Appendices

Appendix A

Tool Code Before Development

A.1 Transformations

A.1.1 IsaBasicTypesConv

```
package org.overturetool.cgisa.transformations;

import org.overture.cgisa.isair.analysis.DepthFirstAnalysisIsaAdaptor;
import org.overture.codegen.ir.*;
import org.overture.codegen.ir.declarations.*;
import org.overture.codegen.ir.types.AIntNumericBasicTypeIR;
import org.overture.codegen.trans.assistants.TransAssistantIR;

import java.util.Map;
import java.util.stream.Collectors;

/***
 * Visitor to convert basic VDM types to VDMToolkit types
 */
public class IsaBasicTypesConv extends DepthFirstAnalysisIsaAdaptor {

    private final Map<String, ATypeDeclIR> isaTypeDeclIRMap;
    private final TransAssistantIR t;
    private final AModuleDeclIR vdmToolkitModuleIR;
    private final IRInfo info;
    private final static String isa_VDMInt = "isa_VDMInt";

    public IsaBasicTypesConv(IRInfo info, TransAssistantIR t, AModuleDeclIR
        vdmToolkitModuleIR) {
        this.t = t;
        this.info = info;
        this.vdmToolkitModuleIR = vdmToolkitModuleIR;

        this.isaTypeDeclIRMap = this.vdmToolkitModuleIR.getDecls()
                .stream()
                .filter(d -> {
                    if (d instanceof ATypeDeclIR)
                        return true;
                    else
                        return false;
                }).map(d -> (ATypeDeclIR) d)
                .collect(Collectors.toMap(x -> ((ANamedTypeDeclIR)
                    x.getDecl()).getName().getName(), x -> x));
```

```
    }

    //Transform int to isa_VDMInt
    public void caseAIntNumericBasicTypeIR(AIntNumericBasicTypeIR x){
        if(x.getNamedInvType() == null)
        {
            AIntNumericBasicTypeIR a = new AIntNumericBasicTypeIR();
            // Retrieve isa_VDMInt from VDMToolkit
            ATypeDeclIR isa_td = isaTypeDeclIRMap.get(this.isa_VDMInt);

            x.setNamedInvType((ANamedTypeDeclIR)isa_td.getDecl().clone());
        }

    }
}
```

A.1.2 IsaInvGenTrans.java

```
public class IsaInvGenTrans extends DepthFirstAnalysisIsaAdaptor {

    private final AModuleDeclIR vdmToolkitModule;
    private final Map<String, ATypeDeclIR> isaTypeDeclIRMap;
    private IRInfo info;
    private final Map<String, AFuncDeclIR> isaFuncDeclIRMap;

    public IsaInvGenTrans(IRInfo info, AModuleDeclIR vdmToolkitModuleIR) {
        this.info = info;
        this.vdmToolkitModule = vdmToolkitModuleIR;

        this.isaFuncDeclIRMap =
            this.vdmToolkitModule.getDecls().stream().filter(d ->
        {
            if (d instanceof AFuncDeclIR)
                return true;
            else
                return false;
        }).map(d -> (AFuncDeclIR) d).collect(Collectors.toMap(x -> x.getName(), x
            -> x));

        this.isaTypeDeclIRMap =
            this.vdmToolkitModule.getDecls().stream().filter(d -> {
            if (d instanceof ATypeDeclIR)
                return true;
            else
                return false;
        }).map(d -> (ATypeDeclIR) d).collect(Collectors.toMap(x ->
            ((ANamedTypeDeclIR) x.getDecl()).getName().getName(), x -> x));


    }

    @Override
    public void caseATypeDeclIR(ATypeDeclIR node) throws AnalysisException {
        super.caseATypeDeclIR(node);
```

```
        String typeName = IsaInvNameFinder.findName(node.getDecl());
        SDeclIR decl = node.getDecl();
        SDeclIR invFun = node.getInv();

        if(invFun == null)
        {
            // Invariant function
            AFuncDeclIR invFun_ = new AFuncDeclIR();
            invFun_.setName("inv_" + typeName);

            // Define the type signature
            //TODO: Type should be XTypeInt - correct?
            AMethodTypeIR methodType = new AMethodTypeIR();
            STypeIR t = IsaDeclTypeGen.apply(node.getDecl());
            methodType.getParams().add(t.clone());
            methodType.setResult(new ABoolBasicTypeIR());
            invFun_.setMethodType(methodType);

            // Generate the pattern
            AIdentifierPatternIR identifierPattern = new AIdentifierPatternIR();
            identifierPattern.setName("x");
            AFormalParamLocalParamIR afp = new AFormalParamLocalParamIR();
            afp.setPattern(identifierPattern);
            afp.setType(t.clone()); // Wrong to set entire methodType?
            invFun_.getFormalParams().add(afp);

            // Generate the expression
            SExpIR expr = IsaInvExpGen.apply(decl, identifierPattern,
                methodType.clone(), isaFuncDeclIRMap);
            invFun_.setBody(expr);

            // Insert into AST
            AModuleDeclIR encModule = node.getAncestor(AModuleDeclIR.class);
            if(encModule != null)
            {
                encModule.getDecls().add(invFun_);
            }

            System.out.println("");

        }
    }

    public String GenInvTypeDefinition(String arg){
        return "Definition\n" +
               "   inv_" + arg+ " :: \"" + arg + " \\<Rightarrow> \\<bool>\"\n" +
               "   where\n" +
               "";
    }

}
```

A.1.3 IsaBasicTypesConv

```
/***
 * Visitor to convert basic VDM types to VDMToolkit types
 */
public class IsaBasicTypesConv extends DepthFirstAnalysisIsaAdaptor {

    private final Map<String, ATypeDeclIR> isaTypeDeclIRMap;
    private final TransAssistantIR t;
    private final AModuleDeclIR vdmToolkitModuleIR;
    private final IRInfo info;
    private final static String isa_VDMInt = "isa_VDMInt";

    public IsaBasicTypesConv(IRInfo info, TransAssistantIR t, AModuleDeclIR
        vdmToolkitModuleIR) {
        this.t = t;
        this.info = info;
        this.vdmToolkitModuleIR = vdmToolkitModuleIR;

        this.isaTypeDeclIRMap = this.vdmToolkitModuleIR.getDecls()
                .stream()
                .filter(d -> {
                    if (d instanceof ATypeDeclIR)
                        return true;
                    else
                        return false;
                }).map(d -> (ATypeDeclIR) d)
                .collect(Collectors.toMap(x -> ((ANamedTypeDeclIR)
                    x.getDecl()).getName().getName(), x -> x));
    }

    //Transform int to isa_VDMInt
    public void caseAIntNumericBasicTypeIR(AIntNumericBasicTypeIR x){
        if(x.getNamedInvType() == null)
        {
            AIntNumericBasicTypeIR a = new AIntNumericBasicTypeIR();
            // Retrieve isa_VDMInt from VDMToolkit
            ATypeDeclIR isa_td = isaTypeDeclIRMap.get(this.isa_VDMInt);

            x.setNamedInvType((ANamedTypeDeclIR)isa_td.getDecl().clone());
        }

    }
}
```

A.1.4 IsaDeclTypeGen

```
public class IsaDeclTypeGen extends AnswerIsaAdaptor<STypeIR> {

    public static STypeIR apply(INode node) throws AnalysisException {
        IsaDeclTypeGen finder = new IsaDeclTypeGen();
        return node.apply(finder);
    }

    public STypeIR caseANamedTypeDeclIR(ANamedTypeDeclIR n)
    {
        AIntNumericBasicTypeIR a = new AIntNumericBasicTypeIR();
        a.setNamedInvType(n.clone());
```

```
        return a;
    }



    public STypeIR caseARecordTypeDeclIR(ARecordDeclIR n)
    {
        return null;
    }

    @Override
    public STypeIR createNewReturnValue(INode node) throws AnalysisException {
        return null;
    }

    @Override
    public STypeIR createNewReturnValue(Object node) throws AnalysisException {
        return null;
    }
}
```

A.1.5 IsaInvExpGen

```
/*
Generates the expression for an invariant.
Example:
    VDM spec: types
              test = nat
    Invariant expression: isa_inv_VDMNat i
    where i is a parameter to this visitor.
 */
public class IsaInvExpGen extends AnswerIsaAdaptor<SExpIR> {

    AIdentifierPatternIR ps;
    AMethodTypeIR methodType;
    private final Map<String, AFuncDeclIR> isaFuncDeclIRMap;

    public IsaInvExpGen(AIdentifierPatternIR ps, AMethodTypeIR methodType,
        Map<String, AFuncDeclIR> isaFuncDeclIRMap)
    {
        this.ps = ps;
        this.methodType = methodType;
        this.isaFuncDeclIRMap = isaFuncDeclIRMap;
    }

    public static SExpIR apply(SDeclIR decl, AIdentifierPatternIR afp,
        AMethodTypeIR methodType, Map<String, AFuncDeclIR> isaFuncDeclIRMap)
        throws AnalysisException {
        IsaInvExpGen finder = new IsaInvExpGen(afp, methodType, isaFuncDeclIRMap);
        return decl.apply(finder);
    }

    @Override
    public SExpIR caseANamedTypeDeclIR(ANamedTypeDeclIR node) throws
        AnalysisException {
```

```
        STypeIR type = node.getType();

        // Find invariant function
        AFuncDeclIR fInv = this.isaFuncDeclIRMap.get("isa_invTrue");

        // Create ref to function
        AIdentifierVarExpIR fInvIdentifier = new AIdentifierVarExpIR();
        fInvIdentifier.setName(fInv.getName());
        fInvIdentifier.setSourceNode(fInv.getSourceNode());
        fInvIdentifier.setType(fInv.getMethodType());

        // Crete apply expr
        AApplyExpIR exp = new AApplyExpIR();
        exp.setType(new ABoolBasicTypeIR());
        AIdentifierVarExpIR iVarExp = new AIdentifierVarExpIR();
        iVarExp.setName(this.ps.getName());
        iVarExp.setType(this.methodType);
        exp.getArgs().add(iVarExp);
        exp.setRoot(fInvIdentifier);

        return exp;
    }

    @Override
    public SExpIR caseARecordDeclIR(ARecordDeclIR node) throws AnalysisException
        {
        throw new AnalysisException();
    }

    @Override
    public SExpIR createNewReturnValue(INode node) throws AnalysisException {
        return null;
    }

    @Override
    public SExpIR createNewReturnValue(Object node) throws AnalysisException {
            return null;
    }


    public SExpIR caseASeqSeqType(ASeqSeqTypeIR node)
            throws AnalysisException {
        if(node.getSeqOf().getTag()!= null)
        {
            Object t = node.getSeqOf().getTag();

            // We are referring to another type, and therefore we stop here. This
                is the instantiation of the polymorphic function.
            /*
            For VDM:
             */
            // Return expression corresponding to:
                isa_invSeqElemens[token](isa_true[token], p)
        }
        else {
            //We need to keep going
        }
        throw new AnalysisException();
```

```
    }

    public SExpIR caseATokenBasicTypeIR(ATokenBasicTypeIR n) throws
        AnalysisException
    {

        AApplyExp e = new AApplyExp();

        throw new AnalysisException();

    }


    public SExpIR caseASetSetTypeIR(ASetSetTypeIR node) throws AnalysisException
        {
        throw new AnalysisException();

    }


}
```

A.1.6 IsaInvNameFinder

```
public class IsaInvNameFinder extends AnswerIsaAdaptor<String>
{
    public static String findName(INode node) throws AnalysisException {
        IsaInvNameFinder finder = new IsaInvNameFinder();
        return node.apply(finder);
    }

    @Override
    public String caseANamedTypeDeclIR(ANamedTypeDeclIR node) throws
        AnalysisException {
        return node.getName().getName();
    }

    @Override
    public String caseARecordDeclIR(ARecordDeclIR node) throws AnalysisException
        {
        return node.getName();
    }

    @Override
    public String createNewReturnValue(INode node) throws AnalysisException {
        return null;
    }

    @Override
    public String createNewReturnValue(Object node) throws AnalysisException {
        return null;
    }
}
```

A.2 CodeGen Platform

A.2.1 IsaGen

```
/*
 * #%~
 * VDM to Isabelle Translation
 * %%
 * Copyright (C) 2008 - 2015 Overture
 * %%
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
 * License along with this program. If not, see
 * <http://www.gnu.org/licenses/gpl-3.0.html>.
 * #~%
 */

package org.overturetool.cgisa;

import java.io.File;
import java.io.StringReader;
import java.io.StringWriter;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;

import org.apache.velocity.Template;
import org.apache.velocity.app.Velocity;
import org.apache.velocity.runtime.RuntimeServices;
import org.apache.velocity.runtime.RuntimeSingleton;
import org.apache.velocity.runtime.parser.ParseException;
import org.apache.velocity.runtime.parser.node.SimpleNode;
import org.overture.ast.analysis.AnalysisException;
import org.overture.ast.definitions.SClassDefinition;
import org.overture.ast.expressions.PExp;
import org.overture.ast.modules.AModuleModules;
import org.overture.codegen.ir.*;
import org.overture.codegen.ir.declarations.AModuleDeclIR;
import org.overture.codegen.merging.MergeVisitor;
import org.overture.codegen.utils.GeneratedData;
import org.overture.codegen.utils.GeneratedModule;
import org.overture.typechecker.util.TypeCheckerUtil;
import org.overturetool.cgisa.transformations.*;

/**
 * Main facade class for VDM 2 Isabelle IR
 *
 * @author ldc
```

```
 */
public class IsaGen extends CodeGenBase {

    public IsaGen()
    {
        this.addInvTrueMacro();

        this.getSettings().setAddStateInvToModule(false);
        this.getSettings().setGenerateInvariants(true);
    }
    //TODO: Auto load files in macro directory
    public static void addInvTrueMacro(){
        StringBuilder sb = new StringBuilder("#macro ( invTrue $node )\n" +
                "    definition\n" +
                "        inv_$node.Name :: $node.Name \\<RightArrow> \\<bool>\n" +
                "        where\n" +
                "        \"inv_$node.Name \\<equiv> inv_True\"\n" +
                "#end");
        addMacro("invTrue",new StringReader(sb.toString()));
        Template template = new Template();
    }

    public static void addMacro(String name, StringReader reader){
        try {
            Template template = new Template();
            RuntimeServices runtimeServices =
                RuntimeSingleton.getRuntimeServices();

            SimpleNode simpleNode = runtimeServices.parse(reader, name);
            template.setRuntimeServices(runtimeServices);
            template.setData(simpleNode);
            template.initDocument();
        } catch (ParseException e)
        {
            System.out.println("Failed with: " + e);
        }
    }

    public static String vdmExp2IsaString(PExp exp) throws AnalysisException,
            org.overture.codegen.ir.analysis.AnalysisException {
        IsaGen ig = new IsaGen();
        GeneratedModule r = ig.generateIsabelleSyntax(exp);
        if (r.hasMergeErrors()) {
            throw new
                org.overture.codegen.ir.analysis.AnalysisException(exp.toString()
                   + " cannot be generated. Merge errors:"
                   + r.getMergeErrors().toString());
        }
        if (r.hasUnsupportedIrNodes()) {
            throw new
                org.overture.codegen.ir.analysis.AnalysisException(exp.toString()
                   + " cannot be generated. Unsupported in IR:"
                   + r.getUnsupportedInIr().toString());
        }
        if (r.hasUnsupportedTargLangNodes()) {
            throw new
                org.overture.codegen.ir.analysis.AnalysisException(exp.toString()
                   + " cannot be generated. Unsupported in TargLang:"
```

```
                + r.getUnsupportedInTargLang().toString());
        }

        return r.getContent();
    }


    /**
     * Main entry point into the Isabelle Translator component. Takes an AST and
         returns corresponding Isabelle Syntax.
     *
     * @param statuses The IR statuses holding the nodes to be code generated.
     * @return The generated Isabelle syntax
     * @throws AnalysisException
     *
     */
    @Override
    protected GeneratedData genVdmToTargetLang(List<IRStatus<PIR>> statuses)
        throws AnalysisException {


        // Typecheck the VDMToolkit module and generate the IR
        TypeCheckerUtil.TypeCheckResult<List<AModuleModules>>
            listTypeCheckResult1 =
               TypeCheckerUtil.typeCheckSl(new
                   File("src/test/resources/VDMToolkit.vdmsl"));
        AModuleModules isaToolkit = listTypeCheckResult1.result.
               stream().
               filter(mod -> mod.getName().getName().equals("VDMToolkit")).
               findAny().
               orElseThrow(() -> new AnalysisException("Failed to find VDMToolkit
                   module"));
        super.genIrStatus(statuses, isaToolkit);

        // Get the VDMToolkit module IR
        IRStatus<PIR> vdmToolkitIR = statuses.stream().filter(x ->
            x.getIrNodeName().equals("VDMToolkit")).findAny().orElseThrow(() ->
            new AnalysisException("Failed to find VDMToolkit IR node"));
        AModuleDeclIR vdmToolkitModuleIR = (AModuleDeclIR)
            vdmToolkitIR.getIrNode();


        GeneratedData r = new GeneratedData();
        try {


           // Apply transformations
           for (IRStatus<PIR> status : statuses) {
               if(status.getIrNodeName().equals("VDMToolkit")){
                   System.out.println("");
               } else {


                   // make init expression an op
                   StateInit stateInit = new StateInit(getInfo());
                   generator.applyPartialTransformation(status, stateInit);

                   // transform away any recursion cycles
```

```
                GroupMutRecs groupMR = new GroupMutRecs();
                generator.applyTotalTransformation(status, groupMR);

                if (status.getIrNode() instanceof AModuleDeclIR) {
                    AModuleDeclIR cClass = (AModuleDeclIR) status.getIrNode();
                    // then sort remaining dependencies
                    SortDependencies sortTrans = new
                        SortDependencies(cClass.getDecls());
                    generator.applyPartialTransformation(status, sortTrans);
                }


                // Transform all token types to isa_VDMToken
                // Transform all nat types to isa_VDMNat
                // Transform all nat1 types to isa_VDMNat
                // Transform all int types to isa_VDMInt

                IsaBasicTypesConv invConv = new IsaBasicTypesConv(getInfo(),
                    this.transAssistant, vdmToolkitModuleIR);
                generator.applyPartialTransformation(status, invConv);

                IsaInvGenTrans invTrans = new IsaInvGenTrans(getInfo(),
                    vdmToolkitModuleIR);
                generator.applyPartialTransformation(status, invTrans);
            }
        }

        r.setClasses(prettyPrint(statuses));
    } catch (org.overture.codegen.ir.analysis.AnalysisException e) {
        throw new AnalysisException(e);
    }
    return r;

}

public GeneratedModule generateIsabelleSyntax(PExp exp)
        throws AnalysisException,
        org.overture.codegen.ir.analysis.AnalysisException {
    IRStatus<SExpIR> status = this.generator.generateFrom(exp);

    if (status.canBeGenerated()) {
        return prettyPrint(status);
    }

    throw new
        org.overture.codegen.ir.analysis.AnalysisException(exp.toString()
            + " cannot be code-generated");
}


private List<GeneratedModule> prettyPrint(List<IRStatus<PIR>> statuses)
        throws org.overture.codegen.ir.analysis.AnalysisException {
    // Apply merge visitor to pretty print Isabelle syntax
    IsaTranslations isa = new IsaTranslations();
    MergeVisitor pp = isa.getMergeVisitor();

    List<GeneratedModule> generated = new ArrayList<GeneratedModule>();
```

```
        for (IRStatus<PIR> status : statuses) {
            if(status.getIrNodeName().equals("VDMToolkit")){
                System.out.println("");
            } else {
                generated.add(prettyPrintNode(pp, status));
            }

        }

        // Return syntax
        return generated;
    }

    private GeneratedModule prettyPrint(IRStatus<? extends INode> status)
            throws org.overture.codegen.ir.analysis.AnalysisException {
        // Apply merge visitor to pretty print Isabelle syntax
        IsaTranslations isa = new IsaTranslations();
        MergeVisitor pp = isa.getMergeVisitor();
        return prettyPrintNode(pp, status);
    }

    private GeneratedModule prettyPrintNode(MergeVisitor pp,
                                            IRStatus<? extends INode> status)
            throws org.overture.codegen.ir.analysis.AnalysisException {
        INode irClass = status.getIrNode();

        StringWriter sw = new StringWriter();

        irClass.apply(pp, sw);

        if (pp.hasMergeErrors()) {
            return new GeneratedModule(status.getIrNodeName(), irClass,
                pp.getMergeErrors(), false);
        } else if (pp.hasUnsupportedTargLangNodes()) {
            return new GeneratedModule(status.getIrNodeName(), new
                HashSet<VdmNodeInfo>(), pp.getUnsupportedInTargLang(), false);
        } else {
            // Code can be generated. Ideally, should format it
            GeneratedModule generatedModule = new
                GeneratedModule(status.getIrNodeName(), irClass, sw.toString(),
                false);
            generatedModule.setTransformationWarnings(status.getTransformationWarnings());
            return generatedModule;
        }
    }
}
```

