D7011E Compiler Construction

Compiler written in Haskell using Parsec.

Sebastian Larsson
seblar-0@student.ltu.se

Emil Svensson
emisve-1@student.ltu.se

Summary

The best compiler mankind has ever seen, written in Haskell using Parsec.

Introduction

Stuff

I now opened a room in Fronter where you turn in your final assignment. If we (during the face-face meeting) discussion some additional things to turn in, make it part of the zip archive and submit ti all together. Here is the instructions for the fronter submission.

"Here you turn in your final assignment, i.e., your compiler.
make a zip archive con tainting
1: your source code
2: a file describing the well-formedness rules
3: a README, on how to compile and run
4: your LOST counter in cOOre
5: additional test files, to show (the part) of type checking rules that your compiler handles"

Let me know if you have any questions, or run into any problem.

Grammar in EBNF form

```
digit = “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”;
int = {digit};
floating = {digit}, “.”, {digit};
all characters = ? all visible characters ?;
strings = ““‘, {all characters - ““‘}, ““‘;
expr = exprnum | exprctrl;
exprnum = factor, (“=” | “+” | “-” | “/” | “*”), factor;
exprctrl = if | for | return | claim;
variable = identifier;
function = “funktion” | “hel” | “flyt” | “sträng”, “(“, identifier, {arg}, “)”, {expr}, “klar”;
reset = “reset”, {expr}, “klar”;
idle = “idle”, {expr}, “klar”;
arg = variable, “=”, types;
call = callInside | callOutside;
callInside = identifier, “(“, {expr}, “)”;
callOutside = identifier, “§”, identifier, “(“, {expr}, “)”;
factor = types | function | call | async | sync | variable | (“(“, {expr}, “)”);
types = floating | int | strings;
claim = “begär”, identifier, {expr}, “klar”;
```

```
return = “återvänd”, [expr];
if = “om”, expr, {expr}, [“annars”, {expr}], “klar”;
for = “för”, expr, expr, expr, {expr}, “klar”;
async = “async”, [“efter”, int], [“före”, int], call;
sync = “sync”, call;
defn = function | expr;
toplevel = klass | include | includecore | reset | idle;
klass = “struktur”, identifier, {defn}, “meep”;
include = “referera”, str;
includecore = “refereracore”, str, {defn}, “meep”;
```

Clearly indicate what part of the grammar is covered in each section of your implementation

All grammar is in Parser.hs. The naming follow the EBNF quite closely.

Type checking rules

Expression rules

```
Env => e1 : typ    Env => e2 : typ
-------------------------------      typ is hel or flyt
Env => e1 + e2 : typ

Env => e1 : typ    Env => e2 : typ
-------------------------------    typ is hel or flyt
Env => e1 - e2 : typ


Env => e1 : typ    Env => e2 : typ
-------------------------------        typ is hel or flyt
Env => e1 * e2 : typ

Env => e1 : typ    Env => e2 : typ
-------------------------------        typ is hel or flyt
Env => e1 / e2 : typ
```

3 Define the type checking rules, (e.g., by inference trees)

type checking examples:

variables:

foo = 10 °hel (int)	ok
bar = “iam a happy string” °string (char*)	ok
foobar = 2.0 °flyt (float)	ok
foo + foo	ok
foo + bar	fail
foo + foobar	fail
foobar + foobar	ok
referera “std.coore” hel pow(factor = 0) °index is of type hel return factor*factor klar	ok
foo = fib(5)	ok
bar = fib(5)	fail
foo = fib(foo)	ok
foo = fib(bar)	fail

Define the language in terms of associativity on operators, and precedences.

Opperator	Associativity
*	Left
/	Left
+	Left

-	Left
=	Right

The table is sorted from highest to lowest precedences.

Conclusion

The language is usable altough it could be improved in several areas. First of all boolean expressions should be added, to make ifs and for loops easier, it is currently solved by using external functions in the std.coore library. The include paths when using referera requires a path relative to working directory when compiling, it might be a good idea to change this to use relative paths to the file. To handle includes the compiler uses unsafePerformIO, which could probably be replaced by something nicer.