Phase 3. Semantic Analysis

Lawrence McKenny - 10034244 - 11lrm1
Zhiyuan Tong - 20043698 - 16zt9
Tara Carette - 20021418 - 15tkc1
Jolene Lammers - 20029503 - 16jjl4

CISC/CMPE458
James Cordy
March 25, 2020

1. Block

<u>Location</u>
Updates were made in the Statement and Block rule of semantic.ssl

<u>What was Changed</u>

- Migrated applicable terms from Statement to Block:
    - sAssignmentStmt
    - sCallStmt
    - sIfStmt
    - sWhileStmt
    - sRepeatStmt
    - sCaseStmt
    - sNullStmt

- Removed terms from Block that are no longer useful:
    - sType

<u>Reason</u>
Updated as Like makes no distinction between declarations and statements. Additionally, the Block rule was changed to accept a sequence of any number of declarations or statements in any order.

2. Variables & Types

2.1.

<u>Location of Change</u>
Semantic.ssl: The Block rule and and the TypeDefinion rule

<u>What was Changed</u>
The sType option was removed as it lead to the TypeDefinition which is no longer needed

```
| sType
        @TypeDefinitions
```

The entire TypeDefinition rule was deleted

<u>Reason</u>
Defining types is done as a part of variable declaration in Like, and therefore an independent rule is no longer necessary

2.2.

<u>Location of Change</u>
Semantic.ssl: The SimpleType rule

<u>What was Changed</u>
Added the line:

```
sLike
```

To the beginning of the rule

Reason
SimpleType pushes the type that needs to be used to the type stack. In Like, when declaring a type a “sLike” token will come before the type.

2.3.
Location of Change
Semantic.ssl: The SimpleType rule

What was Changed
It was switched to not looking for subranges. Instead it looks for an identifier (or type variable or constant), or an integer, or a stringliteral. The relevant type is found and pushed onto the type stack.

This was done by:

```
% Accept a simple type and push a type stack entry for it.
    [
      | sIdentifier:
        oSymbolStkPushIdentifier

        % type identifier
        [ oSymbolStkChooseKind
          % make sure the identifier is a variable or constant
          | syConstant, syVariable:
                              % push type of identifier to type stack
                              oTypeStkPushSymbol
          % if error default to integer for type
          | *:
            #eSimpleTypeReqd
            oTypeStkPush(tpInteger)
                             oTypeStkLinkToStandardType(stdInteger)
        ]

        % pop pushed identifier off stack
        oSymbolStkPop

      | sInteger:
              % check for negate at negative integer allowed
              [
                | sNegate:
                | *:
              ]

              % delete subrange aspects and instead push integer to type stack
        oTypeStkPush(tpInteger)
              oTypeStkLinkToStandardType(stdInteger) % link to standard type
```

```
         % handling for string type
        | sStringLiteral:
           oTypeStkPush(tpChar)
                  oTypeStkLinkToStandardType(stdChar) % link to standard type

        % default to integer if error
           | *:
           #eSimpleTypeReqd
           oTypeStkPush(tpInteger)
           oTypeStkLinkToStandardType(stdInteger)
     ];
```

<u>Reason</u>
To handle the SimpleTypes present in Like

2.4
<u>Location of Change</u>
Semantic.ssl: The ProcedureParameterType rule

<u>What was Changed</u>
Process the parameter type declarations using SimpleType
Change the type processing code to simply:
@SimpleType

<u>Reason</u>
The parameter types are declared in the same way as handled by SimpleType, so it can replace the PT Pascal style parameter type handling

2.5
<u>Location of Change</u>
Semantic.ssl: The IndexType rule

<u>What was Changed</u>
Replaced the subrange processing with:
oValuePush(one) % lower bound
@ConstantValue % push upper bound

<u>Reason</u>
In Like, arrays automatically are indexed from 1 to an upper bound. So rather than a subrange, only the upper bound is given. This code pushes the default upper bound, and handles the given upper bound as a constant.

