Solving the TTC 2011 Compiler Optimization Task with **meta**-tools

Markus Lepper

\<semantics/\> GmbH, Berlin, Germany

post@markuslepper.eu

Baltasar Trancón y Widemann

Universität Bayreuth, Germany

Baltasar.Trancon@uni-bayreuth.de

The authors' **meta**-tools are a collection of tools for generic programming. This includes generating Java sources from mathematically well-founded specifications, as well as the creation of strictly typed document object models for XML encoded texts. In this context, almost every computer-internal structure is treated as a "model", and every computation is a kind of model transformation.

This concept differs significantly from "classical model transformation" executed by specialized tools and languages. Therefore it seemed promising to the organizers of the Transformation Tool Contest 2011 in Zürich, as well as to the authors, to apply **meta**-tools to one of the challenges, namely to the "compiler optimization task". This is a report on the resulting experiences.

1 Principles of **meta**-tools and Context of this Report

The authors' view on model transformation [5] is in some aspects a "dual" approach to the construction of dedicated model transformation tools: It implies that *every* computer-internal data structure can (and should!) be seen as a model, and every computation is a kind of model transformation. Consequently, models are realized as type definitions, invariants and processing code in the hosting computer language: In a first step, source code is generated automatically from a mathematical description. In a second step, the programmer is free to handle the model objects by combining the generated API and the corresponding runtime libraries with his/her familiar coding techniques. During the last decade the authors' coding activities in this area have been collected into **meta**-tools [4], a tool set for JAVA source code generation. In this context, even source code generation itself is realized as a transformation between models which adhere to different meta-models.

Obviously this view differs significantly from dedicated model transformation languages. Both approaches have their merits. It may be valuable to compare the experiences, it may be possible to learn from each other, and maybe both approaches turn out to be two sides of a bridge which slowly grow and will meet somewhere in the middle.

Regrettably, the authors did not get to know about the "Transformation Tool Contest 2011" [7] until it had started. Nevertheless, the organizers invited them to present their approach and the authors afterwards developed a solution to the "compiler optimization task" from TTC 2011 using **meta**-tools. The results are presented in this paper.

1.1 Solved Tasks

The source text and a runnable demonstration of this solution is found at [6]. It can read, visualize and optimize the files `min.gxl`, `const.gxl`, `zero.gxl`, and `testcase.gxl`, which are included as copies from the task's original test data. The other files employ "memory operations", which are not yet supported by the importing code.
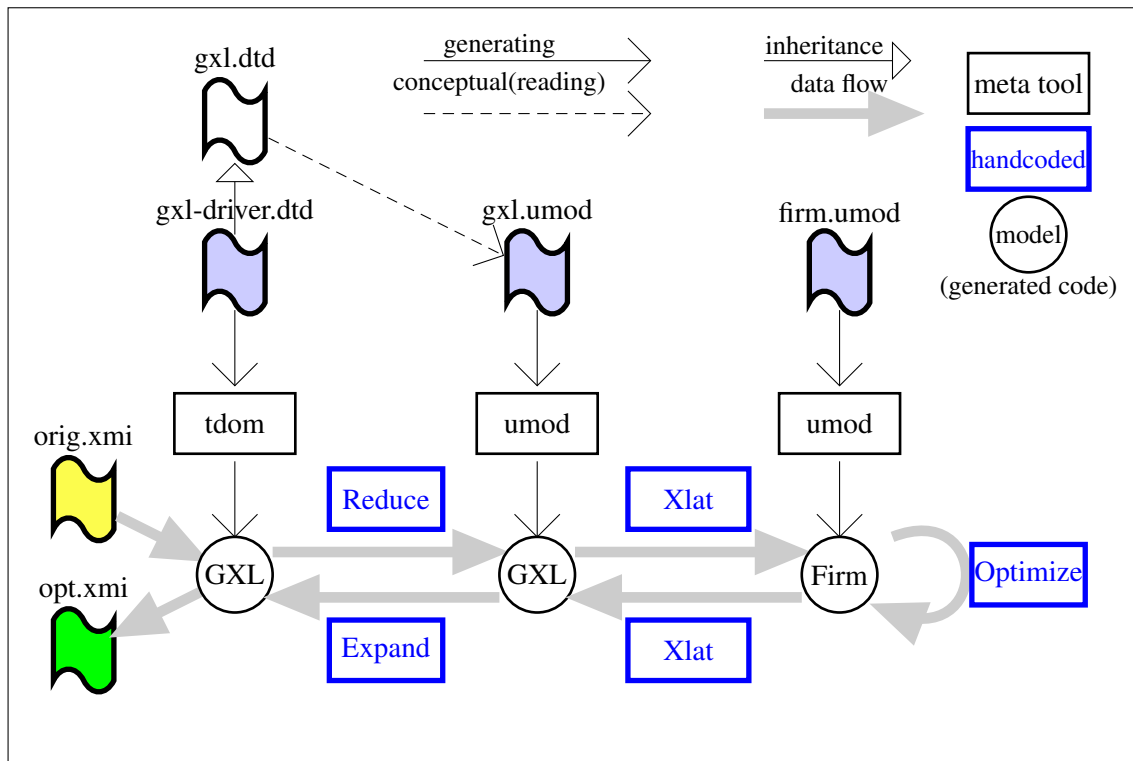
Figure 1: The processing pipeline for solving the "Compiler Optimization Task" with ^{meta}-tools

2 Stacking Models for Solving the Compiler Optimization Task

2.1 Import and Export Pipeline

The "Compiler Optimization Task" is presented in [3], containing an informal description and the necessary test data. The latter is given as an intermediate graph following the "Firm" syntax [9]. This in turn is encoded in the "GXL" format, a standardized XML format for exchanging graph-like data [8].

Given this setting, it soon turned out that a *stack of models* is an adequate approach. Not only due to the fact that two programmers were involved who created code for GXL and Firm independently, but also for clean, modular and correct implementation. Figure 1 shows the transformation pipeline.

Two kinds of model transformations are involved: First, identical information is transformed from one representation to another. Then the optimization task itself is performed as a transformation on the best-suited representation. Afterwards the first transformation chain is reversed.

Directly from the DTD at [8, /gxl-1.0.dtd], the `tdom` generator from ^{meta}-tools generated a typesafe document object model, in form of source text for JAVA classes. Only a small driver DTD (see appendix A) had to be prepended, for manually identifying some common attributes and abstractions of common content models. These will be translated to abstract super classes. Appendix B shows the code snippet for creating a model instance from a SAX event stream.

Then this model is transformed into a equivalent `umod` model, (see next section for explanation and appendix C for the source). The inheritance relation which spans the `umod` model resemble that from the "XML Schema" version of the GXL definition, see [8, xmlschema/xmlschema.html], for natural

reasons. The decoders and encoders between both models of GXL are written manually, and derived from the automatically generated base visitors. The source is in appendices D and E. The structure of this code clearly expresses the inheritance relation of the visited model.

2.2   The `umod` Model of Firm

Finally, the data to process is transformed from the GXL model into a `umod` model for representing Firm structures. `umod` is one of the central tools in ^meta^-tools for defining computer-internal data models, [4], [5]. It takes a mathematical description of a model, i.e. of classes, inheritance relation, types and field structures, and creates an API which allows to create and process JAVA objects which represent the model elements in a type-safe way. Esp., it supports the free composition of non-syntactic data types like sets, sequences and maps, and eliminates the ubiquitous threat of the `null` value. The generated code includes configurable, general purpose *visitors* and *rewriters*, serving as base classes for user code, as well as dedicated subclasses for visualization and serialization. By extending this generated code, user-defined code which realizes the intended semantics becomes very compact, cf. appendices I and J. The `umod` source is contained in appendix F. Its design closely follows [1, table 1] and [9, table 2.1].

A program graph in Firm is a kind of "inverted" program control flow graph: Every node points to its predecessors which have to be evaluated in advance. This structure corresponds to those of pure expressions and mathematical terms. Consequently, in the `umod` version each program graph is identified and represented *solely by its end node*, since from this every other node and block is reachable.

Nodes and edges are *typed*, from a small fixed set of *node types* and *edge types*. The latter are called "modes" for historical reasons. Every node class defines for its instances the allowed combinations of types for the incoming and the outgoing edges.

While the GXL model contains the Firm edges as reified, the `umod` version reduces them to uni-directional references. As a consequence, all backward information necessary during processing must (and always can!) be locally memorized. This elegant and efficient technique has already been proposed by [9, section 2.2.2.8]. It is only possible because in general no role information or sequential order is imposed on *incoming* edges.

The drawings in [9] introduce "sockets", which impose a partitioning of the incoming and on the outgoing side of each node. These sockets are required only on the conceptual level, not in the implementation, because this partitioning is uniquely determined by the *type* of the connected edges. This type, in turn, is uniquely determined by the "role" of the definition of the object's field which contains the reference which realizes the edge.

Whenever a definition line in [1, table 1] or [9, table 2.1] mentions two different letters on the right side of the arrow, then an instance of this node type can be the end point of edges of two different edge types. In the `umod` model, edge types and node roles are realized by JAVA "interfaces". On the conceptual level, e.g. each node of type "Start" can be the target of a "data flow edge" and of a "control flow edge". So the corresponding `umod` class implements the `Numeric` and the `ControlFlow` interface, and a reference to an instance object of this class can be held by a field of `Numeric` and of `ControlFlow` type.

The only node definitions which break this concept are those in which the same edge type appears more than once on the right side of the arrow, representing output of the same type, but with different roles. This can be seen as a *sequential order* on incoming edges. In this case, the proposal from [9, section 2.2.2.8] is followed: The node is the target of many edges, but not directly from a consuming node, but from a special *projection node*, inserted to select one component of the incoming tuple. In the `umod` model there exist two such node classes, `Proj_X` and `Proj_N`, for control flow and for data flow.

The property below is currently . . .				
. . . checked on the fly by the translation algorithm in Gxl2Firm =				T
. . . checked explicitly in Checker =			C	
. . . guaranteed implicitly, by the properties of Firm.umod =		i		
. . . currently still unchecked, but urgently required! =	?			
only one start node				T
start node only node in block				
start block no predecs				
only one end node		i		T
end node only node in block				
end block has no successors			C	
all blocks reach end node		i		
start node reaches all blocks	?			
inter-block graphs are acyclic	?			
phi node predecs ≡ block predecs			C	
inter-block edge requires corresponding control flow edge	?			
dto., w.r.t. phi nodes and predecessor numbering	?			
edge from node to block is on position "-1"				T
all constants contained in start block				
each block (w/o end block) has one control flow node			C	

Table 1: Consistency checks and their implementation.

The sequential order of the *outgoing* edges is realized most naturally by the distinction of the fields which contain the references, and by special container classes like lists and maps for containing more than one outgoing edge.

2.3 Consistency Checks on the Firm Model

Both the textual description of the task in [2, section 3 "Getting started. Verifier"] and the Firm documentation in [9, section 2.1.3 "Further restrictions to Firm Graphs"], suggest to check certain consistency properties. In our approach, some of these are already guaranteed "by construction", i.e. by the algorithm which translates from the GXL into the Firm umod model, see appendix G. Further properties are checked explicitly be a dedicated Checker class, see appendix I. Others are guaranteed implicitly, by the structure of the umod model. Table 1 lists these properties (just by a few keywords for the reader familiar with both documents) and indicates which part of our solution does cover them.

The Checker class is derived from the generated visitor class, but indirectly, because we have to deal with cycles explicitly. We assume that *the only cycles* in the graph are made by blocks and their final "Jump/Cond" nodes. The class VisitBlocksOnce cuts these cycles. It will be compiled as a static inner class in Firm.java, since it is contained as a JAVA escape directly in the umod source, see appendix F.

2.4 Optimizing Transformations on the Firm Model

The code of the class ConstantFolding, see appendix J, performs simple constant folding in arithmetic operations, and the resulting control flow simplification. Most of the transformation-performing classes

are derived from the generated *rewriter* code, a framework for copy-on-write updating of model data. Similar to a visitor, this generated class contains an generated `action(C)` method for every model element class. This method creates a clone of the visited object, and calls the method `rewriteFields(C clone)`. The *generated version* of `rewriteFields()` first calls `rewriteFields((D) clone)`, when D is the superclass of C. Then it initiates the rewriting process recursively on each field f defined on the level of C by calling `rewrite(clone.get_f())`. When this returns, (1) it compares the result of rewriting with the original, for detecting changes, (2) stores the result into the field of the clone, and (3) re-adjusts its own result value, which at the beginning pointed to the original, to refer to the clone, whenever such a change of a field value has been detected.

The generated methods are now partly overridden by user defined, specialized methods:

The method `rewriteFields(Binary)` will be called on each binary numeric node in the model. Like the generated version, it first calls the rewrite process bottom-up on all sub-objects referred to. Additionally it *substitutes* the result by a new constant, where appropriate.

User-defined `action(Proj_X)` looks whether the selecting value of the Cond is a constant after rewriting. In this case it substitutes a constant `Jmp` or an empty list, reflecting whether the index number of the `Proj_X` is equal to the selecting constant. References to control flow nodes can only occur in the `.predecs` map of a block. In the context of such an aggregate, a single reference can be re-written to a multiplicity of references, or to the empty list. This decrease in alternatives is in turn checked by `action(Phi)`, which may replace the visited object by its single surviving input.

3 Conclusion

We have presented a solution for the "Compiler Optimization Task" from the "Transformation Tool Contest 2011" by generic programming, i.e. by automated generation of model source code, as supported by the ^{meta}-tools toolkit. During the last decade, generated source code of this kind has been employed in different software projects, serving as the starting point for very different styles of of JAVA programming. This shows that the mental model of "coding as model transformation" is adequate for developing general purpose mid-scale software architectures.

