Phase 3. Semantic Analysis

The Compildres

Arthur Margulies
Harold Treen
Paul Wells

March 18th, 2015

1 - Updating Input Tokens

We updated the input tokens for semantic.ssl to be the same as the output tokens in parser.ssl. Below are the changes in semantic.ssl

```
Input :
        sIdentifier
        firstInputToken = sIdentifier
        sInteger
        sLiteral
-------------------------------------------------------------------------------------
        sParmBegin
        sParmEnd
        sConst
        sType
        sVar
        sProcedure
+-- 14 lines: sBegin-----------------------------------------------------------------
        sCallStmt
        sFieldWidth
        sIfStmt
        sThen
        sElse
        sWhileStmt
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
        sEq
        sNE
        sLT
        sLE
        sGT
        sGE
+--  5 lines: sAdd-------------------------------------------------------------------
        sInfixOr
        sOr
        sInfixAnd
        sAnd
        sNot
        sNewLine
        % Added tokens
        sPublic
        sDefault
        sExtern
        sModule
        sLoopStmt
        sLoopBreakIf
        sLoopEnd
        sSubstring
```

```
Input :
        sIdentifier
        firstInputToken = sIdentifier
        sInteger
        sLiteral
        sProgram
        sParmBegin
        sParmEnd
        sConst
        sType
        sVar
        sProcedure
+-- 14 lines: sBegin--------------------
        sCallStmt
        sFieldWidth
        sIfStmt
        sThen
        sElse
        sWhileStmt
        sRepeatStmt
        sRepeatEnd
        sEq
        sNE
        sLT
        sLE
        sGT
        sGE
+--  5 lines: sAdd----------------------
        sInfixOr
        sOr
        sInfixAnd
        sAnd
        sNot
        sNewLine
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
```

2 - Extending the T-Code Machine Model

We added all of the new required tCodes and removed the ones that no longer applied. Below are the non-compound tCodes that were added/removed in semantic.ssl:

```
tVarParm
tFetchAddress
tFetchInteger
tFetchBoolean
tAssignBegin
tAssignAddress
tAssignInteger
tAssignBoolean
tStoreParmAddress
tStoreParmInteger
tStoreParmBoolean
tSubscriptBegin
tSubscriptAddress
tSubscriptInteger
tSubscriptBoolean
tArrayDescriptor
tFileDescriptor
tIfBegin
tIfEnd
tCaseBegin
+--  5 lines: tWhileBegin----------------------------------------
tProcedureEnd
tWriteBegin
tReadBegin
tTrapBegin
tWriteEnd
tReadEnd
% Added non-compound T-Codes
tFetchString
tAssignString
tStoreParmString
tSubscriptString
tConcatenate
tSubstring
tLength
tStringEqual
tLoopBegin
tLoopBreakIf
tCaseDefault
```

```
tVarParm
tFetchAddress
tFetchInteger
tFetchChar
tFetchBoolean
tAssignBegin
tAssignAddress
tAssignInteger
tAssignChar
tAssignBoolean
tStoreParmAddress
tStoreParmInteger
tStoreParmChar
tStoreParmBoolean
tSubscriptBegin
tSubscriptAddress
tSubscriptInteger
tSubscriptChar
tSubscriptBoolean
tArrayDescriptor
tFileDescriptor
tIfBegin
tIfEnd
tCaseBegin
+--  5 lines: tWhileBegin----------------------------------------
tProcedureEnd
tWriteBegin
tReadBegin
tTrapBegin
tWriteEnd
tReadEnd
```

Here are the changes to the compound tCodes in semantic.ssl:

```
% Compound T-codes are those that take operands
tLiteralAddress
firstCompoundOutputToken = tLiteralAddress
tLiteralInteger
tLiteralBoolean
tLiteralString
tStringDescriptor
tSkipString
tIfThen
tIfMerge
tCaseSelect
tCaseMerge
tCaseEnd
tCaseElse
tWhileTest
tWhileEnd
tRepeatTest
tSkipProc
tCallEnd
tLineNumber
% Added Compound T-Codes
tLoopTest
tLoopEnd
%%
```

```
% Compound T-codes are those that take operands
tLiteralAddress
firstCompoundOutputToken = tLiteralAddress
tLiteralInteger
tLiteralChar
tLiteralBoolean
tLiteralString
tStringDescriptor
tSkipString
tIfThen
tIfMerge
tCaseSelect
tCaseMerge
tCaseEnd
tWhileTest
tWhileEnd
tRepeatTest
tSkipProc
tCallEnd
tLineNumber
```

These changes changed semantic.def and we updated semantic.pt with those changes. (Code omitted)

3 - Adding Modules

In order to add modules we added two new semantic operations oSymbolTblStripScope and oSymbolTblMergeScope to semantic.pt. oSymbolTblStripScope hides all the symbols in the current scope except for public modules. It's similar to oSymbolTblPopScope except that we don't pop the symbol table display and we don't change the top of the symbol table. We added a new kind of symbol called syPublicProcedure and we skip symbols of that type.