Appendix B

Tool Code After Development

B.1 Transformations

B.1.1 IsaInvNameFinder

```
public class IsaInvNameFinder extends AnswerIsaAdaptor<String>
{
    public static String findName(INode node) throws AnalysisException {
        IsaInvNameFinder finder = new IsaInvNameFinder();
        return node.apply(finder);
    }

    @Override
    public String caseANamedTypeDeclIR(ANamedTypeDeclIR node) throws
        AnalysisException {
        return node.getName().getName();
    }

    @Override
    public String caseANotImplementedExpIR(ANotImplementedExpIR node) {
    return "True";

    }

    @Override
    public String caseAStateDeclIR(AStateDeclIR node) throws AnalysisException {
        return node.getName();
    }

    @Override
    public String caseASetSetTypeIR(ASetSetTypeIR node) throws AnalysisException
        {
        return "SetElems";
    }

    @Override
    public String caseASeqSeqTypeIR(ASeqSeqTypeIR node) throws AnalysisException
        {
      ANamedTypeDeclIR n = new ANamedTypeDeclIR();
      return "SeqElems";
    }
    @Override
```

```
public String caseANatNumericBasicTypeIR(ANatNumericBasicTypeIR node) throws
    AnalysisException {
  return "VDMNat";
}

@Override
public String caseAIntNumericBasicTypeIR(AIntNumericBasicTypeIR node) throws
    AnalysisException {
  return "True";
}

@Override
public String caseARealNumericBasicTypeIR(ARealNumericBasicTypeIR node)
    throws AnalysisException {
  return "True";
}

@Override
public String caseARatNumericBasicTypeIR(ARatNumericBasicTypeIR node) throws
    AnalysisException {
  return "True";
}


@Override
public String caseABoolBasicTypeIR(ABoolBasicTypeIR node) throws
    AnalysisException {
  return "True";
}

@Override
public String caseACharBasicTypeIR(ACharBasicTypeIR node) throws
    AnalysisException {
  return "True";
}

@Override
public String caseAMapMapTypeIR(AMapMapTypeIR node) throws AnalysisException
    {
  return "True";
}

@Override
public String caseATokenBasicTypeIR(ATokenBasicTypeIR node) throws
    AnalysisException {
  return "True";
}

@Override
public String caseANat1NumericBasicTypeIR(ANat1NumericBasicTypeIR node)
    throws AnalysisException {
  return "VDMNat1";
}



@Override
```

```
    public String caseARecordDeclIR(ARecordDeclIR node) throws AnalysisException
        {
        return node.getName();
    }

    @Override
    public String createNewReturnValue(INode node) throws AnalysisException {
      String typeName;
        STypeIR n = (STypeIR) node;
        //if not a toolkit or IR node type
      if (n.getNamedInvType() == null) typeName = "True";
      else typeName = n.getNamedInvType().getName().getName();
      return typeName;
    }

    @Override
    public String createNewReturnValue(Object node) throws AnalysisException {
      String typeName;
        STypeIR n = (STypeIR) node;
        //if not a toolkit or IR node type
      if (n.getNamedInvType() == null) typeName = "True";
      else typeName = n.getNamedInvType().getName().getName();
      return typeName;
    }
}
```

B.1.2 IsaFuncDeclConv

```
public class IsaFuncDeclConv extends DepthFirstAnalysisIsaAdaptor {


    private final AModuleDeclIR vdmToolkitModuleIR;
    private final Map<String, AFuncDeclIR> isaFuncDeclIRMap;

    public IsaFuncDeclConv(IRInfo info, TransAssistantIR t, AModuleDeclIR
        vdmToolkitModuleIR) {
        this.vdmToolkitModuleIR = vdmToolkitModuleIR;

        this.vdmToolkitModuleIR.getDecls()
                .stream()
                .filter(d -> {
                    if (d instanceof ATypeDeclIR)
                        return true;
                    else
                        return false;
                }).map(d -> (ATypeDeclIR) d)
                .collect(Collectors.toMap(x -> ((ANamedTypeDeclIR)
                    x.getDecl()).getName().getName(), x -> x));

        this.isaFuncDeclIRMap =
            this.vdmToolkitModuleIR.getDecls().stream().filter(d ->
        {
            if (d instanceof AFuncDeclIR)
                return true;
            else
                return false;
```

```
    }).map(d -> (AFuncDeclIR) d).collect(Collectors.toMap(x -> x.getName(), x
        -> x));


}


// Transform AFuncDeclIR
@Override
public void caseAFuncDeclIR(AFuncDeclIR x) throws AnalysisException {
  super.caseAFuncDeclIR(x);
  //we need to stop post conditions of postconditions of post conditions...
      being formed
  if (!x.getName().contains("inv") &&
      !x.getName().contains("post") && !x.getName().contains("pre"))
  {

    if (x.parent() instanceof AStateDeclIR)
    {
      transStateInit(x);

    }
    else
    {
      transformPreConditions(x);

      transformPostConditions(x);

      // If no parameter function set params to null to make this more
          concrete for velocity
      if (x.getFormalParams().size() == 0)
      {
        x.getMethodType().setParams(null);
      }

      formatIdentifierPatternVars(x);
      if (x.getImplicit()) removeFromAST(x);
    }



  }
}

private void transStateInit(AFuncDeclIR node) {
  AStateDeclIR st = node.getAncestor(AStateDeclIR.class);

  AMethodTypeIR methodType = new AMethodTypeIR();


  st.getFields().forEach(f ->
      methodType.getParams().add(f.getType().clone()));
  methodType.setResult(new ABoolBasicTypeIR());

  AFuncDeclIR postInit = new AFuncDeclIR();
  postInit.setMethodType(methodType.clone());
  postInit.setName("post_"+node.getName());
```

```
    AApplyExpIR app = new AApplyExpIR();
    AIdentifierVarExpIR root = new AIdentifierVarExpIR();
    root.setName("inv_"+st.getName());
    System.out.println(IsaGen.funcGenHistoryMap.keySet());
    root.setType(IsaGen.funcGenHistoryMap.get("inv_"+st.getName()).getMethodType().clone());
    app.setRoot(root);

    AIdentifierVarExpIR arg = new AIdentifierVarExpIR();
    arg.setName(node.getName());
    arg.setType(node.getMethodType().clone());
    app.getArgs().add(arg);

    postInit.setBody(app);
      addToAST(postInit, node);

      System.out.println("Post condition has been added");

}


private void removeFromAST(AFuncDeclIR x) {
    // Insert into AST
      AModuleDeclIR encModule = x.getAncestor(AModuleDeclIR.class);
      if(encModule != null)
      {
          encModule.getDecls().remove(x);
      }

}

  private void addToAST(INode node, INode parent) {
    // Insert into AST
      AModuleDeclIR encModule = parent.getAncestor(AModuleDeclIR.class);
      if(encModule != null)
      {
          encModule.getDecls().add((SDeclIR) node);
      }


}


private void transformPreConditions (AFuncDeclIR node) throws
    AnalysisException {
    AMethodTypeIR mt = node.getMethodType().clone();

    /*The final pre condition that will be populated with a generated pre
       condition,
    a modeller written pre condition or both or neither.*/
    AFuncDeclIR finalPreCondition = null;

    //Generated pre condition will be populated if one can be generated
    AFuncDeclIR generatedPre = null;

      // If there are parameters with which to build a pre condition then build
         one
    if (!mt.getParams().isEmpty())
    {
```

```
  generatedPre = createPre(node.clone());
  //Copy across all generated properties into final pre condition.
  finalPreCondition = generatedPre;

}

  // If there are pre written pre conditions and one was generated add them
      both
if (node.getPreCond() != null && generatedPre != null)
{
  AFuncDeclIR preCond_ = (AFuncDeclIR) node.getPreCond();

  AAndBoolBinaryExpIR andExisting = new AAndBoolBinaryExpIR();
  andExisting.setLeft(generatedPre.getBody());
  andExisting.setRight(preCond_.getBody());
  finalPreCondition.setBody(andExisting);
}

//If there is only a pre written pre condition add that
else if (node.getPreCond() != null && generatedPre == null)
{
  //Copy across all pre written properties into final pre condition.
  finalPreCondition = new AFuncDeclIR();

  //No need to add formal params again they're all already put there above
  AFuncDeclIR preCond_ = (AFuncDeclIR) node.getPreCond();
  finalPreCondition.setBody(preCond_.getBody());
  finalPreCondition.setFormalParams(preCond_.getFormalParams());
  finalPreCondition.setMethodType(preCond_.getMethodType());
  finalPreCondition.setName(preCond_.getName());
}
 /* If no pre condition is written, none has been generated then
  there are no parameter types to use as invariant checks and no relevant
      checks provided
  by modeller, so pre condition is added but left empty as a reminder to
      the modeller to add one later.*/
else if (node.getPreCond() == null && generatedPre == null)
{
  //Copy across all pre written properties into final pre condition.
  finalPreCondition = new AFuncDeclIR();
  ANotImplementedExpIR n = new ANotImplementedExpIR();
  n.setTag("TODO");
  finalPreCondition.setBody(n);
  // Set up method type for post condition
    AMethodTypeIR mty = new AMethodTypeIR();
    mty.setResult(new ABoolBasicTypeIR());
mty.setParams(null);
    finalPreCondition.setMethodType(mty);
  finalPreCondition.setName("unimplemented_pre_"+node.getName());
}

formatIdentifierPatternVars(finalPreCondition);
node.setPreCond(finalPreCondition);
IsaGen.funcGenHistoryMap.put(finalPreCondition.getName(),
    finalPreCondition.clone());
addToAST(finalPreCondition, node);

System.out.println("Pre condition has been added");
```

```
}



private void transformPostConditions (AFuncDeclIR node) throws
    AnalysisException {
  AMethodTypeIR mt = node.getMethodType().clone();

  /*The final post condition that will be populated with a generated post
      condition,
  a modeller written post condition or both or neither.*/
  AFuncDeclIR finalPostCondition = null;

  //Generated post condition will be populated if one can be generated
  AFuncDeclIR generatedPost = null;

    // If there are parameters and results with which to build a post
        condition then build one
  if (!mt.getParams().isEmpty() && mt.getResult() != null)
  {
    generatedPost = createPost(node.clone());
    //Copy across all generated properties into final post condition.
      finalPostCondition = generatedPost;

  }

  // If there are pre written post conditions and one was generated add them
      both
    if (node.getPostCond() != null && generatedPost != null)
    {
      AFuncDeclIR postCond_ = (AFuncDeclIR) node.getPostCond();


      AAndBoolBinaryExpIR andExisting = new AAndBoolBinaryExpIR();
      andExisting.setLeft(generatedPost.getBody());
      andExisting.setRight(postCond_.getBody());
      finalPostCondition.setBody(andExisting);

    }

    //If there is only a pre written post condition add that
    else if (node.getPostCond() != null && generatedPost == null)
    {
      //Copy across all pre written properties into final post condition.
      finalPostCondition = new AFuncDeclIR();

      //No need to add formal params again they're all already put there above
      AFuncDeclIR postCond_ = (AFuncDeclIR) node.getPostCond();
      finalPostCondition.setBody(postCond_.getBody());
      finalPostCondition.setFormalParams(postCond_.getFormalParams());
      finalPostCondition.setMethodType(postCond_.getMethodType());
      finalPostCondition.setName(postCond_.getName());
    }
   /* If no post condition is written, none has been generated then
    there are no parameter types to use as invariant checks and no relevant
        checks provided
```

```
    by modeller, so post condition is added but left empty as a reminder to
        the modeller to add one later.*/
    else if (node.getPostCond() == null && generatedPost == null)
    {
      //Copy across all pre written properties into final post condition.
      finalPostCondition = new AFuncDeclIR();
      ANotImplementedExpIR n = new ANotImplementedExpIR();
      n.setTag("TODO");
      finalPostCondition.setBody(n);

      AMethodTypeIR mty = new AMethodTypeIR();
      mty.setResult(new ABoolBasicTypeIR());
  mty.getParams().add(mt.getResult());
      finalPostCondition.setMethodType(mty);
      finalPostCondition.setName("unimplemented_post_"+node.getName());
    }

    formatIdentifierPatternVars(finalPostCondition);
    node.setPostCond(finalPostCondition);
  IsaGen.funcGenHistoryMap.put(finalPostCondition.getName(),
       finalPostCondition.clone());
    addToAST(finalPostCondition, node);

    System.out.println("Post condition has been added");
}
```

```
private AFuncDeclIR createPre(AFuncDeclIR node) throws AnalysisException {
  // Post condition function
    AFuncDeclIR preCond = new AFuncDeclIR();
  AMethodTypeIR mt = node.getMethodType();
    SExpIR expr;

   // Set post_[function name] as post function name
  preCond.setName("pre_" + node.getName());

  // Set up method type for post condition
    AMethodTypeIR type = new AMethodTypeIR();
    type.setResult(new ABoolBasicTypeIR());
type.setParams(mt.getParams());
    preCond.setMethodType(type);



    AIdentifierPatternIR identifierPattern = new AIdentifierPatternIR();
    identifierPattern.setName("");
    if (node.getFormalParams() != null && !node.getFormalParams().isEmpty())
    {
      // Loop through all but result type
      for (int i = 0; i < preCond.getMethodType().getParams().size(); i++)
      {
        identifierPattern = new AIdentifierPatternIR();
          identifierPattern.setName(node.getFormalParams().get(i).getPattern().toString());
        AFormalParamLocalParamIR afp = new AFormalParamLocalParamIR();
        afp.setPattern(identifierPattern);
```

```
        afp.setType(preCond.getMethodType().getParams().get(i).clone());
        preCond.getFormalParams().add(afp);
      }
    }
  expr = IsaInvExpGen.apply(preCond.clone(), identifierPattern,
      preCond.getMethodType().clone(), isaFuncDeclIRMap);
    preCond.setBody(expr);
    return preCond;
  }


private AFuncDeclIR createPost(AFuncDeclIR node) throws AnalysisException {
    // Post condition function
    AFuncDeclIR postCond = new AFuncDeclIR();
  AMethodTypeIR mt = node.getMethodType();
    SExpIR expr;

   // Set post_[function name] as post function name
  postCond.setName("post_" + node.getName());

  // Set up method type for post condition
    AMethodTypeIR type = new AMethodTypeIR();
    type.setResult(new ABoolBasicTypeIR());
    List<STypeIR> params = mt.getParams();
    params.add(mt.getResult().clone());
type.setParams(params);
    postCond.setMethodType(type);



    AIdentifierPatternIR identifierPattern = new AIdentifierPatternIR();
    identifierPattern.setName("");
    if (node.getFormalParams() != null && !node.getFormalParams().isEmpty())
    {
      // Loop through all but result type
      for (int i = 0; i < postCond.getMethodType().getParams().size() -1; i++)
      {
        identifierPattern = new AIdentifierPatternIR();
          identifierPattern.setName(node.getFormalParams().get(i).getPattern().toString());
        AFormalParamLocalParamIR afp = new AFormalParamLocalParamIR();
        afp.setPattern(identifierPattern);
        afp.setType(postCond.getMethodType().getParams().get(i).clone());
        postCond.getFormalParams().add(afp);
      }
    }

    // Add RESULT pattern if the function has a result
    if (mt.getResult() != null)
    {
      identifierPattern = new AIdentifierPatternIR();
      if (node.getPostCond() != null)
        identifierPattern.setName(((AFuncDeclIR)
            node.getPostCond()).getFormalParams().getLast().getPattern().toString());
      else
        identifierPattern.setName("RESULT");
      AFormalParamLocalParamIR afp = new AFormalParamLocalParamIR();
      afp.setPattern(identifierPattern);
      afp.setType(mt.getResult());
```

```
        postCond.getFormalParams().add(afp);
      }
   //an and expression of all of the parameter invariants do nothing if the
       body is not implemented
      expr = IsaInvExpGen.apply(postCond.clone(), identifierPattern,
          postCond.getMethodType().clone(), isaFuncDeclIRMap);
      postCond.setBody(expr);


      return postCond;
  }


  /*space out identifier variables, e.g. for two variable t and x we should
      have t x not tx.
    "inv_t t x \<equiv> isa_invTrue t \<and> isa_invTrue x"*/
   private AFuncDeclIR formatIdentifierPatternVars (AFuncDeclIR node) {
    /* This puts a space between different parameters in the Isabelle function
        body
    , xy is misinterpreted as one variable whereas x y is correctly interpreted
        as two
    */
    node.getFormalParams().forEach
    (
       p -> {

         AIdentifierPatternIR ip = new AIdentifierPatternIR();
         ip.setName(p.getPattern().toString() + " ");
         p.setPattern(ip);

       }
    );

    return node;
   }

}
```

B.1.3 IsaBasicTypesConv

```
/***
 * Visitor to convert sequence or set VDM types to VDMToolkit types
 */
public class IsaTypeTypesConv extends DepthFirstAnalysisIsaAdaptor {

   private final Map<String, ATypeDeclIR> isaTypeDeclIRMap;
   private final TransAssistantIR t;
   private final AModuleDeclIR vdmToolkitModuleIR;
   private final IRInfo info;

   private final static String isa_VDMSet = "isa_VDMSet";

   private final static String isa_VDMSeq = "isa_VDMSeq";

   public IsaTypeTypesConv(IRInfo info, TransAssistantIR t, AModuleDeclIR
       vdmToolkitModuleIR) {
```

```
        this.t = t;
        this.info = info;
        this.vdmToolkitModuleIR = vdmToolkitModuleIR;

        this.isaTypeDeclIRMap = this.vdmToolkitModuleIR.getDecls()
                .stream()
                .filter(d -> {
                    if (d instanceof ATypeDeclIR)
                        return true;
                    else
                        return false;
                }).map(d -> (ATypeDeclIR) d)
                .collect(Collectors.toMap(x -> ((ANamedTypeDeclIR)
                    x.getDecl()).getName().getName(), x -> x));
    }

   //transform seq into VDMSeq
    public void caseASeqSeqTypeIR(ASeqSeqTypeIR x) {
      if(x.getNamedInvType() == null)
        {

            // Retrieve isa_VDMSeq from VDMToolkit
            ATypeDeclIR isa_td = isaTypeDeclIRMap.get(IsaTypeTypesConv.isa_VDMSeq);

            x.setNamedInvType((ANamedTypeDeclIR)isa_td.getDecl().clone());

        }
    }
  //transform set into VDMSet
    public void caseASetSetTypeIR(ASetSetTypeIR x) {
      if(x.getNamedInvType() == null)
        {

            // Retrieve isa_VDMSet from VDMToolkit
            ATypeDeclIR isa_td = isaTypeDeclIRMap.get(IsaTypeTypesConv.isa_VDMSet);

            x.setNamedInvType((ANamedTypeDeclIR)isa_td.getDecl().clone());
        }
    }



}
```