The model definitions in appendices C and F show the compactness of model definitions in our framework, compared to the generated API doc at [6, ttc2011/apidoc].

The sources in appendices D, E, G, and H show how the structure of the model organizes the transformation code in a natural and readable way.

The code in appendices I and J shows how an intended transformation is broken down into simple steps, and then combined in a pipeline.

The main *advantages* of this kind of model code generation are, in our experience:

1. After the initial step, the generation of sources by ^{meta}-tools, no further dedicated tools are required, and no additional programming language front-end has to be learned.

2. Nearly everything (beside some primitive run-time functions) is visible to the programmer, nothing mystic happens "behind the scene".

3. Many properties of a correct model instance are checked automatically by the generated constructors and setter methods, or mapped to the type system of the hosting language, see table 1 above.

4. Visitor and rewriter based code is seamlessly integrated with all other handwritten code: Both kinds of code can control, call and parameterize each other, freely and recursively. For this, only the native constructs from the hosting language are required.

The special problems and *disadvantages*:

1. The first disadvantage is caused by the same fact as the preceding advantage: Due to the integration of generated and hand-written code, the consistency and integrity of the model's internal state variables can only be guaranteed in the limits imposed by the hosting language. E.g., in case of JAVA, putting both kinds of code into the same "package" would allow the programmer to clutter anything, with unforeseeable effects.

2. The design of the model definition is crucial: All benefits mentioned above are only fully enjoyable with an adequate design of the `.umod` source. E.g., in case of `Firm.umod`: whether the different numeric data types shall be mapped to subclasses of '`Numeric`", — whether all three interface types are really required, — whether one or more "`Proj_<>`" nodes are sensible, — all these questions require some experience, or even some experiments.

3. The visitor and rewriter API has to be understood and the protocol has to be respected by the programmer. Otherwise effects can occur which seem "mystical" indeed.

4. Due to the freedom of calling any code, inferring properties of the transformation system as a whole, like termination, completeness, confluence, may become infeasible, or at least more restricted than in case of a dedicated model transformation tool. But the authors' work on visitor optimization [5] showed that even in the open environment of an imperative hosting language some analysis is possible.

References

[1] Mathias Braun, Sebastian Buchwald & Andreas Zwinkau (2011): *Firm – A Graph-Based Intermediate Representation*. In: *Workshop on Intermediate Representations*, authors, Chamonix. Available at `http://pp.info.uni-karlsruhe.de/uploads/publikationen/braun11wir.pdf`.

[2] Sebastian Buchwald & Edgar Jakumeit: *TTC 2011 Compile Optimization Test Case / Data and Description*. Karlsruhe Institute of Technology. Available at `http://is.ieis.tue.nl/staff/pvgorp/events/TTC2011/cases/ttc2011_submission_5.zip`.

[3] Sebastian Buchwald & Edgar Jakumeit (2011): *Compiler Optimization: A Case for the Transformation Tool Contest*. In Pieter Van Gorp, Steffen Mazanek & Louis Rose, editors: *TTC 2011: Fifth Transformation Tool Contest, Zürich, Switzerland, June 29-30 2011*, EPTCS.

[4] Markus Lepper & Baltasar Trancón y Widemann (2011): *metatools Users' Documentation*. website. Available at `http://www.bandm.eu/metatools`.

[5] Markus Lepper & Baltasar Trancón y Widemann (2011): *Optimization of Visitor Performance by Reflection-Based Analysis*. In Jordi Cabot & Eelco Visser, editors: *Theory and Practice of Model Transformations, ICMT 2011*, *LNCS* 6707, Organization, Springer, Berlin, New York, Heidelberg, pp. 15–30, doi:10.1007/978-3-642-21732-6_2.

[6] Markus Lepper & Baltasar Trancón y Widemann (2011): *SHARE demo for the paper Solving the TTC 2011 Compiler Optimization Task with meta-tools*. Available at `http://is.ieis.tue.nl/staff/pvgorp/share/?page=ConfigureNewSession&vdi=Ubuntu_10.04_TTC11_metatools.vdi`.

[7] Steffen Mazanek, Louis Rose & Pieter Van Gorp: *TTC 2011 Home Page*. Available at `http://is.ieis.tue.nl/staff/pvgorp/events/TTC2011`.

[8] Andy Schuerr, Susan Elliott Sim, Ric Holt & Andreas Winter (2002): *Graph eXchange Language — Homepage*. website. Available at `http://www.gupro.de/GXL`.

[9] Martin Trapp, Götz Lindenmaier & Boris Boesler (1999): *Documentation of the Intermediate Representation FIRM*. Technical Report 1999-14, Universität Karlsruhe, Fakultät für Informatik. Available at `http://www.info.uni-karlsruhe.de/papers/firmdoc.ps.gz`.

Contents of Appendices

A   The GXL DTD Driver File

```
0......<?tdom default private?>
1......<?tdom public gxl?>
2......
3......<?tdom attribute
4......      xlink:type (simple) #FIXED "simple"
5......      xlink:href CDATA #REQUIRED
6......      id ID #IMPLIED
7......      isdirected (true | false) #IMPLIED
8...... ?>
9......
10......<!ENTITY % gxl.dtd SYSTEM "../gxl-1.0.dtd">
11......%gxl.dtd;
12......
13......<!ENTITY % anyval "(%val;)">
14......<?tdom abstract-entity anyval ?>
15......<?tdom abstract part (node | edge | rel) ?>
```

B   Code which initiates the creation of the tdom model from some SAX event stream

```
import org.xml.sax.InputSource ;
import eu.bandm.tools.xantlrtdom.TdomReader ;
import eu.bandm.ttc2011.case2.tdom.Document_gxl ;
import eu.bandm.ttc2011.case2.tdom.DTD ;
...
  try{
    final Document_gxl allGxl
      = TdomReader.parseXmlFile
```

```
      ( new InputSource(inputfilename),
        Document_gxl.class,
        DTD.dtd,
        /*debug=*/false);
    } catch (TdomException ex){ ...}
```

C   The GXL Model, as source to `umod`

```
0......// http://www.gupro.de/GXL/dtd/gxl-1.0.html
1......
2......MODEL GxlModel  =
3......// MODEL GXL = geht nicht unter cygwin/ntfs wg. Konflikt GXL.java Gxl.java
4......
5......ENUM Edgemode =  directed, undirected, defaultdirected, defaultundirected
6......ENUM Direction =  in, out, none
7......
8......VISITOR 0 SinglePass  ;
9......
10......EXT Location = eu.bandm.tools.message.Location
11......    <eu.bandm.tools.message.XMLDocumentIdentifier>
12......
13......TOPLEVEL CLASS
14......
15......GxlObject ABSTRACT
16......        location OPT Location
17......
18......| Gxl
19......        graphs SEQ Graph                !       V 0/0 ;
20......
21......| Type
22......        href string                     ! C 0/0 ;
23......
24......| Attributed ABSTRACT
25......        attrs SEQ Attr                  !       V 0/0 ;
26......| | Typed ABSTRACT
27......        type OPT Type                   !       V 0/1 ;
28......| | | Graph
29......        id string                       ! C 0/0 ;
30......        parts SEQ Part                  !       V 0/2 ;
31......        role OPT string
32......        edgeids bool = "false"
33......        hypergraph bool = "false"
34......        edgemode Edgemode = "Edgemode.directed"
35......| | | Part ABSTRACT
36......        id ABSTRACT GETTER OPT string
37......        graphs SEQ Graph                !       V 0/2 ;
38......| | | | Node
39......        id string                       ! C 0/0 ;
40......| | | | Edgy ABSTRACT
41......        id OPT string
42......        isdirected OPT bool
43......| | | | | Edge
44......        from Part                       ! C 0/0 ;
45......        to Part                         ! C 0/1 ;
46......        fromorder OPT int
47......        toorder OPT int
48......| | | | | Rel
49......        relends SEQ Relend              !       V 0/3 ;
50......| | Relend
51......        target Part                     ! C 0/0 ;
52......        role OPT string
53......        direction OPT Direction
```

```
54......        startorder OPT int
55......        endorder OPT int
56......| | Attr
57......        id OPT string
58......        name string                    ! C 0/0 ;
59......        kind OPT string
60......        val Val                        ! C 0/1 V 0/1 ;
61......
62......| Val ABSTRACT
63......| | Locator
64......        href string                    ! C 0/0 ;
65......| | Bool
66......        value bool                     ! C 0/0 ;
67......| | Int
68......        value int                      ! C 0/0 ;
69......| | Float
70......        value float                    ! C 0/0 ;
71......| | String
72......        value string                   ! C 0/0 ;
73......| | Enum
74......        value string                   ! C 0/0 ;
75......| | Aggregate ABSTRACT
76......        elems SEQ Val                  !       V 0/0 ;
77......| | | Seq
78......| | | Set
79......| | | Bag
80......| | | Tup
81......
82......END MODEL
```

