MEMORY

Nicolás Vázquez Cancela & Martín Castro Fernández

User manual

1.1 Multiline expressions

The program input will be read until the "$" symbol is found, regardless of any line changes previously entered.

Example:

```
let mas1 = L n:Nat.
    succ n
in
    mas1 0
$
```

2.1 Internal fixed-point combiner

The ability to implement recursive functions has been added, as well as the *prod*, *fib* and *fact* functions (in the examples.txt file).

Example:

```
letrec iseven:(Nat->Bool) = L x:Nat.
    if iszero x then true
    else if iszero (pred x) then false
    else iseven (pred (pred x))
in
    iseven 2
$
```

2.2 Context of global definitions

The ability to associate free variables with values or terms has been added.

Example:

```
x = 1$
y = 2$
{x, y}$
```

2.3 String type

The `String` type has been added.

Example:

```
"example"          String
"exa" ^ "mple"     Concatenation of Strings
```

2.4 Pars

Pairs are a special case of tuples, so it was not necessary to implement them directly.

Example:

```
{0}$            Tuple
{0, 1}$         Par
{0, 1, 2}$      Tuple
```

2.5 Tuples

Tuples are a special case of records where the labels are natural numbers,

but for convenience, these are inferred according to the member position and must not be explicitly specified. Empty tuples are not allowed.

The format for writing a tuple is as follows:

{x, y...} where x, y are natural numbers.

The format for projecting a member is as follows:

{x0, y1, zi, ...}.i being i the position of the member starting at zero.

Example:

```
{0, 1, 2}$ Type {Nat, Nat, Nat}
{0, 1, 2}.0$ Returns 0 of Nat type
{0, 1, 2}.1$ Returns 1 of Nat type
{0, 1, 2}.2$ Returns 2 of Nat type
```

2.6 Registers

The ability to implement records has been added. These have alphanumeric fields that must begin with a letter. The empty record (top) can exist.

Example:

```
{}$            Register
```

2.7 Lists

The ability to implement lists has been added, as well as the *length*, *append* and *map* functions (in the examples.txt file). Lists can be constructed with a more convenient syntax.

Example:

```
cons[Nat] 0 (cons[Nat] 1 nil[Nat])       List

[0, 1]: Nat                              List
```

2.8 Subtyping

Subtyping capability has been added for records and functions.

Example:

```
(L x:{x:Nat}. x.x) {x=1,y=2}$

(L f:{x:Nat,y:Nat}->{x:Nat,y:Nat}. f {x=1,y=2}) (L x:{x:Nat}.
{x=x.x,y=x.x})

(* Good *)$

(L f:{x:Nat,y:Nat}->{x:Nat,y:Nat}. f {x=1,y=2}) (L
x:{x:Nat,y:Nat}. {x=x.x,y=x.x,z=x.x})

(* Good *)$

(L f:{x:Nat,y:Nat}->{x:Nat,y:Nat}. f {x=1,y=2}) (L x:{x:Nat}.
{x=x.x,y=x.x,z=x.x})

(* Good *)$

More examples in examples.txt
```

2.9 Unit type

The `Unit` type has been added.

Example:

```
unit; unit; true$ Chain of Unit expresions that returns true
```

2.10 Input/output operations

The ability to perform input/output operations has been added. In addition, a program has been implemented that receives a list by command line, sorts it and finally prints it (msort.txt).

Example:

```
print_string “hello\n”;

print_nat 10;

read_string unit;

read_nat unit$
```

Technical manual

The contents of this manual have been replaced by comments in the source code.