Overview.md 1.77 KB

Implementing a small imperative language

We implement an interpreter for a small imperative programming language.

Example

This is the sample file fibonacci.xy:

```
{
 y := 0;
 z := 1;
 while (x > 0)
  {
   x := x - 1;
   t := y + z;
   y := z;
   z := t;
  }
}
```

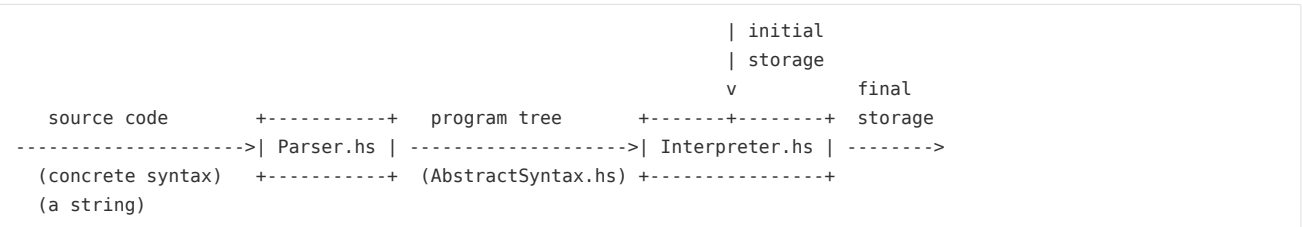
There are no IO commands in our language. The input is in the variable `x` and the output is in the variable `y`. Hence we call our language `xy`. We run this as follows:

```
  $ runhaskell Runxy.hs fibonacci.xy 11
  89
```

Overall structure

Given a program source code, that is, a `String`, in *concrete syntax* as above:

- We parse it, to produce a program tree in abstract syntax.
- Given this and an initial storage, that is, an assignment of values to program variables, we run the program to get a new storage.



We use a program Runxy.hs to read a file into a `String` and read a value `x` from the command line arguments, which then calls the parser and then the interpreter, with an initial storage assigning the value `x` to the program variable `"x"` and finally prints the value of the variable `"y"`, as in the above example.

Next: Concrete syntax