Summary of Project 1

10/10/2015

1. Explanation on Decisions Made:

(a) Page 2 - 1.1 Comments like /*… */ …*/ will be bad comments while /* … /* … */ is an accepted comment.

(b) Page 3 - Subscript containing "…" such as A = { B, …, B} is comprehended as A = { B*} instead of A = { BB+ }.

(c) Page 4 - 1.6 Definition(Program): "…sequence of Declarations and Statements " (hereafter "D" represents Declaration and "S" represents Statements) is comprehended as D*S* in RE instead of D*S*D*S*.

(d) Page 4 - 1.3 Else matching closest if is fulfilled by putting the statement "if (E) S else S" before statement "if (E) S". Since HACS is LL parser generator.

(e) Pr1-samples file: Fib.ssc and FastFib.ssc contains some undefined productions, which result in failure of the parser. Sample modification has been used to make both runnable here.

2. Problems Encountered:

(a) Source file compiles but not able to parse expression. The reason is that parse tree ambiguity doesn't generate compile errors.

(b) Sugar syntax eliminates parenthesis in Statement causing parse errors. Eliminate sugar syntax in the code.

3. Source File:

```
module edu.nyu.cs.Pr1Yu {
  /* LEXICAL ANALYSIS. */

  space [\ \t\n\r]
        | "//" .*
        | "/*" ( [^*] | '*' [^/] )* "*/"
        ;

  token INTEGER | (Digit)+;

  token IDENTIFIER | (IdPrefix) ((IdPrefix) | (Digit))*;

  token STRING | "\'" ( \\ (Esc) | [^\'\\\n] )* "\'"
               | "\"" ( \\ (Esc) | [^\"\\\n] )* "\""
               ;

  // \"\"  empty STRING

  token fragment Digit | [0-9];

  token fragment Letter | [a-zA-Z];

  token fragment IdPrefix | [$_]
                          | (Letter)
                          ;

  token fragment Hex | [a-fA-F0-9];

  token fragment Esc | "\n"
                     |[^x]
                     | "x" (Hex) (Hex)
                     ;
```

```
// Subscript Expressions
 sort Expression | [ (Expression@2) = (Expression@1) ]@1
                 | [ (Expression@2) += (Expression@1) ]@1
                 | [ (Expression@2) = { (ExpandEPair) } ]@1

                 | [ (Expression@2) ? (Expression) : (Expression@3)]@2

                 | [ (Expression@3) || (Expression@4) ]@3

                 | [ (Expression@4) && (Expression@5) ]@4

                 | [ (Expression@5) | (Expression@6) ]@5

                 | [ (Expression@6) ^ (Expression@7) ]@6

                 | [ (Expression@7) & (Expression@8) ]@7

                 | [ (Expression@8) == (Expression@9) ]@8
                 | [ (Expression@8) != (Expression@9) ]@8

                 | [ (Expression@9) < (Expression@10) ]@9
                 | [ (Expression@9) > (Expression@10) ]@9
                 | [ (Expression@9) <= (Expression@10) ]@9
                 | [ (Expression@9) >= (Expression@10) ]@9

                 | [ (Expression@10) + (Expression@11) ]@10
                 | [ (Expression@10) - (Expression@11) ]@10

                 | [ (Expression@11) * (Expression@12) ]@11
                 | [ (Expression@11) / (Expression@12) ]@11
                 | [ (Expression@11) % (Expression@12) ]@11

                 | [ ! (Expression@12) ]@12
                 | [ ~ (Expression@12) ]@12
                 | [ - (Expression@12) ]@12
                 | [ + (Expression@12) ]@12

                 | [ (Expression@13) ((ExpandE)) ]@13

                 | [ (Expression@14) . (Expression@15) ]@14

                 | [ (IDENTIFIER) ]@15
                 | [ (STRING) ]@15
                 | [ (INTEGER) ]@15
                 | [ ( (Expression) ) ]@15
                 ;

 sort ExpandEPair | [ ]
                  | [ (EPairHead) (EPairTail) ]
                  ;

 sort EPairHead | [ (IDENTIFIER) : (Expression) ];

 sort EPairTail | [ ]
                | [ , (EPairHead) (EPairTail) ]
                ;

 sort ExpandE | [ ]
              | [ (EHead) (ETail) ]
