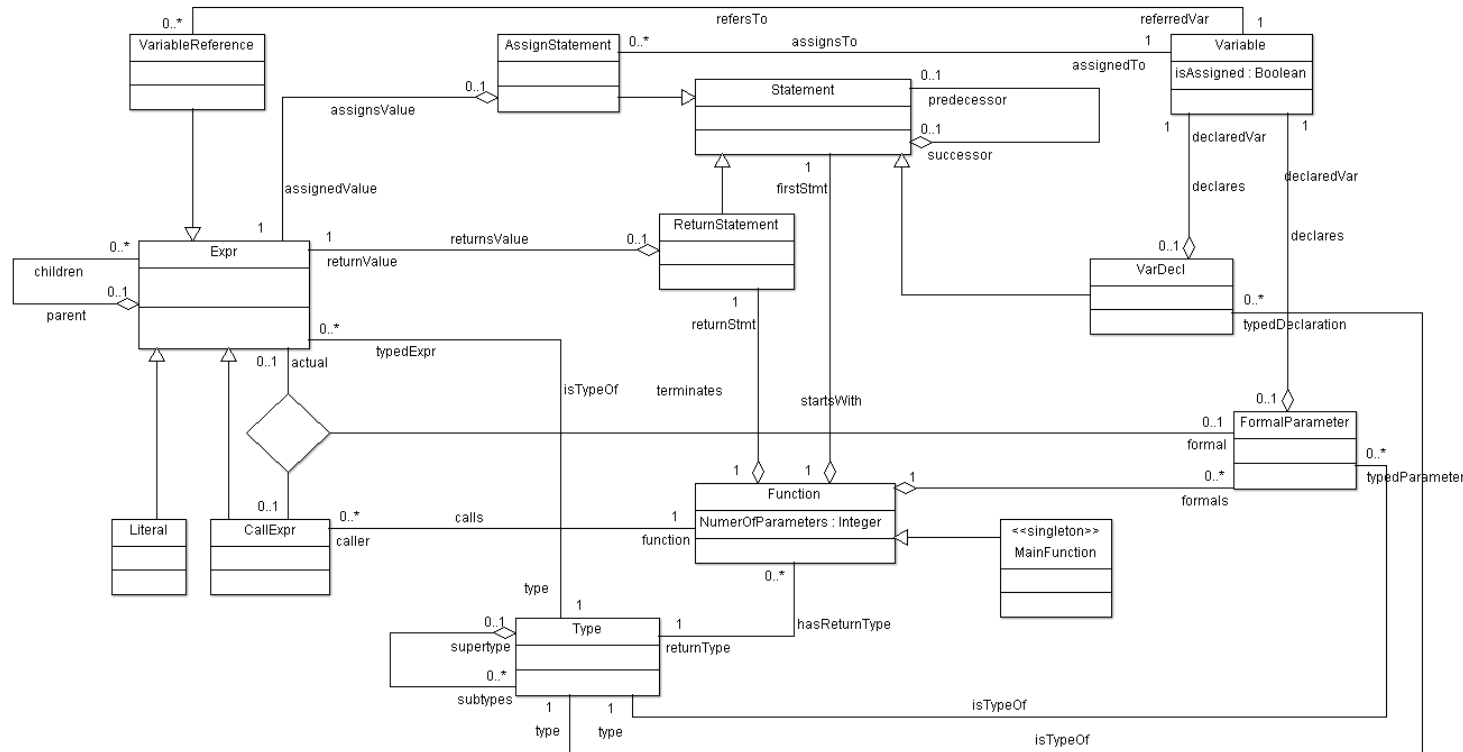


ETH SAE Project 1 - Sherlock

Source code and XML files are in *hand-in.zip*.