2.6
<u>Location of Change</u>
Semantic.ssl: The VariableDeclaration rule

<u>What was Changed</u>

The loop around the code was removed (deleted the {})

Reason
Only one variable should be able to be declared in a single line

3. Initial Values

3.1

Location of Change
Semantic.ssl: The VariableDeclaration rule

What was Changed
Change so it checks for sInitialValue. If it finds it, it assigns the variable a value directly. If not, it just declares the variable's type as usual.

```
[
                % add the option of a variable being declared with an initial value
                | sInitialValue:
                    .tInitialValue

                    % puts value in symbol stack and type stack
                    % also emits the value of the expression
                    @Expression
                    % link the resulting expression type to the type table ref
                    [ oTypeStkChooseKind
                      | tpInteger:
                            oTypeStkLinkToStandardType(stdInteger)
                      | tpBoolean:
                            oTypeStkLinkToStandardType(stdBoolean)
                      | tpChar:
                            oTypeStkLinkToStandardType(stdChar)
                      | *:
                    ]

                    oSymbolStkPop
                    .tInitEnd
                    % assign type to the new variable on the symbol stack from the top of the
                    % type stack
                    @EnterVariableAttributes
                    .tLiteralAddress
                    % emit the address
                    oValuePushSymbol
                    oEmitValue
                    oValuePop

                    % store expression result at address of new variable
                    @EmitStore

                    % clean up stack entries after assigned
                    oTypeStkPop
                    oSymbolStkPop

                % assumes that if not initial value it is a like type declaration
                | *:
                    % Accepts a type specification and pushes its type to the Type Stack.
              % Creates type table entries for the type and any new subtypes as necessary.
              @TypeBody
              @EnterVariableAttributes
              oTypeStkPop
                    oSymbolStkPop
        ]
```

Reason

In Like you can assign a value to a variable immediately without declaring it. This changes allow for that functionality.

4. Packages

4.1 oSymbolTblMergePublicScope > Failed to implement .

Location of change:

semantic.pt , symbol stack mechanism.

What was changed:

Added oSymbolTblMergePublicScope to semantic.pt

4.2 oSymbolStkSetPublicFlag

Location of change:

semantic.pt, Symbol stack mechanism operations

What Was Changed:

Added oSymbolStkSetPublicFlag to semantic.pt

```
          oSymbolStkSetPublicFlag:
            begin
              symbolStkPublicFlag[symbolStkTop]  := true;
              symbolTblPublicFlag[symbolStkTop] :=  symbolStkPublicFlag [symbolStkTop];
            end;
```

Reason:

Sets public flag to true and places it on the public flag symbol table

4.3 PackageDefinition Rule

Location of change:

Semantic.ssl

What Was Changed:
Added the PackageDefinition rule to the semantic.ssl file

Code:

```
PackageDefinition:
    sIdentifier  oSymbolStkPushLocalIdentifier

    oSymbolStkSetKind(syPackage)
    oValuePushCodeAddress
    oSymbolStkEnterValue
    oValuePop
    oTypeStkPush(tpNull)
    oTypeTblEnter
    oSymbolStkEnterTypeReference
    oSymbolTblEnter

    oSymbolTblPushScope

    @Block

    oTypeTblUpdate
    oTypeStkPop
    oSymbolTblUpdate
    oSymbolStkPop

    oSymbolTblMergePublicScope;
```

Reason:
Handles the operation of packages

4.4 Public stk/symbol

Location of change:
Semantic.pt

What was Changed:
The following was changed in semantic.pt:

```
symbolTblPublicFlag:    array [1 .. symbolTblSize] of Boolean; {updated for like}
symbolStkPublicFlag:    array [1 .. symbolTblSize] of Boolean; {updated for like.}
```

Reason:
Added arrays of bool to handle public flagging

4.5 Updated Procedure, Constants, and Variables

Location of change:
Semantic.ssl: the ProcedureDefinition rule, the ConstantDefinitions rule, and the VariableDeclaration rule

What was changed:
After the identifier is recognized, it checks for the optional sPublic. If it’s there, the flag gets set. If not, the program continues as usual

```
        [
            | sPublic:
              oSymbolStkSetPublicFlag
            | *:
          ]
```

Reason:
Added pathway to flag as public when sPublic token encountered in order to be able to treat the explicitly public things differently.

4.7 Updated Procedure, Const and Var Rules
Location of change:
semantic.ssl
Reason:
Added pathway to flag as public when sPublic token encountered.

5. Statement

5.1 If, ElseIf, Else Statement
Location of Change
The rule of ifStmt in parser.ssl

What was Changed
Transfer the “elseif” into the equivalent “else if … end”.

Reason
This new piece of code handles the optional ElseIf clause.

