Transforming Truth Tables to Binary Decision Diagrams using Relational Reference Attribute Grammars

Johannes Mey, René Schöne, Christopher Werner and Uwe Aßmann
{first.last}@tu-dresden.de

Software Technology Group
Technische Universität Dresden

Abstract

The TTC 2019 case describes the computation of a binary decision diagram from a given truth table. In this paper, we show a detailed view on a solution which uses relational reference attribute grammars to represent both input and output model. To transform the former into the latter, we implemented different strategies and present in detail an algorithm using a reduced order binary decision diagram utilizing higher-order attributes. We further present transformation times and the number of decision nodes in the result model showing the feasibility of our approach.

1 Introduction

In the TTC 2019 case, a given truth table has to be transformed into a logically equivalent binary decision diagram. Both truth table and binary decision diagram (BDD) are two different ways to represent a function over Boolean tuples, where on the one hand truth tables can be seen as a set of assignments, each given as a row of the table. On the other hand, a binary decision diagram (BDD) represents this function as a directed acyclic graph (DAG) with a single source. The nodes of the DAG are partitioned into decision nodes and leaf nodes. Decision nodes refer to input variables and have edges annotated with either 0 or 1, representing the corresponding decision. Leaf nodes are annotated with an output tuple.[1]

In Section 2, important basic concepts are explained, which are built upon in our approach described in Section 3. The application of our approach to the given case is presented in Section 4 and evaluated in Section 5. Section 6 concludes this paper with an outlook on how the application of relational RAGs in a modelling context can be facilitated further.

The implementation can be found in a public Git repository.[2]

2 Background

Attribute Grammars [Knu68] use a context-free grammar for the declarative definition of an abstract syntax tree (AST) and attributes to define analyses on this tree. In [Hed00], this concept was extended to Reference Attribute Grammars (RAGs), which allow attributes to compute nodes of the AST, effectively enable the definition of graphs instead of trees. Our solution uses Relational Reference Attribute Grammars [MSH+18], which in turn extend RAGs with first-class relations, i.e, edges between nodes in the AST, to provide an easy definition of those relations and to enable bidirectional relations.

Furthermore, we use an extended feature of RAGs, namely nonterminal attributes [VSK89]. They are attributes computing a new subtree, which after computation is treated like a normal nonterminal. With this, implicit knowledge of the tree can be manifested and later be reused.

[1] BDDs are usually specified for one single output variable. If there are more output variables, *shared BDDs* with multiple source nodes can be used [MT12].

[2] https://git-st.inf.tu-dresden.de/ttc/bdd

In our implementation, we use *JastAdd* [EH07], a RAG system to define both grammar and attributes. The grammar is specified using a BNF syntax with inheritance and relations. Every nonterminal defined there is compiled to a Java class with accessors for its children, attributes, and relations. Attribute definitions are specified using a Java-based DSL and are woven into the Java class of the nonterminal they are defined in. As described in [MSH+18], a preprocessor is used to transform the defined relations into basic grammar rules and special accessors.

3 Computing a Binary Decision Diagrams with Relational RAGs

In this section, we show how truth tables, BDTs, and BDDs can be modelled using relational RAGs and, in particular, how relations help the transformations and tracability between the models. Initially, the case description demanded a binary decision *tree* (BDT) and not a binary decision diagram (BDD). Therefore, we present different configurable algorithms to transform a truth table into either a BDT and a BDD. While the outputs of all provided transformations are semantically equivalent for one given input, they have different characteristics, both in runtime of the transformation and properties of the resulting diagrams.

3.1 Describing Conceptual Models

Many conceptual models can be described by relational RAGs. This is possible, because many model specification languages, e.g., ecore, require models to have a *containment hierarchy*, and thus a spanning tree. To represent the other relations of conceptual models, relational RAGs add the feature of intrinsic non-containment relations to (reference attribute) grammars. Listing 1 shows the grammar of the truth table that corresponds to the provided model. Since the grammar specification rules of *JastAdd* use concepts like inheritance and abstract types, the model can be described accurately in a straight-forward manner.

```
1  LocatedElement ::= <Location:String>;
2  TruthTable : LocatedElement ::= <Name:String> Port:Port* Row:Row* ;  // The start symbol of this grammar
3
4  abstract Port : LocatedElement ::= <Name:String>;                    // Port is an abstract node that cannot be instantiated
5  InputPort : Port;                                                    // InputPort and OutputPort are a special ports, i.e., inheriting
6  OutputPort : Port;                                                   //  name and location from it
7  Row : LocatedElement ::= Cell:Cell*;
8  Cell : LocatedElement ::= <Value:Boolean>;                           // The cell has a intrinsic attribute, in this case
9                                                                       //  the boolean value of the cell
10 rel Port.Cell* <-> Cell.Port;                                        // A bidirectional relation stating, that a cell refers to a port
```

Listing 1: Grammar for a truth table in *JastAdd* syntax.

