1. Marc Bertelsen, berte20@student.sdu.dk

2. Xtext Grammar:

   (a) Does your grammar support all of the example programs? If not, what are the limitations?
   - Yes, all the example programs are supported.

   (b) How did you implement operator precedence and associativity?
   - They are implemented in order of Union/Intersection -> Product -> Projection/Filter -> Primitives.
   - In the tree of operations each have a 'left' and optional 'right' values, this happens recursively, allowing for chaining operations.
   - This is done by having all the operations extend a common parent (Expression) and having a hierarchy were each operation uses the type of the layer below.

   (c) How did you implement the syntax of variables (`ID` in rule `Exp` of the X24 BNF) such that they can refer both to record's parameters and members?
   - To reference both Parameters and Member a 'Ref' rule was used as a Primitive value type, i.e., the very lowest in the Expression hierarchy, together with the scope provider, it allowed for referencing the valid parameters and member following the inheritance of the record.

3. Scoping Rules

   (a) Did you implement scoping rules that allow variables to refer to members and parameters? If not, what are the limitations?
   - Yes, parameters and member, both local and inherited are provided in the scope of the correct records.

   (b) Did you implement scoping rules that allow for correct record member access? If not, what are the limitations?
   - Yes, all the members of a record are available to access, even the inherited ones.

   (c) Describe your implementation of any scoping rules included with your system.
   - The scoping implemented look to the reference of the 'Member' in the context of a 'RecordAccess', from here navigating up to

the root record in the hierarchy and get all the members while traversing.
- Second part is in the context of a 'Ref' (Member/Parameter binding). Here the parent record, which contains the 'Ref', is gotten. Combine all the parameters from the record with the members from navigating to the root record in the hierarchy and getting all the members while traversing.

4. Validation

   (a) What validation rule did you implement, if any?
   - Rules 4 and 5 are implemented. They check that only a single type is present within a set, both in the context of parameters and raw value usage.

   (b) Describe the implementation of any validation rule.
   - Checking on each of the Set objects by iterating over each of the declared values. If any value is of type 'Int' the 'hasInt' flag is set, same for 'Tuple' with the 'hasTuple' flag. If both flags are set report an error, notifying the user that both types are not allowed within a Set.

   (c) For the non-implemented validation rules, describe the problems of not having them.
   - Without the missing validations, it is possible to write invalid syntax, which can cause the generator to encounter problems and potentially have the runtime crash.

5. Generator

   (a) Does your code generator correctly generate code for all examples provided, and are you confident that it will also work for "similar" programs? If not, then what limitations are there?
   - Yes, the implementation can generate correct code. It passes all the test, and I can write other programs where I test the record inheritance.

   (b) Briefly describe how your code generator works.
   - The generation is broadly split into 3 sections:

      - generateExternalDefs -> if needed, generates the external interface.

- generateComputes -> generates the compute function with all the Set operations.
- generateRecords -> generates all the records with all their parts and inheritance.

- All the different operations and primitive types all extend Expression, which allows for using dispatch to handle all the cases.

6. Implementation: Include your Xtext grammar file and all implemented Xtend files (scoping, type validation, generator, and any additional file).

Xtext Grammar

```
grammar dk.sdu.mmmi.mdsd.ex.SetLang with org.eclipse.xtext.common.Terminals

generate setLang "http://www.sdu.dk/mmmi/mdsd/ex/SetLang"

Program:
 'program' name=ID
 externalDefs+=ExternalDef*
 records+=Record*
 computes+=Compute*
;

ExternalDef:
 'external' name=ID '(' type=Type ')'
;

Record:
 'record' name=ID (':' parent=[Record])?
 ('(' parameters+=Parameter (',' parameters+=Parameter)*')')?
 '{' members+=Member* '}'
;

Parameter:
 name=ID ':' type=Type
;
```

```
Type:
 'Int' |
 'Tuple' |
 'Set' '(' Type ')'
;

Member:
 name=ID ':' exp=UnionIntersection
;

Compute:
 'compute' exp=UnionIntersection
;

UnionIntersection returns Expression:
 Product ((
  {Union.left=current} 'U' |
  {Intersection.left=current} '&'
 ) right=Product)*
;

Product returns Expression:
 ProjectionFilter (({Product.left=current} '*') right=ProjectionFilter)*
;

ProjectionFilter returns Expression:
 Primitive ('#'
  (({Projection.op=current} '<' indices+=INT (',' indices+=INT)* '>')  |
  ({Filter.op=current}'[' external=[ExternalDef] ']'))
  )?
;

Binding:
 Member | Parameter
;

Primitive returns Expression:
 IntValue |
```

```
  Set |
  Tuple |
  RecordAccess |
  Ref |
  Parenthesis
;

Ref:
 ref=[Binding]
;

IntValue returns Expression:
 {IntValue} value=INT
;

Set returns Expression:
 {Set} '{' (values+=UnionIntersection (',' values+=UnionIntersection)*)?  '}'
;

Tuple returns Expression:
 {Tuple} '[' values+=UnionIntersection (',' values+=UnionIntersection*)* ']'
;

Parenthesis returns Expression:
 {Parenthesis} '(' exp=UnionIntersection ')'
;



RecordAccess returns Expression:
 {RecordAccess} '!' record=[Record]
  ('(' parameters+=UnionIntersection (',' parameters+=UnionIntersection)* ')')?
  '->' member=[Member]
;
```