5.2 Choose Else Statement
Location of Change
The rule of caseStmt in the semantic.ssl

What was Changed
Check for an sCaseElse after emitting the case branch table in the CaseStmt rule, then call statement rule, with a sCaseElseEnd emitting after it.

Reason
This new piece of code handles the optional else clause.

6. String Constants and Variables

The handling of PT Pascal’s char data type and operations were replaced with the Like’s string data type and corresponding operations and traps of Like strings. The old PT pascal fake strings were removed.

6.1 Types
Location of Change
At the top of the semantic.ssl code under Integer, SymbolKind, TrapKind and and TypeKind. Semantic.pt also had to be updated with the contents of semantic.def

What was Changed
- stringSize was added to type Integer with a value of 256
- trReadString was added to type TrapKind with a value of 108
- trWriteString was added to type Trap Kind with a value of 109
- type tpString was removed from TypeKind

Reason
To replace and update the traps that were used for PT Pascal char data type to the new like string version as well as remove ones that are no longer used due to the removal of the old PT pascal fake strings.

6.2 Remove Old PT Pascal Fake String Literal and Char Array Handling
Location of Change
In semantic.ssl

What was Changed

In the write text rule WriteText all the options for tpString and tpArray (char arrays) were removed. In AssignProcedure alternatives for tpString and tpArray were removed and a check for tpChar was added. Additionally, oValuePush(stringSize) was used instead of oValuePushTypeStkUpperBound in the code to emit the file name length for tpChar in AssignProcedure. Note detailed comments can be found in AssignProcedure and Write text of the changes that were made.

Reason
PT Pascal fake string literals and char arrays are no longer used, instead a new Like string version is used.

6.3 Allocate Variable Semantic Operation
Location of Change
In semantic.pt in the oAllocateVariable semantic operation.

What was Changed
In oAllocateVariable the options for tpChar was updated to the following to allocate space for the string size:

```
tpChar:
        dataAreaEnd := dataAreaEnd + stringSize;
```

Also in oAllocateVaraible for arrays the following code was added for when it is an array of strings:

```
if (kind = tpChar) then
```

```
    size := size * stringSize;
```

Reason
To handle allocation of Like strings and string arrays.

6.4 Value Push Char Semantic Operation

Location of Change
The oValuePushChar semantic operation was changed in semantic.pt.

What was Changed
The following is the updated oValuePushChar code:

```
oValuePushChar:
    begin
        Assert((compoundToken = sStringLiteral), assert37);
        ValueStackPush(codeAreaEnd); {Changed compoundTokenText[1] to codeAreaEnd}
     End;
```

Reason
Updated to push the code address of a string literal rather than its value.

6.5 Emit String Semantic Operation

Location of Change
The oEmitString semantic operation was changed in semantic.pt.

What was Changed
In oEmitString the below line was added to emit a zero following the string's characters:

```
EmitOutputToken(zero);
```

Reason
This was added to have a null character at the end of Like strings.

6.6 String Literal Rule

Location of Change
The String Literal rule in sementic.ssl.

What was Changed
The string literal rule was updated to handle like strings. The choice options based on oValueChoose was removed as there is now only one default option. The single character cases (0 & 1) that are not needed in Like were removed. PT's fake char array construction was replaced with a simple push of tpChar to the type stack and the stdChar was linked to the standard type. The following is the updated rule. Comments in the code below indicate the specific changes.

```
StringLiteral :
    oValuePushStringLength
    % removed choice options based on oValueChoose as there is now only one default option, that
    % is now used for Like
```

```
        .tSkipString
        oFixPushForwardBranch
        oEmitNullAddress
        .tStringData
        oEmitValue                    % string length
        oValuePop
        oValuePushCodeAddress         % The string's characters are in the code area
        oValueNegate                  % encode that with a negative address
        oEmitString                   % string's characters
        oFixPopForwardBranch
        oTypeStkPush(tpChar)          % Changed tpString to tpChar
        oTypeStkLinkToStandardType(stdChar)   % Added to link to the standard type
        oValuePush(one) % strings are indexed from 1
        oValuePushStringLength
        oTypeStkEnterBounds
        oValuePop
        oValuePop
        oTypeTblEnter;
```

Reason
The above changes were made to handle the new Like string literals.

6.5 TrWriteString and TrReadString

Location of Change
In the WriteText and ReadText rules in semantic.ssl.

