COL 226: Programming Languages
Assignment3: Type Checking and Evaluation

Submission Lifeline: Sun 25 Apr 23:59
Submission Deadline (with Late Penalty): Fri 30 Apr 23:59

---

1. (easy) Extend the language for Boolean algebra from assignment 2 to support the following:

   - Integer arithmetic with operators `PLUS, MINUS, TIMES, NEGATE` (unary operator), `EQUALS, LESSTHAN, GREATERTHAN`. Suitably extend the `AST` type specification.
   - Extend the expression language with the following:
     - `if` exp `then` exp `else` exp `fi`. You must raise appropriate exceptions when the first expression does not evaluate to a boolean value type or the expression in the `then` and `else` evaluate to different types.
     - `let` var = exp `in` exp `end`. This is to create temporary identifier bindings.

     Here, exp is either a valid formula from assignment 2[1] or a valid integer expression.
   - You can use the usual precedence and associativity rules for integer operations
   - Parenthesis can be used to define order the evaluation over all of the integer and boolean operations.

   You may refer to the demo code shown in the class (on April 1) to implement the above mentioned tasks.

2. (medium) Provide support for functions in your language implementation. That is, you should provide support for the following concrete syntax:

   - `fn` $(x : typ) : typ \Rightarrow$ `exp`
   - `fun` $x(y : typ) : typ \Rightarrow$ `exp`

   Some examples of concrete and abstract syntax pertaining to functions are given below:

Concrete syntax	Abstract syntax
`(f 3)`	`AppExp ( VarExp ("f"), NumExp (3))`
`fn (`x `: int): int` $\Rightarrow 1$	`Fn("x", INT, INT, NumExp(1))`
`fun g (f: int` $\rightarrow$ `int): int` $\Rightarrow$ `(f 1)`	`Fun(:"g", "f", ARROW(INT,INT), INT, AppExp ( VarExp ("f"), NumExp (1)))`

[1] except IF THEN ELSE

Recall the datatypes `value` and `typ` defined for AST specification in our evaluator demo. You may have to carefully think about the extensions to these datatypes to provide support for functions.

3. (medium) Implement a type checker for the extended language sought in this assignment. Each expression is either *well-typed* (*i.e.*, associated with at most one type), or an exception must be raised. For instance, `if e1 then e2 else e3 end` expression is well-typed when $e1$ is a boolean type, and $e2$ and $e3$ must have the same types. To implement the type checker, you must implement a type grammar. The type grammar provides rules and *syntax-directed* derivation that are required to conclude the type checking. HINT: You may have to define a type environment and domain and range types for each operator. Finally, if the type checking goes through without any hiccups, then evaluate an input program in the extended language using the evaluation strategy call-by-value.

Submission Instruction

Submit a zip file named ⟨EntryNumber⟩.zip. You may provide a README.md that contains suggestions for the evaluator.

Important Notes

1. Do not change any of the names given in this document. You are not even allowed to change upper-case letters to lower-case letters or vice-versa.

2. We have created a piazza post titled "A3 Queries". If you have doubts, please post only in this thread. The queries outside this thread or over email *will not* be entertained.

3. Instead of using `IF-THEN-ELSE` from assignment 2, use the syntax provided here (`if` exp `then` exp `else` exp `fi`).