B.1.4 IsaTypeTypesConv

B.1.5 IsaInvExpGen

```
/*
Generates the expression for an invariant.
Example:
    VDM spec: types
              test = nat
    Invariant expression: isa_inv_VDMNat i
    where i is a parameter to this visitor.
```

```
 */
public class IsaInvExpGen extends AnswerIsaAdaptor<SExpIR> {

    AIdentifierPatternIR ps;
    AMethodTypeIR methodType;

    private final Map<String, AFuncDeclIR> isaFuncDeclIRMap;
  private AIdentifierVarExpIR targetIP;
  private final LinkedList<ANamedTypeDeclIR> invArr = new
      LinkedList<ANamedTypeDeclIR>();


    public IsaInvExpGen(AIdentifierPatternIR ps, AMethodTypeIR methodType,
        Map<String, AFuncDeclIR> isaFuncDeclIRMap)
    {
        this.ps = ps;
        this.methodType = methodType;
        this.isaFuncDeclIRMap = isaFuncDeclIRMap;
    }

    public static SExpIR apply(SDeclIR decl, AIdentifierPatternIR afp,
        AMethodTypeIR methodType, Map<String, AFuncDeclIR> isaFuncDeclIRMap)
        throws AnalysisException {
        IsaInvExpGen finder = new IsaInvExpGen(afp, methodType, isaFuncDeclIRMap);
        return decl.apply(finder);
    }

    @Override
    public SExpIR caseANamedTypeDeclIR(ANamedTypeDeclIR node) throws
        AnalysisException {
        node.getType();
        //TODO make for different types invariants
        // Find invariant function
        AFuncDeclIR fInv = this.isaFuncDeclIRMap.get("isa_invTrue");
        // Create ref to function
        AIdentifierVarExpIR fInvIdentifier = new AIdentifierVarExpIR();
        fInvIdentifier.setName(fInv.getName());
        fInvIdentifier.setSourceNode(fInv.getSourceNode());
        fInvIdentifier.setType(fInv.getMethodType());

        // Crete apply expr
        AApplyExpIR exp = new AApplyExpIR();
        exp.setType(new ABoolBasicTypeIR());
        AIdentifierVarExpIR iVarExp = new AIdentifierVarExpIR();
        iVarExp.setName(this.ps.getName());
        iVarExp.setType(this.methodType);
        exp.getArgs().add(iVarExp);
        exp.setRoot(fInvIdentifier);

        return exp;
    }


    @Override
    public SExpIR caseAStateDeclIR(AStateDeclIR node) throws AnalysisException {
      //TODO e.g. where "inv_recType r \<equiv> isa_invVDMSeq isa_invVDMNat1 (x
          r) etc.
      LinkedList<AFieldDeclIR> fields = new LinkedList<AFieldDeclIR>();
```

```
node.getFields().forEach(f -> fields.add(f.clone()));
    AApplyExpIR completeExp = new AApplyExpIR();
    LinkedList<AApplyExpIR> fieldInvariants = new LinkedList<AApplyExpIR>();

    for (int i = 0; i < fields.size(); i++)
      {
        STypeIR type = fields.get(i).getType();
      AIdentifierVarExpIR invExp = new AIdentifierVarExpIR();
          invExp.setName("("+node.getName().substring(0,1).toLowerCase()+
              node.getName().toString().substring(1,
                  node.getName().toString().length())+"_"+
              fields.get(i).getName()+" "+this.ps.toString()+")");
          invExp.setType(this.methodType);
          this.targetIP = invExp;

          completeExp.setType(new ABoolBasicTypeIR());
          //Recursively build curried inv function e.g. (inv_VDMSet
              (inv_VDMSet inv_Nat1)) inv_x

          try {
      fieldInvariants.add(buildInvForType(type.clone()));
    } catch (AnalysisException e) {
      e.printStackTrace();
    }

      }

 // Link numerous apply expressions together in an and expression
    if (fieldInvariants.size() >= 2)
      return genAnd(fieldInvariants);
    else
    // Just one field return it as an apply expression
      return fieldInvariants.get(0);
}
```

```
@Override
public SExpIR caseARecordDeclIR(ARecordDeclIR node) throws AnalysisException
    {
  //TODO e.g. where "inv_recType r \<equiv> isa_invVDMSeq isa_invVDMNat1 (x
      r) etc.
  LinkedList<AFieldDeclIR> fields = new LinkedList<AFieldDeclIR>();
node.getFields().forEach(f -> fields.add(f.clone()));
    AApplyExpIR completeExp = new AApplyExpIR();
    LinkedList<AApplyExpIR> fieldInvariants = new LinkedList<AApplyExpIR>();

    for (int i = 0; i < fields.size(); i++)
      {
        STypeIR type = fields.get(i).getType();
      AIdentifierVarExpIR invExp = new AIdentifierVarExpIR();
          invExp.setName("("+node.getName().substring(0,1).toLowerCase()+
              node.getName().toString().substring(1,
                  node.getName().toString().length())+"_"+
              fields.get(i).getName()+" "+this.ps.toString()+")");
```

```
            invExp.setType(this.methodType);
            this.targetIP = invExp;

            completeExp.setType(new ABoolBasicTypeIR());
            //Recursively build curried inv function e.g. (inv_VDMSet
                (inv_VDMSet inv_Nat1)) inv_x

            try {
          fieldInvariants.add(buildInvForType(type.clone()));
        } catch (AnalysisException e) {
          e.printStackTrace();
        }

          }

     // Link numerous apply expressions together in an and expression
        if (fieldInvariants.size() >= 2)
          return genAnd(fieldInvariants);
        else
        // Just one field return it as an apply expression
          return fieldInvariants.get(0);
    }



    @Override
    public SExpIR caseAFieldDeclIR(AFieldDeclIR node) throws AnalysisException {
        STypeIR t = node.getType().clone();
        AApplyExpIR completeExp = new AApplyExpIR();
        // Crete apply to the inv_ expr e.g inv_x inv_y
        AIdentifierVarExpIR invExp = new AIdentifierVarExpIR();
        invExp.setName(node.getName());
        invExp.setType(this.methodType);
        this.targetIP = invExp;

        completeExp.setType(new ABoolBasicTypeIR());
        //Recursively build curried inv function e.g. (inv_VDMSet (inv_VDMSet
            inv_Nat1)) inv_x

    completeExp = buildInvForType(t);



    return completeExp;
    }


    @Override
    public SExpIR caseAFuncDeclIR(AFuncDeclIR node) throws AnalysisException {
        LinkedList<AFormalParamLocalParamIR> t = node.getFormalParams();
        node.setMethodType(this.methodType);
        LinkedList<AApplyExpIR> paramInvariants = new LinkedList<AApplyExpIR>();

        for (int i = 0; i < node.getFormalParams().size(); i++) {
          STypeIR type = t.get(i).getType();
          AApplyExpIR completeExp = new AApplyExpIR();

          // Create apply to the inv_ expr e.g inv_x inv_y
```

```
        AIdentifierVarExpIR invExp = new AIdentifierVarExpIR();
        invExp.setName(node.getFormalParams().get(i).getPattern().toString());
        invExp.setType(this.methodType);
        this.targetIP = invExp;

        completeExp.setType(new ABoolBasicTypeIR());
        //Recursively build curried inv function e.g. (inv_VDMSet (inv_VDMSet
            inv_Nat1)) inv_x

    try {
      completeExp = buildInvForType(type);
    } catch (AnalysisException e) {
      e.printStackTrace();
    }

    paramInvariants.add(completeExp);


      }


      // Link numerous apply expressions together in an and expression
      if (paramInvariants.size() >= 2)
        return genAnd(paramInvariants);
      else
      // Just one parameter return it as an apply expression
        return paramInvariants.get(0);



  }


  private SExpIR genAnd(LinkedList<AApplyExpIR> paramInvariants) {

    AAndBoolBinaryExpIR and = new AAndBoolBinaryExpIR();

    //base case
  if (paramInvariants.size() == 2)
      {
      and.setLeft(paramInvariants.get(0));
      and.setRight(paramInvariants.get(1));
      }
  else
    {
      and.setLeft(paramInvariants.get(0));
      paramInvariants.remove(0);
      and.setRight( genAnd(paramInvariants) );
    }
  return and;

}

//build curried invariant
  public AApplyExpIR buildInvForType(STypeIR seqtNode) throws
      AnalysisException {

    String typeName = IsaInvNameFinder.findName(seqtNode);
```

```
    AFuncDeclIR fInv;
    if (this.isaFuncDeclIRMap.get("isa_inv"+typeName) != null)
    {
      fInv = this.isaFuncDeclIRMap.get("isa_inv"+typeName).clone();
    }
    else
    {
      fInv = IsaGen.funcGenHistoryMap.get("inv_"+typeName).clone();

    }
    if (fInv.getMethodType() == null)
    {
      AMethodTypeIR mt = new AMethodTypeIR();
      mt.setResult(new ABoolBasicTypeIR());
      mt.getParams().add(seqtNode);
      fInv.setMethodType(mt.clone());
    }

       // Create ref to function
      AIdentifierVarExpIR curriedInv = new AIdentifierVarExpIR();
      curriedInv.setName(fInv.getName());
      curriedInv.setSourceNode(fInv.getSourceNode());
      curriedInv.setType(fInv.getMethodType().clone());//Must always clone
    AApplyExpIR accum = new AApplyExpIR();
    accum.setRoot(curriedInv);


    //if this type is not the last in the nested types, then keep rescursing
        until we get to the final nested type
    if ( seqtNode instanceof ASetSetTypeIR && ((ASetSetTypeIR)
        seqtNode).getSetOf() != null )
    {
      accum.getArgs().add(buildInvForType(((ASetSetTypeIR)
          seqtNode).getSetOf().clone()));
    }
    else if (seqtNode instanceof ASeqSeqTypeIR && ((ASeqSeqTypeIR)
        seqtNode).getSeqOf() != null)
    {

      accum.getArgs().add(buildInvForType(((ASeqSeqTypeIR)
          seqtNode).getSeqOf().clone()));
    }
    else
    {
      accum.getArgs().add(targetIP);
    }
    return accum;

}


  @Override
  public SExpIR createNewReturnValue(INode node) throws AnalysisException {
      return null;
  }

  @Override
```

```
    public SExpIR createNewReturnValue(Object node) throws AnalysisException {
           return null;
    }

    public SExpIR caseATokenBasicTypeIR(ATokenBasicTypeIR n) throws
        AnalysisException
    {

       new AApplyExp();

       throw new AnalysisException();

    }


    public SExpIR caseASetSetTypeIR(ASetSetTypeIR node) throws AnalysisException
        {
       throw new AnalysisException();

    }


}
```

B.1.6 IsaDeclTypeGen

```
public class IsaDeclTypeGen extends AnswerIsaAdaptor<STypeIR> {

    public static STypeIR apply(INode node) throws AnalysisException {
        IsaDeclTypeGen finder = new IsaDeclTypeGen();
        return node.apply(finder);
    }

    public STypeIR caseANamedTypeDeclIR(ANamedTypeDeclIR n)
    {
      IsaGen.typeGenHistoryMap.put(n.getType(), n.getName().toString());
        AIntNumericBasicTypeIR a = new AIntNumericBasicTypeIR();
        a.setNamedInvType(n.clone());
        return a;
    }

    public STypeIR caseAStateDeclIR(AStateDeclIR n)
    {
      ARecordTypeIR a = new ARecordTypeIR();
      ATypeNameIR o = new ATypeNameIR();
      o.setName(n.getName());
      a.setName(o);
        return a;

    }

    public STypeIR caseARecordDeclIR(ARecordDeclIR n)
    {
      ARecordTypeIR a = new ARecordTypeIR();
      ATypeNameIR o = new ATypeNameIR();
      o.setName(n.getName());
```

```
      a.setName(o);
        return a;

    }

    @Override
    public STypeIR createNewReturnValue(INode node) throws AnalysisException {
        return null;
    }

    @Override
    public STypeIR createNewReturnValue(Object node) throws AnalysisException {
        return null;
    }
}
```

B.1.7 IsaInvGenTrans

```
public class IsaInvGenTrans extends DepthFirstAnalysisIsaAdaptor {

    private final AModuleDeclIR vdmToolkitModule;
    private final Map<String, ATypeDeclIR> isaTypeDeclIRMap;
    private IRInfo info;
    private final Map<String, AFuncDeclIR> isaFuncDeclIRMap;

    public IsaInvGenTrans(IRInfo info, AModuleDeclIR vdmToolkitModuleIR) {
        this.info = info;
        this.vdmToolkitModule = vdmToolkitModuleIR;

        this.isaFuncDeclIRMap =
            this.vdmToolkitModule.getDecls().stream().filter(d ->
        {
            if (d instanceof AFuncDeclIR)
                return true;
            else
                return false;
        }).map(d -> (AFuncDeclIR) d).collect(Collectors.toMap(x -> x.getName(), x
            -> x));

        this.isaTypeDeclIRMap =
            this.vdmToolkitModule.getDecls().stream().filter(d -> {
            if (d instanceof ATypeDeclIR)
                return true;
            else
                return false;
        }).map(d -> (ATypeDeclIR) d).collect(Collectors.toMap(x ->
            ((ANamedTypeDeclIR) x.getDecl()).getName().getName(), x -> x));


    }


    @Override
    public void caseAStateDeclIR(AStateDeclIR node) throws AnalysisException {
      super.caseAStateDeclIR(node);
```

```
SDeclIR decl = node.clone();
String typeName = IsaInvNameFinder.findName(node.clone());
SExpIR invExp = node.getInvExp();
  // Invariant function
  AFuncDeclIR invFun_ = new AFuncDeclIR();
  invFun_.setName("inv_" + typeName); //inv_t

  AMethodTypeIR methodType = new AMethodTypeIR();

  STypeIR t = IsaDeclTypeGen.apply(decl.clone());
  methodType.getParams().add(t.clone());


methodType.setResult(new ABoolBasicTypeIR());
  invFun_.setMethodType(methodType);




  // Translation for VDMToolkit and modeller written invariants
  if (invExp != null)
  {
    AAndBoolBinaryExpIR multipleInvs = new AAndBoolBinaryExpIR();
    //change (a_c a) to (c A) for Isabelle field access
      //if (decl instanceof ARecordDeclIR)
          formatExistingRecordInvExp(inv.getBody());

      multipleInvs.setRight(invExp);

AIdentifierPatternIR identifierPattern = new AIdentifierPatternIR();
identifierPattern.setName(typeName.substring(0, 1).toLowerCase());

//set Inv pattern if one does not exist
if (node.getInvPattern() != null) node.setInvPattern(identifierPattern);

SExpIR expr = IsaInvExpGen.apply(decl,
    identifierPattern ,
    methodType.clone(), isaFuncDeclIRMap);

multipleInvs.setLeft(expr);

    invFun_.setBody(multipleInvs);
    node.setInvExp(multipleInvs);
  }
  //translation for no inv types
  else
  {

    SExpIR expr;
AIdentifierPatternIR identifierPattern = new AIdentifierPatternIR();
    identifierPattern.setName(typeName.substring(0, 1).toLowerCase());
    AFormalParamLocalParamIR afp = new AFormalParamLocalParamIR();
    afp.setPattern(identifierPattern.clone());
    afp.setType(t.clone());

    node.setInvPattern(identifierPattern);

    invFun_.getFormalParams().add(afp);
```

```
      expr = IsaInvExpGen.apply(decl.clone(), identifierPattern,
          methodType.clone(), isaFuncDeclIRMap);


      invFun_.setBody(expr.clone());
      node.setInvExp(expr);
    }
    node.setInvDecl(invFun_.clone());

    IsaGen.funcGenHistoryMap.put(invFun_.getName(), invFun_.clone());
    System.out.println("");
    }
```

```
@Override
public void caseATypeDeclIR(ATypeDeclIR node) throws AnalysisException {
    super.caseATypeDeclIR(node);

    /*We do not want invariants built for each type declaration field
    instead we would like one invariant for the whole declaration type
    we skip subsequent record fields so that we do not get
    inv_field1 inv_field2 inv_record instead we get inv_record which accesses
    field1 and field2.*/

    String typeName = IsaInvNameFinder.findName(node.getDecl());
    SDeclIR decl = node.getDecl().clone();

    SDeclIR invFun;
    if (node.getDecl() instanceof ARecordDeclIR)
      invFun = ( (ARecordDeclIR) decl).getInvariant();
    else
      invFun = node.getInv();
    // Invariant function
    AFuncDeclIR invFun_ = new AFuncDeclIR();
    invFun_.setName("inv_" + typeName); //inv_t

    // Define the type signature
    //TODO: Type should be XTypeInt - correct?
    AMethodTypeIR methodType = new AMethodTypeIR();

    STypeIR t = IsaDeclTypeGen.apply(decl);
    methodType.getParams().add(t.clone());


  methodType.setResult(new ABoolBasicTypeIR());
    invFun_.setMethodType(methodType);



    // Translation for VDMToolkit and modeller written invariants
    if (invFun != null)
    {
```

```
    AFuncDeclIR inv = (AFuncDeclIR) invFun;//cast invariant function
        declaration to AFuncDeclIR
    AAndBoolBinaryExpIR multipleInvs = new AAndBoolBinaryExpIR();

    for (int i = 0; i < inv.getMethodType().getParams().size(); i++)
    {
      AFormalParamLocalParamIR afplp = new AFormalParamLocalParamIR();
        afplp.setPattern(inv.getFormalParams().get(i).getPattern());
        afplp.setType(inv.getMethodType().getParams().get(i).clone());
        invFun_.getFormalParams().add(afplp);
    }

    //change (a_c a) to (c A) for Isabelle field access
      //if (decl instanceof ARecordDeclIR)
          formatExistingRecordInvExp(inv.getBody());

      multipleInvs.setRight(inv.getBody());

AIdentifierPatternIR identifierPattern = new AIdentifierPatternIR();
identifierPattern.setName(typeName.substring(0, 1).toLowerCase());
SExpIR expr = IsaInvExpGen.apply(decl.clone(),
    identifierPattern ,
    methodType.clone(), isaFuncDeclIRMap);

multipleInvs.setLeft(expr);

    invFun_.setBody(multipleInvs);
  }
  //translation for no inv types
  else
  {
    SExpIR expr;
AIdentifierPatternIR identifierPattern = new AIdentifierPatternIR();
    identifierPattern.setName(typeName.substring(0, 1).toLowerCase());
    AFormalParamLocalParamIR afp = new AFormalParamLocalParamIR();
    afp.setPattern(identifierPattern);
    afp.setType(t.clone());
    invFun_.getFormalParams().add(afp);
    expr = IsaInvExpGen.apply(decl.clone(), identifierPattern,
        methodType.clone(), isaFuncDeclIRMap);


    invFun_.setBody(expr);
  }


  // Insert into AST and get rid of existing invariant functions forEach
      field in record type
  AModuleDeclIR encModule = node.getAncestor(AModuleDeclIR.class);
  if (decl instanceof ARecordDeclIR) encModule.getDecls().removeIf(
      d -> d instanceof AFuncDeclIR &&
          d.getChildren(true).get("_name").toString().contains("inv"));

  if(encModule != null)
  {

      encModule.getDecls().add(invFun_);
  }
```

```
    IsaGen.funcGenHistoryMap.put(invFun_.getName(), invFun_.clone());

    System.out.println("");


}

@Override
public void caseAFieldDeclIR(AFieldDeclIR node) throws AnalysisException {
    super.caseAFieldDeclIR(node);
    if (node.parent() instanceof AStateDeclIR){
      System.out.println("Redirecting State Invariants...");
    }
    else {
    STypeIR t = node.getType();// Invariant function
    AFuncDeclIR invFun_ = new AFuncDeclIR();
    invFun_.setName("inv_" + node.getName());

    AMethodTypeIR mt = new AMethodTypeIR();

  mt.setResult(new ABoolBasicTypeIR()); //set return type to bool
    invFun_.setMethodType(mt.clone());



    AIdentifierPatternIR identifierPattern = new AIdentifierPatternIR();
    identifierPattern.setName("");//abbreviations have no params so do not
        use identifier pattern


    AFormalParamLocalParamIR afp = new AFormalParamLocalParamIR();
    afp.setPattern(identifierPattern);
    afp.setType(t.clone());
    invFun_.getFormalParams().add(afp);


    SExpIR expr = IsaInvExpGen.apply(node, identifierPattern, mt.clone(),
        isaFuncDeclIRMap);

  invFun_.setBody(expr);
  IsaGen.funcGenHistoryMap.put(invFun_.getName(), invFun_);
    // Insert into AST
    AModuleDeclIR encModule = node.getAncestor(AModuleDeclIR.class);
    if(encModule != null)
    {
        encModule.getDecls().add(invFun_.clone());
    }
    System.out.println("");
    }
}



public String GenInvTypeDefinition(String arg){
    return "Definition\n" +
           "   inv_" + arg+ " :: \"" + arg + " \\<Rightarrow> \\<bool>\"\n" +
           "   where\n" +
```

```
            "";
    }

}
```