D   Decoding from the `tdom` model into the `umod` model of GXL

```
0......package eu.bandm.ttc2011.case2.gxlcodec ;
1......
2......import eu.bandm.ttc2011.case2.tdom.* ;
3......
4......import eu.bandm.tools.tdom.runtime.* ;
5......import eu.bandm.tools.graph.GraphModels ;
6......import eu.bandm.tools.graph.CycleException ;
7......import eu.bandm.tools.message.MessageReceiver ;
8......import eu.bandm.tools.message.SimpleMessage ;
9......import eu.bandm.tools.message.Location ;
10......import eu.bandm.tools.message.XMLDocumentIdentifier ;
11......import eu.bandm.tools.message.MessageThrower ;
12......
13......import java.util.* ;
14......
15......
16......public class GxlDecoder {
17......
18......  private MessageReceiver<? super SimpleMessage<XMLDocumentIdentifier>> msg =
19......    new MessageThrower<SimpleMessage>() ;
20......
21......  public void setMessageReceiver(MessageReceiver<? super SimpleMessage<XMLDocumentIdentifier>> msg) {
22......    this.msg = msg ;
23......  }
24......
25......  protected void error(Location<XMLDocumentIdentifier> location,
26......                       java.lang.String text) {
27......    msg.receive(SimpleMessage.error(location, text)) ;
28......  }
```

```
29......
30......    protected void hint(Location<XMLDocumentIdentifier> location,
31......                        java.lang.String text) {
32......      msg.receive(SimpleMessage.hint(location, text)) ;
33......    }
34......
35......    @SuppressWarnings("unchecked")
36......    public Gxl decode(Document_gxl d) {
37......      final Gxl result = new Gxl() ;
38......      try {
39......        final GxlGraphElementModel gem = new GxlGraphElementModel(d) ;
40......        new Visitor() {
41......          @Override public void visit(Element_graph e) {
42......            result.get_graphs().add(new GraphDecoder(gem).decode(e)) ;
43......          }
44......        }.visit(d) ;
45......        result.set_location(d.getDocumentElement().getLocation()) ;
46......        return result ;
47......      }
48......      catch (HomonymousIdException e) {
49......        error(e.getElement2().getLocation(),
50......              "illegal homonymous element ID: " + e.getId()) ;
51......        if (e.getElement().getLocation() != null)
52......          hint(e.getElement().getLocation(), "previous use here") ;
53......      }
54......      catch (SynonymousIdException e) {
55......        error(e.getElement().getLocation(),
56......              "illegal synonymous element ID: " + e.getId2()) ;
57......        hint(e.getElement().getLocation(), "previous ID: " + e.getId()) ;
58......      }
59......      return null ;
60......    }
61......
62......
63......    private abstract class AttributedDecoder<R extends Attributed>
64......      extends Visitor {
65......
66......      protected R result ;
67......
68......      @Override public void visit(Element_attr e) {
69......        result.get_attrs().add(new AttrDecoder().decode(e)) ;
70......      }
71......
72......      protected final void visitNoMatch(Element_attr e) {
73......        super.visit(e) ;
74......      }
75......
76......    }
77......
78......    private abstract class TypedDecoder<R extends Typed> extends AttributedDecoder<R> {
79......
80......      @Override public void visit(Element_type e) {
81......        result.set_type(new Type(e.getAttr_xlink_href().getValue())) ;
82......      }
83......
84......    }
85......
86......    private class GraphDecoder extends TypedDecoder<Graph> {
87......
88......      final GxlGraphElementModel gem ;
89......      final Map<java.lang.String, Part> parts = new HashMap<java.lang.String, Part>() ;
90......
91......      GraphDecoder(GxlGraphElementModel gem) {
92......        this.gem = gem ;
93......      }
```

```
94......
95......      Part getPart(Element e, java.lang.String id) {
96......        if (parts.containsKey(id))
97......          return parts.get(id) ;
98......        else {
99......          error(e.getLocation(), "undefined IDREF: " + id) ;
100.....          return null ;
101.....        }
102.....      }
103.....
104.....      public Graph decode(Element_graph e) {
105.....        result = new Graph(e.getId()) ;
106.....        visit(e) ;
107.....        final Pass2 pass2 = new Pass2() ;
108.....        try {
109.....          for (Element_part p : GraphModels.postorder(gem.narrow(e)))
110.....            pass2.visit(p) ;
111.....        }
112.....        catch (CycleException ex) {
113.....          final Element_part p = (Element_part)ex.getNode() ;
114.....          error(p.getLocation(), "cyclic cross-reference") ;
115.....          // FIXME: find all members?
116.....          return null ;
117.....        }
118.....        result.set_location(e.getLocation()) ;
119.....        return result ;
120.....      }
121.....
122.....      @Override public void visit(Element_graph.Attr_role a) {
123.....        result.set_role(a.getValue()) ;
124.....      }
125.....
126.....      @Override public void visit(Element_graph.Attr_edgeids a) {
127.....        switch (a.getValue()) {
128.....        case Value_true:
129.....          result.set_edgeids(true) ; break ;
130.....        case Value_false:
131.....          result.set_edgeids(false) ; break ;
132.....        }
133.....      }
134.....
135.....      @Override public void visit(Element_graph.Attr_hypergraph a) {
136.....        switch (a.getValue()) {
137.....        case Value_true:
138.....          result.set_edgeids(true) ; break ;
139.....        case Value_false:
140.....          result.set_edgeids(false) ; break ;
141.....        }
142.....      }
143.....
144.....      @Override public void visit(Element_graph.Attr_edgemode a) {
145.....        switch (a.getValue()) {
146.....        case Value_directed:
147.....          result.set_edgemode(Edgemode.directed) ; break ;
148.....        case Value_undirected:
149.....          result.set_edgemode(Edgemode.undirected) ; break ;
150.....        case Value_defaultdirected:
151.....          result.set_edgemode(Edgemode.defaultdirected) ; break ;
152.....        case Value_defaultundirected:
153.....          result.set_edgemode(Edgemode.defaultundirected) ; break ;
154.....        }
155.....      }
156.....
157.....
158.....      @Override public void visit(Element_node e) {}
```

```
159......    @Override public void visit(Element_edge e) {}
160......    @Override public void visit(Element_rel e) {}
161......
162......    private class Pass2 extends Visitor {
163......
164......      @Override public void visit(Element_node e) {
165......        result.get_parts().add(register(new NodeDecoder().decode(e))) ;
166......      }
167......
168......      @Override public void visit(Element_edge e) {
169......        result.get_parts().add(register(new EdgeDecoder().decode(e))) ;
170......      }
171......
172......      @Override public void visit(Element_rel e) {
173......        result.get_parts().add(register(new RelDecoder().decode(e))) ;
174......      }
175......
176......      private Part register(Part x) {
177......        if (x.get_id() != null)
178......          parts.put(x.get_id(), x) ;
179......        return x ;
180......      }
181......
182......    }
183......
184......
185......    private abstract class PartDecoder<R extends Part> extends TypedDecoder<R> {
186......
187......      @Override public void visit(Element_graph e) {
188......        result.get_graphs().add(new GraphDecoder(gem).decode(e)) ;
189......      }
190.....
191......    }
192.....
193......    private abstract class EdgyDecoder<R extends Edgy> extends PartDecoder<R> {
194......
195......      Boolean isdirected ;
196......
197......      @Override public void visit(Attr_isdirected a) {
198......        if (a.getValue() != null)
199......          switch (a.getValue()) {
200......          case Value_true:
201......            isdirected = true ; break ;
202......          case Value_false:
203......            isdirected = false ; break ;
204......          }
205......      }
206......
207......      @Override public void visit(Attr_id a) {
208......        result.set_id(a.getValue()) ;
209......      }
210......
211......    }
212......
213......
214......    private class NodeDecoder extends PartDecoder<Node> {
215......
216......      public Node decode(Element_node e) {
217......        result = new Node(e.getId()) ;
218......        visit(e) ;
219......        result.set_location(e.getLocation()) ;
220......        return result ;
221......      }
222......
223......    }
```

```
224......
225......
226......        private class EdgeDecoder extends EdgyDecoder<Edge> {
227......
228......            private Location<XMLDocumentIdentifier> location ;
229......
230......            public Edge decode(Element_edge e) {
231......              final Part from = getPart(e, e.getAttr_from().getValue()) ;
232......              final Part to = getPart(e, e.getAttr_to().getValue()) ;
233......              result = new Edge(from, to) ;
234......              location = e.getLocation() ;
235......              visit(e) ;
236......              result.set_location(e.getLocation()) ;
237......              return result ;
238......            }
239......
240......            @Override public void visit(Element_edge.Attr_fromorder a) {
241......              try {
242......                if (a.getValue() != null)
243......                  result.set_fromorder(Integer.parseInt(a.getValue())) ;
244......              }
245......              catch (NumberFormatException ex) {
246......                error(location, "illegal integer: " + a.getValue()) ;
247......              }
248......            }
249......
250......            @Override public void visit(Element_edge.Attr_toorder a) {
251......              try {
252......                if (a.getValue() != null)
253......                  result.set_toorder(Integer.parseInt(a.getValue())) ;
254......              }
255......              catch (NumberFormatException ex) {
256......                error(location, "illegal integer: " + a.getValue()) ;
257......              }
258......            }
259......
260......        }
261......
262......
263......        private class RelDecoder extends EdgyDecoder<Rel> {
264......
265......            public Rel decode(Element_rel e) {
266......              result = new Rel() ;
267......              visit(e) ;
268......              result.set_location(e.getLocation()) ;
269......              return result ;
270......            }
271......
272......            @Override public void visit(Attr_isdirected a) {
273......              if (a.getValue() != null)
274......                switch (a.getValue()) {
275......                case Value_true:
276......                  result.set_isdirected(true) ; break ;
277......                case Value_false:
278......                  result.set_isdirected(false) ; break ;
279......                }
280......            }
281......
282......            @Override public void visit(Element_relend e) {
283......              result.get_relends().add(new RelendDecoder().decode(e)) ;
284......            }
285......
286......        }
287......
288......        private class RelendDecoder extends AttributedDecoder<Relend> {
```

```
289......
290......        private Location<XMLDocumentIdentifier> location ;
291......
292......        public Relend decode(Element_relend e) {
293......          final Part target = getPart(e, e.getAttr_target().getValue()) ;
294......          result = new Relend(target) ;
295......          location = e.getLocation() ;
296......          visit(e) ;
297......          result.set_location(e.getLocation()) ;
298......          return result ;
299......        }
300......
301......        @Override public void visit(Element_relend.Attr_role a) {
302......          result.set_role(a.getValue()) ;
303......        }
304......
305......        @Override public void visit(Element_relend.Attr_direction a) {
306......          if (a.getValue() != null)
307......            switch (a.getValue()) {
308......            case Value_in:
309......              result.set_direction(Direction.in) ; break ;
310......            case Value_out:
311......              result.set_direction(Direction.out) ; break ;
312......            case Value_none:
313......              result.set_direction(Direction.none) ; break ;
314......            }
315......        }
316......
317......        @Override public void visit(Element_relend.Attr_startorder a) {
318......          try {
319......            if (a.getValue() != null)
320......              result.set_startorder(Integer.parseInt(a.getValue())) ;
321......          }
322......          catch (NumberFormatException ex) {
323......            error(location, "illegal integer: " + a.getValue()) ;
324......          }
325......        }
326......
327......        @Override public void visit(Element_relend.Attr_endorder a) {
328......          try {
329......            if (a.getValue() != null)
330......              result.set_endorder(Integer.parseInt(a.getValue())) ;
331......          }
332......          catch (NumberFormatException ex) {
333......            error(location, "illegal integer: " + a.getValue()) ;
334......          }
335......        }
336......
337......      }
338......
339......    }
340......
341......  private class AttrDecoder extends AttributedDecoder<Attr> {
342......
343......    public Attr decode(Element_attr e) {
344......      result = new Attr(e.getAttr_name().getValue(),
345......                        new ValDecoder().decode(e.getElem_1_anyval())) ;
346......      visitNoMatch(e) ;
347......      result.set_location(e.getLocation()) ;
348......      return result ;
349......    }
350......
351......    @Override public void visit(Attr_id a) {
352......      result.set_id(a.getValue()) ;
353......    }
```

```
354......
355......    @Override public void visit(Element_attr.Attr_kind a) {
356......      result.set_kind(a.getValue()) ;
357......    }
358......
359......  }
360......
361......  private class ValDecoder extends Visitor {
362......
363......    private Val val ;
364......
365......    public Val decode(Element_anyval e) {
366......      visit((Element)e) ;
367......      val.set_location(e.getLocation()) ;
368......      return val ;
369......    }
370......
371......    @Override public void visit(Element_locator e) {
372......      val = new Locator(e.getAttr_xlink_href().getValue()) ;
373......    }
374......
375......    @Override public void visit(Element_bool e) {
376......      final java.lang.String text = e.getPCData() ;
377......      if (text.equals("true"))
378......        val = new Bool(true) ;
379......      else if (text.equals("false"))
380......        val = new Bool(false) ;
381......      else
382......        error(e.getLocation(), "illegal boolean: " + text) ;
383......    }
384......
385......    @Override public void visit(Element_int e) {
386......      try {
387......        val = new Int(Integer.parseInt(e.getPCData())) ;
388......      }
389......      catch (NumberFormatException ex) {
390......        error(e.getLocation(), "illegal integer: " + e.getPCData()) ;
391......      }
392......    }
393......
394......    @Override public void visit(Element_float e) {
395......      try {
396......        val = new Float(Double.parseDouble(e.getPCData())) ;
397......      }
398......      catch (NumberFormatException ex) {
399......        error(e.getLocation(), "illegal float: " + e.getPCData()) ;
400......      }
401......    }
402......
403......    @Override public void visit(Element_string e) {
404......      val = new String(e.getPCData()) ;
405......    }
406......
407......    @Override public void visit(Element_enum e) {
408......      val = new Enum(e.getPCData()) ;
409......    }
410......
411......    @Override public void visit(Element_set e) {
412......      val = subvals(new Set(), e.getElems_1_anyval()) ;
413......    }
414......
415......    @Override public void visit(Element_seq e) {
416......      val = subvals(new Seq(), e.getElems_1_anyval()) ;
417......    }
418......
```

```
419......    @Override public void visit(Element_bag e) {
420......      val = subvals(new Bag(), e.getElems_1_anyval()) ;
421......    }
422......
423......    @Override public void visit(Element_tup e) {
424......      val = subvals(new Tup(), e.getElems_1_anyval()) ;
425......    }
426......
427......    private Aggregate subvals(Aggregate parent, Element_anyval... es) {
428......      for (Element_anyval e : es) parent.get_elems().add(decode(e)) ;
429......      return parent ;
430......    }
431......
432......  }
433......
434......}
```