Here's the code of oSymbolTblStripScope in semantic.pt:

```
oSymbolTblStripScope:
    { Hide all symbols in current scope except for public proced

    begin
        Assert((lexicLevelStackTop >= 1), assert31);
        i := symbolTblTop;
        { Set the identifier table pointer to the identifier
          entry in the closest enclosing scope if there is
          one }
        { Iterate through every symbol in current scope }
        while i > symbolTblDisplay[lexicLevelStackTop] do
            begin
                    { Don't hide public procedures }
                    if(symbolTblKind[i] <> syPublicProcedure) th
                    begin
                        link := symbolTblIdentLink[i];

                        if link <> null then
                            { This is not a dummy identifier
                              generated by the parser's syntax
                              error recovery procedure.       }
                            begin
                                while link > 0 do
                                    link := symbolTblIdentLink[l
                                identSymbolTblRef[-link] :=
                                    symbolTblIdentLink[i];
                            end;
                    end;

                i := i - 1
            end;

    end;
```

In addition to syPublicProcedure we had to add a symbol kind for modules called syModule:

```
syProcedure = 4;                 syProcedure = 4;
syPublicProcedure = 5;           syFunction = 5;
syFunction = 6;                  syExternal = 6;
syExternal = 7;                  syExpression = 7;
syExpression = 8;                syUndefined = 8;
syUndefined = 9;
syModule = 10;
```

We also had to modify several assertions that previously required a symbol to be of kind syProcedure so that it also accepted syPublicProcedure. We've shown one such assertion below:

```
Assert((symbolStkKind[symbolStkTop] = syProcedure) or          Assert((symbolStkKind[symbolStkTop] = syProcedure) or
    (symbolStkKind[symbolStkTop] = syFunction) or                  (symbolStkKind[symbolStkTop] = syFunction),
    (symbolStkKind[symbolStkTop] = syPublicProcedure)
```

Here is the code in AllocateVar for assigning the kind of procedure symbols to be either syPublicProcedure or syProcedure:

```
[
    | sPublic:
        oSymbolStkSetKind(syPublicProcedure)
    | *:
        oSymbolStkSetKind(syProcedure)
]
```

```
oSymbolStkSetKind(syProcedure)
```

Below is the code for oSybmolTblMergeScope in semantic.pt. It simply pops the lexical level stack and changes the top of the type table but not the symbol table. This removes any types declared in modules but leaves the symbols (of which only the public procedures are visible)

```
oSymbolTblMergeScope:
begin
        { No local type definitions need be saved since
      parameters must be declared with global type
      names. }
    typeTblTop := typeTblDisplay[lexicLevelStackTop];
    lexicLevelStackTop := lexicLevelStackTop - 1;
end;
```

We also created a rule for ModuleDeclarations in semantic.ssl and added a choice for sModule in the Block rule:

```
            | sType:
                @TypeDefinitions
            | sVar:
                @VariableDeclarations
                | sModule:
                @ModuleDeclaration % Added module declaration
            | *:
                >
        ]}
        sBegin
        @BeginStmt;

ModuleDeclaration :
        % Require module name
        [
            | sIdentifier:
            | *:
                #eUndefinedIdentifier
        ]

        oSymbolStkPushLocalIdentifier
        oSymbolStkSetKind(syModule)
        oSymbolTblEnter
        oSymbolStkPop

        oSymbolTblPushScope
        @Block

        % Pop module scope, leaving public procedures
        %  visible
        oSymbolTblStripScope
        oSymbolTblMergeScope;
```

4 - Creating the String Type

Changes to Semantic.ssl

First, the supported input and output tokens needed to be updated.

Tokens were added to the input:

```
sSubstring
sLength
```

And tokens were added to the output:

```
tFetchString
tAssignString
tStoreParmString
tSubscriptString
tConcatenate
tSubstring
tLength
tStringEqual
```

Char related tokens (tFetchChar, tAssignChar, tStoreParmChar, tSubscriptChar, tLiteralChar) were removed.

Error tokens were updated to also refer to strings instead of chars (eCharExpnReqd, eCharFileVarReqd). Other types were also updated, such as pidCha, stdChar, trWriteChar, trReadChar and tpChar. stringSize was also added to define the memory for a string.