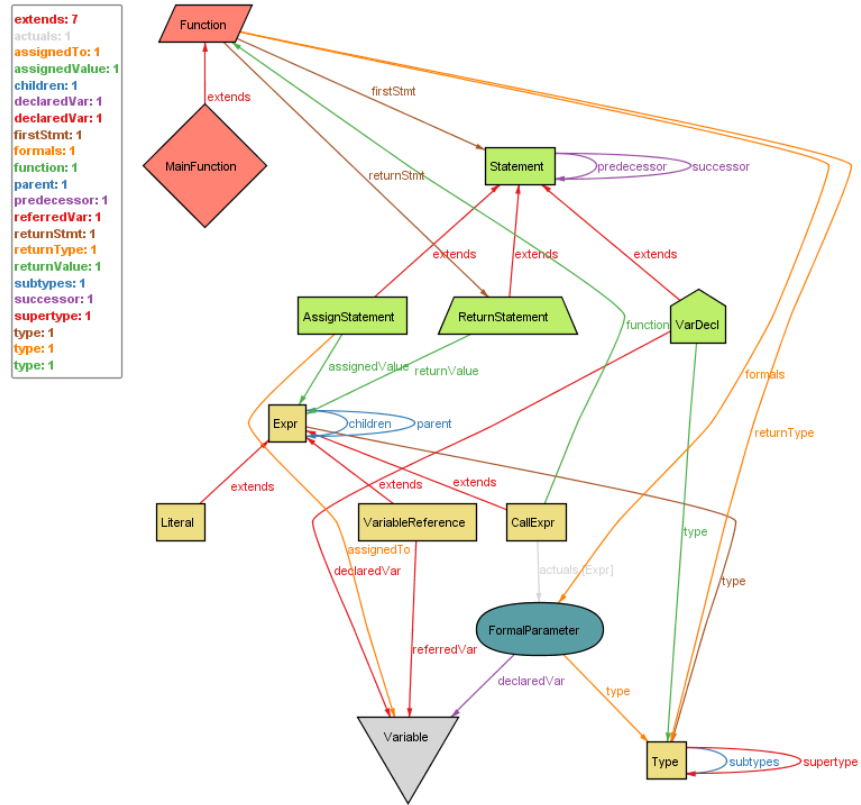
UML class diagram



Elements that are not encodable in the UML class diagram

- The first statement has no predecessor.
- The return statement has no successor.
- Actual parameters are mapped to the formal parameters of the function our expression calls.
- A return statement terminates the execution of the function body.
- A function may not contain unreachable statements.
- Recursion is not allowed.
- The call expressions are in parent-child relation with the actual parameters.
- There is one main function that takes no parameters.
- All functions are transitively called from the main function.
- A variable has to be declared at least once. (We only modelled it has to be declared at most once, but we want to say that variables are declared exactly once.)
- Variables must be declared in the same function before the first use such as the first assignment.
- Variables can only be used in expressions after they have been assigned to.
- We do not allow dead variables or dead assignments.
- There are no assignments to parameters.
- Typing rules for assignments, function calls and return statements.