B.2 CodeGen Platform

B.2.1 IsaGen

```
/**
 * Main facade class for VDM 2 Isabelle IR
 *
 * @author ldc
 */
public class IsaGen extends CodeGenBase {


  public static Map<String, AFuncDeclIR> funcGenHistoryMap = new HashMap<>();;
  public static Map<STypeIR, String> typeGenHistoryMap = new HashMap<>();;

    public IsaGen()
    {
        this.addInvTrueMacro();

        this.getSettings().setAddStateInvToModule(false);
        this.getSettings().setGenerateInvariants(true);
    }
    //TODO: Auto load files in macro directory
    public static void addInvTrueMacro(){
        StringBuilder sb = new StringBuilder("#macro ( invTrue $node )\n" +
                "    definition\n" +
                "        inv_$node.Name :: $node.Name \\<Rightarrow> \\<bool>\n" +
                "        where\n" +
                "        \"inv_$node.Name \\<equiv> inv_True\"\n" +
                "#end");
        addMacro("invTrue",new StringReader(sb.toString()));
        Template template = new Template();
    }

    public static void addMacro(String name, StringReader reader){
        try {
            Template template = new Template();
            RuntimeServices runtimeServices =
                RuntimeSingleton.getRuntimeServices();

            SimpleNode simpleNode = runtimeServices.parse(reader, name);
            template.setRuntimeServices(runtimeServices);
            template.setData(simpleNode);
            template.initDocument();
        } catch (ParseException e)
        {
            System.out.println("Failed with: " + e);
        }
    }
```

```
public static String vdmExp2IsaString(PExp exp) throws AnalysisException,
       org.overture.codegen.ir.analysis.AnalysisException {
    IsaGen ig = new IsaGen();
    GeneratedModule r = ig.generateIsabelleSyntax(exp);
    if (r.hasMergeErrors()) {
        throw new
            org.overture.codegen.ir.analysis.AnalysisException(exp.toString()
               + " cannot be generated. Merge errors:"
               + r.getMergeErrors().toString());
    }
    if (r.hasUnsupportedIrNodes()) {
        throw new
            org.overture.codegen.ir.analysis.AnalysisException(exp.toString()
               + " cannot be generated. Unsupported in IR:"
               + r.getUnsupportedInIr().toString());
    }
    if (r.hasUnsupportedTargLangNodes()) {
        throw new
            org.overture.codegen.ir.analysis.AnalysisException(exp.toString()
               + " cannot be generated. Unsupported in TargLang:"
               + r.getUnsupportedInTargLang().toString());
    }

    return r.getContent();
}


/**
 * Main entry point into the Isabelle Translator component. Takes an AST and
     returns corresponding Isabelle Syntax.
 *
 * @param statuses The IR statuses holding the nodes to be code generated.
 * @return The generated Isabelle syntax
 * @throws AnalysisException
 *
 */
@Override
protected GeneratedData genVdmToTargetLang(List<IRStatus<PIR>> statuses)
    throws AnalysisException {


    // Typecheck the VDMToolkit module and generate the IR
    TypeCheckerUtil.TypeCheckResult<List<AModuleModules>>
        listTypeCheckResult1 =
           TypeCheckerUtil.typeCheckSl(new
               File("src/test/resources/VDMToolkit.vdmsl"));
    AModuleModules isaToolkit = listTypeCheckResult1.result.
           stream().
           filter(mod -> mod.getName().getName().equals("VDMToolkit")).
           findAny().
           orElseThrow(() -> new AnalysisException("Failed to find VDMToolkit
               module"));
    super.genIrStatus(statuses, isaToolkit);

    // Get the VDMToolkit module IR
    IRStatus<PIR> vdmToolkitIR = statuses.stream().filter(x ->
        x.getIrNodeName().equals("VDMToolkit")).findAny().orElseThrow(() ->
        new AnalysisException("Failed to find VDMToolkit IR node"));
```

```
AModuleDeclIR vdmToolkitModuleIR = (AModuleDeclIR)
    vdmToolkitIR.getIrNode();


GeneratedData r = new GeneratedData();
try {


    // Apply transformations
    for (IRStatus<PIR> status : statuses) {
        if(status.getIrNodeName().equals("VDMToolkit")){
            System.out.println("");
        } else {


            // transform away any recursion cycles
            GroupMutRecs groupMR = new GroupMutRecs();
            generator.applyTotalTransformation(status, groupMR);

            if (status.getIrNode() instanceof AModuleDeclIR) {

                AModuleDeclIR cClass = (AModuleDeclIR) status.getIrNode();

                // then sort remaining dependencies
                SortDependencies sortTrans = new
                    SortDependencies(cClass.getDecls());
                generator.applyPartialTransformation(status, sortTrans);
            }


            // Transform all token types to isa_VDMToken
            // Transform all nat types to isa_VDMNat
            // Transform all nat1 types to isa_VDMNat
            // Transform all int types to isa_VDMInt
            IsaBasicTypesConv invConv = new IsaBasicTypesConv(getInfo(),
                this.transAssistant, vdmToolkitModuleIR);
            generator.applyPartialTransformation(status, invConv);


            // Transform Seq and Set types into isa_VDMSeq and isa_VDMSet
            IsaTypeTypesConv invSSConv = new IsaTypeTypesConv(getInfo(),
                this.transAssistant, vdmToolkitModuleIR);
            generator.applyPartialTransformation(status, invSSConv);


            IsaInvGenTrans invTrans = new IsaInvGenTrans(getInfo(),
                vdmToolkitModuleIR);
            generator.applyPartialTransformation(status, invTrans);

            IsaFuncDeclConv funcConv = new IsaFuncDeclConv(getInfo(),
                this.transAssistant, vdmToolkitModuleIR);
            generator.applyPartialTransformation(status, funcConv);



        }
    }
```

```
            r.setClasses(prettyPrint(statuses));
        } catch (org.overture.codegen.ir.analysis.AnalysisException e) {
            throw new AnalysisException(e);
        }
        return r;

    }

    public GeneratedModule generateIsabelleSyntax(PExp exp)
            throws AnalysisException,
            org.overture.codegen.ir.analysis.AnalysisException {
        IRStatus<SExpIR> status = this.generator.generateFrom(exp);

        if (status.canBeGenerated()) {
            return prettyPrint(status);
        }

        throw new
            org.overture.codegen.ir.analysis.AnalysisException(exp.toString()
                + " cannot be code-generated");
    }


    private List<GeneratedModule> prettyPrint(List<IRStatus<PIR>> statuses)
            throws org.overture.codegen.ir.analysis.AnalysisException {
        // Apply merge visitor to pretty print Isabelle syntax

        IsaTranslations isa = new IsaTranslations();
        MergeVisitor pp = isa.getMergeVisitor();


        List<GeneratedModule> generated = new ArrayList<GeneratedModule>();

        for (IRStatus<PIR> status : statuses) {
            if(status.getIrNodeName().equals("VDMToolkit")){
                System.out.println("");
            } else {
                generated.add(prettyPrintNode(pp, status));
            }

        }
        // Return syntax
        return generated;
    }


    //feed to velocity monster
    private GeneratedModule prettyPrint(IRStatus<? extends INode> status)
            throws org.overture.codegen.ir.analysis.AnalysisException {
        // Apply merge visitor to pretty print Isabelle syntax
        IsaTranslations isa = new IsaTranslations();
        MergeVisitor pp = isa.getMergeVisitor();
        return prettyPrintNode(pp, status);
    }

    private GeneratedModule prettyPrintNode(MergeVisitor pp,
                                            IRStatus<? extends INode> status)
            throws org.overture.codegen.ir.analysis.AnalysisException {
```

```
        INode irClass = status.getIrNode();

        StringWriter sw = new StringWriter();

        irClass.apply(pp, sw);

        if (pp.hasMergeErrors()) {
            return new GeneratedModule(status.getIrNodeName(), irClass,
                pp.getMergeErrors(), false);
        } else if (pp.hasUnsupportedTargLangNodes()) {
            return new GeneratedModule(status.getIrNodeName(), new
                HashSet<VdmNodeInfo>(), pp.getUnsupportedInTargLang(), false);
        } else {
            // Code can be generated. Ideally, should format it
            GeneratedModule generatedModule = new
                GeneratedModule(status.getIrNodeName(), irClass, sw.toString(),
                false);
            generatedModule.setTransformationWarnings(status.getTransformationWarnings());
            return generatedModule;
        }
    }
}
```

Appendix C

Testing

C.0.1 IsaGenParamTest

```
/**
 * Main parameterized test class. Runs tests on modules with minimal
 * definitions to exercise the translation with a single construct
 * at a time.
 *
 * @author ldc
 */
@RunWith(Parameterized.class)
public class IsaGenParamTest extends ParamStandardTest<CgIsaTestResult> {

    public IsaGenParamTest(String nameParameter, String inputParameter,
                           String resultParameter) {
        super(nameParameter, inputParameter, resultParameter);
    }

    private static final String UPDATE = "tests.update.isagen";
    private static final String CGISA_ROOT = "src/test/resources/modules";
    private static final List<String> skippedTests =
        Arrays.asList();//"NoParamPrePost.vdmsl",
//       "2ParamsPrePost.vdmsl",
//       "NoParamNoPre.vdmsl",
//       "1ParamNoPre.vdmsl","1ParamPrePost.vdmsl",
//       "FuncPrePost.vdmsl",
////       "NotYetSpecified.vdmsl",
//       "FuncPre.vdmsl",
//       "FuncApply3Params.vdmsl",
//       "FuncDecl2Params.vdmsl",
//       "FuncDeclNoParam.vdmsl",
//       "FuncDepSimple.vdmsl",
//       "FuncApplyNoParam.vdmsl",
//       "FuncPost.vdmsl",
//       "FuncApply1Param.vdmsl",
//       "FuncDecl1Param.vdmsl",
//       "EqualsInit.vdmsl","PredicateInit.vdmsl",
//
    "IntExpVarExp.vdmsl","ExplicitInt.vdmsl","ExplicitNat.vdmsl","ExplicitNat1.vdmsl",
//       "ExplicitReal.vdmsl","IndependentDefsOrder.vdmsl",
//       "ImplicitNumericExp.vdmsl","VarExp.vdmsl",
//       "SeqNat.vdmsl",
//       "BoolType.vdmsl",
//       "InvSet.vdmsl",
```

```
//       "InvRecordDummyInv.vdmsl",
//       "InvInt.vdmsl",
//
    "Rec2Fields.vdmsl","SeqInt.vdmsl","Real.vdmsl","CharSeqIntSetTuple.vdmsl","IntIntTuple.vdmsl",
//       "MapIntChar.vdmsl",
//       "Char.vdmsl",
//       "Rec1Field.vdmsl",
//       "IntCharTuple.vdmsl","Token.vdmsl",
//
    "CharNatTokenTuple.vdmsl","Rat.vdmsl","SetInt.vdmsl","Nat.vdmsl","Nat1.vdmsl",
//       "Rec2FieldsDiffTypes.vdmsl");//,// "MapIntInt.vdmsl");

    @Override
    public CgIsaTestResult processModel(List<INode> ast) {
        IsaGen gen = new IsaGen();
        GeneratedData genData = null;

        try {
            genData = gen.generate(ast);
        } catch (AnalysisException e) {
            fail("Could not process test file " + testName);
        }

        List<AModuleModules> classes = new LinkedList<>();
        for (INode n : ast) {
            classes.add((AModuleModules) n);
        }

        List<GeneratedModule> result = null;
            result = genData.getClasses();
            if (!result.get(0).canBeGenerated()) {
                StringBuilder sb = new StringBuilder();
                sb.append(result.get(0).getMergeErrors());
                sb.append(result.get(0).getUnsupportedInIr());
                sb.append(result.get(0).getUnsupportedInTargLang());
                fail(sb.toString());
            }

        return CgIsaTestResult.convert(result);
    }

    @Parameters(name = "{index} : {0}")
    public static Collection<Object[]> testData() {
        return PathsProvider.computePaths(CGISA_ROOT);
    }

    @Override
    public Type getResultType() {
        Type resultType = new TypeToken<CgIsaTestResult>() {
        }.getType();
        return resultType;
    }

    @Override
    protected String getUpdatePropertyString() {
        return UPDATE;
    }
```

```
    @Override
    public void compareResults(CgIsaTestResult actual, CgIsaTestResult expected)
        {
        assertTrue("\n --- Expected: ---\n" + expected.translation
               + "\n --- Got: ---\n" + actual.translation,
                   expected.compare(actual));
        if(expected.compare(actual))
        {
            System.out.println("\n --- Got: ---\n" + actual.translation);
        }

    }

    @Override
    protected void checkAssumptions() {
        Assume.assumeTrue("Test in skip list.",notSkipped());
    }

    private boolean notSkipped() {
        return !skippedTests.contains(testName);
    }
}
```

C.0.2 IsaGenModelTest

```
/*
 * #%~
 * VDM to Isabelle Translation
 * %%
 * Copyright (C) 2008 - 2015 Overture
 * %%
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
 * License along with this program. If not, see
 * <http://www.gnu.org/licenses/gpl-3.0.html>.
 * #~%
 */
package org.overturetool.cgisa;

import java.util.Arrays;
import java.util.Collection;
import java.util.List;

import org.junit.Assume;
import org.junit.Ignore;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
```

```
import org.junit.runners.Parameterized.Parameters;
import org.overture.core.testing.PathsProvider;

/**
 * Main integration test class. Runs tests on complete models.
 *
 * @author ldc
 */
@RunWith(Parameterized.class)

public class IsaGenModelTest extends IsaGenParamTest
{

    public IsaGenModelTest(String nameParameter, String inputParameter,
            String resultParameter)
    {
        super(nameParameter, inputParameter, resultParameter);
    }

    private static final String UPDATE = "tests.update.isagen.model";
    private static final String MODELS_ROOT = "src/test/resources/models";
    private static final List<String> skippedTests =
        Arrays.asList();//"CustomAlarm.vdmsl","dummy.vdmsl","Alarm1.vdmsl");


    @Parameters(name = "{index} : {0}")
    public static Collection<Object[]> testData()
    {
        return PathsProvider.computePaths(MODELS_ROOT);
    }

    @Override
    protected String getUpdatePropertyString()
    {
        return UPDATE;
    }

    protected void checkAssumptions() {
        Assume.assumeTrue("Test in skip list.",notSkipped());
    }

    private boolean notSkipped() {
        return !skippedTests.contains(testName);
    }


}
```

C.1 VDM-SL Test Files

C.1.1 FuncDecl1Param.vdmsl / .vdmsl.result

```
module A

definitions
```

```
functions

f : nat -> nat
f(x) == x;

end A
```

```
{"translation":"theory A\n imports VDMToolkit\n
begin\n\n
definition \n
f :: \"VDMNat \\<Rightarrow> VDMNat\"\n
where \n
\"f x \\<equiv> x\"\n

definition \n
pre_f \:\: \"VDMNat \\<Rightarrow> \\<bool>\"\n
where \n
\"pre_f x \\<equiv> isa_invVDMNat x\"\n

definition\n
post_f \:\: \"VDMNat \\<Rightarrow> VDMNat \\<Rightarrow> \\<bool>\"\n
where\n
\" post_f x RESULT \\<equiv> (isa_invVDMNat x \\<and> isa_invVDMNat RESULT)\"\n


\n\nend","errors":false}
```

C.1.2 FuncApply3Params.vdmsl / .vdmsl.result

```
module A

definitions

functions
f : int * int * int -> int
f (x,y,z) == 0;

values
x = f(1,2,3);

end A
```

```
{"translation":"\ntheory A\nimports VDMToolkit\nbegin

definition\nf \:\: \"VDMInt \\<Rightarrow> VDMInt \\<Rightarrow> VDMInt
    \\<Rightarrow> VDMInt\"\nwhere\n\"f x y z \\<equiv> 0\"\n\nabbreviation\n x
    \:\: VDMInt\n where\n\"x \\<equiv> f 1 2 3\"

definition\ninv_x \:\: \"\\<bool>\"\nwhere\n\"inv_x \\<equiv> isa_invTrue
    x\"\n\ndefinition\n pre_f \:\: \"VDMInt \\<Rightarrow> VDMInt \\<Rightarrow>
    VDMInt \\<Rightarrow> \\<bool>\"\nwhere\n\"pre_f x y z \\<equiv>
    (isa_invTrue x \\<and> (isa_invTrue y \\<and> isa_invTrue
    z))\"\n\ndefinition\npost_f \:\: \"VDMInt\\<Rightarrow> VDMInt\\<Rightarrow>
```

```
    VDMInt\\<Rightarrow> VDMInt\\<Rightarrow> \\<bool>\"\nwhere\n\"post_f x y z
    RESULT \\<equiv> (isa_invTrue x \\<and> (isa_invTrue y \\<and> (isa_invTrue
    z \\<and> isa_invTrue RESULT)))\"\n\nend","errors":false}
```

C.1.3 NotYetSpecified.vdmsl / .vdmsl.result

```
functions

f : int -> token
f (x) == is not yet specified
```

```
{"translation":"
theory DEFAULT
  imports VDMToolkit
begin


definition
    f \:\: \"VDMInt
 \\<Rightarrow> VDMToken
\"
    where
    \"f x \\<equiv> undef\"


definition
    pre_f \:\: \"VDMInt
 \\<Rightarrow> \\<bool>\"
    where
    \"pre_f x \\<equiv> isa_invTrue x\"


definition
    post_f \:\: \"VDMInt
 \\<Rightarrow> VDMToken
 \\<Rightarrow> \\<bool>\"
    where
    \"post_f x RESULT \\<equiv> (isa_invTrue x \\<and> isa_invTrue RESULT)\"

end","errors":false}
```

C.1.4 1ParamNoPre.vdmsl / .vdmsl.result

```
functions

f (x:int) r: int
post r = x
```

```
{"translation":"theory DEFAULT\n imports VDMToolkit\nbegin\n\n

definition \n
pre_f \:\: \"VDMInt \\<Rightarrow> \\<bool>\"\n
where \n
```

```
\"pre_f x \\<equiv> isa_invTrue x \"\n

definition\n
post_f \:\: \"VDMInt \\<Rightarrow> VDMInt \\<Rightarrow> \\<bool>\"\n
where\n
\"post_f x r \\<equiv> ((isa_invTrue x \\<and> isa_invTrue r) \\<and> (r = x))\"



\n\nend","errors":false}
```

C.1.5 NoParamNoPre.vdmsl / .vmdsl.result

```
functions

f () r: int
post true
```

```
{"translation":"theory DEFAULT\n imports VDMToolkit\nbegin\n\n

definition \n
unimplemented_pre_f \:\: \"\\<bool>\"\n
where \n
\"unimplemented_pre_f \\<equiv> undef\"\n

definition\n
post_f \:\: \"VDMInt \\<Rightarrow> \\<bool>\"\n
where\n
\" post_f r \\<equiv> true\"\n



\n\n end","errors":false}
```

C.1.6 PredicateInit.vdmsl / .vdmsl.result

```
state S of
  x : nat
  init s == s.x >0
end
```

```
{"translation":"
theory DEFAULT
  imports VDMToolkit
begin

record S =
       s_x \:\: VDMNat



definition
    inv_S \:\: \"S \\<Rightarrow> \\<bool>\"
```

```
    where
    \"inv_S s \\<equiv> isa_invVDMNat (s_x s)\"


definition
    init_S \:\: \"S \\<Rightarrow> \\<bool>\"
    where
    \"init_S s \\<equiv> ((s_x s) > 0)\"


definition
    post_init_S \:\: \"VDMNat
 \\<Rightarrow> \\<bool>\"
    where
    \"post_init_S \\<equiv> inv_S init_S\"

end","errors":false}
```

