Homework Assignment #3

ESML: Embedded SML

Due to the increasing popularity of embedded devices, the SML community has decided that it is time to design a version of SML targeted at small computing devices, such as handhelds and pocket PCs. A committee was appointed, and after three years of laborious work, they have come up with Embedded SML, or ESML for short (pronounced essmel). ESML is a pure subset of SML: it contains only the core constructs of SML, those absolutely necessary to achieve Turing completeness. This reduced language design will allow the interpreter to have a slim memory footprint, making it ideal for memory constrained devices.

One the important parts of the ESML implementation is its type inference algorithm. The committee has already hired a person, whom they affectionately call the code monkey, to write most of the type inference code. However, the code monkey is not very smart, and was only able to implement the mundane and simple parts of the code: parsing, data structures, and printing of the data structures. The main part of the code, the core of the type inference algorithm, remains to be designed. The ESML committee now turns to you, experts of type inference, to implement this core functionality of the type inference algorithm.

ESML supports the following constructs, which you will have to handle in your type inference algorithm:

- Top-level function declarations (top-level let) and references to identifiers
- Function calls
- If-then-else constructs
- Arithmetic, boolean, and list expressions (ESML does not have a "`::`" operator, but rather a "`cons`" function to construct lists)

ESML however does not contain the following, and you will NOT have to handle these cases:

- Pattern matching
- User-defined type or datatype declarations
- Structures and functors
- Records or tuples
- Overloaded operators
- Type annotations on expressions
- Mutually recursive functions (the "`and`" construct)
- Nested let constructs
- Lambda expressions

The committee is actually very interested in lambdas and nested lets, and you will be given a bonus if you can add support for these constructs.

Your task

You can find the parser and some skeleton code for the type inference implementation on the course web page, in gzipped, tar'd form, under the "sample code snippets" link. Save the link to a file, e.g. `type_infer.tar.gz`, then do "`zcat type_infer.tar.gz | tar xfBpv -`" to unpack it into a directory named `type_infer`.

Take a look at the `README` file for information on the different files you are given. Your goal is to complete the type inference code located in the `type_infer.sml` file.

The parser you are given actually parses a much larger subset of SML than the one in ESML. You will however only be required to handle the constructs in ESML, which were outlined above. If the user inputs a file that contains a construct that you are not supposed to handle, you should raise the `UnhandledLanguageConstruct` exception.

What to hand in

You will need to hand in:

1. A printout of only your part of the code, or the parts that you changed. Do not printout the whole `type_infer.sml` file.

2. A clear, concise description of how your algorithm works, and how it handles each construct in the language. You can either hand this in as a separate print-out, or you can embed good and clear comments in your code at the appropriate places.

3. A soft copy of your whole `type_infer` directory, not just the `type_infer.sml` file. You should do this by running "`tar cvfz type_infer.tar.gz type_infer`", and then emailing the file `type_infer.tar.gz` as an attachment to both `lerns@cs` and `todd@cs`.

Grading

We will test your code by running the following commands in the `sml` interpreter on ceylon (you can develop your code on any of the instructional Linux machines, but please make sure at some point that your code does work on ceylon):

```
> use "load_ast.sml";
> use "type_infer.sml";
> test_type_infer("type_infer_test.sml");
> test_type_infer("new_type_infer_test.sml");
```

The file "`type_infer_test.sml`", which is included in the package you've downloaded from the web, already contains some test cases. You should add more test cases to this file to demonstrate that your code works. We will first run your type inference algorithm on these test cases. These should presumably work because you've already tested these yourself. After this, we will test your type inference algorithm against the ESML committee's test cases, which are located in "`new_type_infer_test.sml`" (and which you haven't seen).

Bonus

For bonus, and extra respect from the ESML committee, add support for handling lambda expressions. For even more bonus, add support for nested non-polymorphic let expressions. For a big slap on the back, support nested polymorphic let declarations.