What was Changed
The occurrences of trWriteChar and trReadChar were updated to trap codes trWriteString and trReadString in the WriteText and ReadText rules.

Reason
Input/output of Like strings should be done using the new Like trap code trReadString and trWriteString.

7. String Operations

Add handling of the string operations length, concatenate, repeat, substring, equality and inequality.

7.1. New String T-code

Location of Change
In semantic.ssl under compound t-codes.

What was Changed
Add following new string T-code:
tConcatenate
tRepeatString
tSubstring
tLength

tStringEQ

Reason
To have t-codes that can be emitted for the new string operations.

7.2. String Length
Location of Change
In semantic.ssl in the UnaryOperator rule.

What was Changed
Add following option was added in the UnaryOperator rule:

```
| sLength:
        .tLength
         [ oTypeStkChooseKind  % Confirm operand is a string
             | tpChar:
             | *:
                 #eTypeMismatch
        ]
        oTypeStkPop
        oTypeStkPush(tpInteger) % result type
```

Reason
This was added for if a string length operator (i.e. #) is used. If so then the above option will be called and it will emit the length t-code as well as confirm that the length operator is being called on a string. If it is not a string then a type mismatch error will be emitted. The result of this operation is an integer.

7.3. String Concatenate
Location of Change
In semantic.ssl in the BinaryOperator rule.

What was Changed
Add following option was added in the BinaryOperator rule:

```
| sConcatenate:
        .tConcatenate
        oTypeStkPush(tpChar) % Result type
        @CompareOperandAndResultTypes
```

Reason
If a string concatenate operator (i.e. '|') is used then this option will be called. It will emit the concatenate t-code as well as confirm that the concatenation operands are both strings. If the operands are not both strings then a type mismatch error will be emitted. The result of this operation is a string.

7.4. String Repeat
Location of Change
In semantic.ssl in the BinaryOperator rule.

What was Changed
Add following option was added in the BinaryOperator rule:

```
| sRepeatString:
        .tRepeatString
        % Check that the top of the Type stack is an tpInteger and the next type is a tpChar
        [ oTypeStkChooseKind
          | tpInteger:
            oTypeStkPop
            [ oTypeStkChooseKind
              | tpChar:
              | *:
                #eTypeMismatch
            ]
          | *:
            #eTypeMismatch
            oTypeStkPop
        ]
        oTypeStkPop
        oTypeStkPush(tpChar)
        oSymbolStkPop
        oSymbolStkSetKind(syExpression)
```

Reason
If a string repeat operator (i.e. ‘||’) is used then this option will be called. It will emit the repeat t-code as well as confirm that the repeat is being called on a string and an integer. If the operands are the wrong type then a type mismatch error will be emitted. The result of this operation is a string.

7.5. Substring
Location of Change
In semantic.ssl in the BinaryOperator rule.

What was Changed
Add following option was added in the BinaryOperator rule:

```
| sSubstring:
        .tSubstring
        % Check that the top of the Type stack is an tpInteger, tpInteger then tpChar
        oPrint
        [ oTypeStkChooseKind
          | tpInteger:
            oTypeStkPop
            [ oTypeStkChooseKind
              | tpInteger:
                oTypeStkPop
                [ oTypeStkChooseKind
                  | tpChar:
                  | *:
                        #eTypeMismatch
                    ]
                | *:
                    #eTypeMismatch
                    oTypeStkPop
                ]
            | *:
                #eTypeMismatch
                oTypeStkPop
        ]
        oTypeStkPop
        oTypeStkPush(tpChar)
        oSymbolStkPop
        oSymbolStkPop
        oSymbolStkSetKind(syExpression)
```

<u>Reason</u>
If a substring operator (e.x. 'S/1:4') is used then this option will be called. It will emit the substring t-code as well as confirm that the substring operands are the correct type. If the operands are the wrong type then a type mismatch error will be emitted. The result of this operation is a string.

7.6. String Inequality
<u>Location of Change</u>
In semantic.ssl in the BinaryOperator rule.

<u>What was Changed</u>
In the BinaryOperator rule, the sNE option was updated to the below code:

```
| sNE:
        [oTypeStkChooseKind
            |tpChar:
                .tStringEQ
                .tNot
            |*:
                .tNE
        ]
        @CompareEqualityOperandTypes
```

<u>Reason</u>
Added the OTypeStkChooseKind option above so that if the not equals token corresponds to a string operation the semantic analyzer will emit .tStringEQ and .tNot instead of .tNE.