E   Encoding from the `umod` model into the `tdom` model of GXL

```
0......package eu.bandm.ttc2011.case2.gxlcodec ;
1......
2......import eu.bandm.ttc2011.case2.tdom.* ;
3......import eu.bandm.tools.tdom.runtime.* ;
4......
5......import java.util.* ;
6......
7......
8......public class GxlEncoder {
9......
10......  private static final Element_graph[] GRAPHS = new Element_graph[0] ;
11......  private static final Element_attr[] ATTRS = new Element_attr[0] ;
12......  private static final Element_part[] PARTS = new Element_part[0] ;
13......  private static final Element_relend[] RELENDS = new Element_relend[0] ;
14......
15......  public Document_gxl encode(Gxl x) {
16......    final Collection<Element_graph> graphs = new ArrayList<Element_graph>() ;
17......    new SinglePass() {
18......      @Override protected void action(Graph x) {
19......        graphs.add(new GraphEncoder().encode(x)) ;
20......      }
21......    }.match(x) ;
22......    final Element_gxl e = new Element_gxl(graphs.toArray(GRAPHS)) ;
23......    return new Document_gxl(e) ;
24......  }
25......
26......
27......  private static abstract class AttributedEncoder extends SinglePass {
28......
29......    final Collection<Element_attr> attrs = new ArrayList<Element_attr>() ;
30......
31......    @Override protected void action(Attr x) {
32......      attrs.add(new AttrEncoder().encode(x)) ;
33......    }
34......
35......    protected final void actionNoMatch(Attr x) {
36......      super.action(x) ;
37......    }
38......
39......  }
40......
41......
```

```
42......   private static abstract class TypedEncoder extends AttributedEncoder {
43......
44......     Element_type type ;
45......
46......     @Override protected void action(final Type x) {
47......       type = new Element_type() {
48......           @Override protected void initAttrs() {
49......             getAttr_xlink_href().setValue(x.get_href()) ;
50......           }
51......         } ;
52......     }
53......
54......   }
55......
56......
57......   private static class GraphEncoder extends TypedEncoder {
58......
59......     final Collection<Element_part> parts = new ArrayList<Element_part>() ;
60......
61......     Element_graph encode(final Graph x) {
62......       match(x) ;
63......       return new Element_graph(type,
64......                               attrs.toArray(ATTRS),
65......                               parts.toArray(PARTS)) {
66......         @Override protected void initAttrs() {
67......           getAttr_id().setValue(x.get_id()) ;
68......         }
69......       } ;
70......     }
71......
72......     @Override protected void action(Node x) {
73......       parts.add(new NodeEncoder().encode(x)) ;
74......     }
75......
76......     @Override protected void action(Edge x) {
77......       parts.add(new EdgeEncoder().encode(x)) ;
78......     }
79......
80......     @Override protected void action(Rel x) {
81......       parts.add(new RelEncoder().encode(x)) ;
82......     }
83......
84......   }
85......
86......   private static abstract class PartEncoder extends TypedEncoder {
87......
88......     final Collection<Element_graph> graphs = new ArrayList<Element_graph>() ;
89......
90......     @Override protected void action(Graph x) {
91......       graphs.add(new GraphEncoder().encode(x)) ;
92......     }
93......
94......   }
95......
96......   private static class NodeEncoder extends PartEncoder {
97......
98......     Element_node encode(final Node x) {
99......       match(x) ;
100......      return new Element_node(type,
101......                              attrs.toArray(ATTRS),
102......                              graphs.toArray(GRAPHS)) {
103......        @Override protected void initAttrs() {
104......          getAttr_id().setValue(x.get_id()) ;
105......        }
106......      } ;
```

```
107......    }
108......  }
109......
110......  private static abstract class EdgyEncoder extends PartEncoder {
111......
112......    java.lang.String id ;
113......    Attr_isdirected.Value isdirected ;
114......
115......    @Override protected void action(Edgy x) {
116......      super.action(x) ;
117......      id = x.get_id() ;
118......      if (x.get_isdirected() != null)
119......        isdirected = x.get_isdirected()
120......          ? Attr_isdirected.Value.Value_true
121......          : Attr_isdirected.Value.Value_false ;
122......    }
123......
124......  }
125......
126......  private static class EdgeEncoder extends EdgyEncoder {
127......
128......    Element_edge encode(final Edge x) {
129......      match(x) ;
130......      return new Element_edge(type,
131......                              attrs.toArray(ATTRS),
132......                              graphs.toArray(GRAPHS)) {
133......        @Override protected void initAttrs() {
134......          getAttr_id().setValue(id) ;
135......          getAttr_from().setValue(x.get_from().get_id()) ;
136......          getAttr_to().setValue(x.get_to().get_id()) ;
137......          if (x.get_fromorder() != null)
138......            getAttr_fromorder().setValue(x.get_fromorder().toString()) ;
139......          if (x.get_toorder() != null)
140......            getAttr_toorder().setValue(x.get_toorder().toString()) ;
141......          getAttr_isdirected().setValue(isdirected) ;
142......        }
143......      } ;
144......    }
145......
146......  }
147......
148......  private static class RelEncoder extends EdgyEncoder {
149......
150......    private Collection<Element_relend> relends = new ArrayList<Element_relend>() ;
151......
152......    Element_rel encode(final Rel x) {
153......      match(x) ;
154......      return new Element_rel(type,
155......                             attrs.toArray(ATTRS),
156......                             graphs.toArray(GRAPHS),
157......                             relends.toArray(RELENDS)) {
158......        @Override protected void initAttrs() {
159......          getAttr_id().setValue(id) ;
160......          getAttr_isdirected().setValue(isdirected) ;
161......        }
162......      } ;
163......    }
164......
165......    @Override protected void action(Relend x) {
166......      relends.add(new RelendEncoder().encode(x)) ;
167......    }
168......
169......  }
170......
171......  private static class RelendEncoder extends AttributedEncoder {
```

```
172......
173......     Element_relend encode(final Relend x) {
174......       match(x) ;
175......       return new Element_relend(attrs.toArray(ATTRS)) {
176......         @Override protected void initAttrs() {
177......           getAttr_target().setValue(x.get_target().get_id()) ;
178......           getAttr_role().setValue(x.get_role()) ;
179......           if (x.get_direction() != null)
180......             switch (x.get_direction()) {
181......             case in:
182......               getAttr_direction().setValue(Element_relend.Attr_direction.Value.Value_in) ; break ;
183......             case out:
184......               getAttr_direction().setValue(Element_relend.Attr_direction.Value.Value_out) ; break ;
185......             case none:
186......               getAttr_direction().setValue(Element_relend.Attr_direction.Value.Value_none) ; break ;
187......             }
188......           if (x.get_startorder() != null)
189......             getAttr_startorder().setValue(x.get_startorder().toString()) ;
190......           if (x.get_endorder() != null)
191......             getAttr_endorder().setValue(x.get_endorder().toString()) ;
192......         }
193......       } ;
194......     }
195......
196...... }
197......
198......   private static class AttrEncoder extends AttributedEncoder {
199......
200......     private Element_anyval val ;
201......
202......     Element_attr encode(final Attr x) {
203......       actionNoMatch(x) ;
204......       return new Element_attr(attrs.toArray(ATTRS), val) {
205......         @Override protected void initAttrs() {
206......           getAttr_id().setValue(x.get_id()) ;
207......           getAttr_name().setValue(x.get_name()) ;
208......           getAttr_kind().setValue(x.get_kind()) ;
209......         }
210......       } ;
211......     }
212......
213......     @Override protected void action(Val x) {
214......       val = new ValEncoder().encode(x) ;
215......     }
216......
217...... }
218......
219......   private static class ValEncoder extends SinglePass {
220......
221......     private Element_anyval val ;
222......
223......     Element_anyval encode(Val x) {
224......       match(x) ;
225......       return val ;
226......     }
227......
228......     @Override protected void action(final Locator x) {
229......       val = new Element_locator() {
230......         @Override protected void initAttrs() {
231......           getAttr_xlink_href().setValue(x.get_href()) ;
232......         }
233......       } ;
234......     }
235......
236......     @Override protected void action(Bool x) {
```

```
237......        val = new Element_bool(java.lang.String.valueOf(x.get_value())) ;
238......      }
239......
240......      @Override protected void action(Int x) {
241......        val = new Element_int(java.lang.String.valueOf(x.get_value())) ;
242......      }
243......
244......      @Override protected void action(Float x) {
245......        val = new Element_float(java.lang.String.valueOf(x.get_value())) ;
246......      }
247......
248......      @Override protected void action(String x) {
249......        val = new Element_string(x.get_value()) ;
250......      }
251......
252......      @Override protected void action(Enum x) {
253......        val = new Element_enum(x.get_value()) ;
254......      }
255......
256......      @Override protected void action(Seq x) {
257......        val = new Element_seq(subvals(x.get_elems())) ;
258......      }
259......
260......      @Override protected void action(Set x) {
261......        val = new Element_set(subvals(x.get_elems())) ;
262......      }
263......
264......      @Override protected void action(Bag x) {
265......        val = new Element_bag(subvals(x.get_elems())) ;
266......      }
267......
268......      @Override protected void action(Tup x) {
269......        val = new Element_tup(subvals(x.get_elems())) ;
270......      }
271......
272......      private Element_anyval[] subvals(Collection<Val> xs) {
273......        final Collection<Element_anyval> elems = new ArrayList<Element_anyval>(xs.size()) ;
274......        for (Val x : xs)
275......          elems.add(new ValEncoder().encode(x)) ;
276......        return elems.toArray(new Element_anyval[xs.size()]) ;
277......      }
278......
279......  }
280......
281......}
```

F   The Firm Mode, as source to `umod`

```
0......MODEL Firm =
1......
2....../** Firm Model version 0.1
3......
4......    This model is a more recent version (20110813). It is
5......    based on the documentation contained in
6......    http://www.info.uni-karlsruhe.de/papers/firmdoc.ps.gz
7......    <p>
8......    (The more recent table in
9......    http://pp.info.uni-karlsruhe.de/uploads/publikationen/braun11wir.pdf
10......    differs slightly, and is taken instead for some definitios not yet used in
11......    the testcase, e.g. "alloc" etc.)
12......    <p>
```

```
13......    The fundamental principles of the this Firm realization by an umod model:
14......    <p>
15......    Node classes are translated to umod model element classes / java classes.
16......    Edges are translated to references.
17......    <p>
18......    The above-mentioned documentation introduces "sockets" for separating
19......    and qualifying edges.
20......    <p>
21......    With the outgoing edges this is realized explicitly by
22......    mapping "sockets" to fields. Further sequential order on the edges is realized
23......    by using appropriate container types (SEQ, MAP, etc)
24......    <p>
25......    With the incoming edges the "sockets" are always given implicitly,
26......    by the "starting socket" of the edge.
27......    <p>
28......    Whenever further order of incoming edges is required, e.g. when a
29......    "call" or "start" node produces more than one numeric data, or
30......    with the "phi" and "cond" node, the data is
31......    treated as forming a  "Tuple" and explicit "Proj" nodes are inserted
32......    between the producing socket and the reading conumer.
33......  */
34......
35......EXT Location = eu.bandm.tools.message.Location
36......     <eu.bandm.tools.message.XMLDocumentIdentifier>
37......
38......
39......VISITOR O Matcher MULTIPHASE ;
40......VISITOR O SimpleVisitor ;
41......VISITOR O Rewriter IS REWRITER ;
42......VISITOR O CoRewriter IS COREWRITER ;
43......
44......ENUM NumericType =   p, Iu, Is, Su, Ss, Bu, Bs, E, F, D, C, b,
45......                    NotYetComputed
46......
47......INTERFACE
48......   MemoryState
49......   Numeric
50......   ControlFlow
51......
52......TOPLEVEL CLASS
53......
54......FirmNode
55......        location OPT Location          ! C 0/0 ;
56......        gxlId OPT string
57......
58......| Unknown IMPLEMENTS MemoryState, Numeric, ControlFlow
59......
60......| Bad IMPLEMENTS MemoryState, Numeric, ControlFlow
61......
62......<< JAVA public static final Bad BAD = new Bad(); $$
63......
64......| Block
65......        predecs  int->ControlFlow       !                          V 0/0 R ; // 1/0 R ;
66......
67......| BlockNode
68......        block Block                     ! C 0/1                    V 0/0 ;
69......
70......| | End
71......| | Cond
72......        selector Numeric                ! C 0/2                    V 0/0 ;
73......        // int or boolean !!
74......
75......| | ControlFlowNode IMPLEMENTS ControlFlow
76......| | MemoryNode IMPLEMENTS MemoryState
77......| | NumericNode IMPLEMENTS Numeric
```

```
78......
79......
80......EXTEND CLASS
81......
82......ControlFlowNode
83......| Start IMPLEMENTS MemoryState, Numeric
84......| Return
85......        memstate MemoryState              ! C 0/2                    V 0/0;
86......        results SEQ Numeric               !                         V 0/1 ;
87......| Jmp
88......| Proj_X
89......        input Cond                        ! C 0/2                    V 0/0 ;
90......        selection int                     ! C 0/3                    ;
91......
92......MemoryNode
93......| NoMem
94......| Sync
95......        predecs SET MemoryState           !                         V 0/0 ;
96......| MemModification
97......    preState MemoryState                  ! C 0/2                    V 0/1 ;
98......    position Numeric                      ! C 0/3                    V 0/2 ;
99......| | Store
100......    valueNumeric Numeric                 ! C 0/4                    V 0/3 ;
101......| | Free
102......
103......
104......NumericNode
105......        type NumericType                  ! C 0/2 ;
106......
107......| Phi
108......        alternatives int->Numeric         !                         V 0/0 R ;
109......// must be in sync with "this.block.predecs" !
110......
111......
112......| Proj_N // de-compose a tupel
113......        predec Numeric /*Tuple_N*/        ! C 0/3                    V 0/0 R ;
114......        pos int                           ! C 0/4 ;
115......
116......| Tuple_N IMPLEMENTS Numeric
117......        combines SEQ Numeric              !                         V 0/0 ;
118......| | Call IMPLEMENTS MemoryState
119......        preState MemoryState              ! C 0/3                    V 0/1 ;
120......
121......| Nullary
122......| | NumericConst
123......    unparsedValue string                  ! C 0/3;
124......    intValue        OPT int
125......    floatValue      OPT float
126......    stringValue     OPT string
127......    charValue       OPT char
128......    booleanValue    OPT bool
129......
130......| | SymConst
131......    unparsedValue string                  ! C 0/3;
132......
133......| Unary
134......    on  Numeric                           ! C 0/3           V 0/0 ;
135......| | Conv
136......| | Minus
137......| | Not
138......| | Rotl
139......| | Shl
140......| | Shr
141......| | Shrs
142......
```

```
143......| Binary
144......   left Numeric                          ! C 0/3        V 0/0 ;
145......   right Numeric                         ! C 0/4        V 0/1 ;
146......| | Add
147......| | And
148......| | Div
149......| | Eor
150......| | Mod
151......| | Mul
152......| | Or
153......| | Sub
154......
155......| | Cmp
156......
157......| Ternary
158......   first  Numeric                        ! C 0/3        V 0/1 ; // boolean!
159......   second Numeric                        ! C 0/4        V 0/2 ;
160......   third  Numeric                        ! C 0/5        V 0/3 ;
161......| | Mux
162......
163......| MemOperations IMPLEMENTS MemoryState
164......   preState MemoryState                  ! C 0/3        V 0/1 ;
165......| | Alloc
166......   size Numeric                          ! C 0/4        V 0/2 ;
167......| | Load
168......   position Numeric                       ! C 0/4        V 0/2 ;
169......| | Sel
170......   position Numeric                       ! C 0/4        V 0/2 ;
171......
172......| BadNumeric
173......| UnknownNumeric
174......
175......
176......<<JAVA
177......   public static class VisitBlocksOnce extends SimpleVisitor {
178......     final protected java.util.Set<Block> visitedBlocks
179......     = new java.util.HashSet<Block>();
180......     public java.util.Set<Block> visited(){
181......       return java.util.Collections.unmodifiableSet(visitedBlocks);
182......     }
183......     @Override public void action (final Block block){
184......       if (visitedBlocks.contains(block))
185......         return ;
186......       visitedBlocks.add(block);
187......       super.action(block);
188......     }
189......   }
190......$$
191......
192....../* == NOT YET IMPLEMENTED:
193......ASM             B x variable  -->   variable   Inline assembler
194......=================*/
195......
196......END MODEL
```