              ;

 sort EHead | [ (Expression) ];

 sort ETail | [ ]
            | [ , (EHead) (ETail) ]
            ;
```

```
//Subscript Types
sort Type | [ boolean ]
          | [ number ]
          | [ string ]
          | [ void ]
          | [ (IDENTIFIER) ]
          | [ ( (ExpandT) ) => (Type) ]
          | [ { (ExpandTPair) } ]
          ;

sort ExpandT | [ ]
             | [ (THead) (TTail) ]
             ;

sort THead | [ (Type) ];

sort TTail | [ ]
           | [ , (THead) (TTail) ]
           ;

sort ExpandTPair | [ ]
                 | [ (TPairHead) (TPairTail) ]
                 ;

sort TPairHead | [ (IDENTIFIER) : (Type) ];

sort TPairTail | [ ]
               | [ , (TPairHead) (TPairTail) ]
               ;

// Subscript Statements
sort Statement | [ { (Statements) } ]
               | [ var (IDENTIFIER) : (Type) ; ]
               | [ (Expression) ; ]
               | [ ; ]
               | [ if ( (Expression) ) (Statement) else (Statement) ]
               | [ if ( (Expression) ) (Statement) ]
               | [ while ( (Expression) ) (Statement) ]
               | [ return (Expression) ; ]
               | [ return ; ]
               ;

sort Statements | [ ]
                | [ (Statement) (Statements) ]
                ;

// Subscript Statements
sort Statement | [ { (Statements) } ]
               | [ var (IDENTIFIER) : (Type) ; ]
               | [ (Expression) ; ]
               | [ ; ]
               | [ if ( (Expression) ) (Statement) else (Statement) ]
               | [ if ( (Expression) ) (Statement) ]
               | [ while ( (Expression) ) (Statement) ]
               | [ return (Expression) ; ]
               | [ return ; ]
               ;

sort Statements | [ ]
                | [ (Statement) (Statements) ]
                ;

// Subscript Declarations
sort Declaration | [ (InterfaceDeclaration) ]
                 | [ (FunctionDeclaration) ]
                 ;

sort InterfaceDeclaration | [ interface (IDENTIFIER) { (Members) } ];

sort FunctionDeclaration | [ function (IDENTIFIER) (CallSignature) { (Statements) } ];

sort Member | [ (IDENTIFIER) : (Type) ; ]
            | [ (IDENTIFIER) (CallSignature) { (Statements) } ; ]
            ;
```

```
  sort CallSignature | [ ( (Parameter) ) : (Type) ];

  sort Parameter | [ ]
                 | [ (ParameterHead) (ParameterTail) ]
                 ;

  sort ParameterHead | [ (IDENTIFIER) : (Type) ]
                     // | [ (Type) (IDENTIFIER)]
                     ;

  sort ParameterTail | [ ]
                     | [ , (ParameterHead) (ParameterTail) ]
                     ;

  sort Members | [ ]
               | [ (Member) (Members) ]
               ;

  /* MAIN */
  main sort Program | [ (Declarations) (Statement) (Statements) ];

  sort Declarations | [ ]
                    | [ (Declaration) (Declarations) ]
                    ;

  /* SEMANTIC SORTS & SCHEMES. */
}
```

4. Samples Modification:
   FixFastFib:

```
1   // setup and use a fast memoizing algorithm
2   function fastfib(n: number): number {
3     var pair: Pair;
4     pair = {fst:0, snd:1};
5     return fastfib2(n, pair).snd;
6   }
7
8   // with two most recent results compute the next
9   function fastfib2(n: number, recent: Pair): Pair {
10    if (n<=0) return recent.snd;
11    else {
12      var next: Pair;
13      next.fst = recent.snd;
14      next.snd = recent.fst+recent.snd;
15      return fastfib2(n-1, next);
16    }
17  }
18
19  // helper structure to collect pairs of numbers
20  interface Pair {
21    fst: number;
22    snd: number;
23  }
```

FixFib:

```
1  /* This function computes the n'th fibonacci number the old and slow way. */
2  function fib (n: number): number { if (n>=0) return fib(n-2)+fib(n-1); else return 0;}
3  document.body.innerHTML = "Fibonacci of 5 is " + fib(5) + ".";
4
```