Andrew Krall
10-24-2018

BNF (Backus-Naur Form)

BNF grammars correspond almost perfectly to data declarations in Haskell.

Given a BNF grammar, we can use it to show that certain strings are in the language (and others are not!)

Ex: show (12. ()) $\in$ <LON>
"Contained in a language"

You want to go until there is nothing but terminal symbols.

<LON> => (<number>. <LON>)
=> (<number> . ())
=> (12. ())

BNF derivations

General Rule: String $s \in$ <L> if and only if there is a sequence of production steps: <L> => ... => s

- Note that there may be more than one such sequence.

BNF for programming language syntax: The λ-calculus

Lambda calculus

Example question: "Is the following Scheme code in <expr>"?

Processing Languages with Effects

→

Andrew Kvall
10-24-2018

Some strings don't parse. How do you represent syntax errors and parsing in Haskell?

An interpreter is a function.

interp :: AbstractSyntax → "Computation of Values"

We can use Maybe to handle syntax error.

Maybe is a monad.

The big fact is that any programming language fits in this scheme, where the notion that "Computation of Values" is a monad is true.

We can write an interpreter in Haskell for a language we wrote.

An error is a "side effect".

How do we handle this error side effect? We add a value for errors.

We want to change the definition of interp to incorporate the error as a value in the language.

Note: In this example, the fact that errors may happen is now reflected in the type signature.

→

Andrew Kvall

10-24-2018

This method is functional, but we want accurate error handling AND easy-to-read code.

We can use "do notation" in Haskell to accomplish this.

Please check the slides on Professor Harrison's website to view examples of interp and do notation. code

Alternative formulation of "do"

```
do v <- x
   e
```

can also be written as

"bind"

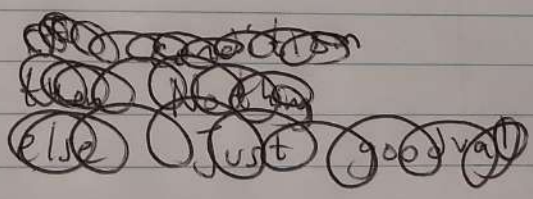
```
x >>= \v -> e
```

In fact, Haskell itself translates the "do"s into "bind"s at compile time.

"return" is an overloaded symbol meaning "Just".

Slides online have none code examples of throwing an error.

Throw is defined as,




```
throw :: Bool -> a -> Maybe a
throw condition goodval = if condition
                          then Nothing
                          else Just goodval
```

→

Andrew Krall

10-24-2018

In the Interp V5.0 example, Just and Nothing don't occur in the code.

Bind and return are overloaded.

There are a whole bunch of Monads in Haskell.

All example code is on Harrison's website!

Side effects in an impure language are not always expressed in the types, whereas in Haskell they are generally expressed in the types.