Xtend Generator

```
  /*
```

```
 * generated by Xtext 2.33.0
 */
package dk.sdu.mmmi.mdsd.ex.generator

import java.util.List
import java.util.ArrayList
import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.AbstractGenerator
import org.eclipse.xtext.generator.IFileSystemAccess2
import org.eclipse.xtext.generator.IGeneratorContext
import org.eclipse.emf.common.util.EList
import dk.sdu.mmmi.mdsd.ex.setLang.Program
import dk.sdu.mmmi.mdsd.ex.setLang.IntValue
import dk.sdu.mmmi.mdsd.ex.setLang.Set
import dk.sdu.mmmi.mdsd.ex.setLang.Tuple
import dk.sdu.mmmi.mdsd.ex.setLang.Filter
import dk.sdu.mmmi.mdsd.ex.setLang.Projection
import dk.sdu.mmmi.mdsd.ex.setLang.Intersection
import dk.sdu.mmmi.mdsd.ex.setLang.Union
import dk.sdu.mmmi.mdsd.ex.setLang.Product
import dk.sdu.mmmi.mdsd.ex.setLang.RecordAccess
import dk.sdu.mmmi.mdsd.ex.setLang.Parenthesis
import dk.sdu.mmmi.mdsd.ex.setLang.Record
import dk.sdu.mmmi.mdsd.ex.setLang.Compute
import dk.sdu.mmmi.mdsd.ex.setLang.Parameter
import dk.sdu.mmmi.mdsd.ex.setLang.Ref
import dk.sdu.mmmi.mdsd.ex.setLang.ExternalDef

/**
 * Generates code from your model files on save.
 *
 * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#co
 */
class SetLangGenerator extends AbstractGenerator {

 override void doGenerate(Resource resource, IFileSystemAccess2 fsa,
    IGeneratorContext context) {
  val program = resource.allContents.filter(Program).next
```

```
 fsa.generateFile('setdsl/' + program.name + '.java',
         program.generateCode)
}

def CharSequence generateCode(Program program) '''
/* generated by SetLang parser */

package setdsl;

import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import utils.*;

public class «program.name» extends AbstractRecordProgram {
 «IF program.externalDefs !== null && program.externalDefs.size > 0»
 «program.generateConstructor»
 «ENDIF»
 «program.computes.generateComputes»
 «program.records.generateRecords»
 «IF program.externalDefs !== null && program.externalDefs.size > 0»
 «program.externalDefs.generateExternalDefs»
 «ENDIF»
}
'''

def CharSequence generateConstructor(Program program) '''

private External external;

public «program.name»(External external) {
 this.external = external;
}
'''

def CharSequence generateExternalDefs(EList<ExternalDef> externalDefs) '''
```

```
public interface External {
 «FOR e:externalDefs»
 public boolean «e.name»(«e.type.toJava» a);
 «ENDFOR»
}
'''

def CharSequence generateComputes(EList<Compute> computes) '''

public List compute() {
 List results = new ArrayList();
 «FOR compute:computes»
 results.add(«compute.exp.generateExp»);
 «ENDFOR»
 return results;
}
'''

def CharSequence generateRecords(EList<Record> records) '''
«FOR p:records»«p.generateRecord»«ENDFOR»'''

def CharSequence generateRecord(Record record) '''

public class «record.name»
 «IF record.parent !== null» extends «record.parent.name»«ENDIF» {
 «FOR p:record.parameters»
 private «p.type.toJava» «p.name»;
 «ENDFOR»
 «FOR m:record.members»
 public Set «m.name»;
 «ENDFOR»

 public «record.name»(«FOR p:record.getOwnAndLineageParameters SEPARATOR ', '»
   «p.type.toJava» «p.name»«ENDFOR») {
  «IF record.parent !== null»
  super(«FOR p:record.getLineageParameters SEPARATOR ', '»«p.name»«ENDFOR»);
  «ENDIF»
```

```
  «FOR p:record.parameters»
  this.«p.name» = «p.name»;
  «ENDFOR»
  «FOR m:record.members»
  this.«m.name» = «m.exp.generateExp»;
  «ENDFOR»
 }
}
'''

def List<Parameter> getOwnAndLineageParameters(Record record) {
 val lineage = new ArrayList();
 lineage.addAll(record.getLineageParameters);
 lineage.addAll(record.parameters);
 lineage;
}

def List<Parameter> getLineageParameters(Record record) {
 val lineage = new ArrayList();
 if (record.parent !== null) {
  record.parent.getParentParameters(lineage);
 }
 lineage;
}

def void getParentParameters(Record record, List<Parameter> parameters) {
 if (record.parent !== null) {
  record.parent.getParentParameters(parameters)
 }
 parameters.addAll(record.parameters);
}

def CharSequence toJava(String type) {
 if (type == 'Int')
  return 'int'
 else if (type.startsWith('Set'))
  return 'Set'
 type
```

```
}

def dispatch CharSequence generateExp(IntValue value) '''
«value.value»'''

def dispatch CharSequence generateExp(Set set) '''
Set.of(«FOR v:set.values SEPARATOR ', '»«v.generateExp»«ENDFOR»)'''

def dispatch CharSequence generateExp(Tuple tuple) '''
Tuple.of(«FOR v:tuple.values SEPARATOR ', '»«v.generateExp»«ENDFOR»)'''

def dispatch CharSequence generateExp(Product product) '''
set_product(«product.left.generateExp», «product.right.generateExp»)'''

def dispatch CharSequence generateExp(RecordAccess recordAccess) '''
new «recordAccess.record.name»(«FOR p:recordAccess.parameters SEPARATOR ',' »
 «p.generateExp»«ENDFOR»).«recordAccess.member.name»'''

def dispatch CharSequence generateExp(Filter filter) '''
set_filter(«filter.op.generateExp», external::«filter.external.name»)'''

def dispatch CharSequence generateExp(Projection projection) '''
set_projection(«projection.op.generateExp»,
 «FOR i:projection.indices SEPARATOR ', '»«i»«ENDFOR»)'''

def dispatch CharSequence generateExp(Intersection intersection) '''
set_intersection(«intersection.left.generateExp»,
 «intersection.right.generateExp»)'''

def dispatch CharSequence generateExp(Union union) '''
set_union(«union.left.generateExp», «union.right.generateExp»)'''

def dispatch CharSequence generateExp(Parenthesis parenthesis) '''
«parenthesis.exp.generateExp»'''

def dispatch CharSequence generateExp(Ref ref)'''
this.«ref.ref.name»'''
}
```

