1 Attribute Grammars for Cool (30 points)

In this homework, you will write the an attribute grammar for semantic analysis of (part of) Cool. This work will be a design of part of what you will program for PA4. Every expression will have two attributes:

- **env** The referencing environment (inherited). This environment contains all the information you need to perform name lookup in an expression. You can assume the environment has the methods `lookup` (returning the type of the identifier, or null), `type_leq` and `type_lub` methods declared for class `Environment`. You can create instances of the `SingleEnvironment` class to create nested environments.

- **type** The type of the expression (synthesized). The type will be a symbol that indicates the static type of the expression.

Recall that for each kind of node (each APS constructor), the attribute grammar must assign the inherited attributes of children nodes and the synthesized attributes of the root.

You may assume the existence of a function `lookup_class` that looks up a class in the (shared global) class table.

Additionally, every node will have a collection attribute **errors** that can be optionally assigned to a child or the root node. Multiple assignments (even to the same node) are possible. All assignments are collected together to get a set of error messages (possibly empty) for each node. You should generate messages for all type errors and errors concerning the object namespace (don’t worry about classes or methods) that occur for each construct.

Please use the following grammar for your attribute grammar:

\[
\begin{align*}
A & ::= \epsilon | AP \\
P & ::= \text{for } ID(L) \ S \ \text{end} \\
L & ::= \epsilon | N \\
N & ::= F | F;N \\
F & ::= ID:ID \\
S & ::= \epsilon | S R \\
R & ::= \text{var } ID:ID = E \\
& \quad | \ U = E \\
& \quad | \ ID\errors > E \\
& \quad | \ \text{unless } (E) \ S \ \text{end} \\
U & ::= ID | ID.ID \\
E & ::= \text{Extended Cool expression}
\end{align*}
\]

For example, the rule for loop would be:

```java
for loop(pred, body : Expression)
    pred.env = this.env
    body.env = this.env
    unless (pred.type == 'Boolean)
        this.errors :> "Predicate should be Boolean, not " + pred.type
    end
    this.type = 'Unit

// The following is just to demonstrate local attributes:
var silly : Symbol = env.lookup('fish);
```
unless (is_null(silly))
    this.errors :> "Fish don't like loops."
end
end

You may omit “this.” in your rules.

For this Homework, please only write the attribute rules for the following constructors: assign, cond, let, add, lt, alloc, unit. For this assignment, it is not necessary to check that formal parameters and branch variables are not assigned.

## 2 Testing (20 points)

Write test files for good and bad testing of the semantic analyzer. As before, we provide a test case tester:

```
test4 [-v] good.cool bad.cool ...
```

You can pass as many files as you wish on the command line—good and bad tests do not need to be distinguished for this Homework, although the tests must be segregated for PA4, since one must be able to test the output AST for the good tests.

CS 754: Homework # 3
due 2018/3/6

Instead of the previous parts, please do the following:

## 3 Attribute Grammars (20 points)

Find at least two papers on attribute grammars. Write a short essay on attribute grammars (plus the two references) that summarizes what attribute grammars are good for and why one should learn about them.

## 4 Related Work (30 points)

Read the “Featherweight Java” paper:

> Atsushi Igarashi, Benjamin C. Pierce, Philip Wadler. *Featherweight Java: a minimal core calculus for Java and GJ*. ACM Transactions on Programming Languages and Systems (TOPLAS), 23(3), May 2001, pp. 396 - 450.

(Former CompSci 732 students may instead read Chapter 19 of TAPL.)

Compare the type system of FJ with that of Cool (Section 13). What is the connection (if any) between $O, M$ in the Cool Manual and $\Gamma$ for FJ? Why does FJ have a “stupid cast” rule? Why does Cool not? In what ways is Cool’s type system simpler? FJ’s?