A new ternary operator rule was added to expression analysis to catch substring operations.

```
TernaryOperator:
[
    | sSubstring:
        .tSubstring
        oTypeStkPush(tpString) % Push substring
        @CompareTernaryOperandAndResultTypes % Verify results
    | *:
];
```

```
CompareTernaryOperandAndResultTypes :
[ oTypeStkChooseKind
    |tpString : % String
        oTypeStkSwap
        [ oTypeStkChooseKind
            |tpInteger :  % Lower bound
                oTypeStkPop
                oTypeStkSwap
                [oTypeStkChooseKind
                    |tpInteger: % Upper bound
                        oTypeStkPop
                        oTypeStkSwap
                        [oTypeStkChooseKind
                            |tpString:
                            oTypeStkPop
                            |*:
                            #eOperandOperatorTypeMismatch % Err
                        ]
                    |*:
                        #eOperandOperatorTypeMismatch
                ]
            |* :
            #eOperandOperatorTypeMismatch
        ]
    |* :
        #eOperandOperatorTypeMismatch

]
oSymbolStkPop
oSymbolStkPop
oSymbolStkSetKind(syExpression);
```

Since drift strings are first class citizens, the analysis of literal operands was greatly simplified.

```
    | sLiteral:
            oValuePush(one) % implicit lower bound of strin
            oValuePushStringLength
            oSymbolStkPush(syExpression)
            oTypeStkPush(tpString)
            .tLiteralString
            oEmitValue % string length
            oEmitString
            oValuePop
            oValuePop
```

To support the sLength operation, a new unary operation needed to be added.

```
CompareAndSwapTypesLength :
[ oTypeStkChooseKind
    |tpInteger:
    oTypeStkSwap
    [oTypeStkChooseKind
        |tpString:
        |*:
        #eTypeMismatch
    ]
    |* :
    #eTypeMismatch
];
```

```
        | sLength:
        .tLength
        [ oTypeStkChooseKind
            | tpString:
                oTypeStkPush(tpInteger)
                @CompareAndSwapTypesLength
            | *:
                #eTypeMismatch
        ]
```

Strings support concatenation and comparison for equality. The binary operator rule was updated to support these operations.

```
BinaryOperator :
    % Choice should be ordered by frequency of occurrence of al
    % Could make this a cycle (comment applies to UnaryOperator
    % but that would probably be less efficient on average.
    [
        | sAdd:
        .tAdd
        oTypeStkPush(tpInteger) % result type
        @CompareOperandAndResultTypes
        | sSubtract:
        .tSubtract
        oTypeStkPush(tpInteger)
        @CompareOperandAndResultTypes
        | sInfixAnd:    % marker without semantic significance
        .tInfixAnd
        | sAnd:
        .tAnd
        oTypeStkPush(tpBoolean)
        @CompareOperandAndResultTypes
        | sInfixOr:
        .tInfixOr
        | sOr:
        .tOr
        oTypeStkPush(tpBoolean)
        @CompareOperandAndResultTypes
        | sEq:
        .tEq
        @CompareRelationalOperandTypes
        | sNE:
        .tNE
        @CompareRelationalOperandTypes
        | sGT:
        .tGT
        @CompareRelationalOperandTypes
        | sGE:
        .tGE
        @CompareRelationalOperandTypes
        | sLT:
        .tLT
        @CompareRelationalOperandTypes
        | sLE:
        .tLE
        @CompareRelationalOperandTypes
```

```
    [
        | sAdd:
            [ oTypeStkChooseKind
                | tpInteger:
                    .tAdd
                        oTypeStkPush(tpInteger) % result type
                | tpString:
                    .tConcatenate
                    oTypeStkPush(tpString)
            ]
                @CompareOperandAndResultTypes
        | sSubtract:
        .tSubtract
        oTypeStkPush(tpInteger)
        @CompareOperandAndResultTypes
        | sInfixAnd:    % marker without semantic significance
        .tInfixAnd
        | sAnd:
        .tAnd
        oTypeStkPush(tpBoolean)
        @CompareOperandAndResultTypes
        | sInfixOr:
        .tInfixOr
        | sOr:
        .tOr
        oTypeStkPush(tpBoolean)
        @CompareOperandAndResultTypes
        | sEq:
                [ oTypeStkChooseKind
                    | tpInteger:
                        .tEq
                    | tpString:
                        .tStringEqual
                ]
                @CompareRelationalOperandTypes
        | sNE:
                [ oTypeStkChooseKind
                    | tpInteger:
                        .tNE
                    | tpString:
                        .tStringEqual
                        .tNot
                ]
        @CompareRelationalOperandTypes
```

Many other places were updated to look for String tokens instead of Char tokens or call string rules instead of char rules.