Xtend Validator

```
  /*
 * generated by Xtext 2.33.0
 */
package dk.sdu.mmmi.mdsd.ex.validation

import java.util.ArrayList
import java.util.List
import org.eclipse.xtext.validation.Check
import dk.sdu.mmmi.mdsd.ex.setLang.Expression
import dk.sdu.mmmi.mdsd.ex.setLang.Filter
import dk.sdu.mmmi.mdsd.ex.setLang.IntValue
import dk.sdu.mmmi.mdsd.ex.setLang.Intersection
import dk.sdu.mmmi.mdsd.ex.setLang.Product
import dk.sdu.mmmi.mdsd.ex.setLang.Projection
import dk.sdu.mmmi.mdsd.ex.setLang.Record
import dk.sdu.mmmi.mdsd.ex.setLang.Ref
import dk.sdu.mmmi.mdsd.ex.setLang.Set
import dk.sdu.mmmi.mdsd.ex.setLang.SetLangPackage
import dk.sdu.mmmi.mdsd.ex.setLang.Tuple
import dk.sdu.mmmi.mdsd.ex.setLang.Union

/**
 * This class contains custom validation rules.
 *
 * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#va
 */
class SetLangValidator extends AbstractSetLangValidator {

 public static val MULTIPLE_VALIE_TYPES = 'multiple_valie_types'

 // Make sure that a Set only contains one type of values
 @Check
 def checkSetTypeRestriction(Set set) {
  var hasInt = false;
```

```
 var hasTuple = false;
 for (v : set.values) {
  if (v instanceof IntValue) {
   hasInt = true;
  }
  else if (v instanceof Tuple) {
   hasTuple = true;
  }

  // Both are not allowed
  if (hasInt && hasTuple) {
   error('Set cannot contain both Int and Tuple!',
    SetLangPackage.Literals.SET__VALUES,
    MULTIPLE_VALIE_TYPES
   )
   return
  }
 }
}

// Make sure that a Set only uses parameter of the correct type
@Check
def checkSetParameterUse(Record record) {
 for (m : record.members) {
  for (e : m.exp.expandExp) {

   if (e instanceof Set) {
    checkSetParameters(e, record)
   }
  }
 }
}

protected def void checkSetParameters(Set e, Record record) {
 var hasInt = false;
 var hasTuple = false;
 for (v : e.values) {
  if (v instanceof IntValue) {
```

```
   hasInt = true;
  }
  else if (v instanceof Tuple) {
   hasTuple = true;
  }
  else if (v instanceof Ref) {
   val type =  record.getRefType(v)
   if (type.equals('Int')) {
    hasInt = true;
   }
   else if (type.equals('Tuple')) {
    hasTuple = true;
   }
  }
  // Both are not allowed
  if (hasInt && hasTuple) {
   error('Set cannot contain both Int and Tuple!',
    SetLangPackage.Literals.RECORD__MEMBERS,
    MULTIPLE_VALIE_TYPES
   )
  }
 }
}

def getRefType(Record record, Ref ref) {
 record.parameters.filter[e | e.name == ref.ref.name].toList.get(0).type
}

def expandExp(Expression exp) {
 val expanded = new ArrayList();
 exp.expand(expanded)
 expanded
}

def dispatch void expand(Ref ref, List<Object> expanded) {
 expanded.add(ref.ref.name)
}
```

```
 def dispatch void expand(Set set, List<Object> expanded) {
  expanded.add(set)
 }

 def dispatch void expand(Union union, List<Object> expanded) {
  union.left.expand(expanded)
  union.right.expand(expanded)
 }

 def dispatch void expand(Intersection intersection, List<Object> expanded) {
  intersection.left.expand(expanded)
  intersection.right.expand(expanded)
 }

 def dispatch void expand(Product product, List<Object> expanded) {
  product.left.expand(expanded)
  product.right.expand(expanded)
 }

 def dispatch void expand(Projection projection, List<Object> expanded) {
  projection.op.expand(expanded)
 }

 def dispatch void expand(Filter filter, List<Object> expanded) {
  filter.op.expand(expanded)
 }

}
```