Alloy model from task B

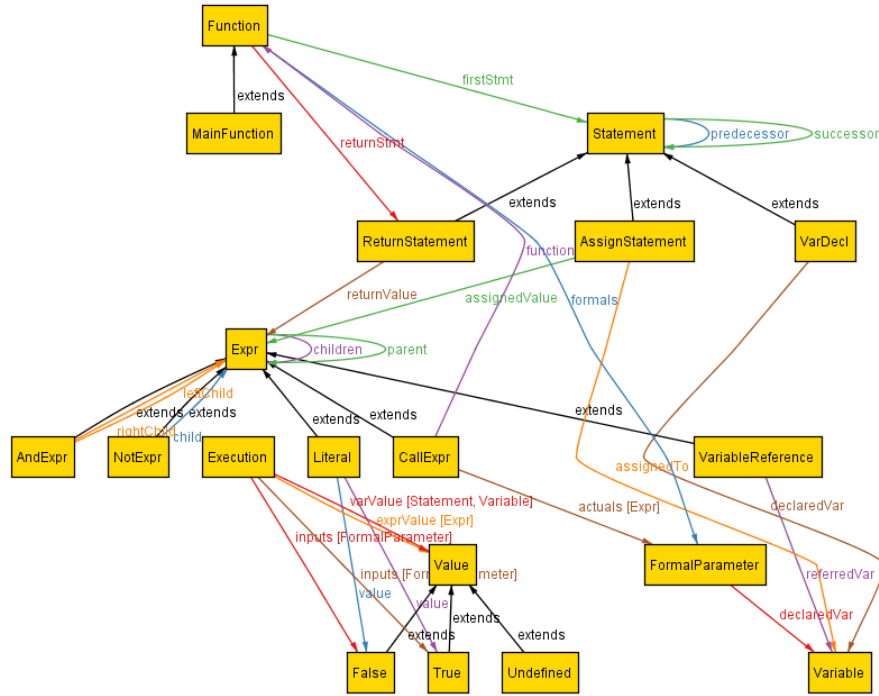


Elements that are not encodable in the Alloy model for task B

- A return statement terminates the execution of the function body. (This is not a static constraint.)
- A function may not contain unreachable statements.

Alloy model from task D

extends: 12
 actuals: 1
 assignedTo: 1
 assignedValue: 1
 child: 1
 children: 1
 declaredVar: 1
 declaredVar: 1
 exprValue: 1
 firstStmt: 1
 formals: 1
 function: 1
 inputs: 1
 inputs: 1
 leftChild: 1
 parent: 1
 predecessor: 1
 referredVar: 1
 returnStmt: 1
 returnValue: 1
 rightChild: 1
 successor: 1
 value: 1
 value: 1
 varValue: 1



Instances that are not feasible in task C

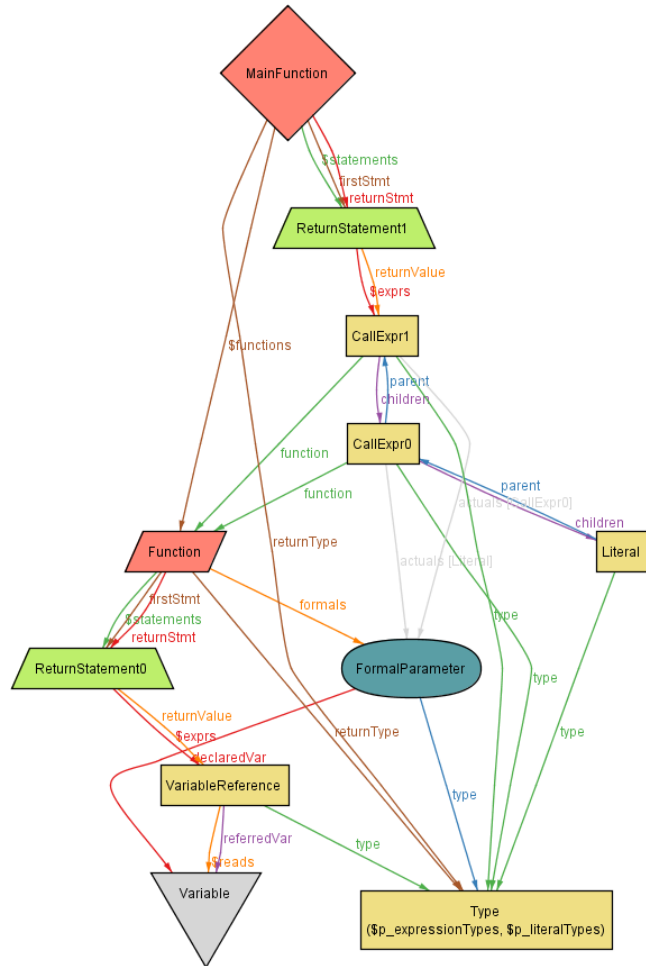
- Instance 1 is not feasible. We would need two function calls from the main function to itself. However, the LINEAR language does not allow recursion.
- Instance 5 is not feasible since it is not possible to have incomparable types at all due to the restricted set of expressions the language provides.

Image exports from task C

(We only show one screen per generated instance.)