Changes to Semantic.pt

All token changes in Semantic.ssl also needed to be added to Semantic.pt after being generated into semantic.def.

When allocating a string variable, more memory needs to be allocated. The amount is calculated using the new stringSize constant.

```
dataAreaEnd := dataAreaEnd + wordSize;
tpChar, tpBoolean:
dataAreaEnd := dataAreaEnd + byteSize;
tpArray, tpPackedArray:
begin
    size := typeStkUpperBound[typeStkTop]
    - typeStkLowerBound[typeStkTop] + 1;
```

```
dataAreaEnd := dataAreaEnd + wordSize;
tpBoolean:
dataAreaEnd := dataAreaEnd + byteSize;
tpString:
dataAreaEnd := dataAreaEnd + stringSize;
tpArray, tpPackedArray:
begin
```

5 - The Elsif Clause

The elsif clause was handled in the parser phase by converting the elsif blocks into nested if statements. This meant no changes needed to be made in the semantic phase for elsif to work.

6 - Loop Statements

The first step to handling loop statements was the addition of the following tokens input tokens: sLoopStmt, sLoopBreakIf, sLoopEnd. As well, the following output tokens needed to be added: tLoopBegin, tLoopBreakIf, tLoopTest, tLoopEnd. Tokens referring to “repeat” were also removed as that syntax has been removed.

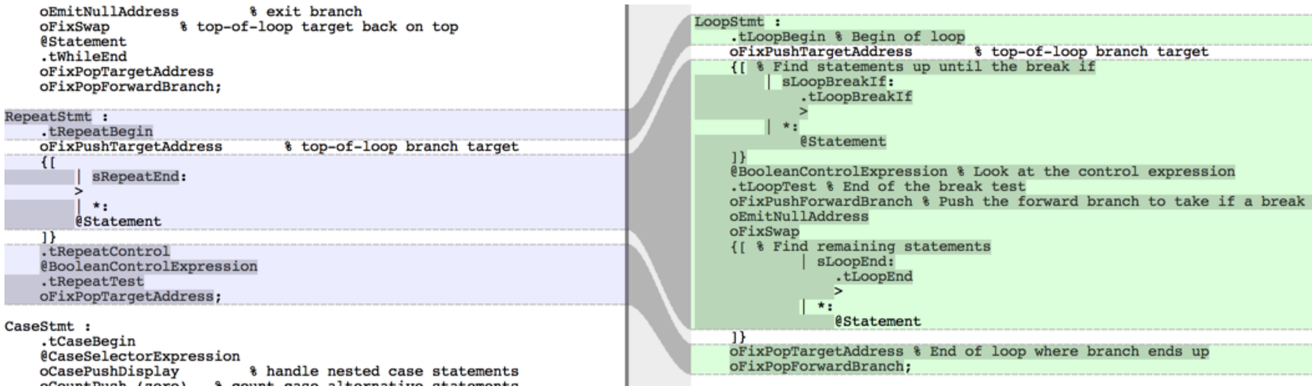
Next, loops replaced repeats within the Statement rule.

```
Statement :
    [
        | sAssignmentStmt:
        @AssignmentStmt
        | sCallStmt:
        @CallStmt
        | sBegin:
        @BeginStmt
        | sIfStmt:
        @IfStmt
        | sWhileStmt:
        @WhileStmt
        | sCaseStmt:
        @CaseStmt
        | sRepeatStmt:
        @RepeatStmt
        | sNullStmt:
    ];
```

```
Statement :
    [
        | sAssignmentStmt:
        @AssignmentStmt
        | sCallStmt:
        @CallStmt
        | sBegin:
        @BeginStmt
        | sIfStmt:
        @IfStmt
        | sWhileStmt:
        @WhileStmt
        | sCaseStmt:
        @CaseStmt
        | sLoopStmt:
        @LoopStmt
        | sNullStmt:
    ];
```

semantic.ssl

Next, a rule for analyzing the loops tokens was created.



```
    oEmitNullAddress        % exit branch
    oFixSwap        % top-of-loop target back on top
    @Statement
    .tWhileEnd
    oFixPopTargetAddress
    oFixPopForwardBranch;

RepeatStmt :
    .tRepeatBegin
    oFixPushTargetAddress       % top-of-loop branch target
    {[
        | sRepeatEnd:
        >
        | *:
        @Statement
    ]}
    .tRepeatControl
    @BooleanControlExpression
    .tRepeatTest
    oFixPopTargetAddress;

CaseStmt :
    .tCaseBegin
    @CaseSelectorExpression
    oCasePushDisplay        % handle nested case statements
```