C.1.7 EqualsInit.vdmsl / .vdmsl.result

```
state S of
  x : nat
  init s == s = mk_S(0)
end
```

```
{"translation":"
theory DEFAULT
  imports VDMToolkit
begin

record S =
       s_x \:\: VDMNat



definition
    inv_S \:\: \"S \\<Rightarrow> \\<bool>\"
    where
    \"inv_S s \\<equiv> isa_invVDMNat (s_x s)\"


definition
    init_S \:\: \"S \\<Rightarrow> \\<bool>\"
    where
    \"init_S s \\<equiv> (s = (| x = 0 |))\"


definition
    post_init_S \:\: \"VDMNat
 \\<Rightarrow> \\<bool>\"
    where
    \"post_init_S \\<equiv> inv_S init_S\"

end","errors":false}
```

C.1.8 InvSet.vdmsl / .vdmsl.result

```
types

t = set of int
inv t == t <> {}
```


```
{"translation":"theory DEFAULT\n imports VDMToolkit\nbegin\n\ntype_synonym t
    \u003d \"VDMInt VDMSet\"\n\n\n\n\ndefinition\n inv_t :: \"(t)
    \\\u003cRightarrow\u003e \\\u003cbool\u003e\"\n where\n \"inv_t t
    \\\u003cequiv\u003e (isa_invTrue t \\<and> (t <>
    {}))\"\n\nend","errors":false}
```


C.1.9 InvRecordDummyInv.vdmsl / .vdmsl.result

```
types
    A :: b : seq of char
         c : int
    inv a == a.c > 1
```


```
{"translation":"theory DEFAULT
  imports VDMToolkit
begin

record A =
       a_b \:\: char VDMSeq

       a_c \:\: VDMInt




definition
    inv_A \:\: \"A \\<Rightarrow> \\<bool>\"
    where
    \"inv_A a \\<equiv> ((isa_invSeqElems isa_invTrue (a_b a) \\<and>
        isa_invTrue (a_c a)) \\<and> ((a_c a) > 1))\"

end

","errors":false}
```


C.1.10 Token.vdmsl / .vdmsl.result

```
types

t = token
```


```
{"translation":"theory DEFAULT\n imports VDMToolkit\nbegin\n\ntype_synonym t
    \u003d \"VDMToken\"\n\n\n\n\ndefinition\n inv_t :: \"(t)
```

```
    \\\u003cRightarrow\u003e \\\u003cbool\u003e\"\n where\n \"inv_t t
    \\\u003cequiv\u003e isa_invTrue t\"\n\nend","errors":false}
```

C.1.11 SeqNat.vdmsl / .vdmsl.result

```
types

t = seq of nat
```

```
{"translation":"theory DEFAULT\n imports VDMToolkit\nbegin\n\ntype_synonym t
    \u003d \"VDMNat VDMSeq\"\n\n\n\n\ndefinition\n inv_t :: \"(t)
    \\\u003cRightarrow\u003e \\\u003cbool\u003e\"\n where\n \"inv_t t
    \\\u003cequiv\u003e isa_invTrue t\"\n\nend","errors":false}
```

C.1.12 Rec2FieldsDiffTypes.vdmsl / .vdmsl.result

```
types

RecType :: x : char
           y: realR
```

```
{"translation":"
theory DEFAULT
  imports VDMToolkit
begin

record RecType =
       recType_x \:\: VDMNat

       recType_y \:\: VDMNat




definition
    inv_RecType \:\: \"RecType \\<Rightarrow> \\<bool>\"
    where
    \"inv_RecType r \\<equiv> (isa_invVDMNat (recType_x r) \\<and> isa_invVDMNat
        (recType_y r))\"

end","errors":false}
```

C.1.13 CharNatTokenTuple.vdmsl / .vdmsl.result

```
types
t = char * nat * token
```

```
{"translation":"theory DEFAULT\n imports VDMToolkit\nbegin\n\ntype_synonym t
    \u003d \"char * VDMNat * VDMToken\"\n\n\n\n\ndefinition\n inv_t :: \"(t)
```

```
    \\\u003cRightarrow\u003e \\\u003cbool\u003e\"\n where\n \"inv_t t
    \\\u003cequiv\u003e isa_invTrue t\"\n\nend","errors":false}
```

C.1.14 NestedVarExp.vdmsl / .vdmsl.result

```
values
x = {[1]};
y = x;
```

```
{"translation":"theory DEFAULT\n imports VDMToolkit
begin

abbreviation
 x \:\: VDMNat1 VDMSeq VDMSet where
\" x \\<equiv> {[1]} \"
\nabbreviation
 y \:\: VDMNat1 VDMSeq VDMSet where
\" y \\<equiv> x \"

definition
inv_x \:\: \"\\<bool>\" where
\"inv_x \\<equiv> isa_invSetElems isa_invSeqElems isa_invVDMNat1 x\"

definition
inv_y \:\: \"\\<bool>\" where
\"inv_y \\<equiv> isa_invSetElems isa_invSeqElems isa_invVDMNat1 y\"

end","errors":false}
```

C.1.15 IndependentDefsOrder.vdmsl / .vdmsl.result

```
values
a = 10;
b = 20;
c = 30;
```

```
{"translation":"theory DEFAULT\n imports VDMToolkit
begin

abbreviation
 a \:\: VDMNat1 where
\" a \\<equiv> 10 \"
\n
abbreviation
 b \:\: VDMNat1 where
\" b \\<equiv> 20 \"
\n
abbreviation
 c \:\: VDMNat1 where
\" c \\<equiv> 30 \"
\n\n
definition
inv_a \:\: \"\\<bool>\" where
```

```
\"inv_a \\<equiv> isa_invVDMNat1 a\"
\n
definition
inv_b \:\: \"\\<bool>\" where
\"inv_b \\<equiv> isa_invVDMNat1 b\"
\n
definition
inv_c \:\: \"\\<bool>\" where
\"inv_c \\<equiv> isa_invVDMNat1 c\"
\n\n
end","errors":false}
```

C.1.16 ImplicitNumericExp.vdmsl / .vdmsl.result

```
values
x = 1;
```

```
{"translation":"theory DEFAULT\n imports VDMToolkit
begin

abbreviation
 x \:\: VDMNat1 where
\" x \\<equiv> 1 \"
\n
definition
inv_x \:\: \"\\<bool>\" where
\"inv_x \\<equiv> isa_invVDMNat1 x\"
\n\n
end","errors":false}
```

C.1.17 ExplicitNat.vdmsl / .vdmsl.result

```
values
x : nat = 1;
```

```
{"translation":"theory DEFAULT\n imports VDMToolkit
begin

abbreviation
 x \:\: VDMNat where
\" x \\<equiv> 1 \"

definition
inv_x \:\: \"\\<bool>\" where
\"inv_x \\<equiv> isa_invVDMNat x\"
\n\n
end","errors":false}
```

C.1.18 BoolType.vdmsl / .vdmsl.result

```
values
```

```
x : bool = true;
```


```
{"translation":"theory DEFAULT\n imports VDMToolkit\nbegin\n\nabbreviation\n x
    \:\: \\<bool>\nwhere \" x \\<equiv> true \"\n

definition
inv_x \:\: \"\\<bool>\" where
\"inv_x \\<equiv> isa_invTrue x\"
\n\n

 \n\nend","errors":false}
```


C.1.19 MapIntChar.vdmsl / .vdmsl.result

```
types

t = map int to char
```


```
{"translation":"theory DEFAULT\n imports VDMToolkit\nbegin\n\ntype_synonym t
    \u003d \"VDMInt \\<rightharpoonup> char\"\n\n\n\n\ndefinition\n inv_t ::
    \"(t) \\\u003cRightarrow\u003e \\\u003cbool\u003e\"\n where\n \"inv_t t
    \\\u003cequiv\u003e isa_invTrue t\"\n\nend","errors":false}
```

Appendix D

Apache Velocity

D.1 Velocity Before Development

D.1.1 AFuncDeclIR

```
#macro ( transIdentifiers $node )
#foreach($p in $node.FormalParams)
$Isa.trans($p.pattern)##
#end
#end

definition
    $node.Name :: "$Isa.transTypeParams($node.MethodType.Params) \<RightArrow>
        $Isa.trans($node.MethodType.Result)"
    where
    "$node.Name #transIdentifiers($node) \<equiv> $Isa.trans($node.Body)"
```

D.2 Velocity After Development

D.2.1 AFuncDeclIR

```
#macro ( transIdentifiers $node )
#foreach($p in $node.FormalParams)
$Isa.trans($p.pattern)##
#end
#end

definition
#if ("$Isa.transTypeParams($node.MethodType.Params)" == "")
  $node.Name :: "$Isa.trans($node.MethodType.Result)"
#else
  $node.Name :: "$Isa.transTypeParams($node.MethodType.Params) \<Rightarrow>
      $Isa.trans($node.MethodType.Result)"
#end
    where
    "$node.Name #transIdentifiers($node) \<equiv> $Isa.trans($node.Body)"
```

Appendix E

Results

E.0.1 Individual Construct Translation

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "VDMInt
 \<rightharpoonup> VDMInt
"



definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

record RecType =
       recType_x :: char
       recType_y :: \<real>



definition
    inv_RecType :: "RecType \<Rightarrow> \<bool>"
    where
    "inv_RecType r \<equiv> (isa_invTrue (recType_x r) \<and> isa_invTrue
        (recType_y r))"
```

```
end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym XType = "VDMInt
"


definition
    inv_XType :: "(XType) \<Rightarrow> \<bool>"
    where
    "inv_XType x \<equiv> isa_invTrue x"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "VDMNat1
"


definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "VDMNat
"


definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
```

```
    "inv_t t \<equiv> isa_invTrue t"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "VDMInt
 VDMSet
"


definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "\<rat>"


definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "char * VDMNat
 * VDMToken
"


definition
```

```
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end

 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "VDMToken
"



definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end



 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "VDMInt
 * char"



definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end



 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

record RecType =
       recType_x :: VDMNat
```

```
definition
    inv_RecType :: "RecType \<Rightarrow> \<bool>"
    where
    "inv_RecType r \<equiv> isa_invVDMNat (recType_x r)"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "char"


definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "VDMInt
 \<rightharpoonup> char"


definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "VDMInt
 * VDMInt
"
```

```
definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "char VDMSeq
 * VDMInt
 VDMSet
"
```

```
definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "\<real>"
```

```
definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin
```

```
type_synonym t = "VDMNat
 VDMSeq
"



definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end




 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "VDMInt
 VDMSeq
"



definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> isa_invTrue t"

end






 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

record RecType =
       recType_x :: VDMNat

       recType_y :: VDMNat




definition
    inv_RecType :: "RecType \<Rightarrow> \<bool>"
    where
```

```
    "inv_RecType r \<equiv> (isa_invVDMNat (recType_x r) \<and> isa_invVDMNat
        (recType_y r))"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

type_synonym t = "VDMInt
"


definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> (isa_invTrue t \<and> (t > 0))"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

record A =
       a_b :: char VDMSeq

       a_c :: VDMInt


definition
    inv_A :: "A \<Rightarrow> \<bool>"
    where
    "inv_A a \<equiv> ((isa_invSeqElems isa_invTrue (a_b a) \<and> isa_invTrue
        (a_c a)) \<and> ((a_c a) > 1))"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
```

```
begin

type_synonym t = "VDMInt
 VDMSet
"



definition
    inv_t :: "(t) \<Rightarrow> \<bool>"
    where
    "inv_t t \<equiv> (isa_invTrue t \<and> (t <> {}))"

end



 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

abbreviation
 x :: \<bool> where
"x \<equiv> true"


definition
    inv_x :: "\<bool>"
    where
    "inv_x \<equiv> isa_invTrue x"

end



 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

abbreviation
 x :: VDMNat1
 where
"x \<equiv> 1"

abbreviation
 y :: VDMNat1
 where
"y \<equiv> x"


definition
    inv_x :: "\<bool>"
    where
    "inv_x \<equiv> isa_invVDMNat1 x"
```

```
definition
    inv_y :: "\<bool>"
    where
    "inv_y \<equiv> isa_invVDMNat1 y"

end


 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

abbreviation
 x :: VDMNat1
 where
"x \<equiv> 1"


definition
    inv_x :: "\<bool>"
    where
    "inv_x \<equiv> isa_invVDMNat1 x"

end


 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

abbreviation
 a :: VDMNat1
 where
"a \<equiv> 10"

abbreviation
 b :: VDMNat1
 where
"b \<equiv> 20"

abbreviation
 c :: VDMNat1
 where
"c \<equiv> 30"


definition
    inv_a :: "\<bool>"
    where
    "inv_a \<equiv> isa_invVDMNat1 a"


definition
    inv_b :: "\<bool>"
    where
    "inv_b \<equiv> isa_invVDMNat1 b"
```

```
definition
    inv_c :: "\<bool>"
    where
    "inv_c \<equiv> isa_invVDMNat1 c"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

abbreviation
 x :: \<real> where
"x \<equiv> 1.2"


definition
    inv_x :: "\<bool>"
    where
    "inv_x \<equiv> isa_invTrue x"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

abbreviation
 x :: VDMNat1
 where
"x \<equiv> 1"


definition
    inv_x :: "\<bool>"
    where
    "inv_x \<equiv> isa_invVDMNat1 x"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

abbreviation
 x :: VDMNat1
 VDMSeq
 VDMSet
```

```
 where
"x \<equiv> {[1]}"

abbreviation
 y :: VDMNat1
 VDMSeq
 VDMSet
 where
"y \<equiv> x"


definition
    inv_x :: "\<bool>"
    where
    "inv_x \<equiv> isa_invSetElems isa_invSeqElems isa_invVDMNat1 x"


definition
    inv_y :: "\<bool>"
    where
    "inv_y \<equiv> isa_invSetElems isa_invSeqElems isa_invVDMNat1 y"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

abbreviation
 x :: VDMNat
 where
"x \<equiv> 1"


definition
    inv_x :: "\<bool>"
    where
    "inv_x \<equiv> isa_invVDMNat x"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

abbreviation
 x :: VDMInt
 where
"x \<equiv> 1"
```

```
definition
    inv_x :: "\<bool>"
    where
    "inv_x \<equiv> isa_invTrue x"

end


 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin

abbreviation
 x :: VDMNat1
 where
"x \<equiv> 1"

abbreviation
 y :: VDMNat1
 where
"y \<equiv> x"


definition
    inv_x :: "\<bool>"
    where
    "inv_x \<equiv> isa_invVDMNat1 x"


definition
    inv_y :: "\<bool>"
    where
    "inv_y \<equiv> isa_invVDMNat1 y"

end




 --- Got: ---
theory A
  imports VDMToolkit
begin


definition
    f :: "VDMNat
 \<Rightarrow> VDMNat
"
    where
    "f x \<equiv> x"


definition
    pre_f :: "VDMNat
```

```
 \<Rightarrow> \<bool>"
    where
    "pre_f x \<equiv> isa_invVDMNat x"


definition
    post_f :: "VDMNat
 \<Rightarrow> VDMNat
 \<Rightarrow> \<bool>"
    where
    "post_f xRESULT \<equiv> (isa_invVDMNat x \<and> isa_invVDMNat RESULT)"

end



f(1)


 --- Got: ---
theory A
  imports VDMToolkit
begin


definition
    f :: "VDMInt
 \<Rightarrow> VDMInt
"
    where
    "f x \<equiv> 0"

abbreviation
 x :: VDMInt
 where
"x \<equiv> f 1"


definition
    inv_x :: "\<bool>"
    where
    "inv_x \<equiv> isa_invTrue x"


definition
    pre_f :: "VDMInt
 \<Rightarrow> \<bool>"
    where
    "pre_f x \<equiv> isa_invTrue x"


definition
    post_f :: "VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> \<bool>"
    where
    "post_f xRESULT \<equiv> (isa_invTrue x \<and> isa_invTrue RESULT)"
```

```
end
```

```
 --- Got: ---
theory A
  imports VDMToolkit
begin


definition
    f :: "VDMInt
 \<Rightarrow> VDMInt
"
    where
    "f x \<equiv> 0"


definition
    pre_f :: "VDMInt
 \<Rightarrow> \<bool>"
    where
    "pre_f x \<equiv> isa_invTrue x"


definition
    post_f :: "VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> \<bool>"
    where
    "post_f xRESULT \<equiv> ((isa_invTrue x \<and> isa_invTrue RESULT) \<and>
        true)"

end
```

```
f()
```

```
 --- Got: ---
theory A
  imports VDMToolkit
begin


definition
    f :: "VDMInt
"
    where
    "f  \<equiv> 0"

abbreviation
 x :: VDMInt
 where
"x \<equiv> f "
```

```
definition
    inv_x :: "\<bool>"
    where
    "inv_x \<equiv> isa_invTrue x"


definition
    unimplemented_pre_f :: "\<bool>"
    where
    "unimplemented_pre_f \<equiv> undef"


definition
    unimplemented_post_f :: "VDMInt
 \<Rightarrow> \<bool>"
    where
    "unimplemented_post_f \<equiv> undef"

end
```

```
 --- Got: ---
theory A
  imports VDMToolkit
begin


definition
    g :: "VDMInt
 \<Rightarrow> VDMInt
"
    where
    "g x \<equiv> 0"


definition
    f :: "VDMInt
 \<Rightarrow> VDMInt
"
    where
    "f x \<equiv> g x"


definition
    pre_g :: "VDMInt
 \<Rightarrow> \<bool>"
    where
    "pre_g x \<equiv> isa_invTrue x"


definition
    post_g :: "VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> \<bool>"
```

```
    where
    "post_g xRESULT \<equiv> (isa_invTrue x \<and> isa_invTrue RESULT)"


definition
    pre_f :: "VDMInt
 \<Rightarrow> \<bool>"
    where
    "pre_f x \<equiv> isa_invTrue x"


definition
    post_f :: "VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> \<bool>"
    where
    "post_f xRESULT \<equiv> (isa_invTrue x \<and> isa_invTrue RESULT)"

end
```

```
 --- Got: ---
theory A
  imports VDMToolkit
begin


definition
    f :: "VDMNat
"
    where
    "f  \<equiv> 0"


definition
    unimplemented_pre_f :: "\<bool>"
    where
    "unimplemented_pre_f \<equiv> undef"


definition
    unimplemented_post_f :: "VDMNat
 \<Rightarrow> \<bool>"
    where
    "unimplemented_post_f \<equiv> undef"

end
```

```
 --- Got: ---
theory A
  imports VDMToolkit
begin
```

```
definition
    f :: "VDMNat
 \<Rightarrow> VDMNat
 \<Rightarrow> VDMNat
"
    where
    "f x y \<equiv> x"


definition
    pre_f :: "VDMNat
 \<Rightarrow> VDMNat
 \<Rightarrow> \<bool>"
    where
    "pre_f x y \<equiv> (isa_invVDMNat x \<and> isa_invVDMNat y)"


definition
    post_f :: "VDMNat
 \<Rightarrow> VDMNat
 \<Rightarrow> VDMNat
 \<Rightarrow> \<bool>"
    where
    "post_f xyRESULT \<equiv> (isa_invVDMNat x \<and> (isa_invVDMNat y \<and>
        isa_invVDMNat RESULT))"

end



f(1, 2, 3)