G   Decoding a GXL model into a Firm model

```
0......package eu.bandm.ttc2011.case2.transformations ;
1......
2......import java.util.Map;
3......import java.util.HashMap;
```

```
4......import java.util.Set;
5......import java.util.HashSet;
6......
7......import eu.bandm.tools.message.MessageReceiver ;
8......import eu.bandm.tools.message.MessageTee ;
9......import eu.bandm.tools.message.MessageCounter   ;
10......import eu.bandm.tools.message.SimpleMessage ;
11......import eu.bandm.tools.message.XMLDocumentIdentifier ;
12......import eu.bandm.tools.message.Location;
13......
14......import eu.bandm.tools.ops.Multimap ;
15......import eu.bandm.tools.ops.HashMultimap ;
16......
17......import eu.bandm.tools.ops.Pattern ;
18......import static eu.bandm.tools.ops.Pattern.eq ;
19......import static eu.bandm.tools.ops.Pattern.any;
20......import static eu.bandm.tools.ops.Pattern.Variable ;
21......
22......
23......import eu.bandm.ttc2011.case2.gxlcodec.* ;
24......import eu.bandm.ttc2011.case2.firm_01.* ;
25......
26......import java.lang.String ;
27......
28......
29....../** Translates between two umod-generated java models,
30......namely from the {@link eu.bandm.ttc2011.case2.gxlcodec.GxlModel gxl model}
31......into the {@link eu.bandm.ttc2011.case2.firm_01.Firm firm model}.
32......
33......
34......Please note the nomenclature:
35......<pre>
36......    MODEL Glx
37......       Node
38......       Edge
39......
40......    MODEL Firm
41......      FirmNode
42......      | Block
43......      | BlockNode // node "contained in a block"
44......</pre>
45......*/
46......
47......public class Gxl2Firm {
48......
49......   final Set <Node> all_gxl_nodes = new HashSet<Node>();
50......
51......   final Map <String, String> metanodeid2nodeclassname
52......     = new HashMap<String, String>();
53......   final Map <String, String> nodeclassname2metanodeid
54......     = new HashMap<String, String>();
55......
56......   final Map <Node, Map<Integer, Edge>> node2outgoing
57......     = new HashMap<Node, Map<Integer, Edge>>();
58......
59......   final Map <Node, String> node2nodetype = new HashMap<Node, String>();
60......
61......   final Map <Node, FirmNode> correspondent = new HashMap <Node, FirmNode>();
62......
63......   final Set <Node> undertranslation = new HashSet<Node>();
64......
65......
66......   protected void ERROR(Exception ex){
67......     msgT.receive(SimpleMessage.<XMLDocumentIdentifier>error(ex, "exception"));
68......   }
```

```
69......  protected void ERROR(Location<XMLDocumentIdentifier> loc, String ex){
70......    msgT.receive(SimpleMessage.<XMLDocumentIdentifier>error(loc, ex));
71......  }
72......  protected void ERROR(String ex){
73......    msgT.receive(SimpleMessage.<XMLDocumentIdentifier>error(ex));
74......  }
75......  protected void WARNING(Location<XMLDocumentIdentifier> loc, String ex){
76......    msgT.receive(SimpleMessage.<XMLDocumentIdentifier>warning(loc, ex));
77......  }
78......
79......
80......  protected MessageTee<SimpleMessage<XMLDocumentIdentifier>> msgT
81......    = new  MessageTee<SimpleMessage<XMLDocumentIdentifier>>();
82......  protected MessageCounter<SimpleMessage<XMLDocumentIdentifier>> msgC
83......    = new MessageCounter<SimpleMessage<XMLDocumentIdentifier>>();
84......  {msgT.add(msgC);}
85......
86......  protected boolean severeErrors(){
87......    return msgC.getCriticalCount()>0;
88......  }
89......
90......
91......  protected Gxl gxl ;
92......  protected Graph metamodel, objectmodel ;
93......  Node startnode = null, glx_endnode = null ;
94......
95......  End firm_endnode = null ;
96......
97......  public static final String STRING_HREF_METAMODEL
98......    = "http://www.gupro.de/GXL/gxl-1.0.gxl#gxl-1.0" ;
99......  public static final String STRING_HREF_OBJECTMODEL            = "#Firm" ;
100......  public static final String STRING_HREF_OBJECTMODEL_2
101......          = "#InstructionSelection"   ;
102......  public static final String STRING_NODENAME_START             = "Start" ;
103......  public static final String STRING_NODENAME_END               = "End" ;
104......  public static final String ATTRIBUTE_NAME_NAME        = "name" ;
105......  public static final String ATTRIBUTE_NAME_POSITION    = "position" ;
106......  public static final String ATTRIBUTE_NAME_VALUE       = "value" ;
107......
108......  public static final String STRING_BLOCKTYPE                  = "Block" ;
109......  public static final String STRING_STARTBLOCKTYPE             = "StartBlock" ;
110......  public static final String STRING_ENDBLOCKTYPE               = "EndBlock" ;
111......  public static final int ORDER_BLOCK_CONTAINMENT = -1 ;
112......
113......  public static final String STRING_EDGETYPE_TRUE              = "#True" ;
114......  public static final String STRING_EDGETYPE_FALSE             = "#False" ;
115......
116......  // numeric parts of right side of a start node is a tuple as follows:
117......  public static final int POSITION_STACKFRAME_ARGUMENT = 0 ;
118......  public static final int POSITION_HEAPPOINTER_ARGUMENT = 1 ;
119......  public static final int POSITION_ARGUMENTS_ARGUMENT = 2 ;
120......
121......
122......  protected Pattern<Attributed> hasAttr(String name, Pattern<? super Val> val) {
123......    return Attributed.get_attrs(Attr.get_name(eq(name))
124......                            .and(Attr.get_val(val)).somewhere()) ;
125......  }
126......
127......
128......  /** ASSUME the GXL model contains TWO graphs:
129......      <ol>
130......      <li> A first Graph representing the meta-model, i.e. describing "Firm" itself.
131......      <br>
132......      Its "type" field conatains the
133......      href-value "{@code http://www.gupro.de/GXL/gxl-1.0.gxl#gxl-1.0}" <br>
```

```
134......        Its "id" contains "SCE_Firm". <br>
135......        Most of its nodes represent node classes of the object graph.<br>
136......        With these nodes, the child element {@code <Gxl:Attr name="name">}
137......        gives a hint which node class is meant.
138......        <li>
139......        As second graph in the gxl structure, there is the object model itself.
140......        </ol>
141......        Its "type" field contains a reference (local to this file!)
142......        to the node with id = "Firm" in the meta-model <br>
143......        <br>
144......        All firm nodes in the are represented as gxl nodes in the second
145......        graph. Theur "type" attribute
146......        is a href (local to this file!) to a node in the meta model.
147......        <br>
148......        All edges go between nodes (gxl would permit others!).
149......        <br>(NB The order
150......        of the outgoing edges is NOT given by the gxl-defined field "endorder",
151......        but by a dedicated gxl-attribute with name {@link ATTRIBUTE_NAME_VALUE}!)
152......        <br>
153......        This method is the central entry method. <br>
154......        <br>
155......        It decodes the "model" graph contained in the Gxl into a Firm object.
156......        <ol>
157......        <li> It calls "new Collector().match()" to separate both models
158......        and to collect all edges between nodes.
159......        <li> It calls "find_start_and_end()" to identify the start and the end node
160......        in the gxl model.
161......        <li> It calls "convert()" on the end node to produce the "firm" end node.
162......        </ol>
163......        Afterwards the user can inquire for
164......        <ol>
165......        <li> the "endnode" of the Firm model,
166......        <li> the meta-model as a Gxl graph,
167......        <li> and the map from node class names to identifiers of the meta model.
168......        </ol>
169......        The last two are required for subsequent re-encoding of a tranformed Firm model.
170......    */
171......    public void decode_meta_and_object
172......      (final MessageReceiver<SimpleMessage<XMLDocumentIdentifier>>msg,
173......       final Gxl gxl){
174......      this.msgT.add(msg);
175......      this.gxl = gxl ;
176......      metamodel=objectmodel=null ;
177......      new Collector().match(gxl);
178......      if (all_gxl_nodes.isEmpty())
179......        ERROR("no model nodes found in file "
180......              +"(perhaps due to a wrong <graph><type xlink:href='#???'> entry ?)");
181......      find_start_and_end();
182......      if (glx_endnode==null)
183......        ERROR("no endnode in graph");
184......      if (startnode==null)
185......        ERROR("no startnode in graph");
186......      if (severeErrors())
187......        return;
188......      undertranslation.add(glx_endnode);
189......      firm_endnode = (End)convert(glx_endnode);
190......      return  ;
191......    }
192......
193......    public End get_resulted_endnode(){
194......      return firm_endnode;
195......    }
196......    public Graph get_resulted_metamodel(){
197......      return metamodel;
198......    }
```

```
199......   public Map<String,String> get_nodeClassName2metaNodeId(){
200......     return java.util.Collections.unmodifiableMap(nodeclassname2metanodeid);
201......   }
202......
203......   protected Map<Integer, Edge> get_node2outgoing (final Node from){
204......     Map<Integer,Edge> map = node2outgoing.get(from);
205......     if (map==null){
206......       map = new HashMap<Integer,Edge>();
207......       node2outgoing.put(from, map);
208......     }
209......     return map ;
210......   }
211......
212......   // ===========================================================================
213......
214......   /** Analyses the gxl files as presented by the authors of the task.
215......       <br/>
216......       1) Memorizes the meta-model graph as a whole, for subsequent re-encoding.
217......       <br/>
218......       2) makes a map from meta-model node ids to firm class names, also for re-encoding
219......       <br/>
220......       3) identifies the model graph, and then <br/>
221......       <br/>
222......          - collects all of its nodes into {@link #all_gxl_nodes} <br/>and memorizes
223......          - collects all edges into {@link #ndoe2outgoing} : Node -> int -> Node <br/>
224......   */
225......
226......   public class Collector  extends SinglePass {
227......
228......     boolean metamode = true ;
229......     String metanodeid ;
230......
231......     @Override public void action (final Graph g){
232......       final Type t = g.get_type();
233......       if (t==null){
234......         ERROR("Type == null not forseen.");
235......         return;
236......       }
237......       final String href = t.get_href();
238......       if (href.equals(STRING_HREF_METAMODEL))
239......         if (metamodel!=null){
240......           ERROR("More than one metamodel?");
241......           return;
242......         }
243......         else{
244......           metamodel=g ;
245......           metamode=true ;
246......         }
247......       else if (href.equals(STRING_HREF_OBJECTMODEL)
248......                 || href.equals(STRING_HREF_OBJECTMODEL_2))
249......         if (objectmodel!=null){
250......           ERROR("More than one objectmodel?");
251......           return;
252......         }
253......         else{
254......           objectmodel=g ;
255......           metamode=false;
256......         }
257......       super.action(g);
258......     }
259......
260......     @Override public void action(final Node n){
261......       if (metamode){
262......         final Variable<String> var = Pattern.<String>variable();
263......         if(hasAttr(ATTRIBUTE_NAME_NAME,
```

```
264......                        eu.bandm.ttc2011.case2.gxlcodec.String.cast
265......                        (eu.bandm.ttc2011.case2.gxlcodec.String.get_value(var))
266......                        .or(any)).match(n)){
267......              final String s = var.getValue();
268......              if (s==null)
269......                ERROR(n.get_location(),
270......                      "name attribute of a meta-node must carry a string value");
271......              else{
272......                final String metanodeid = n.get_id();
273......                metanodeid2nodeclassname.put(metanodeid, s);
274......                nodeclassname2metanodeid.put(s, metanodeid);
275......              }
276......            }
277......            else
278......              ;
279......            //        WARNING(n.get_location(),
280......            //                "meta-node should have an attribute named \"name\"");
281......        }
282......        else
283......          all_gxl_nodes.add(n);
284......    }
285......
286......
287......
288......    @Override public void action(final Edge e){
289......      if (metamode)
290......        return ;
291......      /** NO, the examples included in the task use <attr name="position"><int>
292......      Integer fromOrder = e.get_fromorder();
293......       */
294......      Integer fromOrder = findFromOrder(e);
295......
296......      if (fromOrder==null){
297......        ERROR(e.get_location(), "from order requrired for all edges");
298......        return ;
299......      }
300......      if (!(e.get_from() instanceof Node)){
301......        ERROR(e.get_location(), "start of each edge muat be of node type");
302......        return ;
303......      }
304......      final Node from = (Node)e.get_from();
305......      final Map<Integer,Edge> map = get_node2outgoing(from);
306......      final Edge oldEdge = map.get(fromOrder);
307......      if (oldEdge!=null){
308......        ERROR (e.get_location(),
309......              "duplicate use of order number "+fromOrder+" for this node and "
310......              +" previously for "+oldEdge.get_location());
311......        return ;
312......      }
313......      map.put(fromOrder, e);
314......    }
315......
316......
317......    protected Integer findFromOrder(Edge e){
318......      Variable<Integer> i = Pattern.<Integer>variable();
319......      /*Edge.get_attrs(Attr.get_id(eq(ATTRIBUTE_NAME_POSITION))
320......                          .and(Attr.get_val(Int.cast(Int.get_value(i)))
321......                          ).somewhere()*/
322......      hasAttr(ATTRIBUTE_NAME_POSITION,
323......            Int.cast(Int.get_value(i))
324......            ).match(e);
325......      return i.getValue();
326......    }
327......
328......  }//class Collector
```

```
329......
330......
331......  // ============================================================================
332......
333......  protected void find_start_and_end (/*GLO IN all_gxl_nodes,
334......                                      GLO OUT startnode, endnode, node2nodeclass */){
335......    for (final Node node : all_gxl_nodes){
336......      final Type t = node.get_type();
337......      if (t==null){
338......        ERROR(node.get_location(), "empty node type attribute?");
339......        return ;
340......      }
341......      final String s = t.get_href();
342......      final String s1 = s.substring(1);
343......      final String nodeclass = metanodeid2nodeclassname.get(s1);
344......      if (nodeclass==null){
345......        ERROR(node.get_location(), "unknown node type \""+s+"\" ?");
346......        return ;
347......      }
348......      node2nodetype.put(node, nodeclass);
349......      if (nodeclass.equals(STRING_NODENAME_END))
350......        if (glx_endnode != null)
351......          ERROR(node.get_location(), "A second end node found");
352......        else
353......          glx_endnode = node;
354......      else if (nodeclass.equals(STRING_NODENAME_START))
355......        if (startnode != null)
356......          ERROR(node.get_location(), "A second start node found");
357......        else
358......          startnode = node;
359......    }
360......  }
361......
362......  // ----------------------------------------------------------------------------
363......
364......  protected Map<Start, Proj_N> allArguments = new HashMap<Start, Proj_N>();
365......
366......  protected Proj_N getAllArgs (final Start start){
367......    if (allArguments.containsKey(start))
368......      return (allArguments.get(start));
369......    final Proj_N proj = new Proj_N(start.get_location(), start.get_block(),
370......                                  NumericType.NotYetComputed,
371......                                  start, POSITION_ARGUMENTS_ARGUMENT);
372......    allArguments.put(start,proj);
373......    return proj;
374......  }
375......
376......
377......  protected String noEdgeFound_errormsg (final Node node,
378......                                          final int position,
379......                                          final Class cls){
380......    return "Expecting an edge starting at position "+position
381......      +" of node "+node.get_id()+" of type "+node2nodetype.get(node)
382......      +" pointing to an encoding of a "+cls.getSimpleName();
383......  }
384......
385......
386......
387......  /** Follow the Gxl.Edge object of the Gxl.Node at the given position
388......      and deliver the un-decoded Gxl.Node.
389......      Generate error message if no such Node can be found and strict=true.
390......      Only called from {@link #convert_target(Node,int,Class,boolean)}.
391......  */
392......  protected Node find_targetencoding (final Node node,
393......                                        final int position,
```

```
394......                                        final Class cls, // DIAGNOSIS only!
395......                                        final boolean strict){
396......     final Edge edge  = get_node2outgoing(node).get(position);
397......     if (edge==null){
398......       if (strict)
399......         ERROR(node.get_location(), "No edge at all at this position. "
400......               +noEdgeFound_errormsg(node, position, cls));
401......       return null ;
402......     }
403......     final Part target = edge.get_to();
404......     if (!(target instanceof Node)){
405......       ERROR(node.get_location(), "Edge points to something not a graph node. "
406......               +noEdgeFound_errormsg(node, position, cls));
407......       return null ;
408......     }
409......     return (Node) target ;
410......   }
411......
412......
413......   /** Follow the Gxl.Edge object of the Gxl.Node at the given position,
414......       create the corresponding Firm object by calling {@link #convert(Node)},
415......       and ensure that this result is of type T.
416......   */
417......   protected <T> T convert_target
418......   //protected <T extends FirmNode> T convert_target
419......                       (final Node node,
420......                        final int position,
421......                        final Class<T> cls,
422......                        final boolean strict){
423......     final Node targetnode = find_targetencoding(node, position, cls, strict);
424......     if (targetnode==null) // Error messages are already generated.
425......       return null;
426......     final FirmNode oldtranslation = correspondent.get(targetnode);
427......     if (oldtranslation!=null){
428......       if (!(cls.isInstance(oldtranslation))){
429......         ERROR(targetnode.get_location(),
430......               "translation result is of type "
431......               +oldtranslation.getClass().getSimpleName()
432......               +"but now "+cls.getSimpleName()+" is expected? "
433......               +noEdgeFound_errormsg(node,position,cls));
434......         return null ;
435......       }
436......       else
437......         return cls.cast(oldtranslation);
438......     }
439......     if (undertranslation.contains(targetnode)){
440......       /**/System.err.println("BACK PATCH not yet implemented for LOOP: "
441......                         +noEdgeFound_errormsg(node, position, cls));
442......       return null ;
443......     }
444......     undertranslation.add(targetnode);
445......     final FirmNode newTranslation = convert(targetnode);
446......     if (newTranslation!=null)
447......       if (!(cls.isInstance(newTranslation))){
448......         ERROR(targetnode.get_location(),
449......               "Translation result is of class "
450......               +newTranslation.getClass().getSimpleName()
451......               +", but should be of "+cls.getSimpleName()+" ."
452......               +noEdgeFound_errormsg(node, position, cls));
453......         return null ;
454......       }
455......     correspondent.put(targetnode, newTranslation);
456......     return cls.cast(newTranslation);
457......   }
458......
```

```
459......  // ------------------------------------------------------------------
460......
461......  /** Comprehend all three block types ("StartBlock", "EndBlock", etc.)
462......      into one single model class.
463......  */
464......  protected boolean typeIsBlocktype (final String type){
465......    return type.equals(STRING_BLOCKTYPE)||type.equals(STRING_STARTBLOCKTYPE)
466......      ||type.equals(STRING_ENDBLOCKTYPE) ;
467......  }
468......
469......
470......  Location<XMLDocumentIdentifier> sourceLocation ;
471......
472......  /** ASSUME Only two callers are (1) frequently convert_target(),
473......      and (2) only once top-level, the external call for the end-node.
474......      Only called if node is not yet translated.
475......   */
476......  protected FirmNode convert (final Node node /* GLO ... */){
477......
478......    final String type = node2nodetype.get(node);
479......    if (typeIsBlocktype(type))
480......      return convert_block(node);
481......
482......    final Block block
483......      = convert_target (node, ORDER_BLOCK_CONTAINMENT, Block.class, true);
484......    if (block==null) // Error messages are already emitted!
485......      return null ;
486......
487......    sourceLocation = node.get_location();
488......
489......    if (type.equals("Return")){
490......      final MemoryState ms = convert_target(node, 0, MemoryState.class, true/*???*/);
491......      final Return ret = new Return(sourceLocation, block, ms);
492......      for (Integer i : node2outgoing.get(node).keySet()){
493......        if (i<=0) continue;
494......        ret.get_results().add(convert_target(node, i, Numeric.class, true));
495......      }
496......      return ret ;
497......    }
498......    if (type.equals("Start")){
499......      return new Start(sourceLocation, block);
500......    }
501......    if (type.equals("Jmp")){
502......      return new Jmp(sourceLocation, block);
503......    }
504......    if (type.equals("Phi")){
505......      final Phi phi = new Phi(sourceLocation, block,NumericType.NotYetComputed);
506......      for (Integer i : get_node2outgoing(node).keySet()){
507......        if (i==-1) continue;
508......        phi.get_alternatives().put(i,convert_target(node, i, Numeric.class, true));
509......      }
510......      return phi ;
511......    }
512......    if (type.equals("Cond")){
513......      final Numeric selector = convert_target(node, 0, Numeric.class, true);
514......      return new Cond(sourceLocation, block, selector);
515......    }
516......    if (type.equals("End"))
517......      return new End(sourceLocation, block);
518......
519......    /*
520......    if (type.equals("Store")){
521......      return new Start(sourceLocation, block);
522......    }
523......    */
```

```
524......      if (type.equals("Argument")){
525......        // "#Argument" is not documented, but used in testdata/testcase.gxl
526......        // we realize it as a projection out of a projection.
527......        final Start start = convert_target(node, 0, Start.class, true);
528......        // the Gxl Attribute named "position" gives the index of the argument
529......        // into the stack image with "Start"
530......
531......        Variable<Integer> i = Pattern.<Integer>variable();
532......        if(!hasAttr(ATTRIBUTE_NAME_POSITION,
533......                      Int.cast(Int.get_value(i))
534......                      ).match(node)){
535......          ERROR(sourceLocation, "position attribute missing for Argument");
536......          return Firm.BAD ;
537......        }
538......        final Start start0 = convert_target(node, 0, Start.class, true);
539......        return new Proj_N(sourceLocation, block, NumericType.NotYetComputed,
540......                          getAllArgs(start0), i.getValue());
541......      }
542......      if (is_numeric_nullary(type))
543......        return convert_numeric_nullary(block, type, node);
544......      if (is_numeric_unary(type))
545......        return convert_numeric_unary(block, type, node);
546......      if (is_numeric_binary(type))
547......        return convert_numeric_binary(block, type, node);
548......      if (is_numeric_ternary(type))
549......        return convert_numeric_ternary(block, type, node);
550......      throw new Error("No rule for gxl node  "+node.get_id()+" of type "+type);
551......      //     return null ;
552......    }
553......
554......    protected boolean is_numeric_nullary(String t){
555......      return t.equals("Const") ;
556......    }
557......
558......    // ----------------------------------------------------------------------
559......
560......    protected class DecodeConstValue extends SinglePass{
561......      public NumericType type ;
562......      public int intValue ;
563......      public double floatValue ;
564......
565......      @Override protected void action(Attr a){
566......        if (!(a.get_name().equals(ATTRIBUTE_NAME_VALUE)))
567......          return;
568......        match(a.get_val());
569......      }
570......      @Override protected void action(Locator i){
571......        ERROR(i.get_location(), "constants of lcator type should not appear");
572......      }
573......      @Override protected void action(Int i){
574......        intValue = i.get_value();
575......        type = NumericType.Is ; // PROVIS !
576......      }
577......      @Override protected void action(Bool i){
578......        ERROR(i.get_location(), "constants of boolean type should not appear");
579......      }
580......      @Override protected void action(eu.bandm.ttc2011.case2.gxlcodec.Float i){
581......        floatValue = i.get_value();
582......        type = NumericType.F ; // PROVIS !
583......      }
584......      @Override protected void action(eu.bandm.ttc2011.case2.gxlcodec.String i){
585......        ERROR(i.get_location(), "constants of string type should not appear");
586......      }
587......      @Override protected void action(eu.bandm.ttc2011.case2.gxlcodec.Enum i){
588......        ERROR(i.get_location(), "constants of enum type should not appear");
```

```
589......    }
590......    @Override protected void action(Aggregate i){
591......      ERROR(i.get_location(), "constants of aggregate type should not appear");
592......    }
593......  }// class DecodeConstValue
594......
595......  protected NumericNode convert_numeric_nullary
596......    (final Block b, final String type, final Node node){
597......    final DecodeConstValue d = new DecodeConstValue();
598......    d.match(node);
599......    switch(d.type){
600......      case Is:
601......        NumericConst intConst = new NumericConst (sourceLocation, b, d.type,
602......                                                ""+d.intValue);
603......        intConst.set_intValue(d.intValue);
604......        return intConst ;
605......      case F:
606......        NumericConst fConst = new NumericConst (sourceLocation, b, d.type,
607......                                              ""+d.floatValue);
608......        fConst.set_floatValue(d.floatValue);
609......        return fConst ;
610......    }
611......    return null ; // ERROR already generated.
612......  }
613......
614......  protected boolean is_numeric_unary(String t){
615......    return t.equals("Conv")|| t.equals("Minus")|| t.equals("Not")
616......      || t.equals("Rotl")|| t.equals("Shl")
617......      || t.equals("Shr")|| t.equals("Shrs");
618......  }
619......
620......  protected NumericNode convert_numeric_unary
621......    (final Block b, final String type, final Node node){
622......    final Numeric left  = convert_target(node,0, Numeric.class, true);
623......    if (type.equals("Conv"))
624......      return new Conv(sourceLocation, b, NumericType.NotYetComputed, left);
625......    else if (type.equals("Minus"))
626......      return new Minus(sourceLocation, b, NumericType.NotYetComputed, left);
627......    else if (type.equals("Not"))
628......      return new Not(sourceLocation, b, NumericType.NotYetComputed, left);
629......    else if (type.equals("Rotl"))
630......      return new Rotl(sourceLocation, b, NumericType.NotYetComputed, left);
631......    else if (type.equals("Shl"))
632......      return new Shl(sourceLocation, b, NumericType.NotYetComputed, left);
633......    else if (type.equals("Shr"))
634......      return new Shr(sourceLocation, b, NumericType.NotYetComputed, left);
635......    else // if (type.equals("Shrs"))
636......      return new Shrs(sourceLocation, b, NumericType.NotYetComputed, left);
637......  }
638......
639......  protected boolean is_numeric_binary(String t){
640......    return t.equals("Add")|| t.equals("And")|| t.equals("Cmp")|| t.equals("Div")
641......      || t.equals("Eor")|| t.equals("Mod")
642......      || t.equals("Mul")|| t.equals("Or")
643......      || t.equals("Sub")  ;
644......  }
645......
646......  protected NumericNode convert_numeric_binary
647......    (final Block b, final String type,
648......     final Node node){
649......    final Numeric left = convert_target(node, 0, Numeric.class, true);
650......    final Numeric right = convert_target(node, 1, Numeric.class, true);
651......    if (type.equals("Add"))
652......      return new Add(sourceLocation, b, NumericType.NotYetComputed, left, right);
653......    else if (type.equals("And"))
```

```
654......      return new And(sourceLocation, b, NumericType.NotYetComputed, left, right);
655......    else if (type.equals("Cmp"))
656......      return new Cmp(sourceLocation, b, NumericType.NotYetComputed, left, right);
657......    else if (type.equals("Div"))
658......      return new Div(sourceLocation, b, NumericType.NotYetComputed, left, right);
659......    else if (type.equals("Eor"))
660......      return new Eor(sourceLocation, b, NumericType.NotYetComputed, left, right);
661......    else if (type.equals("Mod"))
662......      return new Mod(sourceLocation, b, NumericType.NotYetComputed, left, right);
663......    else if (type.equals("Mul"))
664......      return new Mul(sourceLocation, b, NumericType.NotYetComputed, left, right);
665......    else if (type.equals("Or"))
666......      return new Or(sourceLocation, b, NumericType.NotYetComputed, left, right);
667......    else // if (type.equals("Sub"))
668......      return new Sub(sourceLocation, b, NumericType.NotYetComputed, left, right);
669......  }
670......
671......  protected boolean is_numeric_ternary(String t){
672......    return t.equals("Mux") ;
673......  }
674......
675......  protected NumericNode convert_numeric_ternary
676......    (final Block b, final String type,
677......     final Node node){
678......    final Numeric selector = convert_target(node, 0, Numeric.class, true);
679......    final Numeric left = convert_target(node, 1, Numeric.class, true);
680......    final Numeric right = convert_target(node, 2, Numeric.class, true);
681......    // if (type.equals("Mux"))
682......    return new Mux(sourceLocation, b, NumericType.NotYetComputed,
683......                  selector, left, right);
684......  }
685......
686......  protected Block convert_block(final Node node){
687......    final Block block = new Block(node.get_location());
688......    if (node.get_id()!=null)
689......      block.set_gxlId(node.get_id());
690......
691......    final Map<Integer, Edge> outgoing = get_node2outgoing(node);
692......    for (Integer i : outgoing.keySet()){
693......      final Type edgetype = outgoing.get(i).get_type();
694......      boolean isfalse = false, istrue = false ;
695......      ControlFlow cf = null ;
696......      if (edgetype!=null)
697......        // ATTENTION here also "numeric conditions / switch(){}"
698......        //   has to be decoded into a Proj_X node
699......        if (edgetype.get_href().equals(STRING_EDGETYPE_FALSE))
700......          isfalse=true ;
701......        else if (edgetype.get_href().equals(STRING_EDGETYPE_TRUE))
702......          istrue=true ;
703......      if (isfalse||istrue){
704......        final Cond cond = convert_target(node, i, Cond.class, true);
705......        cf = new Proj_X (cond.get_location(),
706......                        cond.get_block(),
707......                        cond,
708......                        istrue ? 1 : 0);
709......      }
710......      else
711......        cf = convert_target(node, i, ControlFlow.class, true);
712......      if (cf!=null)
713......        block.get_predecs().put (i, cf);
714......      // else error case, has already been signaled above.
715......    }
716......    return block ;
717......  }
718......
```

```
719......}
720......// eof
```