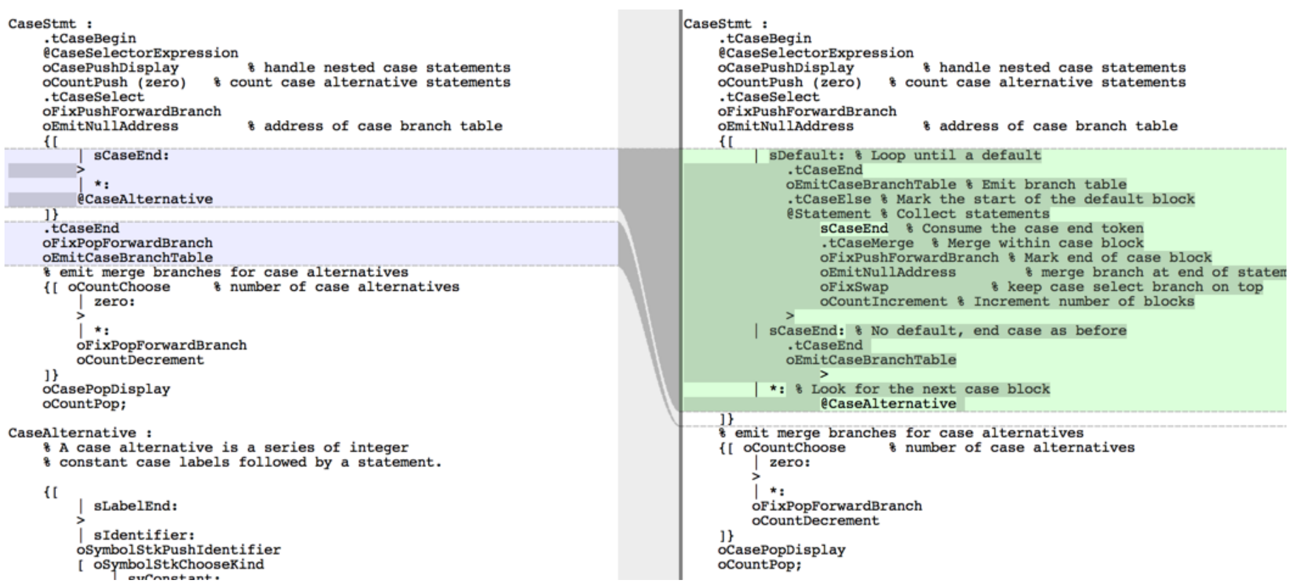
```
LoopStmt :
    .tLoopBegin % Begin of loop
    oFixPushTargetAddress       % top-of-loop branch target
    {[ % Find statements up until the break if
        | sLoopBreakIf:
            .tLoopBreakIf
            >
        | *:
            @Statement
    ]}
    @BooleanControlExpression % Look at the control expression
    .tLoopTest % End of the break test
    oFixPushForwardBranch % Push the forward branch to take if a break
    oEmitNullAddress
    oFixSwap
    {[ % Find remaining statements
            | sLoopEnd:
                .tLoopEnd
                >
            | *:
                @Statement
    ]}
    oFixPopTargetAddress % End of loop where branch ends up
    oFixPopForwardBranch;
```

semantic.ssl

The rule is similar to a broken up while loop with the break condition in the middle. A loop finds statements up until the break clause, assess the conditional and notes a branch should occur if it is true. Further statements are parsed and the target of the branch is found at the end of the loop.

7 - The Switch Statement and Default Clause

Handling of case statements required accepting the new sDefault token from the parser. It also required adding the tCaseDefault and tCase to the list of output tokens.



```
CaseStmt :
    .tCaseBegin
    @CaseSelectorExpression
    oCasePushDisplay        % handle nested case statements
    oCountPush (zero)   % count case alternative statements
    .tCaseSelect
    oFixPushForwardBranch
    oEmitNullAddress        % address of case branch table
    {[
        | sCaseEnd:
        >
        | *:
        @CaseAlternative
    ]}
    .tCaseEnd
    oFixPopForwardBranch
    oEmitCaseBranchTable
    % emit merge branches for case alternatives
    {[ oCountChoose     % number of case alternatives
        | zero:
        >
        | *:
        oFixPopForwardBranch
        oCountDecrement
    ]}
    oCasePopDisplay
    oCountPop;

CaseAlternative :
    % A case alternative is a series of integer
    % constant case labels followed by a statement.

    {[
        | sLabelEnd:
        >
        | sIdentifier:
        oSymbolStkPushIdentifier
        [ oSymbolStkChooseKind
```