 --- Got: ---
theory A
  imports VDMToolkit
begin


definition
    f :: "VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> VDMInt
"
    where
    "f x y z \<equiv> 0"

abbreviation
 x :: VDMInt
 where
"x \<equiv> f 1 2 3"


definition
    inv_x :: "\<bool>"
    where
```

```
    "inv_x \<equiv> isa_invTrue x"


definition
    pre_f :: "VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> \<bool>"
    where
    "pre_f x y z \<equiv> (isa_invTrue x \<and> (isa_invTrue y \<and>
        isa_invTrue z))"


definition
    post_f :: "VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> \<bool>"
    where
    "post_f xyzRESULT \<equiv> (isa_invTrue x \<and> (isa_invTrue y \<and>
        (isa_invTrue z \<and> isa_invTrue RESULT)))"

end
```

```
 --- Got: ---
theory A
  imports VDMToolkit
begin


definition
    f :: "VDMInt
 \<Rightarrow> VDMInt
"
    where
    "f x \<equiv> 0"


definition
    pre_f :: "VDMInt
 \<Rightarrow> \<bool>"
    where
    "pre_f x \<equiv> (isa_invTrue x \<and> true)"


definition
    post_f :: "VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> \<bool>"
    where
    "post_f xRESULT \<equiv> (isa_invTrue x \<and> isa_invTrue RESULT)"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin


definition
    f :: "VDMInt
 \<Rightarrow> VDMToken
"
    where
    "f x \<equiv> undef"


definition
    pre_f :: "VDMInt
 \<Rightarrow> \<bool>"
    where
    "pre_f x \<equiv> isa_invTrue x"


definition
    post_f :: "VDMInt
 \<Rightarrow> VDMToken
 \<Rightarrow> \<bool>"
    where
    "post_f xRESULT \<equiv> (isa_invTrue x \<and> isa_invTrue RESULT)"

end




 --- Got: ---
theory A
  imports VDMToolkit
begin


definition
    f :: "VDMNat
 \<Rightarrow> VDMInt
"
    where
    "f x \<equiv> 0"


definition
    pre_f :: "VDMNat
 \<Rightarrow> \<bool>"
    where
    "pre_f x \<equiv> (isa_invVDMNat x \<and> true)"


definition
    post_f :: "VDMNat
```

```
 \<Rightarrow> VDMInt
 \<Rightarrow> \<bool>"
    where
    "post_f xRESULT \<equiv> ((isa_invVDMNat x \<and> isa_invTrue RESULT) \<and>
        true)"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin


definition
    pre_f :: "VDMInt
 \<Rightarrow> \<bool>"
    where
    "pre_f x \<equiv> (isa_invTrue x \<and> true)"


definition
    post_f :: "VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> \<bool>"
    where
    "post_f xr \<equiv> ((isa_invTrue x \<and> isa_invTrue r) \<and> (r = x))"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin


definition
    pre_f :: "VDMInt
 \<Rightarrow> \<bool>"
    where
    "pre_f x \<equiv> isa_invTrue x"


definition
    post_f :: "VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> \<bool>"
    where
    "post_f xr \<equiv> ((isa_invTrue x \<and> isa_invTrue r) \<and> (r = x))"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin


definition
    unimplemented_pre_f :: "\<bool>"
    where
    "unimplemented_pre_f \<equiv> undef"


definition
    post_f :: "VDMInt
 \<Rightarrow> \<bool>"
    where
    "post_f r \<equiv> true"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin


definition
    pre_f :: "VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> \<bool>"
    where
    "pre_f x y \<equiv> ((isa_invTrue x \<and> isa_invTrue y) \<and> (y <> 0))"


definition
    post_f :: "VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> VDMInt
 \<Rightarrow> \<bool>"
    where
    "post_f xyr \<equiv> ((isa_invTrue x \<and> (isa_invTrue y \<and>
        isa_invTrue r)) \<and> ((x / y) = r))"

end
```

```
 --- Got: ---
theory DEFAULT
  imports VDMToolkit
begin
```

```
definition
    pre_f :: "\<bool>"
    where
    "pre_f \<equiv> true"


definition
    post_f :: "VDMInt
 \<Rightarrow> \<bool>"
    where
    "post_f r \<equiv> true"

end
```

E.0.2 Application to Alarm.vdmsl

```
types

  Plant :: schedule : Schedule
           alarms   : set of Alarm
  inv mk_Plant(schedule,alarms) ==
       forall a in set alarms &
     forall peri in set dom schedule &
       QualificationOK(schedule(peri),a.quali);

  Schedule = map Period to set of Expert
inv sch ==
   forall exs in set rng sch &
         exs <> {} and
         forall ex1, ex2 in set exs &
               ex1 <> ex2 => ex1.expertid <> ex2.expertid;

  Period = token;

  Expert :: expertid : ExpertId
            quali    : set of Qualification
  inv ex == ex.quali <> {};

  ExpertId = token;

  Qualification = <Elec> | <Mech> | <Bio> | <Chem>;

  Alarm :: alarmtext : seq of char
           quali     : Qualification

functions

  NumberOfExperts: Period * Plant -> nat
  NumberOfExperts(peri,plant) ==
    card plant.schedule(peri)
  pre peri in set dom plant.schedule;

  ExpertIsOnDuty: Expert * Plant -> set of Period
  ExpertIsOnDuty(ex,mk_Plant(sch,-)) ==
    {peri| peri in set dom sch & ex in set sch(peri)};

  ExpertToPage(a:Alarm,peri:Period,plant:Plant) r: Expert
  pre peri in set dom plant.schedule and
```

```
    a in set plant.alarms
post r in set plant.schedule(peri) and
     a.quali in set r.quali;

QualificationOK: set of Expert * Qualification -> bool
QualificationOK(exs,reqquali) ==
  exists ex in set exs & reqquali in set ex.quali;
```

```
|    1| theory DEFAULT
|    2|   imports VDMToolkit
|    3| begin
|    4|
|    5| type_synonym Schedule = "(Period) \<rightharpoonup> Expert VDMSet
|    6| "
|    7|
|    8|
|    9| type_synonym Period = "VDMToken
|   10| "
|   11|
|   12|
|   13| record Expert =
|   14|         expert_expertid :: (ExpertId)
|   15|         expert_quali :: Qualification
|   16|  VDMSet
|   17|
|   18|
|   19|
|   20|
|   21| type_synonym ExpertId = "VDMToken
|   22| "
|   23|
|   24|
|   25| datatype Qualification = <Bio>| <Chem>| <Elec>| <Mech>
|   26|
|   27|
|   28| record Alarm =
|   29|         alarm_alarmtext :: char VDMSeq
|   30|
|   31|         alarm_quali :: Qualification
|   32|
|   33|
|   34|
|   35|
|   36|
|   37| definition
|   38|   NumberOfExperts :: "(Period) \<Rightarrow> Plant \<Rightarrow> VDMNat
|   39| "
|   40|     where
|   41|     "NumberOfExperts peri plant  \<equiv> card ((plant_schedule plant)peri)"
|   42|
|   43|
|   44| definition
```

```
|   45|   ExpertIsOnDuty :: "Expert \<Rightarrow> Plant \<Rightarrow> (Period)
VDMSet
|   46| "
|   47|     where
|   48|     "ExpertIsOnDuty ex Plant_(sch, _)  \<equiv> {peri | peri \<in> dom
(sch)  & (ex \<in> sch<peri>)}"
|   49|
|   50|
|   51| definition
|   52|   QualificationOK :: "Expert VDMSet
|   53|  \<Rightarrow> Qualification
|   54|  \<Rightarrow> \<bool>"
|   55|     where
|   56|     "QualificationOK exs reqquali  \<equiv> (exists ex \<in> exs & (reqquali
\<in> (ex_quali ex)))"
|   57|
|   58| record Plant =
|   59|         plant_schedule :: (Period) \<rightharpoonup> Expert VDMSet
|   60|
|   61|         plant_alarms :: Alarm VDMSet
|   62|
|   63|
|   64|
|   65|
|   66|
|   67| definition
|   68|   inv_Plant :: "Plant \<Rightarrow> \<bool>"
|   69|     where
|   70|     "inv_Plant Plant_(schedule, alarms) \<equiv> ((isa_invTrue (plant_schedule
p) \<and> isa_invSetElems isa_invTrue (plant_alarms p)) \<and> (forall a \<in>
alarms & (forall peri \<in> dom (schedule) & QualificationOK schedule<peri>
(a_quali a))))"
|   71|
|   72|
|   73| definition
|   74|   pre_NumberOfExperts :: "(Period) \<Rightarrow> Plant \<Rightarrow>
\<bool>"
|   75|     where
|   76|     "pre_NumberOfExperts peri plant  \<equiv> ((isa_invTrue peri \<and>
isa_invTrue plant) \<and> (peri \<in> dom ((plant_schedule plant))))"
|   77|
|   78|
|   79| definition
|   80|   post_NumberOfExperts :: "(Period) \<Rightarrow> Plant \<Rightarrow>
VDMNat
|   81|  \<Rightarrow> \<bool>"
|   82|     where
|   83|     "post_NumberOfExperts peri plant RESULT  \<equiv> (isa_invTrue peri
\<and> (isa_invTrue plant \<and> isa_invVDMNat RESULT))"
|   84|
|   85|
```

```
|   86| definition
|   87|   pre_ExpertIsOnDuty :: "Expert \<Rightarrow> Plant \<Rightarrow> \<bool>"
|   88|     where
|   89|     "pre_ExpertIsOnDuty ex Plant_(sch, _)  \<equiv> (isa_invTrue ex
\<and> isa_invTrue Plant_(sch, -))"
|   90|
|   91|
|   92| definition
|   93|   post_ExpertIsOnDuty :: "Expert \<Rightarrow> Plant \<Rightarrow> (Period)
VDMSet
|   94|  \<Rightarrow> \<bool>"
|   95|     where
|   96|     "post_ExpertIsOnDuty ex Plant_(sch, _) RESULT  \<equiv> (isa_invTrue
ex \<and> (isa_invTrue Plant_(sch, -) \<and> isa_invSetElems isa_invTrue RESULT))"
|   97|
|   98|
|   99| definition
|  100|   pre_ExpertToPage :: "Alarm \<Rightarrow> (Period) \<Rightarrow> Plant
\<Rightarrow> \<bool>"
|  101|     where
|  102|     "pre_ExpertToPage a peri plant  \<equiv> ((isa_invTrue a \<and>
(isa_invTrue peri \<and> isa_invTrue plant)) \<and> ((peri \<in> dom ((plant_schedule
plant))) \<and> (a \<in> (plant_alarms plant))))"
|  103|
|  104|
|  105| definition
|  106|   post_ExpertToPage :: "Alarm \<Rightarrow> (Period) \<Rightarrow> Plant
\<Rightarrow> Expert \<Rightarrow> \<bool>"
|  107|     where
|  108|     "post_ExpertToPage a peri plant r  \<equiv> ((isa_invTrue a \<and>
(isa_invTrue peri \<and> (isa_invTrue plant \<and> isa_invTrue r))) \<and> ((r
\<in> (plant_schedule plant)peri) \<and> ((a_quali a) \<in> (r_quali r))))"
|  109|
|  110|
|  111| definition
|  112|   pre_QualificationOK :: "Expert VDMSet
|  113|  \<Rightarrow> Qualification
|  114|  \<Rightarrow> \<bool>"
|  115|     where
|  116|     "pre_QualificationOK exs reqquali  \<equiv> (isa_invSetElems isa_invTrue
exs \<and> inv_Qualification reqquali)"
|  117|
|  118|
|  119| definition
|  120|   post_QualificationOK :: "Expert VDMSet
|  121|  \<Rightarrow> Qualification
|  122|  \<Rightarrow> \<bool> \<Rightarrow> \<bool>"
|  123|     where
|  124|     "post_QualificationOK exs reqquali RESULT  \<equiv> (isa_invSetElems
isa_invTrue exs \<and> (inv_Qualification reqquali \<and> isa_invTrue RESULT))"
|  125|
```

```
|  126| end
```

E.0.3 Application to FSM3.vdmsl

```
module FSM3
exports
   types
      struct Command
      struct Event
      struct State

      struct Packet_Data
      struct Address
      struct Packet
      struct Bytes
      struct Count
      struct Flag

      struct CandoFSM
      struct FSM3


   values
      --test1: bool
      --test2: bool
      --test3: bool
      --test4: bool
      test5: bool
definitions

------------------------------------------------------------------------------------------------------------------------
-- Basic types
------------------------------------------------------------------------------------------------------------------------
types
   --@doc code generator doesn't like case sensitive differences in quote types (e.g.
       <A> x <a>). This is allowed in VDM, but in Java,
   --    the code generator has to taken into account the underlying OS file system
       restrictions. For Linux this is okay, but for Macs,
   --      the file system is not usually case-sensitive to file names, which generate
       conflicts. These names are better normalised anyhow.
   --
   --@TODO _C, _E, _S for command, event, state.
   Command = <LED_ON_C>| <LED_OFF_C>| <READ_LED_C>| <SET_BRE_C>| <SET_VLED_C>|
       <READ_DIAG_C>| <PROG_DELAY_DIAG_C>| <PROG_OP_MEM_C>| <LED_ALL_OFF_C>| <RUN_MEM_C>|
                   <READ_LFP_C>| <PROG_CLK_CNT_C>| <RESET_ANA_C>| <SET_ANA_C>|
                       <CONFIG_REC_C>| <PROG_ID_C>| <DUMMY_C>;

   -- columns
   Event = <CONT> | <ERROR> | <SPI_TX_FINISH> | <SPI_RX_FINISH> | <LED_ON_E> |
       <SET_VLED_E> | <SET_BRE_E> | <LED_ALL_OFF_E> |
                <PROG_DELAY_DIAG_E> | <PROG_OP_MEM_E> | <RUN_MEM_E> | <PROG_CLK_CNT_E>
                    | <RESET_ANA_E> | <SET_ANA_E> | <CONFIG_REC_E> |
                <PROG_ID_E> | <DUMMY_E> | <READ_LED_E> | <READ_DIAG_E> | <READ_LFP_E> |
                    <GET_CMD_E>;

   -- rows
   State = <start> | <get_cmd> | <LED_off> | <send_packet_3> | <LED_on> | <set_vLED> |
       <send_packet_6> | <set_sDac> | <set_bre> |
                <send_packet_9> | <set_dDac> | <LED_all_off> | <prog_delay_diag> |
                    <prog_op_mem_1> | <send_packet_14> | <prog_op_mem_2> |
```

```
                <run_mem> | <prog_clk_cnt> | <reset_ana> | <set_ana> | <config_rec> |
                    <prog_ID> | <dummy> | <read_LED> | <send_packet_24> |
                <read_DIAG> | <read_LFP> | <receive_packet_27> | <receive_packet_28> |
                    <receive_packet_29> | <receive_packet_30> | <error_> |
                <chip_rst> | <cmd_finish>;

values
   ALL_STATES: set1 of State =
              {<start> ,                      --0;
              <get_cmd> ,                     --1;
              <LED_off> ,                     --2;
              <send_packet_3> ,     --3;
              <LED_on> ,            --4;
              <set_vLED> ,          --5;
              <send_packet_6> ,     --6;
              <set_sDac> ,          --7;
              <set_bre> ,           --8;
              <send_packet_9> ,     --9;
              <set_dDac> ,         --10;
              <LED_all_off> ,      --11;
              <prog_delay_diag> ,  --12;
              <prog_op_mem_1> ,    --13;
              <send_packet_14> ,   --14;
              <prog_op_mem_2> ,    --15;
              <run_mem> ,          --16;
              <prog_clk_cnt> ,     --17;
              <reset_ana> ,        --18;
              <set_ana> ,          --19;
              <config_rec> ,       --20;
              <prog_ID> ,          --21;
              <dummy> ,            --22;
              <read_LED> ,         --23;
              <send_packet_24> ,   --24;
              <read_DIAG> ,        --25;
              <read_LFP> ,                --26;
              <receive_packet_27> , --27;
              <receive_packet_28> , --28;
              <receive_packet_29> , --29;
              <receive_packet_30> , --30;
              <error_> ,                      --31;
                 <chip_rst> ,                 --32;
                 <cmd_finish>                 --33;
              };

   ALL_EVENTS: set1 of Event =
             {<CONT> , <ERROR> , <SPI_TX_FINISH> , <SPI_RX_FINISH> , <LED_ON_E> ,
                 <SET_VLED_E> , <SET_BRE_E> , <LED_ALL_OFF_E> ,
                 <PROG_DELAY_DIAG_E> , <PROG_OP_MEM_E> , <RUN_MEM_E> , <PROG_CLK_CNT_E>
                     , <RESET_ANA_E> , <SET_ANA_E> , <CONFIG_REC_E> ,
                 <PROG_ID_E> , <DUMMY_E> , <READ_LED_E> , <READ_DIAG_E> , <READ_LFP_E> ,
                     <GET_CMD_E>
             };

  send_states: set1 of State = {<send_packet_3>, <send_packet_6>, <send_packet_9>,
      <send_packet_14>, <send_packet_24>};

  receive_states: set1 of State = {<receive_packet_27>, <receive_packet_28>,
      <receive_packet_29>, <receive_packet_30>};

  packet_creator_states: set1 of State =
       { <LED_off>,             --2;
          <LED_on>,                --4;
          <set_vLED>,         --5;
```

```
        <set_sDac>,          --7;
        <set_bre>,           --8;
        <set_dDac> ,   --10;
        <LED_all_off>,   --11;
        <prog_delay_diag>,--12;
        <prog_op_mem_1>, --13;
        <prog_op_mem_2>, --15;
        <run_mem>,           --16;
        <prog_clk_cnt>,  --17;
        <reset_ana>,       --18;
        <set_ana>,           --19;
        <config_rec>,      --20;
        <prog_ID>,           --21;
        <dummy>,               --22;
        <read_LED>,          --23;
        <read_DIAG>,         --25;
        <read_LFP>         --26;
    };

    stage_one_packet_creator_states: set1 of State =
    { <LED_off>,             --2;
        <LED_on>,              --4;
        <set_vLED>,         --5;
        <set_bre>,           --8;
        <LED_all_off>,   --11;
        <prog_delay_diag>,--12;
        <prog_op_mem_1>, --13;
        <run_mem>,           --16;
        <prog_clk_cnt>,  --17;
        <reset_ana>,       --18;
        <set_ana>,           --19;
        <config_rec>,      --20;
        <prog_ID>,           --21;
        <dummy>,               --22;
        <read_LED>,          --23;
        <read_DIAG>,         --25;
        <read_LFP>         --26;
    };

    stage_two_packet_creator_states: set1 of State =
    { <set_sDac>,            --7;
        <set_dDac> ,   --10;
        <prog_op_mem_2>  --15;
    };

    error_states: set1 of State =
    {<chip_rst> ,        --32;
     <cmd_finish>    --33;
    };

-------------------------------------------------------------------------------------------------------
-- Packet types (created as described in v2)
-------------------------------------------------------------------------------------------------------
types
    Packet_Data = <LED_addr> | <NO_LED_ADDR> | <DAC_value> | <diag_delay> | <mem_len> |
        <constructed_data> | <fs_ratio_to_clk> | <rec_config> ;

    Address = <Optrode_addr> ;

    Packet :: addr : [Address]
              cmd  : [Command]
              data : [Packet_Data];
```

```
    Flag = bool;

    Bytes = nat             -- Bytes sent or received
        inv b == b <= PACKET_LENGTH;

    Count = nat      -- counter for error
        inv t == t <= MAX_COUNT;

values
    PACKET_LENGTH :nat1  =      3;
    MAX_COUNT            :nat1      =   2;
-------------------------------------------------------------------------------------------------------
-- FSM and map (sub-)types
-------------------------------------------------------------------------------------------------------
types
    StateMap = map State to State
    inv s ==
        --@doc states cannot map to the initial state "Start"
        not <start> in set rng s
        and
        --@doc start state can only map to <get_cmd> or <error>
        (<start> in set dom s => s(<start>) in set { <get_cmd>, <error_> })
        and
        --@doc error state can only go to error_, get_cmd, or chip_rst
        (<error_> in set dom s => s(<error_>) in set { <get_cmd>, <chip_rst>, <error_>})
        and
        --@doc cmd_finish can only lead to error
        (<cmd_finish> in set dom s => s(<cmd_finish>) = <error_>)
        and
        --@doc chip_reset can only lead to error
        (<chip_rst> in set dom s => s(<chip_rst>) = <error_>)
        and
        --@doc packet_creator_states can only go to send packet states or error, prevents
            packets being overwritten
        (forall p in set packet_creator_states & p in set dom s => s(p) in set ({<error_>}
            union send_states))
        and
        --receive states map to either a stage 2 packet creator state or the finish state
         (forall r in set receive_states & r in set dom s => s(r) in set {<cmd_finish>, r,
             <error_>} union stage_two_packet_creator_states)