H   Encoding the Firm model into a GXL model

```
0......package eu.bandm.ttc2011.case2.transformations ;
1......
2......import java.util.Map ;
3......import java.util.HashMap ;
4......import java.util.Set ;
5......import java.util.HashSet ;
6......
7......import eu.bandm.ttc2011.case2.gxlcodec.* ;
8......import eu.bandm.ttc2011.case2.firm_01.* ;
9......
10......import java.lang.String ;
11......
12......import static eu.bandm.ttc2011.case2.transformations
13......   .Gxl2Firm.ATTRIBUTE_NAME_POSITION ;
14......import static eu.bandm.ttc2011.case2.transformations
15......   .Gxl2Firm.ORDER_BLOCK_CONTAINMENT ;
16......import static eu.bandm.ttc2011.case2.transformations
17......   .Gxl2Firm.ATTRIBUTE_NAME_VALUE ;
18......
19......
20......public class Firm2Gxl extends SimpleVisitor  {
21......
22......   protected Gxl result ;
23......   protected Graph metamodel ;
24......   protected End endnode ;
25......   protected Graph model ;
26......   protected Map<String, String> nodeType2id ;
27......   protected Map<FirmNode, Node> node2node = new HashMap<FirmNode, Node>();
28......
29......
30......   public Gxl translate (Graph metamodel, Map<String,String> nodeType2id,
31......                         String graphId, End endnode){
32......     this.metamodel=metamodel;
33......     this.nodeType2id=nodeType2id;
34......     this.endnode = endnode ;
35......     result = new Gxl();
36......     result.get_graphs().add(metamodel);
37......     model = new Graph(graphId);
38......     result.get_graphs().add(model);
39......     new Writeout_nodesAndBlocks().match (endnode);
40......     new Writeout_edges().match (endnode);
41......     return result ;
42......   }
43......
44......   protected class Writeout_nodesAndBlocks extends SimpleVisitor{
45......
46......     protected void write_node(final FirmNode n, String type){
47......       if (node2node.containsKey(n))
48......         return ;
49......       final String newId = "node"+String.valueOf(10000+node2node.size());
50......       final Node node = new Node(newId);
51......       node.set_type(new Type("#"+nodeType2id.get(type)));
52......       node2node.put(n, node);
53......       model.get_parts().add(node);
54......     }
55......
```

```
56......
57......      @Override protected void action (final Block n){
58......        if (node2node.containsKey(n))
59......          return ;
60......        write_node(n, "Block");
61......        super.action(n);
62......      }
63......
64......      @Override protected void action (final Start n){
65......        if (node2node.containsKey(n))
66......          return ;
67......        write_node(n, "Start");
68......        super.action(n);
69......      }
70......      @Override protected void action (final Return n){
71......        if (node2node.containsKey(n))
72......          return ;
73......        write_node(n, "Return");
74......        super.action(n);
75......      }
76......      @Override protected void action (final End n){
77......        if (node2node.containsKey(n))
78......          return ;
79......        write_node(n, "End");
80......        super.action(n);
81......      }
82......      @Override protected void action (final Jmp n){
83......        if (node2node.containsKey(n))
84......          return ;
85......        write_node(n, "Jmp");
86......        super.action(n);
87......      }
88......      @Override protected void action (final Cond n){
89......        if (node2node.containsKey(n))
90......          return ;
91......        write_node(n, "Cond");
92......        super.action(n);
93......      }
94......      /* do NOT generate a node in gxl model!
95......      @Override protected void action (final Proj_X n){
96......        super.action(n);
97......      }
98......      */
99......      @Override protected void action (final Sync n){
100......       if (node2node.containsKey(n))
101......         return ;
102......       write_node(n, "Sync");
103......       super.action(n);
104......     }
105......     @Override protected void action (final Phi n){
106......       if (node2node.containsKey(n))
107......         return ;
108......       write_node(n, "Phi");
109......       super.action(n);
110......     }
111......     @Override protected void action (final NumericConst n){
112......       if (node2node.containsKey(n))
113......         return ;
114......       write_node(n, "Const");
115......       eu.bandm.ttc2011.case2.gxlcodec.String s
116......         = new eu.bandm.ttc2011.case2.gxlcodec.String(n.get_unparsedValue());
117......       Attr attr = new Attr(ATTRIBUTE_NAME_VALUE, s);
118......       node2node.get(n).get_attrs().add(attr);
119......       super.action(n);
120......     }
```

```
121......     @Override protected void action (final Conv n){
122......        if (node2node.containsKey(n))
123......           return ;
124......        write_node(n, "Conv");
125......        super.action(n);
126......     }
127......     @Override protected void action (final Minus n){
128......        if (node2node.containsKey(n))
129......           return ;
130......        write_node(n, "Minus");
131......        super.action(n);
132......     }
133......     @Override protected void action (final Not n){
134......        if (node2node.containsKey(n))
135......           return ;
136......        write_node(n, "Not");
137......        super.action(n);
138......     }
139......     @Override protected void action (final Rotl n){
140......        if (node2node.containsKey(n))
141......           return ;
142......        write_node(n, "Rotl");
143......        super.action(n);
144......     }
145......     @Override protected void action (final Shl n){
146......        if (node2node.containsKey(n))
147......           return ;
148......        write_node(n, "Shl");
149......        super.action(n);
150......     }
151......     @Override protected void action (final Shr n){
152......        if (node2node.containsKey(n))
153......           return ;
154......        write_node(n, "Shr");
155......        super.action(n);
156......     }
157......     @Override protected void action (final Shrs n){
158......        if (node2node.containsKey(n))
159......           return ;
160......        write_node(n, "Shrs");
161......        super.action(n);
162......     }
163......     @Override protected void action (final Add n){
164......        if (node2node.containsKey(n))
165......           return ;
166......        write_node(n, "Add");
167......        super.action(n);
168......     }
169......     @Override protected void action (final And n){
170......        if (node2node.containsKey(n))
171......           return ;
172......        write_node(n, "And");
173......        super.action(n);
174......     }
175......     @Override protected void action (final Div n){
176......        if (node2node.containsKey(n))
177......           return ;
178......        write_node(n, "Div");
179......        super.action(n);
180......     }
181......     @Override protected void action (final Eor n){
182......        if (node2node.containsKey(n))
183......           return ;
184......        write_node(n, "Eor");
185......        super.action(n);
```

```
186......    }
187......    @Override protected void action (final Mul n){
188......      if (node2node.containsKey(n))
189......        return ;
190......      write_node(n, "Mul");
191......      super.action(n);
192......    }
193......    @Override protected void action (final Or n){
194......      if (node2node.containsKey(n))
195......        return ;
196......      write_node(n, "Or");
197......      super.action(n);
198......    }
199......    @Override protected void action (final Sub n){
200......      if (node2node.containsKey(n))
201......        return ;
202......      write_node(n, "Sub");
203......      super.action(n);
204......    }
205......    @Override protected void action (final Cmp n){
206......      if (node2node.containsKey(n))
207......        return ;
208......      write_node(n, "Cmp");
209......      super.action(n);
210......    }
211......    @Override protected void action (final Mux n){
212......      if (node2node.containsKey(n))
213......        return ;
214......      write_node(n, "Mux");
215......      super.action(n);
216......    }
217......  }// class Writeout_nodesAndBlocks
218......
219......  // ============================================================================
220......
221......  protected class Writeout_edges extends SimpleVisitor{
222......
223......    final java.util.Set<FirmNode> visited = new HashSet<FirmNode>();
224......
225......    protected boolean done (final FirmNode node){
226......      if (visited.contains(node))
227......        return true ;
228......      visited.add(node);
229......      return false ;
230......    }
231......    protected Edge write_edge(final FirmNode from, final FirmNode to, final int pos){
232......      final Edge edge = new Edge(node2node.get(from), node2node.get(to));
233......      final Int i = new Int(pos);
234......      final Attr attr = new Attr(ATTRIBUTE_NAME_POSITION,i);
235......      edge.get_attrs().add(attr);
236......      model.get_parts().add(edge);
237......      return edge ;
238......    }
239......
240......    protected void write_containing(final BlockNode from){
241......      write_edge(from, from.get_block(), ORDER_BLOCK_CONTAINMENT);
242......    }
243......
244......    @Override protected void action (final Block n){
245......      if (done(n)) return ;
246......      for (Integer i : n.get_predecs().keySet()){
247......        final ControlFlow target = n.get_predecs().get(i);
248......        if (target instanceof Proj_X){
249......          final Proj_X proj = (Proj_X)target;
250......          final Edge e = write_edge(n, proj.get_input(), i);
```

```
251......            e.set_type(new Type(proj.get_selection()==0
252......                              ? Gxl2Firm.STRING_EDGETYPE_FALSE
253......                              : Gxl2Firm.STRING_EDGETYPE_TRUE));
254......          }
255......        else
256......          write_edge(n, (FirmNode)target, i);
257......      }
258......      super.action(n);
259......    }

261......    @Override protected void action (final Start n){
262......      if (done(n)) return ;
263......      write_containing(n);
264......    }
265......    @Override protected void action (final Return n){
266......      if (done(n)) return ;
267......      write_containing(n);
268......      write_edge(n, (BlockNode)n.get_memstate(), 0);
269......      int i = 1 ;
270......      for (Numeric num : n.get_results())
271......        write_edge(n, (FirmNode)num, i++);
272......      super.action(n);
273......    }

275......    @Override protected void action (final End n){
276......      if (done(n)) return ;
277......      write_containing(n);
278......      super.action(n);
279......    }
280......    @Override protected void action (final Jmp n){
281......      if (done(n)) return ;
282......      write_containing(n);
283......      super.action(n);
284......    }
285......    @Override protected void action (final Cond n){
286......      if (done(n)) return ;
287......      write_containing(n);
288......      write_edge(n, (NumericNode)n.get_selector(), 0);
289......      super.action(n);
290......    }
291......    @Override protected void action (final Sync n){
292......      if (done(n)) return ;
293......      write_containing(n);
294......      int i = 0 ;
295......      for (MemoryState mem : n.get_predecs())
296......        write_edge(n, (FirmNode)mem, i++);
297......      super.action(n);
298......    }
299......    @Override protected void action (final Phi n){
300......      if (done(n)) return ;
301......      write_containing(n);
302......      for (Integer i : n.get_alternatives().keySet())
303......        write_edge(n, (BlockNode)n.get_alternatives().get(i), i);
304......      super.action(n);
305......    }
306......    @Override protected void action (final NumericConst n){
307......      if (done(n)) return ;
308......      write_containing(n);
309......      super.action(n);
310......    }
311......    @Override protected void action (final Unary n){
312......      if (done(n)) return ;
313......      write_containing(n);
314......      write_edge(n, (NumericNode)n.get_on(), 0);
315......      super.action(n);
```

```
316......      }
317......      @Override protected void action (final Binary n){
318......        if (done(n)) return ;
319......        write_containing(n);
320......        write_edge(n, (NumericNode)n.get_left(), 0);
321......        write_edge(n, (NumericNode)n.get_right(), 1);
322......        super.action(n);
323......      }
324......      @Override protected void action (final Ternary n){
325......        if (done(n)) return ;
326......        write_containing(n);
327......        write_edge(n, (NumericNode)n.get_first(), 0);
328......        write_edge(n, (NumericNode)n.get_second(), 1);
329......        write_edge(n, (NumericNode)n.get_third(), 2);
330......        super.action(n);
331......      }
332......
333......  }// class Writeout_edges
334......
335......}
336......
337......
338......// eof
```