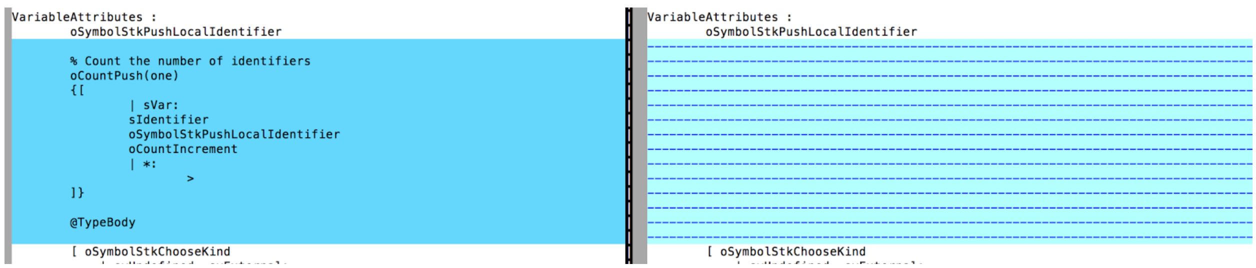
```
CaseStmt :
    .tCaseBegin
    @CaseSelectorExpression
    oCasePushDisplay        % handle nested case statements
    oCountPush (zero)   % count case alternative statements
    .tCaseSelect
    oFixPushForwardBranch
    oEmitNullAddress        % address of case branch table
    {[
        | sDefault: % Loop until a default
            .tCaseEnd
            oEmitCaseBranchTable % Emit branch table
            .tCaseElse % Mark the start of the default block
            @Statement % Collect statements
                sCaseEnd  % Consume the case end token
                .tCaseMerge  % Merge within case block
                oFixPushForwardBranch % Mark end of case block
                oEmitNullAddress        % merge branch at end of statem
                oFixSwap            % keep case select branch on top
                oCountIncrement % Increment number of blocks
            >
        | sCaseEnd: % No default, end case as before
            .tCaseEnd
            oEmitCaseBranchTable
                >
        | *: % Look for the next case block
                @CaseAlternative
    ]}
    % emit merge branches for case alternatives
    {[ oCountChoose     % number of case alternatives
        | zero:
        >
        | *:
        oFixPopForwardBranch
        oCountDecrement
    ]}
    oCasePopDisplay
    oCountPop;
```

semantic.ssl

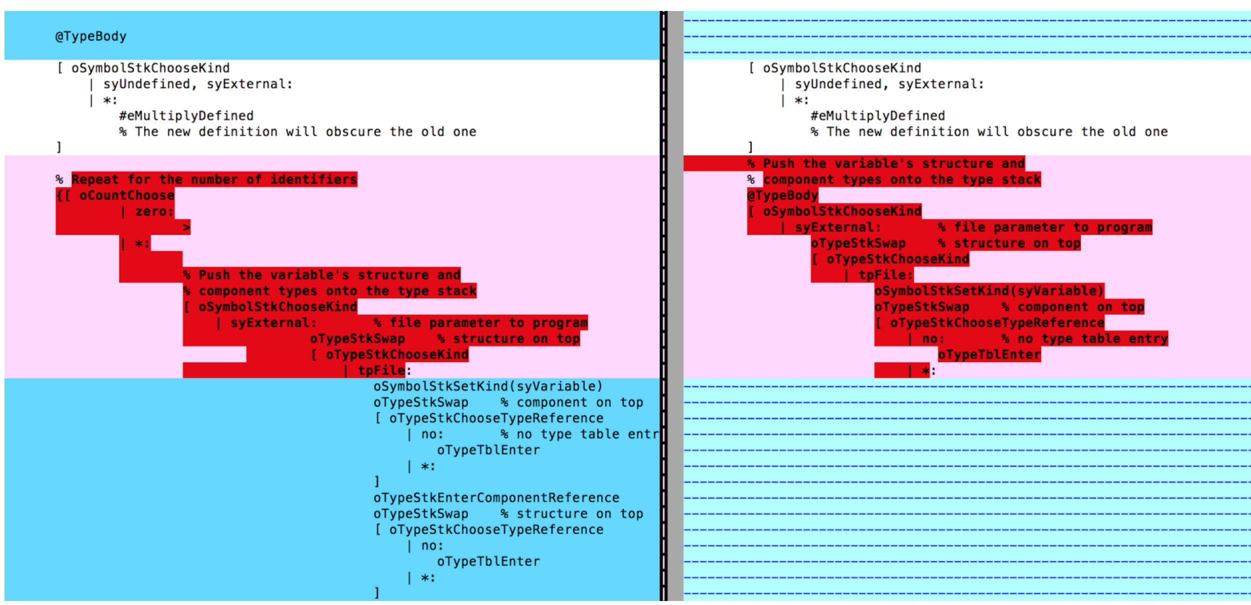
Default statements are optional so a case was added where the sDefault token appears alongside the case where the case statement ends without a default. The regular cases are delimited with the usual .tCaseEnd, but an additional token (tCaseElse) is used to mark the start of statements for the default cased. The final branch is then merged with the rest of the case blocks.