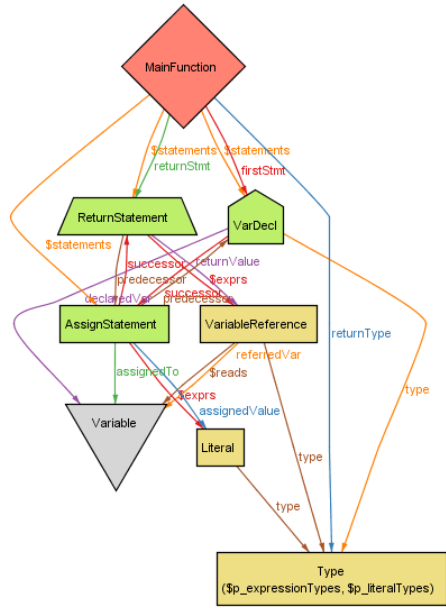
Instance 2

\$exprs: 2
\$functions: 1
\$reads: 1
\$statements: 2
actuals: 2
children: 2
declaredVar: 1
firstStmt: 2
formals: 1
function: 2
parent: 2
referredVar: 1
returnStmt: 2
returnType: 2
returnValue: 2
type: 4
type: 1



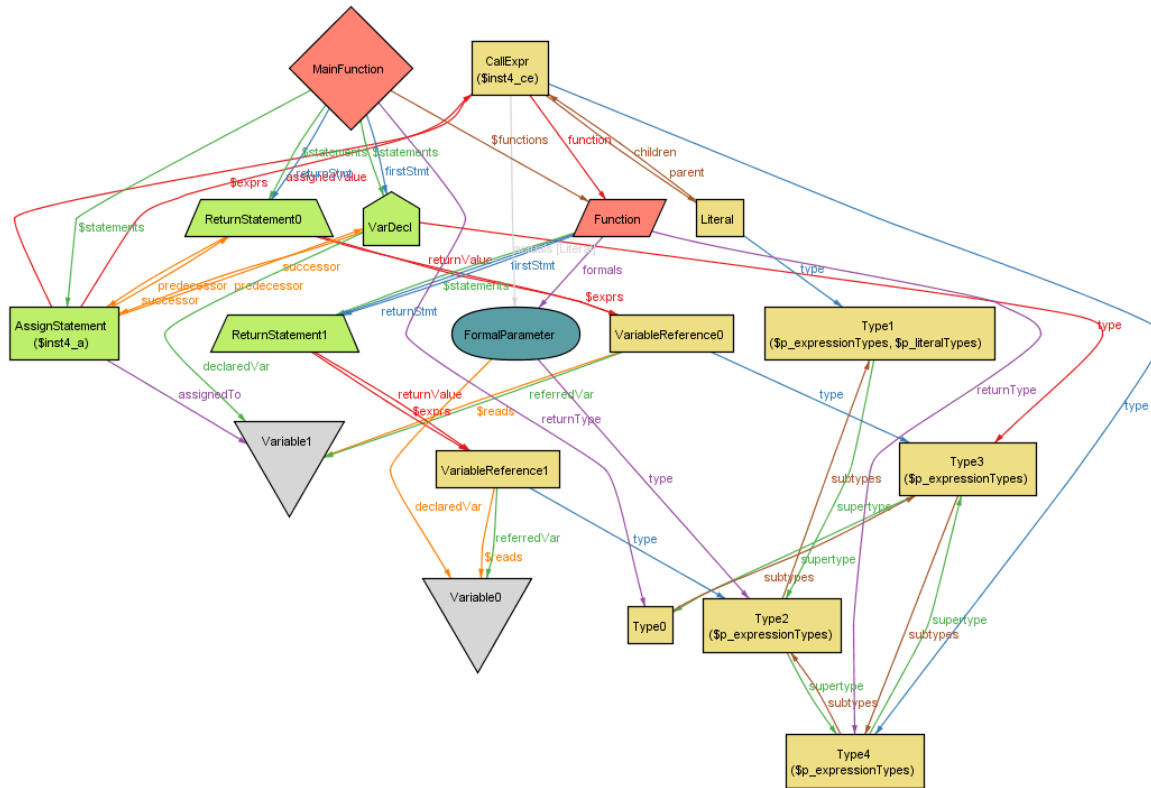
Instance 3

\$exprs: 2
\$reads: 1
\$statements: 3
assignedTo: 1
assignedValue: 1
declaredVar: 1
firstStmt: 1
predecessor: 2
referredVar: 1
returnStmt: 1
returnType: 1
returnValue: 1
successor: 2
type: 2
type: 1



Instance 4

```
$exprs: 3
$functions: 1
$reads: 2
$statements: 4
actuals: 1
assignedTo: 0
assignedValue: 1
children: 1
declaredVar: 1
declaredVar: 1
firstStmt: 2
formals: 1
function: 1
parent: 1
predecessor: 2
referredVar: 2
returnStmt: 2
returnType: 2
returnValue: 2
subtypes: 4
successor: 2
supertype: 4
type: 4
type: 1
type: 1
```



(There are no infeasible instances.)

