

Homework # 6

due 2008/3/11

1 Binding Variables in Cool

Look at the main program from `primes.cl`:

```
class Main : IO is
  Main() begin
    let
      s : Sieve := new Sieve(2);
      i : Integer := 2;
    in
      out_string("2 ");
      while true loop
        -- loop forever
        s.test(i := i + 1)
      pool
    end
  end;
end;
```

For each occurrence of an identifier `Main`, `IO`, `Main`, `s`, ... indicate:

- Whether it is a *defining* occurrence, or a *using* occurrence,
- If defining then indicate where else in the program one *could* have a use (hypothetically).
- If a using occurrence, then indicate where the defining occurrence is.

2 Attribute Grammars for Cool

In this and the following homework, you will write an attribute grammar for name resolution and type checking for Cool expressions. 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. It will support all three namespaces: objects, methods and classes. Looking up an object will get a `VarBinding*`; looking up a class will get a `class__class *` and looking up a method will get a `method_class *`. You may also assume that there is a member function `with_object` (with two symbol parameters) that returns a new referencing environment for an inner scope in which the given identifier has the given type.

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

In this assignment, you will write the rules for name resolution only, while assuming the rules for types are complete.

Before you write the attribute grammar, please answer the following questions:

- (a) Why is **env** an inherited attribute? Please explain what an inherited attribute is and why it makes sense for **env** to be inherited.
- (b) Why does the method lookup member function for an environment need to take a second parameter? What is it?
- (c) What kinds of errors are found when doing name resolution?
- (d) For what AST nodes will new environments need to be created (where the referencing environment changes for a child)?

Now write the attribute grammar with rules for every AST node for expressions and branches. You must define rules for the **env** attribute of child nodes and must also generate errors. Do not worry about cascading errors or error recovery; that aspect will be left to your PA4 programming. The errors should be appropriate to the concrete syntax, not using the abstract syntax names for things. For instance, an AST branch node corresponds to a “when” clause in the concrete syntax, and so error messages should talk about “when clauses” not “branches.”

You may use the **type** attribute. Do not write rules for the **type** attribute, nor generate type errors. Write the attribute grammar rules in a C++-like pseudo-code. You may use the syntax `#identifier` as a shortcut for `idtable.add_string("identifier")`.

Here’s a few examples to get you going:

```
block(body: Expressions)
  for e : Expression in body
    e.env = env

branch(name, type_decl: Symbol; expr: Expression)
  expr.env = env->with_object(name,type_decl)
  unless name != #self
    error << "cannot bind 'self' in when clause."
  unless type_decl == #_void || env->lookup_class(type_decl)
    error << "class " << type_decl << " not declared."

null()
  // no attributes to define
```

3 Naming Test Case

Write a short(!) Cool expression that has a naming error for each of the three namespaces. The errors should be independent (non-cascading). Then give the AST for this expression. Finally, explain how your attribute grammar finds each of the errors.