Xtend ScopeProvider

```
  /*
 * generated by Xtext 2.33.0
 */
package dk.sdu.mmmi.mdsd.ex.scoping

import java.util.List
```

```
import java.util.ArrayList
import org.eclipse.xtext.scoping.IScope
import org.eclipse.emf.ecore.EObject
import org.eclipse.emf.ecore.EReference
import org.eclipse.xtext.scoping.Scopes
import dk.sdu.mmmi.mdsd.ex.setLang.SetLangPackage
import dk.sdu.mmmi.mdsd.ex.setLang.Member
import dk.sdu.mmmi.mdsd.ex.setLang.Record
import dk.sdu.mmmi.mdsd.ex.setLang.RecordAccess
import dk.sdu.mmmi.mdsd.ex.setLang.Ref
import dk.sdu.mmmi.mdsd.ex.setLang.Binding

/**
 * This class contains custom scoping description.
 *
 * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#sc
 * on how and when to use it.
 */
class SetLangScopeProvider extends AbstractSetLangScopeProvider {

 override IScope getScope(EObject context, EReference reference) {
     if (context instanceof RecordAccess
      && reference == SetLangPackage.Literals.RECORD_ACCESS__MEMBER
      ) {
         val candidates = (context as RecordAccess).record.getAllValidMembers;
         return Scopes.scopeFor(candidates);
     }
     else if (context instanceof Ref) {
      val record = (context as Ref).getParentRecord;
         val candidates = record.getAllValidBindings;
          return Scopes.scopeFor(candidates);
     }
        return super.getScope(context, reference);
 }

 def Record getParentRecord(Ref ref) {
  var parent = ref.eContainer
  while(parent !== null) {
```

```
    parent = parent.eContainer
    if (parent instanceof Record) {
     return parent as Record
    }
   }
  }

  def List<Binding> getAllValidBindings(Record record) {
   val bindings = new ArrayList<Binding>();
   bindings.addAll(record.getAllValidMembers)
   bindings.addAll(record.parameters)
   bindings;
  }

  def List<Member> getAllValidMembers(Record record) {
   val members = new ArrayList();
   record.getParentMembers(members);
   members.addAll(record.members);
   members;
  }

  def void getParentMembers(Record record, List<Member> members) {
   if (record.parent !== null) {
    record.parent.getParentMembers(members);
    members.addAll(record.parent.members);
   }
  }
 }
```