

Homework # 5

due 2008/3/4

1 Testing the Cool Parser

For PA3, you will need to provide a file `good.c1` that systematically tests every legal Cool syntactic feature. Part of your assignment will be testing that the operators have the correct precedence.

1. Attach a syntactically valid Cool program that systematically tests the precedence and associativity of every Cool operator. If one looks at the AST created by running your test case, it must be possible to deduce the associativity and relative precedence of every Cool operator. Your test case may assume transitivity: if the test shows that op_1 has higher precedence than op_2 and that op_2 has higher precedence than op_3 then it does not need to test whether op_1 has higher precedence than op_3 . It may also assume that associativity is implemented using bison's `%left`, `%right` and, `%nonassoc` declarations (as opposed to being implemented with multiple nonterminals: `term`, `factor`, `primary` etc.).

Caveats:

- You cannot test the associativity of the nonassociative operators in a legal Cool program. So, for example, each of `<`, `<=`, and `=` must be tested individually against `not`, `:=` and `+` (or `-`).
 - Associativity is basically irrelevant with unary operators.
 - A prefix unary operator's precedence can only be tested with respect to a binary operator or postfix unary operator to its right. A postfix unary operator can only be tested with respect to a binary or a prefix unary operator to its left. Thus an expression such as `x+-y` tests nothing.
 - The relative precedence of two prefix unary operators cannot be tested. So, for example, do not attempt to test the relative precedence of `not` and (unary) `-` against each other. Instead test them against `.` and `*` (or `/`).
 - Because of their syntactic restrictions, assignment (`x := expr`) functions as a prefix unary operator, and dispatch (`expr.m(...)`) function as a postfix unary operator.
2. Take a representative part of your solution to (1) and show how if that part of the test is removed, the test case would fail to detect a hypothetical simple parser error.
 3. Mr. Straw Man says "What's the point of testing? You can never be sure you have found all the bugs." Provide a reasonable (but short, about 100 words) defense of testing. In your answer, explain the purpose of testing and the benefits of systematic tests in particular.

2 Action Rules

Suppose we wished to add array syntax to Cool which is sugar for Cool syntax `new Array(n)`, `a.get_item(i)`, `a.set_item(i,v)` calls. (You will need to generate AST nodes with `dispatch`, `static_dispatch`, etc.) Complete the following bison rules:

```
expr    : ...
        | NEW '[' expr ']'
          { $$ =
            | expr '[' expr ']'
              { $$ =
                | expr '[' expr ']' ASSIGN expr
                  { $$ =
                    ;
```

(In the first rule we do not have a type for the array, because in Cool there's only a single array type: an array of `Objects`.)

Although this is not legal C/C++ syntax, you may use `#s` as short for

```
idtable.add_string("s")
```

You may turn in your solution to this part by writing directly on the assignment handout.