DLP Memory

Practice carried by ***Daniel Olañeta Fariña*** and ***Ángel Álvarez Rey***.

User Manual

1.1 Multiline examples:

```
if false
then true
else false;
```

Expected result: `-: Bool = false`

2.1 Debug Mode examples:

Example without file:

```
/Interprete-de-Lambda-Calculo$ ledit ./top -debugMode
```

```
Evaluator of lambda expressions...
>> x = 5;
4
3
2
1
0
-: Nat = 5
```

Example with file:

```
/Interprete-de-Lambda-Calculo$ ledit ./top -debugMode examples.txt
```

```
-: Bool = true
-: Bool = true
false
-: Bool = false
-: Nat = 0
-: Bool = true
2
1
0
-: Nat = 3
2
1
0
-: Nat = 3
```

1.2 File input examples:

```
Interprete-de-Lambda-Calculo$ ledit ./top examples.txt
```

```
-: Bool = true
-: Bool = true
-: Bool = false
-: Nat = 0
-: Bool = true
-: Nat = 3
-: Nat = 3
-: Nat = 1
-: Bool = false
-: Nat = 1
type error: argument of iszero is not a number in line: 11
```

3.1 Fix examples:

· Recursive sum:

```
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
;
```

Result: `-: Nat = 55`

· Recursive prod:

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

```
lambda n : Nat. lambda m: Nat. if iszero n then 0 else sum m (prod
(pred n) m) in
prod 8 5
;
```

Result: `-: Nat = 40`

· Fibonacci:

```
letrec sum : Nat -> Nat -> Nat =
lambda n : Nat. lambda m : Nat. if iszero n then m else succ (sum
(pred n) m)
in
letrec fib : Nat -> Nat =
lambda n : Nat. if iszero n then 0 else if iszero (pred n) then 1
else sum (fib (pred n) ) (fib (pred (pred n)) ) in
fib 11;
```

Result: `-: Nat = 89`

3.2 Variable context:

```
>> x = true;
-: Bool = true
>> id = lambda x : Bool. x;
-: (Bool) -> (Bool) = (lambda x:Bool. x)
>> id x;
-: Bool = true
```

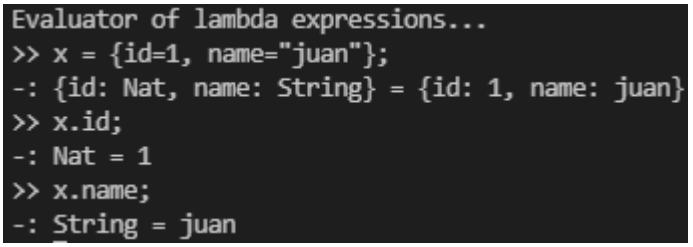
3.3 Type String:

```
Evaluator of lambda expressions...
>> x = "buenos_";
-: String = buenos_
>> y = "dias";
-: String = dias
>> concat x y;
-: String = buenos_dias
```

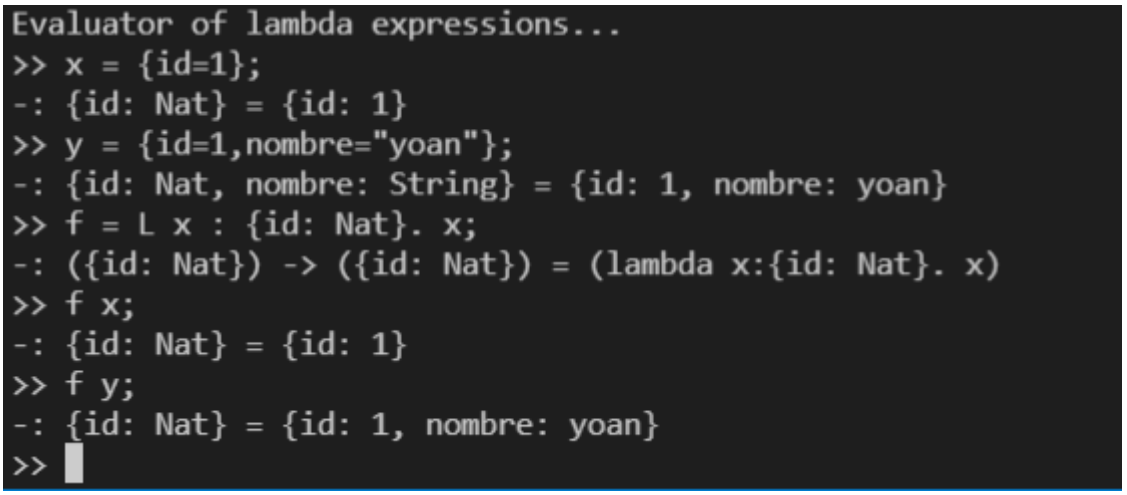
3.4 Type Pair:

```
Evaluator of lambda expressions...
>> x = {1, "hola"};
-: {Nat, String} = {1, hola}
>> x.1;
-: Nat = 1
>> x.2;
-: String = hola
```

3.5 Type Record:



3.7 Subtyping:



Technical Manual

parser.mly.

· Now the start (with token “s”) allows binding variables in addition to evaluating terms.

· New tokens added. These are: CONCAT, PAIR, OPENPAIR, CLOSEPAIR, COMMA, FIRST AND SECOND.

· New options in type term to accept CONCAT.

· New options in type appTerm to accept the Concat, First, Second and Record projection.

· New options in type atomicTerm to accept strings, Pairs and Records.

· New type record to declare how its structure has to be.

· New option in type atomicTy to accept strings.