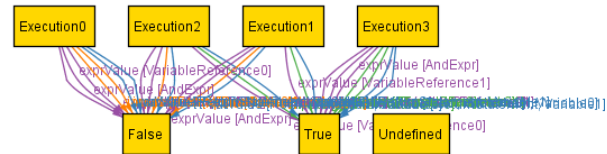
Image exports from task E

Instance 1

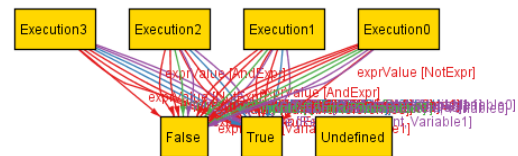
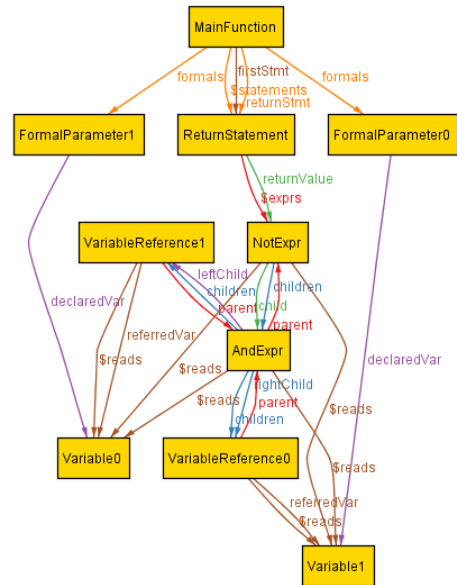
The diagram illustrates the relationships between various components in a compiler, including MainFunction, ReturnStatement, FormalParameter1, FormalParameter0, AndExpr, VariableReference0, VariableReference1, Variable0, and Variable1, connected by labeled edges.

The components are represented as yellow boxes, and the edges are labeled with terms related to compiler semantics, such as `firstStmt`, `formals`, `returnStmt`, `returnValue`, `sExprs`, `parent`, `lightChild`, `declaredVar`, `children`, `referredVar`, and `sReads`.

The diagram shows a hierarchical structure where `MainFunction` is the root, branching into `ReturnStatement` and `FormalParameter1`. `ReturnStatement` further branches into `AndExpr` and `FormalParameter0`. `AndExpr` branches into `VariableReference0` and `VariableReference1`. `VariableReference0` branches into `Variable0`, and `VariableReference1` branches into `Variable1`.