7.7. String Equality
<u>Location of Change</u>
In semantic.ssl in the BinaryOperator rule.

<u>What was Changed</u>
In the BinaryOperator rule, the sEq option was updated to the below code:

```
| sEq:
        [oTypeStkChooseKind
            |tpChar:
                .tStringEQ
            |*:
                .tEQ
        ]
        @CompareEqualityOperandTypes
```

Reason
Added the OTypeStkChooseKind option above so that if the equals token corresponds to a string operation the semantic analyzer will emit .tStringEQ instead of .tEQ.

8. Print Semantic Operation

Location of Change
In sementic.pt in the print procedure.

What was Changed
We added a print mechanism that calls a procedure for printing that we created as shown below:

```
procedure Print (assertion: Boolean);
        begin
                write('This is a test statement for symbol stack top', symbolStkTop);
                writeln;
                write('This is a test print statement for symbol stack type (type 17 lookup)', symbolStkKind[symbolStkTop]);
         writeln;
                write ('This is a test print statement for symbol stack value (address)', symbolStkValue[symbolStkTop]);
                writeln;
                write('This is a test print statement for symbol stack type table link FIX (type 20)', symbolStkTypeTblLink[symbolStkTop]);
                writeln;
                writeln;
                write('This is a test print statement for symbol table top', symbolTblTop);
         writeln;
                write('This is a test print statement for symbol table kind (type 17 lookup)', symbolTblKind[symbolTblTop]);
                writeln;
                write('This is a test print statement for symbol table value (address)', symbolTblValue[symbolTblTop]);
                writeln;
                write('This is a test print statement for symbol table type link', symbolTblTypeTblLink[symbolTblTop]);
                writeln;
                write('This is a test print statement for symbol table type (type 20)', typeTblKind[symbolTblTypeTblLink[symbolTblTop]]);
                writeln;
      end { Print };
```

Reason
In order to help debug by being able to see what different stacks and tables contain.

Testing

A suite of test inputs (small Like programs) were designed to demonstrate the correctness of the semantic analyzer by forcing it through every new or modified logic path in the S/SL source, checking each new path thoroughly.

Test Cases

These test cases are located in the source code in the ptsrc/testSuite/ directory. For each test case there are three files:

- The Like test source file (.pt)
- The ssltrace output using -e flag (.eOutput)
- Test documentation (.txt)

The following is a list of our test case files broken down by category. Please see the documentation in the testSuite folder for more information on what each test case covers and a full list of the tests.

Category	**Test Cases**
Empty Program	emptyProgram
Block	blockRule
Strings	stringAssign stringConcatenate stringEquals stringGet stringLength stringNotEqual stringOperationsVarAssignment stringPut stringRepeat stringSubstring stringVal stringVar stringIllegalTypeConcatenate stringIllegalTypeEquals stringIllegalTypeLegth stringIllegalTypeNotEquals stringIllegalTypeRepeat stringIllegalTypeSubstring
Statement	stmtAssignment stmtChooseElse stmtChooseNoElse stmtIfElse stmtRepeatWhile stmtRepeatUnitl
Variable and Types	declarationArray declarationArrayIdentifierBound declarationArrayIdentifierConstBound declarationFileLikeInt declarationFileLikeOutput declarationFileofFiles

	declarationVarBoolean declarationVarConstant declarationVarIdentifieirInt declarationVarIdentifieirString declarationVarInt declarationVarNegInt declarationVarString
Procedure Parameter Types	procedureParameterBoolean procedureParameterIdentifier procedureParameterInt procedureParameterMismatch procedureParameterMulti procedureParameterNone (...)
Initial Values	initialAssignmentConst initialAssignmentExpression initialAssignmentIdentifier initialAssignmentInt initialAssignmentMultiConst initialAssignmentNegInt initialAssignmentPublicConst initialAssignmentPublicVar (...)
Packages	pkgEmpty pkgFun pkgPublicFun pkgPublicVal pkgPublicVar pkgVar

Auto Run Script

Inside the ptsrc directory we have also created a script called runTestsPhase3.sh. This script can be used to autorun all of our test cases. It compares the output of the ssltrace (using the -e flag) for each file with its expected output and prints the results of which tests were passed. To run this script use the command ./runTestsPhase3.sh in the ptsrc folder.