I   Consistency checks on the Firm model

```
0......package eu.bandm.ttc2011.case2.transformations ;
1......
2......import java.util.Map ;
3......import java.util.HashMap ;
4......import java.util.Set;
5......import java.util.HashSet ;
6......
7......import eu.bandm.tools.message.SimpleMessage ;
8......import eu.bandm.tools.message.Location ;
9......import eu.bandm.tools.message.Locatable ;
10......import eu.bandm.tools.message.XMLDocumentIdentifier ;
11......
12......import eu.bandm.tools.message.MessageReceiver ;
13......import eu.bandm.tools.message.MessageCounter ;
14......import eu.bandm.tools.message.MessageTee ;
15......
16......import eu.bandm.ttc2011.case2.firm_01.* ;
17......
18......
19....../**
20......Performs basic consistenty checks on a firm model.
21......<p>
22......Every firm model is represented by its "endnode", from which all nodes are
23......reachable.
24......<p>
25......In detail it checks
26......<ul>
27......<li> every block (except the end block)
28......contains exactly one(1) control flow node ("jump" or "cond"; which represents
29......what to happen after the complete execution of the block's code)
30......<li>
31......the block which contains the endnode ("end block") does NOT have a
32......control flow node
33......<li> every phi node has as many inputs (outgoing edges)
34...... as the block it is contained in.
```

```
35......<li> the inputs (outgoing edges) of both are numbered consecutively starting with zero.
36......</ul>
37......<p>
38......Additionally some transient cache values are updated to reflect the results,
39......but they are not maintained in the following code!
40......<p>
41......Some properties have already been checked when the Gxl model has
42......been translated, see {@link Gxl2Firm}. These are:
43......<ul>
44......<li>
45......That there is only one "end" and only one "start" node.
46......<li>
47......That every "BlockNode" is related to a "Block".
48......<li>
49......That the numeric nodes have all of their numeric arguments, starting at
50......position number 0(zero), consecutively.
51......</ul>
52......
53......
54......*/
55......
56......public class Checker extends Firm.VisitBlocksOnce  {
57......
58......   final MessageTee<SimpleMessage<XMLDocumentIdentifier>> msg
59......     = new  MessageTee<SimpleMessage<XMLDocumentIdentifier>>();
60......   final MessageCounter<SimpleMessage<XMLDocumentIdentifier>> msgC
61......     = new MessageCounter<SimpleMessage<XMLDocumentIdentifier>>();
62......   {msg.add(msgC);}
63......
64......   protected void ERROR (Location<XMLDocumentIdentifier> loc, String txt){
65......     msg.receive(SimpleMessage.error(loc, txt));
66......   }
67......
68......
69......   public boolean check
70......     ( final End endnode,
71......       final MessageReceiver<SimpleMessage<XMLDocumentIdentifier>> msg){
72......     this.msg.add(msg) ;
73......     match(endnode);
74......     final Block endblock = endnode.get_block();
75......     if (finalNode.containsKey(endblock))
76......       ERROR(finalNode.get(endnode).get_location(),
77......             "The block containing the end node does contain a jump/cond node");
78......     if (visitedBlocks.size()!=finalNode.size()+1)
79......       for (Block b : visitedBlocks)
80......         if (!(b==endblock || finalNode.containsKey(b)))
81......           ERROR(b.get_location(), "this block does not contain a jump/cond node");
82......     return msgC.getCriticalCount()==0;
83......   }
84......
85......
86......   public static boolean isIntegerType (NumericType t){
87......     switch (t){
88......       case Iu: case  Is: case  Su: case  Ss: case  Bu: case  Bs:
89......         return  true ;
90......     default: return false ;
91......     }
92......   }
93......
94......   public static boolean isValidCondSelector (Numeric n){
95......     final NumericType t = ((NumericNode)n).get_type();
96......     return (t==NumericType.b) || isIntegerType(t);
97......   }
98......
99......
```

```
100......   protected Map<Block, BlockNode> finalNode
101......     = new HashMap<Block, BlockNode>();
102......
103......   /** Checks whether "set" contains only numbers from zero to pred(set.size()),
104......       ergo all these numbers.
105......    */
106......   protected boolean checkIndexSet (final Set<Integer> set){
107......     final int count = set.size();
108......     for (final Integer i:set)
109......       if (i<0||i>=count)
110......         return false ;
111......     return true ;
112......   }
113......
114......
115......   @Override public void action (final Jmp cf){
116......     memoControlFlow(cf);
117......     super.action(cf);
118......   }
119......   @Override public void action (final Cond cond){
120......     memoControlFlow(cond);
121......     super.action(cond);
122......   }
123......   @Override public void action (final Start start){
124......     memoControlFlow(start);
125......     super.action(start);
126......   }
127......   @Override public void action (final Return r){
128......     memoControlFlow(r);
129......     super.action(r);
130......   }
131......
132......   /** Checks whether every Block contains only one control flow,
133......       and enters this into the transient map "finalNode"
134......   */
135......     // @Override public void action (final ControlFlowNode cf){
136......   protected void memoControlFlow (final BlockNode cf){
137......     final Block block = cf.get_block();
138......     if (finalNode.containsKey(block))
139......       if (finalNode.get(block)!=cf)
140......         ERROR(block.get_location(), "Block contains more than one control flow, "
141......             +"namely defined at "+finalNode.get(block).get_location()
142......             +" and defined at "+cf.get_location());
143......     finalNode.put(block,cf);
144......   }
145......
146......   /** Checks whether the indices of the predecs of the phi node
147......       are the same as that of the containing block.
148......   */
149......   @Override public void action (final Phi phi){
150......     final Block b = phi.get_block();
151......     final int count = phi.get_alternatives().size();
152......     if (b.get_predecs().size()!=count){
153......       ERROR(phi.get_location(),
154......             "phi node has not the same count of outgoing edges(/input) "
155......             +"as containing block.");
156......     }
157......     else if (!checkIndexSet(phi.get_alternatives().keySet()))
158......       ERROR(phi.get_location(),
159......             "phi node alternatives not numbered from zero, consecutively.");
160......     super.action(phi);
161......   }
162......
163......
164......   /** Checks whether the indices of the predecs of the block are
```

```
165......     starting from zero and consecutive.
166......   */
167......   @Override public void action (final Block b){
168......     if (!checkIndexSet(b.get_predecs().keySet()))
169......       ERROR(b.get_location(),
170......             "block predecessors (/outgoing edges) "
171......             +"not numbered from zero, consecutively.");
172......     super.action(b);
173......   }
174......
175......}
176......
177......
178......
179......
180......// eof
```