8 - Multiple Variable Declarations

To handle multiple variable declarations on one line we had to modify the VariableAttributes rule in semantic.ssl. First we added a loop which counts the number of identifiers and stores the number in the count stack.



Next we put the code after @TypeBody into a loop which exits when the count reaches zero, otherwise it declares a variable and decrements count. We left @TypeBody out of the loop since there is only one type. The diff for this rule is not straightforward but we'll try to describe the changes we made using a series of diffs. Below is the top of the loop where you can see the logic for running until the count is zero.



Here is the logic at the end of the loop for decrementing count, popping the symbol stack so that we can declare the next symbol as well as swapping the type stack back so the component is on top.

```
                    oTypeStkSwap % component on top

                    % Decrement count and move to next identifier
                    oCountDecrement
                    oSymbolStkPop

    ]}
```

9 - Constant and Type Definitions

Editing constant and type definitions to include only one constant or type was simply a matter of taking the logic for each out of a loop. Here are the changes for ConstantDefintions:

```
ConstantDefinitions :           % Process named constant definitions      | ConstantDefinitions :           % Proc
        % Only one constantDefinition allowed                             |         {[
        [                                                                 | --------------------------------------
            | sIdentifier:                                                |             | sIdentifier:
                @ConstantValue                                            |                 @ConstantValue
            | *:                                                          |             | *:
        ];                                                                |                 >
--------------------------------------------------------------------------|         ]};
```

Here are the changes for TypeDefinitions:

```
TypeDefinitions :       % process named type definitions      | TypeDefinitions :       % process named type de
        % Only one type definition allowed                    |         {[
        [                                                     | ----------------------------------------------
            | sIdentifier:                                    |             | sIdentifier:
                oSymbolStkPushLocalIdentifier                 |                 oSymbolStkPushLocalIdentifier
                @TypeBody                                     |                 @TypeBody
                [ oSymbolStkChooseKind                        |                 [ oSymbolStkChooseKind
                    | syUndefined:                            |                     | syUndefined:
                    | syExternal:                             |                     | syExternal:
+-- 37 lines: % Program parameters must be declared as file variables--------------|+ +-- 37 lines: % Program parameters must be dec
                oSymbolStkEnterTypeReference                  |                 oSymbolStkEnterTypeReference
                oSymbolTblEnter                               |                 oSymbolTblEnter
                oSymbolStkPop                                 |                 oSymbolStkPop
                oTypeStkPop                                   |                 oTypeStkPop
                oTypeStkPop                                   |                 oTypeStkPop
            | *:                                              |             | *:
        ];                                                    |                 >
--------------------------------------------------------------|         ]};
```

10 - Testing

An extensive list of drift programs and expected T-code outputs was provided which was used to verify the correct functionality of our changes. As well, previous pt pascal examples were also used to verify old functionality no longer worked. A detailed breakdown of what features these tests covered is as follows.

10.1 - Added new tokens, removed old tokens

New tokens are required for the functionality of all tests. To ensure that old tokens were no longer recognized an implementation of primes in pt was put through the compiler. Doing so yields an error that the tokens aren't recognized, which is expected as the code does not have correct syntax.

10.2 - String handling

From the drift examples, a few have heavy usage of strings: francais.pt, boxes.pt, double.pt, bust.pt.

Since these examples are fairly large and hard to debug, tests such as francais-reduced.pt were used to provide focused testing of the new features (substrings, length, comparisons).

```
extern input, output

var infinitive, root : string
var letter : boolean

infinitive = "hello"
root = infinitive :: 1..(#infinitive-2)
letter = (root::1..1 == "a")
```

francais-reduced.pt

```
  .tLiteralInteger
  oEmitValue
  .tLiteralAddress
  oEmitDataAddress
  .tFileDescriptor
  .tLiteralInteger
  oEmitValue
  .tLiteralAddress
  oEmitDataAddress
  .tFileDescriptor
   .tAssignBegin
     .tLiteralAddress
     oEmitValue
      .tLiteralString
      oEmitValue
      oEmitString
       .tAssignString
        .tAssignBegin
         .tLiteralAddress
         oEmitValue
           .tLiteralAddress
           oEmitValue
           .tFetchString
          .tLiteralInteger
          oEmitValue
           .tLiteralAddress
           oEmitValue
           .tFetchString
          .tLength
          .tLiteralInteger
          oEmitValue
          .tSubtract
          .tSubstring
        .tAssignString
         .tAssignBegin
                .tLiteralAddress
                oEmitValue
                .tFetchString
               .tLiteralInteger
               oEmitValue
               .tLiteralInteger
               oEmitValue
               .tSubstring
               .tLiteralString
               oEmitValue
               oEmitString
               .tStringEqual
         .tAssignBoolean
.tTrapBegin
.tTrap
oEmitTrapKind(trHalt)
```

francais-reduced.pt: t-Codes

Other mini-tests included string-assign.pt and string-length.pt.

