While+ Interpreter

Visentin Filippo

December 3, 2020

Task

Design a While+ Interpreter I such that:

- I takes as input any S $\in$ While+ and some representation of s $\in$ State
- I(S, s) must behave exactly as the $S_{ds}[S]s$ semantic tells
- I relies on Kleene-Knaster-Tarski fixpoint iteration sequence for evaluating the while statements

Interpreter structure

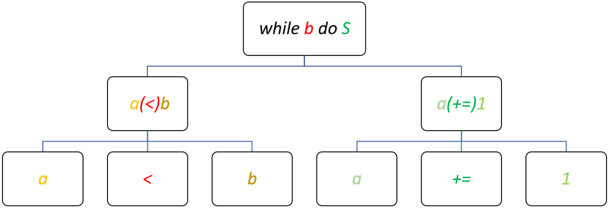
The interpreter is written in **Haskell**. It is composed of various modules:

- The **string parser**, to build the syntax tree
- The **syntax parser**, to match keywords and create statements
- The **interpreter**, to execute statements as stated by the S_{ds}

String parser

Builds a syntax tree starting from the input program.
It relies on parenthesis to distinguish **nested statements**

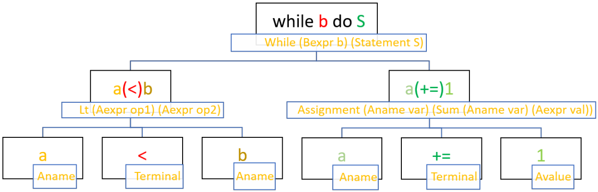
while (a(<)b) do (a(+=)1)



Syntax parser

Starting from the syntax tree creates Haskell's types to encode the statements.
It recursively traverses the tree composing types, each representing a different statement

while (a(<)b) do (a(+=)1)



Interpreter

To give a semantics to the types I need to equip them with a function, distinguishing **expressions** (evalE) and **statements** (evalS).
The 2 functions are overloaded, thanks to Haskell's pattern matching, to accomodate each expression variant.

While Statement

To implement the while semantics I started from the following observations:

$\mathcal{D} : \mathit{While} \rightarrow \mathcal{P}(\mathit{State}) \hookrightarrow \mathcal{P}(\mathit{State})$
$\mathcal{D}[\text{while b do S}]\text{T} = B_c[\neg b](\mathit{lfp}(\lambda T'.T \cup (\mathcal{D}[s] \circ B_c[b]T')))$

I start from a **precondition** and output a **postcondition** after the while.
The precondition is a single state, so I get:

$\mathcal{D} : \mathit{While} \rightarrow \mathit{State} \hookrightarrow \mathcal{P}(\mathit{State})$

I notice that the *lfp* gives the invariant of the loop.

While Statement

The Kleene-Knaster-Tarski fixpoint iteration is done to find this invariant. I need to apply this formulation until i get $FIX(F) = F$

Actually I observe that the only thing I need to keep of the invariant to get the exit state of the loop (if it terminates) is the **last state** computed by the procedure.

$\mathcal{D} : While \to State \hookrightarrow State$

So I can modify the procedure discarding other elements, not performing the union:

$lfp(\lambda T'.\mathcal{D}[s] \circ B_c[b]T')$

Also care must be taken if the loop does not **terminate**. If the fixpoint terminates, giving an invariant but the guard is still true, the wanted behaviour is to keep going with the iteration forever.