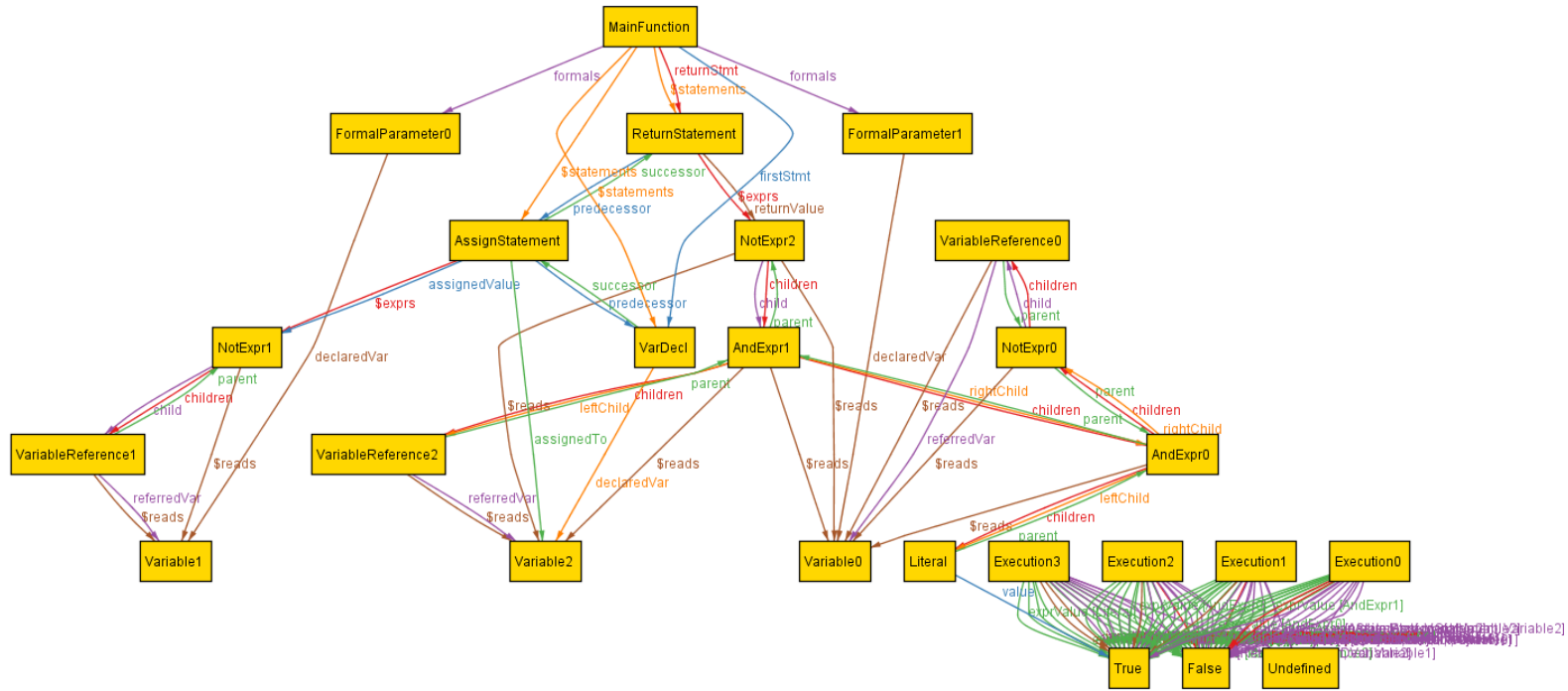


```
$exprs: 1
$reads: 6
$statements: 1
child: 1
children: 3
declaredVar: 2
exprValue: 16
firstStmt: 1
formals: 2
inputs: 4
input: 4
leftChild: 1
parent: 3
referencedVar: 2
returnStmt: 1
returnValue: 1
rightChild: 1
varValue: 8
```

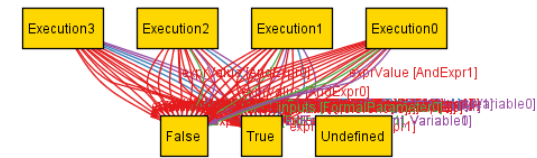


Instance 3

\$exprs: 2
 \$reads: 10
 \$statements: 3
 assignedTo: 1
 assignedValue: 1
 child: 3
 children: 7
 declaredVar: 2
 declaredVar: 1
 exprValue: 36
 firstStmt: 1
 formals: 2
 inputs: 4
 inputs: 4
 leftChild: 2
 parent: 7
 predecessor: 2
 referredVar: 3
 returnStmt: 1
 returnValue: 1
 rightChild: 2
 successor: 2
 value: 1
 varValue: 32




```
$exprs: 1
$reads: 16
$statements: 1
child: 4
children: 10
declaredVar: 2
exprValue: 44
firstStmt: 1
formals: 2
inputs: 4
inputs: 4
leftChild: 3
parent: 10
referredVar: 4
returnStmt: 1
returnValue: 1
rightChild: 3
varValue: 8
```



```

$exprs: 4
$functions: 2
$reads: 13
$statements: 5
$actuals: 3
assignedTo: 1
assignedValue: 1
child: 1
children: 6
declaredVar: 5
declaredVar: 1
exprValue: 40
firstStmt: 3
formals: 5
function: 2
inputs: 4
inputs: 4
leftChild: 1
parent: 6
predecessor: 2
referredVar: 6
returnStmt: 3
returnValue: 3
rightChild: 1
successor: 2
varValue: 46

```