10.3 - Module Definitions

Many examples had heavy usage of modules and functions. The main example was bust.pt. Once again, bust-reduced.pt was created to provide focused testing of the main module features (public functions, modules, scopes).

```
extern input, output
module m
  func randint * (var reslt : integer, modulus : integer)
    reslt = reslt + 1
  end
end
  func doit

    var x:integer
    x = 1
            randint (x, 13)
  end
```

bust-reduced.pt

10.4 - Loop Statements and Removal of Repeat

Loops are also extensively covered through the provided examples (such as bust.pt). To provide a more direct test, loopSyntax.pt was also created to test only loop functionality.

```
extern output

var i: integer

i = 1

loop
    break if i > 50
    i = i + 1
end
```

loopSyntax.pt

```
   .tLiteralInteger
   oEmitValue
   .tLiteralAddress
   oEmitDataAddress
   .tFileDescriptor
     .tAssignBegin
      .tLiteralAddress
      oEmitValue
       .tLiteralInteger
       oEmitValue
     .tAssignInteger
     .tLoopBegin
     .tLoopBreakIf
         .tLiteralAddress
         oEmitValue
         .tFetchInteger
        .tLiteralInteger
        oEmitValue
        .tGT
     .tLoopTest
     oEmitNullAddress
         .tAssignBegin
          .tLiteralAddress
          oEmitValue
            .tLiteralAddress
            oEmitValue
            .tFetchInteger
           .tLiteralInteger
           oEmitValue
           .tAdd
         .tAssignInteger
     .tLoopEnd
 .tTrapBegin
 .tTrap
 oEmitTrapKind(trHalt)
```

loopSyntax.pt: t-Codes

10.5 - Switch Statements Syntax and Default Clause

Many examples were tested which included the new switch syntax (such as bust.pt). To provide more focused testing, simpleSwitch.pt was also created to test the new functionality.

```
extern input

var a : integer
a = 1

switch a
    case 1:
        a = a + 1
    case 2:
        a = a + 1
    default:
        a = a + 1
end
```

simpleSwitch.pt

```
.tLiteralInteger
   oEmitValue
   .tLiteralAddress
   oEmitDataAddress
   .tFileDescriptor
     .tAssignBegin
      .tLiteralAddress
      oEmitValue
       .tLiteralInteger
       oEmitValue
     .tAssignInteger
.tCaseBegin
                    .tLiteralAddress
                    oEmitValue
                    .tFetchInteger
            .tCaseSelect
            oEmitNullAddress
                 .tAssignBegin
                  .tLiteralAddress
                  oEmitValue
                    .tLiteralAddress
                    oEmitValue
                    .tFetchInteger
                   .tLiteralInteger
                   oEmitValue
                   .tAdd
                 .tAssignInteger
             .tCaseMerge
             oEmitNullAddress
                 .tAssignBegin
                  .tLiteralAddress
                  oEmitValue
                    .tLiteralAddress
                    oEmitValue
                    .tFetchInteger
                   .tLiteralInteger
                   oEmitValue
                   .tAdd
                 .tAssignInteger
             .tCaseMerge
             oEmitNullAddress
            .tCaseEnd
            oEmitCaseBranchTable
             .tCaseElse
                 .tAssignBegin
                  .tLiteralAddress
                  oEmitValue
                    .tLiteralAddress
                    oEmitValue
                    .tFetchInteger
                   .tLiteralInteger
                   oEmitValue
                   .tAdd
                 .tAssignInteger
             .tCaseMerge
             oEmitNullAddress
              .tTrapBegin
 .tTrap
 oEmitTrapKind(trHalt)
```

simpleSwitch.pt: T-codes

10.6 - Full Test Suite

The full test suite has 22-tests. An automated test suite was also created with 13-tests using rspec. Of those 13 tests, 9 were full drift program examples provided by the instructor with a pass determined by matching all t-codes.

All tests can be found in **unit_tests/semantic_tests.**
All expected outputs can be found in **unit_tests/semantic_output_e.**