively supported.

In this interpreter, fixed point combinator is called through the "letrec" function. It allows us to define a recursive function on a similar way to the following:

```
letrec sum : Nat -> Nat -> Nat =
        lambda n : Nat. lambda m : Nat. if iszero n then m else succ (sum (pred n) m)
in
sum 21 34;;
```

The format is like:

```
letrec [function name] : [function types] =
        [parameters definition] [recursive function definition] in
[function name] [parameters value]
```

2.2: Global definition context

There's the possibility of assigning a free variable identifier to any value or term so we can use it in further expressions as an alias.

Formatting: [identifying name] = [term]
Example: x = if true then true else false;;

Currently, this functionality is designed to override any id that is assigned twice.
This implementation was decided based on the low complexity and the high similarity with imperative, most spread languages.

2.3: String type

String type is available for users. As in many other languages, string variables are a succession of one or more characters.
They are summoned by "[string]". For instance, "great" is a valid string.

Example: "Hello World";;

Concatenation is also supported for strings. It's called through concat method.

Formatting: concat [str1] [str2]
Example: concat "straw" "berry";;

2.4, 2.5: Pair and Tuple type

Tuples are finite ordered collections of items. They have the following format {item1, item2, ..., itemN}.

A pair is a determined concreteness of the tuple type in which the dimensionality of the tuple is only of two elements.

Pair Example: {if true then false else true, succ 0};;
Tuple Example: {"Hello", succ (succ 0), true};;

Projections are supported for tuples. The N-Projection operation result is the N component of the tuple, when it is fully evaluated.

Formatting: t.N, t = Tuple and N = Int
Example: x = {false, succ (succ 0), "Hello"}.2;;

2.9: Unit type

Unit type is a singleton item that only accepts one type of value.
It's often used as placeholder when the specific value is not as important as its plain existence.

```
() -> a -> a'
↑
Unit
```

In the console prompt, Unit value could be formatted as it follows:

```
unit;;

term1 ; term2;;
     |
(translates |  into  )
     ↓
(lambda x : Unit. term2) term1
     ↓
(λx.term2) term1
```

Meaning that x is the bound variable, term2 is the body of the lambda expression and term1 is the parameter given to the expression.

This is the standart format of a lambda calculus application, that changes coincidences of the bound variable on the body with the input.

In this specific case, (λx.term2) term1 = term1

2.10: I/O Operations

As an upgrade to the system adaptability and usefulness, a bunch of input-output functionalities have been added, such as the following:

print_nat: Prints a Nat term on the console.

```
print_nat 5;;
```

print_string: Prints a String term on the console.

```
print_string "Hi";;
```

print_newline: Prints a newline character, leaping one line.

```
print_newline unit;;
```

read_nat: Given a natural number, entered by command line, assigns it to a Nat term.

```
read_nat unit;;
```

read_string: Given a string, entered by command line, assigns it to a String term.

```
read_string unit;;
```

Batch compiler mode:

Besides the executable file "top" that has been used the whole time to run the program now another executable file "run" has been added.

With "run" users now have the posibility to evaluate whole text files with any desired number of terms via command-line.

Usage:

```
./run [text file]
```