J Constant Folding

```
0......package eu.bandm.ttc2011.case2.transformations ;
1......
2......import java.util.Set ;
3......import java.util.HashSet ;
4......import java.util.Map ;
5......import java.util.HashMap ;
6......
7......import eu.bandm.tools.message.XMLDocumentIdentifier;
8......import eu.bandm.tools.message.Location;
9......import eu.bandm.tools.message.SimpleMessage ;
10......import eu.bandm.tools.message.MessageReceiver ;
11......import eu.bandm.tools.message.MessageCounter ;
12......import eu.bandm.tools.message.MessageTee ;
13......
14......import eu.bandm.tools.umod.runtime.CheckedMap_RD;
15......
16......import static eu.bandm.tools.ops.Collections.the ;
17......
18......import eu.bandm.ttc2011.case2.firm_01.* ;
19......
20......public class ConstantFolding {
21......
22......  final MessageTee<SimpleMessage<XMLDocumentIdentifier>> msg
23......    = new  MessageTee<SimpleMessage<XMLDocumentIdentifier>>();
24......  final MessageCounter<SimpleMessage<XMLDocumentIdentifier>> msgC
25......    = new MessageCounter<SimpleMessage<XMLDocumentIdentifier>>();
26......  {msg.add(msgC);}
27......
28......  protected void ERROR (Location<XMLDocumentIdentifier> loc, String txt){
29......    msg.receive(SimpleMessage.error(loc, txt));
30......  }
31......
32......  public ConstantFolding
33......    (final MessageReceiver<SimpleMessage<XMLDocumentIdentifier>> msgExt){
34......    msg.add(msgExt);
35......  }
36......
37......  public End rewrite (final End endnode, final int visualizationMode){
38......    final End e1 = new ConstPropagator().rewrite_typed(endnode);
39......    //    return e1;
40......    new Visual().visualize(e1, "after constant expression propagation",
41......                       visualizationMode);
```

```
42......
43......      final End e2 = new CoRewriter().rewrite_typed(e1);
44......      new DeadBlockEliminator().checkAlive(e2.get_block());
45......      new Visual().visualize(e2, "after dead block elimination",
46......                             visualizationMode);
47......
48......      final End e3 = new DeadPhiEliminator().rewrite_typed(e2);
49......      // FIXME required for "const.gxl", REICHT NICHT !!
50......      final End e4 = new ConstPropagator().rewrite_typed(e3);
51......      return e4 ;
52......    }
53......
54......    // ============================================================
55......
56......    protected class ConstPropagator extends Rewriter {
57......
58......      /** Replaces a unary numeric operation by a constant, iff operand is constant.
59......       */
60......      @Override public void rewriteFields(final Unary clone){
61......        rewriteFields((NumericNode)clone);
62......        final Numeric on = rewrite_typed(clone.get_on());
63......        if (on instanceof NumericConst) {
64......          final NumericConst nc
65......            = new NumericConst(clone.get_location(),
66......                               clone.get_block(),
67......                               ((NumericConst)on).get_type(),
68......                               clone.get_type()+"("
69......                               +((NumericConst)on).get_unparsedValue()
70......                               +")"
71......                               );
72......          // for demo purpose only, instead of effective value calculation:
73......          nc.set_intValue(new Integer(0));
74......          substitute (nc);
75......        }
76......        else if (clone.set_on(on))
77......          substitute(clone);
78......      }
79......
80......
81......      /** Replaces a binary numeric operation by a constant, iff both
82......          operands are constants.
83......       */
84......      @Override public void rewriteFields(final Binary clone){
85......        rewriteFields((NumericNode)clone);
86......        final Numeric left = rewrite_typed(clone.get_left());
87......        final Numeric right = rewrite_typed(clone.get_right());
88......        if ((left instanceof NumericConst) && (right instanceof NumericConst)){
89......          final NumericConst nc
90......            = new NumericConst(clone.get_location(),
91......                               clone.get_block(),
92......                               ((NumericConst)left).get_type(),
93......                               clone.get_type()+"("
94......                               +((NumericConst)left).get_unparsedValue()
95......                               +","+((NumericConst)right).get_unparsedValue()+")"
96......                               );
97......          // for demo purpose only, instead of effective value calculation:
98......          nc.set_intValue(new Integer(0));
99......          substitute (nc);
100......        }
101......        else {
102......          if (clone.set_left(left))
103......            substitute(clone);
104......          if (clone.set_right(right))
105......            substitute(clone);
106......        }
```

```
107......    }
108......
109......
110......    // out of place for demo purpose, since no explicit type checker provided
111......    protected Set<Cond> checked = new HashSet<Cond>();
112......
113......    /** Replaces a Proj_X  by an unconditional Jump or eliminates it, iff
114......       the input to its Cond is a const.
115......    */
116......    @Override public void action (final Proj_X proj){
117......      super.action(proj);
118......      final Proj_X rewritten = (Proj_X)getResult() ;
119......      final Cond cond = rewritten.get_input();
120......      final NumericNode selector = (NumericNode) cond.get_selector();
121......      if (selector instanceof NumericConst){
122......        if (Checker.isValidCondSelector(selector)){
123......          final int constSelection = ((NumericConst)selector).get_intValue();
124......          if (proj.get_selection()==constSelection){
125......            substitute(new Jmp(proj.get_location(), rewritten.get_block()));
126......          }
127......          else
128......            substitute_empty();
129......        }
130......        else if (!checked.contains(cond)){
131......          ERROR(proj.get_location(),
132......                "type check error, argument to cond must be of integer type");
133......          // thie error should of course have been found earlier,
134......          // during a dedicated type checker's run.
135......          checked.add(cond);
136......        }
137......      }
138......    }
139......
140......    /** Since every loop in the model must pass through a "Block" model element,
141......       it can be cut be cloning the Block and memorizing *in advance*.
142......    */
143......    @Override public void action(final Block block){
144......      final Block clone = breakLoop(block);
145......      if (clone==null)
146......        return ;
147......      rewriteFields (clone);
148......      original=block;
149......      substitute(clone);
150......    }
151......
152......  }//class ConstPropagator
153......
154......
155......  // ================================================================
156......
157......  /** does visit ONLY blocks and Jmp/Proj_X/End
158......   */
159......  protected class DeadBlockEliminator {
160......    protected Set<Block> visited = new HashSet<Block>();
161......
162......    public boolean checkAlive (final Block block){
163......      if (visited.contains(block))
164......        return true ;
165......      final CheckedMap_RD<Integer, ControlFlow> tmp
166......        = new CheckedMap_RD<Integer, ControlFlow>();
167......      for (Map.Entry<Integer, ControlFlow>me : block.get_predecs().entrySet()){
168......        ControlFlowNode cf = (ControlFlowNode)me.getValue();
169......        if ( (cf instanceof Start) || checkAlive(cf.get_block()) )
170......          tmp.put(me.getKey(), cf);
171......      }
```

```
172......        block.set_predecs(tmp);
173......        return  !tmp.isEmpty();
174......     }
175......
176......  } // class DeadBlockEliminator
177......
178......  // ============================================================
179......
180......  public class DeadPhiEliminator extends Rewriter{
181......
182......    // block loop prevention  MISSING
183......
184......    /** replace a Phi with only one input with this input
185......     */
186......    @Override public void action(final Phi phi){
187......      super.action(phi);
188......      final Phi rewritten = (Phi)getResult();
189......      final Set<Integer> keys = rewritten.get_block().get_predecs().keySet() ;
190......      rewritten.get_alternatives().keySet().retainAll(keys);
191......      original = phi;
192......      switch(keys.size()){
193......      case 0: substitute (new Bad()); return ;
194......      case 1: substitute (phi.get_alternatives().get(the(keys)));
195......        return ;
196......      }
197......      substitute(rewritten);
198......    }
199......  }//class DeadPhiEliminator
200......
201......}
202......// eof
```