CS51 Final Project Writeup

Amanda Stetz

May 7 2020

1 Extension for MiniML

The unextended MiniML language while capable of computing basic mathematical operations had the capacity to handle a greater breadth and complexity of operators and functions. For this reason, I chose to extend MiniML by adding the atomic types, float and string, and the operators and functions necessary to carry out division, greater than comparisons, factorials and modular arithmetic. With my extension, MiniML can handle more compound and complex procedures.

2 Atomic Types

2.1 Floats

The first atomic type I added was of type float.
I did so by adding 'Float' as a type of expr in expr.ml and expr.mli and creating a match statement for in each respective function that required it.

After reading the provided guide to the Ocaml parser I added float as a token in miminl-parse.mly and then defined floats as their own type in miniml-lex.ml, where floats are defined as any integer concatenated with a '.' followed by zero or more integers. Shown as:

| digit+ '.' digit* as fnum

In my extension, floats are able to be negated, added, subtracted, multiplied, divided, compared to another float, raised to the power of another float, and have their factorials calculated. These additional operations will be elaborated on later.

2.2 Strings

The second atomic type I added was of type string.
I did so by again adding 'Str' as a type of expr in expr.ml and expr.mli and creating a match statement for in each respective function that required it.

In miminl-parse.mly I added Str as a token and in miniml-lex.ml also defined Str as its own type. Str is defined as any character (expect a quotation mark) enclosed by quotation marks. I then use the String module to return only the contents of the quotations. Shown where the contents sans quotations are a substring of the original input :

STR (String.sub str 1 (String.length str - 2))

In MiniML strings are able to be output without quotations and compared for equality.

3 Functions and Operators

My extension also includes the addition of four binary operators - division, greater than comparision, exponents and modular arithmetic - and one unary operator, the factorial, which is also a function.

3.1 Binary Operators

Implementing each binary operator followed the same overall process. I added each operator under type binop in expr.ml and expr.mli and completed the appropriate binop to string match statements in the appropriate functions. In evaluation.ml I defined the appropriate atomic types that these binops can operate on and the operations themselves.

In miniml-parse.mly I added each binop as a token and declared its association in arithmetic, either left (divide, exponent and mod) or non-associative (greater than).

Then in miniml-lex.mll I added each respective operator to the symbol hashtable and sym definition.

Each respective operator uses the same symbol as it would in Ocaml (backslash, >, percentage sign), with the exception of exponents. I choose to have the exponent operation expressed as a caret rather than $**$ because I felt it was more intuitive for the user in terms of interface.

Consistent with the Ocaml regulations, Divide and Greaterthan can be used on only integers and floats. Mod can only be used on intergers, and Exponent can only be used on floats.

3.2 Functions and Unary Operators

To implement factorials in Miniml first I added Fact as type unop in expr.ml and expr.mli, completing the appropriate unop to string match statements when

necessary, where Fact is expressed concretely as "!".

In evaluation.ml I created helper functions to use within the unop evaluation function. These helper functions, fac and fl-fac perform the factorial operation on integers and floats respectively, therefore, Fact can only be used on these two atomic types.

In miniml-parse.mly I added Fact as a token and its left association. Additionally, I edited expnoapp so the factorial operator could be typed after the number it is supposed to operate on, in consistency with the standard mathematical notation.

Then in miniml-lex.mll I added "!" as a symbol in both the hashtable and the symbol defintion.

4 Conclusion

My extension of MiniML through introducing additional atomic types, binary operators, and unary operator functions, increases the breadth of computation that can be done within the language and thus the potential for mathematical complexity through combining these operations.