        --ERRONOUS INVARIANTS - don't contain errors but throw them for initial FSM
            (commented out)
        and
        --@doc get_cmd maps to either error or a stage 1 packet state
        --@doc get_cmd cannot lead to either send or receive states or stage 2 packet
            states
        --@doc simplified invariatn
        --(<get_cmd> in set dom s =>
        --     s(<get_cmd>) in set ({<error_>} union stage_one_packet_creator_states)
        --                                         \
        --                                         (send_states union receive_states union
            stage_two_packet_creator_states)
        (<get_cmd> in set dom s =>
               s(<get_cmd>) in set ({<error_>} union stage_one_packet_creator_states)
        )
        ;

    --@doc transmission map ensures that send_states are mapped only to receive_states,
         if any
    TXMap = StateMap
    inv m ==
        dom m subset send_states
```

```
    and
    rng m subset receive_states;

--@doc identity state map
IdMap = StateMap
inv m ==
    forall s in set dom m & m(s) = s;

--@doc maps every state to <error_>
ErrorMap = StateMap
inv em ==
    forall s in set dom em & em(s) = <error_>;

--@doc maps every packet creator state to a send state so no packets are overwritten
PacketMap = StateMap
inv pm ==
    dom pm subset packet_creator_states
    and
    rng pm subset send_states;

--@doc maps every receive state maps to a stage 2 packet state or the command finish
    state
ReceiveMap = StateMap
inv rm ==
    dom rm subset receive_states
    and
    rng rm subset stage_two_packet_creator_states union {<cmd_finish>};

--@doc total statemap (or FSM row) must contain all states in the domain of the set
TStateMap = StateMap
inv sm == dom sm = ALL_STATES;

--@doc FSM is a map of event to map of state to state that can be partial
FSM = map Event to StateMap;

--@doc FSM that is total on events and state map for every event
TFSM = FSM
inv fsm ==
    dom fsm = ALL_EVENTS
    and
    forall e in set dom fsm & is_TStateMap(fsm(e));

--@doc Cando FSM is total on events and states
CandoFSM = TFSM
inv fsm ==
    --@doc check that all send states map to receive state if event = CONT
    --@doc that is, the StateMap from <CONT> event over send_states is a transmission
        map
    --@doc if this wasn't total map, then we would need
    --      <CONT> in set dom fsm => is_TXMap(send_states <: fsm(<CONT>))
    is_TXMap(send_states <: fsm(<CONT>))
    and
    --@doc check all send states map to themselves if event = SPI_TX_FINISH
    is_IdMap(send_states <: fsm(<SPI_TX_FINISH>))
    and
    --@doc check all receive states map to themselves if event = SPI_RX_FINISH
    is_IdMap(receive_states <: fsm(<SPI_RX_FINISH>))
    and
    --@doc on packet creation states, CONT leads to a send state
    --previously:
    --rng (packet_creator_states<:fsm(<CONT>)) subset send_states
    is_PacketMap(packet_creator_states <: fsm(<CONT>))
    and
```

```
        --@todo? should all others be error; no. Only those two.
        is_ErrorMap(error_states <: (packet_creator_states<-:fsm(<CONT>)))
        -- is this not the same as is_ErrorMap({cmd_finish>, <chip_rst>} <:
            fsm(<CONT>))?
        and
    --@doc on CONT event, start state leads to get_cmd or error elsewhere
    fsm(<CONT>)(<start>) = <get_cmd>
    and
    (forall x in set dom ({<CONT>}<-:fsm) & fsm(x)(<start>) = <error_>)
    and
    --@doc CONT/GET_CMD_E error behaviour; @todo missing other states?
    fsm(<CONT>)(<error_>) = <chip_rst>
    and
    fsm(<GET_CMD_E>)(<error_>) = <get_cmd>
    and
    --is_IdMap(({<CONT>,<GET_CMD_E>}<-:fsm)); this is not quite right, needs the forall
    --If I wanted is_IdMap, I would have to matrix transpose to make x a row rather
        than a column
    (forall x in set dom ({<CONT>,<GET_CMD_E>}<-:fsm) & fsm(x)(<error_>) = <error_>)
    and
    --@doc receive states map to cmd_finish or to a stage 2 packet state under CONT
    is_ReceiveMap(receive_states <: fsm(<CONT>))
    ;

functions

  --@doc conver partial to total CandoFSM
  --      all undefined state transitions set to <error_>
  sm2tsm_VDM: StateMap -> TStateMap
  sm2tsm_VDM(sm) ==
     --@todo/codegen/FSM3 CG-Error: ATypeMultipleBind
     -- sm munion { s |-> <error_> | s : State & not s in set dom sm };
     sm munion { s |-> <error_> | s in set ALL_STATES \ dom sm };

  --@doc using munion to encode the fact the totalisation does not mess with the
      original fsm
  fsm2tfsm_VDM: FSM -> TFSM
  fsm2tfsm_VDM(fsm) ==
     { e |-> sm2tsm(fsm(e)) | e in set dom fsm }
        munion
     { e |-> sm2tsm({|->}) | e in set ALL_EVENTS \ dom fsm };

  --@doc in Isabelle, we defined this the other way round: make everywone sm2trm({|->})
      then update the dom fsm with fsm(e)
  --      this is necessary because there is no comprehension in Isabelle, and all maps
      are total. So we make it total on empty
  --      for all events, then update it on events of the given FSM.
  sm2tsm_VDM2: StateMap -> TStateMap
  sm2tsm_VDM2(sm) ==
     { s |-> <error_> | s in set ALL_STATES }
     ++
     { s |-> sm(s) | s in set dom sm };

  fsm2tfsm_VDM2: FSM -> TFSM
  fsm2tfsm_VDM2(fsm) ==
     { e |-> sm2tsm({|->}) | e in set ALL_EVENTS }
     ++
     { e |-> sm2tsm(fsm(e)) | e in set dom fsm };

  --@doc because of the various issues with lemmas and complexity, kept it dead simple
      (i.e. no extra operators) and closer
  --      to how Isabelle likes it but without compromising on meaning, albeit on
      clarity perhaps.
```

```
    sm2tsm: StateMap -> TStateMap
    sm2tsm(sm) ==
       { s |-> if s in set dom sm then sm(s) else <error_> | s in set ALL_STATES };

    fsm2tfsm: FSM -> TFSM
    fsm2tfsm(fsm) ==
       { e |-> if e in set dom fsm then sm2tsm(fsm(e)) else sm2tsm({|->}) | e in set
           ALL_EVENTS };

values
    --@doc codegen doesn't like these function equalities
    --test1: bool = sm2tsm_VDM2 = sm2tsm_VDM;
    --test2: bool = sm2tsm    = sm2tsm_VDM;
    --test3: bool = fsm2tfsm_VDM2 = fsm2tfsm_VDM;
    --test4: bool = fsm2tfsm  = fsm2tfsm_VDM;
    test5: bool = ({<SPI_TX_FINISH>,<PROG_OP_MEM_E>}<-:recommended_fsm) =
        ({<SPI_TX_FINISH>,<PROG_OP_MEM_E>}<-:original_initial_fsm)

values
    --recommended_fsm is different to initial_fsm as it includes corrections we believe
        are needed from the initial fsm we were given in the code
    -- the changes are highlighted with @TODO marker
  recommended_fsm: FSM = {
                                        <CONT> |-> {     -- event 0 = CONT
                                                    <start>                    |->
                                                        <get_cmd>,
                                                    <get_cmd>              |-> <LED_off>,
                                                    <LED_off>              |->
                                                        <send_packet_3>,
                                                    <send_packet_3>     |->
                                                        <receive_packet_27>,
                                                    <LED_on>                   |->
                                                        <send_packet_3>,
                                                    <set_vLED>             |->
                                                        <send_packet_6>,
                                                    <send_packet_6>     |->
                                                        <receive_packet_28>,
                                                    <set_sDac>             |->
                                                        <send_packet_3>,
                                                    <set_bre>              |->
                                                        <send_packet_9>,
                                                    <send_packet_9>     |->
                                                        <receive_packet_29>,
                                                    <set_dDac>             |->
                                                        <send_packet_3>,
                                                    <LED_all_off>       |->
                                                        <send_packet_3>,
                                                    <prog_delay_diag> |-> <send_packet_3>,
                                                    <prog_op_mem_1>     |->
                                                        <send_packet_14>,
                                                    <send_packet_14>    |->
                                                        <receive_packet_30>,
                                                    <prog_op_mem_2>  |-> <send_packet_3>,
                                                    <run_mem>              |->
                                                        <send_packet_3>,
                                                    <prog_clk_cnt>      |->
                                                        <send_packet_3>,
                                                    <reset_ana>            |->
                                                        <send_packet_3>,
                                                    <set_ana>              |->
                                                        <send_packet_3>,
                                                    <config_rec>           |->
                                                        <send_packet_3>,
```

```
                                <prog_ID>               |->
                                    <send_packet_3>,
                                <dummy>                     |->
                                    <send_packet_3>,
                                <read_LED>              |->
                                    <send_packet_24>,
                                <send_packet_24>    |->
                                    <receive_packet_27>,
                                <read_DIAG>             |->
                                    <send_packet_24>,
                                <read_LFP>              |->
                                    <send_packet_24>,
                                <receive_packet_27> |-> <cmd_finish>,
                                <receive_packet_28> |-> <set_sDac>,
                                <receive_packet_29> |-> <set_dDac>,
                                <receive_packet_30> |->
                                    <prog_op_mem_2>,
                                <error_>                    |->
                                    <chip_rst>,
                                <chip_rst>              |-> <error_>,
                                <cmd_finish>            |-> <error_>
                                },
  <ERROR> |-> {|->}, -- event 1 = ERROR
  <SPI_TX_FINISH> |-> {       -- event 2 = SPI_TX_FINISH
                    <send_packet_3> |-> <send_packet_3>,
                    <send_packet_6> |-> <send_packet_6>,
                    <send_packet_9> |-> <send_packet_9>,
                                --@TODO this was
                        <send_packet_9> |-> <set_dDAC>
                    <send_packet_14> |-> <send_packet_14>,
                    <send_packet_24> |-> <send_packet_24>
                        },
<SPI_RX_FINISH> |-> { -- event 3 = SPI_RX_FINISH
                    <receive_packet_27> |->
                        <receive_packet_27>,
                    <receive_packet_28> |->
                        <receive_packet_28>,
                    <receive_packet_29> |->
                        <receive_packet_29>,
                    <receive_packet_30> |->
                        <receive_packet_30>
                        },
    <LED_ON_E>              |-> { <get_cmd> |-> <LED_on>},
                 -- event 4 = LED_ON_E
    <SET_VLED_E>        |-> { <get_cmd> |-> <set_vLED>},
                -- event 5 = SET_VLED_E
    <SET_BRE_E>             |-> { <get_cmd> |-> <set_bre>},
               -- event 6 = SET_BRE_E
    <LED_ALL_OFF_E>     |-> { <get_cmd> |->
        <LED_all_off>}, -- event 7 = LED_ALL_OFF_E
    <PROG_DELAY_DIAG_E> |-> { <get_cmd> |->
        <prog_delay_diag>}, -- event 8 = PROG_DELAY_DIAG_E
    <PROG_OP_MEM_E>     |-> { <get_cmd> |->
        <prog_op_mem_1>}, -- event 9 = PROG_OP_MEM_E
        --@TODO this was <get_cmd> |-> <prog_op_mem_2>
    <RUN_MEM_E>             |-> { <get_cmd> |-> <run_mem>},
               -- event 10 = RUN_MEM_E
    <PROG_CLK_CNT_E>    |-> { <get_cmd> |->
        <prog_clk_cnt>},  -- event 11 = PROG_CLK_CNT_E
    <RESET_ANA_E>       |-> { <get_cmd> |-> <reset_ana>},
             -- event 12 = RESET_ANA_E
    <SET_ANA_E>             |-> { <get_cmd> |-> <set_ana>},
               -- event 13 = SET_ANA_E
```

```
                                <CONFIG_REC_E>       |-> { <get_cmd> |-> <config_rec>},
                                      -- event 14 = CONFIG_REC_E
                                <PROG_ID_E>             |-> { <get_cmd> |-> <prog_ID>},
                                            -- event 15 = PROG_IDE_E
                                <DUMMY_E>               |-> { <get_cmd> |-> <dummy>},
                                                    -- event 16 = DUMMY_E
                                <READ_LED_E>            |-> { <get_cmd> |->
                                    <read_LED>},          -- event 17 = READ_LED_E
                                <READ_DIAG_E>       |-> { <get_cmd> |-> <read_DIAG>},
                                        -- event 18 = READ_DIAG_E
                                <READ_LFP_E>            |-> { <get_cmd> |->
                                    <read_LFP>},          -- event 19 = READ_LFP_E
                                <GET_CMD_E>               |-> {

                                    -- event 20 = GET_CMD_E
                                                                          <error_>
                                                                              |->
                                                                              <get_cmd>
                                                                          --<chip_rst>
                                                                              |->
                                                                              <get_cmd>

                                                                              --@TODO
                                                                              this
                                                                              was
                                                                              omitted
                                                                      }
                            };

   --@doc the original initial FSM breaks the discovered invariant
  original_initial_fsm: map Event to (map State to State) = recommended_fsm ++
                        { <SPI_TX_FINISH> |-> {       -- event 2 = SPI_TX_FINISH
                                              <send_packet_3> |-> <send_packet_3>,
                                           <send_packet_6> |-> <send_packet_6>,
                                              <send_packet_9> |-> <set_dDac>,
                                                                  -- @TODO THIS IS AN
                                                  ERROR <send_packet_9> |->
                                                  <send_packet_9>
                                              <send_packet_14> |-> <send_packet_14>,
                                              <send_packet_24> |-> <send_packet_24>
                                                 },
                                <PROG_OP_MEM_E>     |-> { <get_cmd> |->
                                    <prog_op_mem_2>} -- event 9 = PROG_OP_MEM_E
                                    --@TODO THIS IS ALSO AN ERROR <get_cmd> |->
                                    <prog_op_mem_1>
                            };
-------------------------------------------------------------------------------------------
-- FSM3 State
-------------------------------------------------------------------------------------------

state FSM3 of
          fsm                          :            CandoFSM  --1
          currentSt                  :               State          --2
          currentEvt               :                Event     --3
          currentCmd               :                Command   --4
          command_finish_flag :             Flag      --5
          optrode_TX_finish    :               Flag      --6
          optrode_RX_finish    :               Flag      --7
          s_packet                   :                [Packet]  --8
          bytes_received       :               Bytes     --9
          bytes_sent               :               Bytes     --10
          tx_cnt                     :               Count     --11
inv mk_FSM3(-,s,-,-,-,-,-,p,-,-,-) ==
```

```
          --@doc only at the start can the s_packet be nil
          (s = <start> <=> p = nil)
init FSM3 ==
   FSM3 = mk_FSM3(fsm2tfsm(recommended_fsm),
                           <start>, <CONT>, <READ_LFP_C>,
                           false, true, true, nil, 3, 0, 0)

end


end FSM3
```

```
|    1| theory FSM3
|    2|   imports VDMToolkit
|    3| begin
|    4|
|    5| datatype Command = <CONFIG_REC_C>| <DUMMY_C>| <LED_ALL_OFF_C>| <LED_OFF_C>|
<LED_ON_C>| <PROG_CLK_CNT_C>| <PROG_DELAY_DIAG_C>| <PROG_ID_C>| <PROG_OP_MEM_C>|
<READ_DIAG_C>| <READ_LED_C>| <READ_LFP_C>| <RESET_ANA_C>| <RUN_MEM_C>| <SET_ANA_C>|
<SET_BRE_C>| <SET_VLED_C>
|    6|
|    7|
|    8| datatype Event = <CONFIG_REC_E>| <CONT>| <DUMMY_E>| <ERROR>| <GET_CMD_E>|
<LED_ALL_OFF_E>| <LED_ON_E>| <PROG_CLK_CNT_E>| <PROG_DELAY_DIAG_E>| <PROG_ID_E>|
<PROG_OP_MEM_E>| <READ_DIAG_E>| <READ_LED_E>| <READ_LFP_E>| <RESET_ANA_E>| <RUN_MEM_E>|
<SET_ANA_E>| <SET_BRE_E>| <SET_VLED_E>| <SPI_RX_FINISH>| <SPI_TX_FINISH>
|    9|
|   10|
|   11| datatype State = <LED_all_off>| <LED_off>| <LED_on>| <chip_rst>| <cmd_finish>|
<config_rec>| <dummy>| <error_>| <get_cmd>| <prog_ID>| <prog_clk_cnt>| <prog_delay_diag>|
<prog_op_mem_1>| <prog_op_mem_2>| <read_DIAG>| <read_LED>| <read_LFP>| <receive_packet_27
<receive_packet_28>| <receive_packet_29>| <receive_packet_30>| <reset_ana>|
<run_mem>| <send_packet_14>| <send_packet_24>| <send_packet_3>| <send_packet_6>|
<send_packet_9>| <set_ana>| <set_bre>| <set_dDac>| <set_sDac>| <set_vLED>| <start>
|   12|
|   13|
|   14| abbreviation
|   15|  ALL_STATES :: State
|   16|  VDMSet
|   17|  where
|   18| "ALL_STATES \<equiv> start, get_cmd, LED_off, send_packet_3, LED_on,
set_vLED, send_packet_6, set_sDac, set_bre, send_packet_9, set_dDac, LED_all_off,
prog_delay_diag, prog_op_mem_1, send_packet_14, prog_op_mem_2, run_mem, prog_{c}lk_cnt,
reset_ana, set_ana, config_rec, prog_ID, dummy, read_LED, send_packet_24, read_DIAG,
read_LFP, receive_packet_27, receive_packet_28, receive_packet_29, receive_packet_30,
error_, chip_rst, cmd_finish"
|   19|
|   20| abbreviation
|   21|  ALL_EVENTS :: Event
|   22|  VDMSet
|   23|  where
|   24| "ALL_EVENTS \<equiv> CONT, ERROR, SPI_TX_FINISH, SPI_RX_FINISH, LED_ON_E,
SET_VLED_E, SET_BRE_E, LED_ALL_OFF_E, PROG_DELAY_DIAG_E, PROG_OP_MEM_E, RUN{_}MEM_E,
```

```
PROG_CLK_CNT_E, RESET_ANA_E, SET_ANA_E, CONFIG_REC_E, PROG_ID_E, DUMMY_E, READ_LED_E,
READ_DIAG_E, READ_LFP_E, GET_CMD_E"
|   25|
|   26| abbreviation
|   27|  send_states :: State
|   28|  VDMSet
|   29|  where
|   30| "send_states \<equiv> send_packet_3, send_packet_6, send_pa{c}ket_9,
send_packet_14, send_packet_24"
|   31|
|   32| abbreviation
|   33|  receive_states :: State
|   34|  VDMSet
|   35|  where
|   36| "receive_states \<equiv> receive_packet_27, receive_packet_28,{ }receive_packet_2
receive_packet_30"
|   37|
|   38| abbreviation
|   39|  packet_creator_states :: State
|   40|  VDMSet
|   41|  where
|   42| "packet_creator_states \<equiv> LED_off, LED_on, set_vLED, set_sDac,
set_bre, set_dDac, LED_all_off, prog_delay_diag, prog_op_mem_1, prog_op{_}mem_2,
run_mem, prog_clk_cnt, reset_ana, set_ana, config_rec, prog_ID, dummy, read_LED,
read_DIAG, read_LFP"
|   43|
|   44| abbreviation
|   45|  stage_one_packet_creator_states :: State
|   46|  VDMSet
|   47|  where
|   48| "stage_one_packet_creator_states \<equiv> LED_off, LED_on, set_vLED,
set_bre, LED_all_off, prog_delay_diag, prog_op_mem_1, run_mem, {p}rog_clk_cnt,
reset_ana, set_ana, config_rec, prog_ID, dummy, read_LED, read_DIAG, read_LFP"
|   49|
|   50| abbreviation
|   51|  stage_two_packet_creator_states :: State
|   52|  VDMSet
|   53|  where
|   54| "stage_two_packet_creator_states \<equiv> set_sDac, set_dD{a}c, prog_op_mem_2"
|   55|
|   56| abbreviation
|   57|  error_states :: State
|   58|  VDMSet
|   59|  where
|   60| "error_states \<equiv> chip_rst, {c}md_finish"
|   61|
|   62| datatype Packet_Data = <DAC_value>| <LED_addr>| <NO_LED_ADDR>| <constructed_data>
<diag_delay>| <fs_ratio_to_clk>| <mem_len>| <rec_config>
|   63|
|   64|
|   65| type_synonym Address = "<Optrode_addr>"
```

