CS350 Homework 2

Due Date: November 15, 2020

This homework is about implementing an interpreter for the declarative sequential model discussed in Chapter 2 of the book. You can assume that the source code of some Oz program is given in an easy-to-parse Abstract Syntax Tree format.

For example,

```
local X in X=10 end
```

is represented as

```
[var ident(x)
  [bind ident(x) literal(10)]]
```

1 Specification

Implement an interpreter to take the Abstract Syntax Tree as input, and output the sequence of execution states during the execution of the statement.

You will have to implement the single assignment store and the semantic stack. You may use the full Oz language, including Cells, to impement the interpreter.

You are provided with : Code for the unification algorithm (courtesy of Siddharth Agarwal, a former TA for the course.)

Please submit only working code. Non-working code is not eligible for partial credit.

1.1 Subsection

Kindly share the repository link with the Gmail id of the instructor.

2 Questions

1. A program is a statement. The AST of a statement is a list.

   The statement `[nop]` is the skip statement.

   A statement can also be a sequence of statements. Instead of the kernel syntax described in the course, we will represent a sequence of statements as list of statements.

   e.g. `[[nop] [nop] [nop] [nop]]`.

   Implement the semantic stack to execute statements of this form. [15 points]

2. Variable creation. The kernel language statement `local <X> in <s> end` will be represented as:

```
[var ident(x) s]
```

To implement this statement, you must implement the semantic stack and the single assignment store. Note that static scoping must be implemented for this question.

Suggested Oz Data Structure to implement the Single Assignment Store: Dictionary.

At this point, variables cannot be assigned, so the single assignment store will be a set of singleton sets, each singleton set consisting of a single variable.

Example statement:

```
AST syntax                          Kernel Syntax
-----------                         -------------
[var ident(x)                       local X in
  [var ident(y)                       local Y in
    [var ident(x)                       local X in
      [nop]]]                             skip end end
  [nop]]                              skip end
```

(15 points)

3. Variable-to-variable binding $\langle X \rangle = \langle Y \rangle$ is represented as `[bind ident(x) ident(y)]`. Implement the merging algorithm inside SAS. [15 points]

4. Value creation. The kernel language statement $\langle X \rangle = \langle v \rangle$ is represented as follows.

   (a) `[bind ident(x) literal(n)]` is the AST representation of $\langle v \rangle$ is a number n. [10 points]

   (b) If $\langle v \rangle$ is a record $a(feature_1 : x_1 \ldots feature_n : x_n)$, then we represent it by

   ```
   [record literal(a)
     [[literal(feature1) ident(x1)]
      ...
      [literal(featuren) ident(xn)]]]
   ```

   [15 points]

   (c) If $\langle v \rangle$ is a procedure proc$\{\$X_1 \ldots X_n\}\langle s \rangle\}$, then we represent it by `[proc [ident(x1) ... ident(xn)] s]`. Note that there may be free variables in the procedure, so procedure values are closures. [30 points]

5. Pattern matching The kernel language statement **case** X **of** p **then** s_1 **else** s_2 **end** has syntax `[match ident(x) p s1 s2]`. Note that p is a record. [20 points].

6. Procedure application. The kernel language statement $\{FX_1 \ldots X_n\}$ is represented in our syntax by `[apply ident(f) ident(x1) ... ident(xn)]`. [15 points]

3 Files

If you plan to use the unification code provided, then you have to create a file called SingleAssignmentStore.oz. This file must provide the functions

1. RetrieveFromSAS
2. BindRefToKeyInSAS
3. BindValueToKeyInSAS

The unification code can be obtained at: Unify.oz. It also needs ProcessRecords.oz
See Dict.oz for example usage of the Oz Dictionary.