```
|   66|
|   67|
|   68| record Packet =
|   69|         packet_addr :: <Optrode_addr>
|   70|         packet_cmd :: Command
|   71|
|   72|         packet_data :: Packet_Data
|   73|
|   74|
|   75|
|   76|
|   77| type_synonym Flag = "\<bool>"
|   78|
|   79|
|   80| type_synonym Bytes = "VDMNat
|   81| "
|   82|
|   83|
|   84| type_synonym Count = "VDMNat
|   85| "
|   86|
|   87|
|   88| abbreviation
|   89|  PACKET_LENGTH :: VDMNat1
|   90|  where
|   91| "PACKET_LENGTH \<equiv> 3"
|   92|
|   93| abbreviation
|   94|  MAX_COUNT :: VDMNat1
|   95|  where
|   96| "MAX_COUNT \<equiv> 2"
|   97|
|   98| type_synonym StateMap = "State
|   99|  \<rightharpoonup> State
|  100| "
|  101|
|  102|
|  103| type_synonym TXMap = "State
|  104|  \<rightharpoonup> State
|  105| "
|  106|
|  107|
|  108| type_synonym IdMap = "State
|  109|  \<rightharpoonup> State
|  110| "
|  111|
|  112|
|  113| type_synonym ErrorMap = "State
|  114|  \<rightharpoonup> State
|  115| "
|  116|
```

```
|  117|
|  118| type_synonym PacketMap = "State
|  119|  \<rightharpoonup> State
|  120| "
|  121|
|  122|
|  123| type_synonym ReceiveMap = "State
|  124|  \<rightharpoonup> State
|  125| "
|  126|
|  127|
|  128| type_synonym TStateMap = "State
|  129|  \<rightharpoonup> State
|  130| "
|  131|
|  132|
|  133| type_synonym FSM = "Event
|  134|  \<rightharpoonup> State
|  135|  \<rightharpoonup> State
|  136| "
|  137|
|  138|
|  139| type_synonym TFSM = "Event
|  140|  \<rightharpoonup> State
|  141|  \<rightharpoonup> State
|  142| "
|  143|
|  144|
|  145| type_synonym CandoFSM = "Event
|  146|  \<rightharpoonup> State
|  147|  \<rightharpoonup> State
|  148| "
|  149|
|  150|
|  151|
|  152| definition
|  153|   sm2tsm_VDM :: "State
|  154|  \<rightharpoonup> State
|  155|  \<Rightarrow> State
|  156|  \<rightharpoonup> State
|  157| "
|  158|     where
|  159|     "sm2tsm_VDM sm  \<equiv> sm \<union> serror_[s in set org.overture.codegen.ir
|  160|
|  161|
|  162| definition
|  163|   sm2tsm_VDM2 :: "State
|  164|  \<rightharpoonup> State
|  165|  \<Rightarrow> State
|  166|  \<rightharpoonup> State
|  167| "
```

```
|  168|     where
|  169|     "sm2tsm_VDM2 sm  \<equiv> serror_[s in set ALL_STATES]ACompMapExpIR
\<ACompMapExpIR> ssm(s)[s in set org.overture.codegen.ir.expressions.AMapDomainUnaryExpIR(
|  170|
|  171|
|  172| definition
|  173|   sm2tsm :: "State
|  174|  \<rightharpoonup> State
|  175|  \<Rightarrow> State
|  176|  \<rightharpoonup> State
|  177| "
|  178|     where
|  179|     "sm2tsm sm  \<equiv> $node.left \<ACompMapExp> $node.right"
|  180|
|  181| abbreviation
|  182|  recommended_fsm :: Event
|  183|  \<rightharpoonup> State
|  184|  \<rightharpoonup> State
|  185|  where
|  186| "recommended_fsm \<equiv> $node.left \<AEnumMapExpIR> $node.right"
|  187|
|  188| abbreviation
|  189|  original_initial_fsm :: Event
|  190|  \<rightharpoonup> State
|  191|  \<rightharpoonup> State
|  192|  where
|  193| "original_initial_fsm \<equiv> recommended_fsm \<ACompMapExpIR> [SPI_TX_FINISH[se
send_packet_6send_packet_6, send_packet_9set_dDac, send_packet_14send_packet_14,
send_packet_24send_packet_24], PROG_OP_MEM_E[get_cmdprog_op_mem_2]]"
|  194|
|  195| abbreviation
|  196|  test5 :: \<bool> where
|  197| "test5 \<equiv> ([SPI_TX_FINISH, PROG_OP_MEM_E] \<ADomainResByBinaryExp>
recommended_fsm = [SPI_TX_FINISH, PROG_OP_MEM_E] \<ADomainResByBinaryExp> original_initia
|  198|
|  199|
|  200| definition
|  201|   fsm2tfsm :: "Event
|  202|  \<rightharpoonup> State
|  203|  \<rightharpoonup> State
|  204|  \<Rightarrow> Event
|  205|  \<rightharpoonup> State
|  206|  \<rightharpoonup> State
|  207| "
|  208|     where
|  209|     "fsm2tfsm fsm  \<equiv> $node.left \<ACompMapExp> $node.right"
|  210|
|  211| record FSM3 =
|  212|         fSM3_fsm :: Event
|  213|  \<rightharpoonup> State
|  214|  \<rightharpoonup> State
```

```
|  215|
|  216|         fSM3_currentSt :: State
|  217|
|  218|         fSM3_currentEvt :: Event
|  219|
|  220|         fSM3_currentCmd :: Command
|  221|
|  222|         fSM3_command_finish_flag :: \<bool>
|  223|         fSM3_optrode_TX_finish :: \<bool>
|  224|         fSM3_optrode_RX_finish :: \<bool>
|  225|         fSM3_s_packet :: Packet
|  226|         fSM3_bytes_received :: (Bytes)
|  227|         fSM3_bytes_sent :: (Bytes)
|  228|         fSM3_tx_cnt :: (Count)
|  229|
|  230| $Isa.transState( $node.InvDecl )
|  231|
|  232| $Isa.transState( $node.InitDecl )
|  233|
|  234|
|  235| definition
|  236|   fsm2tfsm_VDM2 :: "Event
|  237|  \<rightharpoonup> State
|  238|  \<rightharpoonup> State
|  239|  \<Rightarrow> Event
|  240|  \<rightharpoonup> State
|  241|  \<rightharpoonup> State
|  242| "
|  243|     where
|  244|     "fsm2tfsm_VDM2 fsm  \<equiv> esm2tsm([])[e in set ALL_EVENTS]ACompMapExpIR
\<ACompMapExpIR> esm2tsm(fsm(e))[e in set org.overture.codegen.ir.expressions.AMapDomainU
|  245|
|  246|
|  247| definition
|  248|   fsm2tfsm_VDM :: "Event
|  249|  \<rightharpoonup> State
|  250|  \<rightharpoonup> State
|  251|  \<Rightarrow> Event
|  252|  \<rightharpoonup> State
|  253|  \<rightharpoonup> State
|  254| "
|  255|     where
|  256|     "fsm2tfsm_VDM fsm  \<equiv> esm2tsm(fsm(e))[e in set org.overture.codegen.ir.
\<union> esm2tsm([])[e in set org.overture.codegen.ir.expressions.ASetDifferenceBinaryExp
|  257|
|  258|
|  259| definition
|  260|   inv_Packet :: "Packet \<Rightarrow> \<bool>"
|  261|     where
|  262|     "inv_Packet p \<equiv> (inv_Address (packet_addr p) \<and> (inv_Command
(packet_cmd p) \<and> inv_Packet_Data (packet_data p)))"
```

```
|  263|
|  264|
|  265| definition
|  266|   inv_Flag :: "(Flag) \<Rightarrow> \<bool>"
|  267|     where
|  268|     "inv_Flag f \<equiv> isa_invTrue f"
|  269|
|  270|
|  271| definition
|  272|   inv_Bytes :: "(Bytes) \<Rightarrow> \<bool>"
|  273|     where
|  274|     "inv_Bytes b \<equiv> (isa_invTrue b \<and> b \<le> PACKET_LENGTH)"
|  275|
|  276|
|  277| definition
|  278|   inv_Count :: "(Count) \<Rightarrow> \<bool>"
|  279|     where
|  280|     "inv_Count t \<equiv> (isa_invTrue c \<and> t \<le> MAX_COUNT)"
|  281|
|  282|
|  283| definition
|  284|   inv_PACKET_LENGTH :: "\<bool>"
|  285|     where
|  286|     "inv_PACKET_LENGTH  \<equiv> isa_invVDMNat1 PACKET_LENGTH"
|  287|
|  288|
|  289| definition
|  290|   inv_MAX_COUNT :: "\<bool>"
|  291|     where
|  292|     "inv_MAX_COUNT  \<equiv> isa_invVDMNat1 MAX_COUNT"
|  293|
|  294|
|  295| definition
|  296|   inv_StateMap :: "(StateMap) \<Rightarrow> \<bool>"
|  297|     where
|  298|     "inv_StateMap s \<equiv> (isa_invTrue s \<and> (not (('start' \<in>
rng (s))) \<and> ((not (('start' \<in> dom (s))) or (s<'start'> \<in> {'get_cmd',
'error_'})) \<and> ((not (('error_' \<in> dom (s))) or (s<'error_'> \<in> {'get_cmd',
'chip_rst', 'error_'})) \<and> ((not (('cmd_finish' \<in> dom (s))) or (s<'cmd_finish'>
= 'error_')) \<and> ((not (('chip_rst' \<in> dom (s))) or (s<'chip_rst'> = 'error_'))
\<and> ((forall p \<in> packet_creator_states & (not ((p \<in> dom (s))) or
(s<p> \<in> [error_] \<union> send_states))) \<and> ((forall r \<in> receive_states
& (not ((r \<in> dom (s))) or (s<r> \<in> [cmd_finish, r, error_] \<union> stage_two_pack
\<and> (not (('get_cmd' \<in> dom (s))) or (s<'get_cmd'> \<in> [error_] \<union>
stage_one_packet_creator_states)))))))))"
|  299|
|  300|
|  301| definition
|  302|   inv_TXMap :: "(TXMap) \<Rightarrow> \<bool>"
|  303|     where
|  304|     "inv_TXMap m \<equiv> (isa_invTrue t \<and> ((dom (m) \<subset>
```

```
send_states) \<and> (rng (m) \<subset> receive_states)))"
|  305|
|  306|
|  307| definition
|  308|   inv_IdMap :: "(IdMap) \<Rightarrow> \<bool>"
|  309|     where
|  310|     "inv_IdMap m \<equiv> (isa_invTrue i \<and> (forall s \<in> dom
(m) & (m<s> = s)))"
|  311|
|  312|
|  313| definition
|  314|   inv_ErrorMap :: "(ErrorMap) \<Rightarrow> \<bool>"
|  315|     where
|  316|     "inv_ErrorMap em \<equiv> (isa_invTrue e \<and> (forall s \<in>
dom (em) & (em<s> = 'error_')))"
|  317|
|  318|
|  319| definition
|  320|   inv_PacketMap :: "(PacketMap) \<Rightarrow> \<bool>"
|  321|     where
|  322|     "inv_PacketMap pm \<equiv> (isa_invTrue p \<and> ((dom (pm) \<subset>
packet_creator_states) \<and> (rng (pm) \<subset> send_states)))"
|  323|
|  324|
|  325| definition
|  326|   inv_ReceiveMap :: "(ReceiveMap) \<Rightarrow> \<bool>"
|  327|     where
|  328|     "inv_ReceiveMap rm \<equiv> (isa_invTrue r \<and> ((dom (rm) \<subset>
receive_states) \<and> (rng (rm) \<subset> stage_two_packet_creator_states \<union>
[cmd_finish])))"
|  329|
|  330|
|  331| definition
|  332|   inv_TStateMap :: "(TStateMap) \<Rightarrow> \<bool>"
|  333|     where
|  334|     "inv_TStateMap sm \<equiv> (isa_invTrue t \<and> (dom (sm) = ALL_STATES))"
|  335|
|  336|
|  337| definition
|  338|   inv_FSM :: "(FSM) \<Rightarrow> \<bool>"
|  339|     where
|  340|     "inv_FSM f \<equiv> isa_invTrue f"
|  341|
|  342|
|  343| definition
|  344|   inv_TFSM :: "(TFSM) \<Rightarrow> \<bool>"
|  345|     where
|  346|     "inv_TFSM fsm \<equiv> (isa_invTrue t \<and> ((dom (fsm) = ALL_EVENTS)
\<and> (forall e \<in> dom (fsm) & (fsm<e> hasType State
|  347|  \<rightharpoonup> State
|  348| ))))"
```

```
|  349|
|  350|
|  351| definition
|  352|   inv_CandoFSM :: "(CandoFSM) \<Rightarrow> \<bool>"
|  353|     where
|  354|     "inv_CandoFSM fsm \<equiv> (isa_invTrue c \<and> ((send_states \<ADomainResTo
fsm(CONT) hasType State
|  355|  \<rightharpoonup> State
|  356| ) \<and> ((send_states \<ADomainResToBinaryExp> fsm(SPI_TX_FINISH) hasType
State
|  357|  \<rightharpoonup> State
|  358| ) \<and> ((receive_states \<ADomainResToBinaryExp> fsm(SPI_RX_FINISH)
hasType State
|  359|  \<rightharpoonup> State
|  360| ) \<and> ((packet_creator_states \<ADomainResToBinaryExp> fsm(CONT)
hasType State
|  361|  \<rightharpoonup> State
|  362| ) \<and> ((error_states \<ADomainResToBinaryExp> org.overture.codegen.ir.expressi
hasType State
|  363|  \<rightharpoonup> State
|  364| ) \<and> ((fsm<'CONT'><'start'> = 'get_cmd') \<and> ((forall x \<in>
dom ([CONT] \<ADomainResByBinaryExp> fsm) & (fsm<x><'start'> = 'error_')) \<and>
((fsm<'CONT'><'error_'> = 'chip_rst') \<and> ((fsm<'GET_CMD_E'><'error_'> =
'get_cmd') \<and> ((forall x \<in> dom ([CONT, GET_CMD_E] \<ADomainResByBinaryExp>
fsm) & (fsm<x><'error_'> = 'error_')) \<and> (receive_states \<ADomainResToBinaryExp>
fsm(CONT) hasType State
|  365|  \<rightharpoonup> State
|  366| ))))))))))))"
|  367|
|  368|
|  369| definition
|  370|   inv_recommended_fsm :: "\<bool>"
|  371|     where
|  372|     "inv_recommended_fsm  \<equiv> isa_invTrue recommended_fsm"
|  373|
|  374|
|  375| definition
|  376|   inv_original_initial_fsm :: "\<bool>"
|  377|     where
|  378|     "inv_original_initial_fsm  \<equiv> isa_invTrue original_initial_fsm"
|  379|
|  380|
|  381| definition
|  382|   inv_test5 :: "\<bool>"
|  383|     where
|  384|     "inv_test5  \<equiv> isa_invTrue test5"
|  385|
|  386|
|  387| definition
|  388|   pre_sm2tsm_VDM :: "State
|  389|  \<rightharpoonup> State
```

```
|  390|  \<Rightarrow> \<bool>"
|  391|     where
|  392|     "pre_sm2tsm_VDM sm  \<equiv> isa_invTrue sm"
|  393|
|  394|
|  395| definition
|  396|   post_sm2tsm_VDM :: "State
|  397|  \<rightharpoonup> State
|  398|  \<Rightarrow> State
|  399|  \<rightharpoonup> State
|  400|  \<Rightarrow> \<bool>"
|  401|     where
|  402|     "post_sm2tsm_VDM sm RESULT  \<equiv> (isa_invTrue sm \<and> isa_invTrue
RESULT)"
|  403|
|  404|
|  405| definition
|  406|   pre_sm2tsm_VDM2 :: "State
|  407|  \<rightharpoonup> State
|  408|  \<Rightarrow> \<bool>"
|  409|     where
|  410|     "pre_sm2tsm_VDM2 sm  \<equiv> isa_invTrue sm"
|  411|
|  412|
|  413| definition
|  414|   post_sm2tsm_VDM2 :: "State
|  415|  \<rightharpoonup> State
|  416|  \<Rightarrow> State
|  417|  \<rightharpoonup> State
|  418|  \<Rightarrow> \<bool>"
|  419|     where
|  420|     "post_sm2tsm_VDM2 sm RESULT  \<equiv> (isa_invTrue sm \<and> isa_invTrue
RESULT)"
|  421|
|  422|
|  423| definition
|  424|   pre_sm2tsm :: "State
|  425|  \<rightharpoonup> State
|  426|  \<Rightarrow> \<bool>"
|  427|     where
|  428|     "pre_sm2tsm sm  \<equiv> isa_invTrue sm"
|  429|
|  430|
|  431| definition
|  432|   post_sm2tsm :: "State
|  433|  \<rightharpoonup> State
|  434|  \<Rightarrow> State
|  435|  \<rightharpoonup> State
|  436|  \<Rightarrow> \<bool>"
|  437|     where
|  438|     "post_sm2tsm sm RESULT  \<equiv> (isa_invTrue sm \<and> isa_invTrue
```

```
RESULT)"
|  439|
|  440|
|  441| definition
|  442|   pre_fsm2tfsm :: "Event
|  443|  \<rightharpoonup> State
|  444|  \<rightharpoonup> State
|  445|  \<Rightarrow> \<bool>"
|  446|     where
|  447|     "pre_fsm2tfsm fsm  \<equiv> isa_invTrue fsm"
|  448|
|  449|
|  450| definition
|  451|   post_fsm2tfsm :: "Event
|  452|  \<rightharpoonup> State
|  453|  \<rightharpoonup> State
|  454|  \<Rightarrow> Event
|  455|  \<rightharpoonup> State
|  456|  \<rightharpoonup> State
|  457|  \<Rightarrow> \<bool>"
|  458|     where
|  459|     "post_fsm2tfsm fsm RESULT  \<equiv> (isa_invTrue fsm \<and> isa_invTrue
RESULT)"
|  460|
|  461|
|  462| definition
|  463|   pre_fsm2tfsm_VDM2 :: "Event
|  464|  \<rightharpoonup> State
|  465|  \<rightharpoonup> State
|  466|  \<Rightarrow> \<bool>"
|  467|     where
|  468|     "pre_fsm2tfsm_VDM2 fsm  \<equiv> isa_invTrue fsm"
|  469|
|  470|
|  471| definition
|  472|   post_fsm2tfsm_VDM2 :: "Event
|  473|  \<rightharpoonup> State
|  474|  \<rightharpoonup> State
|  475|  \<Rightarrow> Event
|  476|  \<rightharpoonup> State
|  477|  \<rightharpoonup> State
|  478|  \<Rightarrow> \<bool>"
|  479|     where
|  480|     "post_fsm2tfsm_VDM2 fsm RESULT  \<equiv> (isa_invTrue fsm \<and>
isa_invTrue RESULT)"
|  481|
|  482|
|  483| definition
|  484|   pre_fsm2tfsm_VDM :: "Event
|  485|  \<rightharpoonup> State
|  486|  \<rightharpoonup> State
```

```
|  487|  \<Rightarrow> \<bool>"
|  488|     where
|  489|     "pre_fsm2tfsm_VDM fsm  \<equiv> isa_invTrue fsm"
|  490|
|  491|
|  492| definition
|  493|   post_fsm2tfsm_VDM :: "Event
|  494|  \<rightharpoonup> State
|  495|  \<rightharpoonup> State
|  496|  \<Rightarrow> Event
|  497|  \<rightharpoonup> State
|  498|  \<rightharpoonup> State
|  499|  \<Rightarrow> \<bool>"
|  500|     where
|  501|     "post_fsm2tfsm_VDM fsm RESULT  \<equiv> (isa_invTrue fsm \<and>
isa_invTrue RESULT)"
|  502|
|  503| end
```