Likewise, the meta models of BDTs and BDDs can be represented; the grammar of BDTs is shown in Listing 3. While some concepts like *Port* potentially could be reused between the models, we refrained from doing so, because those concepts do not have exactly the same definition in all models.[3]

Important aspects when dealing with several trees at the same time defined by a relational RAGs are the modularity of their specification and the reachability between those trees. RAGs as specified by *JastAdd* are inherently modular: Both grammar and attributes can be split into aspect files. Therefore, both truth table, decision tree and -diagram models are described by one relational grammar, but specified in different modules. Since the truth table grammar shown in Listing 1 has no relations to the other models, it can be used independently. On the other hand, the diagram models have a relation to the truth table, as shown in Listing 3. Note that this is not required, but a design decision, since those relations enable tracebility between the models.

3.2 Computing Models

A model transformation can be seen as the computation of a target model using information of source model. Relational RAGs provide a mechanism for that: computed attributes that compute a tree of the grammar they are defined in, called *higher-order* or *nonterminal attributes (NTAs)*. There are two types of NTAs, those that describe a *sub-tree* of an AST and those that describe a new tree. In both cases, non-containment relations added by *relational* RAGs provide the means to link these (sub-)trees to the original tree. Figure 1 shows the structure of the trees used in our approach. The only given tree is the *TruthTable* tree. In addition, this tree contains two subtrees defined by NTAs, both of the type *PortOrder* shown in Listing 4.

[3]For example, a *Port* within a truth table is a *LocatedElement*, while one in a BDD is not. In other cases, the types may be the same, but the relations between them are not.

```
BDD ::= <Name:String> Port:BDD_Port* Tree:BDD_Tree*;

abstract BDD_Tree;
BDD_Leaf:BDD_Tree ::= Assignment:BDD_Assignment*;
BDD_Subtree:BDD_Tree;

abstract BDD_Port ::= <Name:String>;
BDD_InputPort : BDD_Port;
BDD_OutputPort : BDD_Port;

BDD_Assignment ::= <Value:boolean>;

rel BDD.Root -> BDD_Tree;
rel BDD_Subtree.TreeForZero <-> BDD_Tree.OwnerSubtreeForZero*;
rel BDD_Subtree.TreeForOne <-> BDD_Tree.OwnerSubtreeForOne*;

rel BDD_InputPort.Subtree* <-> BDD_Subtree.Port;
rel BDD_OutputPort.Assignment* <-> BDD_Assignment.Port;

// relations to TruthTable model
rel BDD.TruthTable -> TruthTable;
rel BDD_InputPort.TruthTableInputPort -> InputPort;
rel BDD_OutputPort.TruthTableOutputPort -> OutputPort;
rel BDD_Leaf.Row* -> Row;
```

Listing 2: Grammar for a Binary Decision Diagram in *JastAdd* syntax

```
BDT ::= <Name:String> Port:BDT_Port* Tree:BDT_Tree;

abstract BDT_Tree;
BDT_Leaf:BDT_Tree ::= Assignment:BDT_Assignment*;
BDT_Subtree:BDT_Tree ::= TreeForZero:BDT_Tree TreeForOne:BDT_Tree;

abstract BDT_Port ::= <Name:String>;
BDT_InputPort : BDT_Port;
BDT_OutputPort : BDT_Port;

BDT_Assignment ::= <Value:boolean>;

rel BDT_InputPort.Subtree* <-> BDT_Subtree.Port;
rel BDT_OutputPort.Assignment* <-> BDT_Assignment.Port;

// relations to TruthTable model
rel BDT.TruthTable -> TruthTable;
rel BDT_InputPort.TruthTableInputPort -> InputPort;
rel BDT_OutputPort.TruthTableOutputPort -> OutputPort;
rel BDT_Leaf.Row* -> Row;
```

Listing 3: Grammar for Binary Decision Tree in *JastAdd* syntax

```
PortOrder;
rel TruthTable.PortOrder -> PortOrder ;
rel PortOrder.Port* -> InputPort;

syn PortOrder TruthTable.getNaturalPortOrder() = //...
syn PortOrder TruthTable.getHeuristicPortOrder() = //...
```

Listing 4: Grammar and NTA definition for *PortOrder*

The results of the transformation are stored in another set of trees that are separate from the truth table, computed by the relational NTAs *BDT()*, *orderedBDD()*, and *reducedOBDD()*. The types of these trees are either BDT or BDD, depending on the transformation. Please note that different NTAs may return the same *type* of subtree with different instances. In addition to the relations computed by NTAs, there may be other intrinsic relations, such as the relation that selects one of the two provided port orders. These relations can also link nodes in computed subtrees to nodes in the primary key.[4]

In the following, we describe the concepts of one particular transformation to compute reduced ordered BDDs, here shown as the relational NTA *reducedOBDD*.

3.3 Computing an Ordered Binary Decision Diagram

Constructing an optimal BDD is a computationally very hard problem. However, since there are many real world applications that require optimised, large BDDs, appropriate simplifications and efficient heuristics exist. One commonly used simplification of BDDs are ordered BDDs (OBDD), in which the order of input variables[5] in all paths is identical, allowing the layering of the graph. Given an OBDD, two simple reduction rules can be applied to reduce the number of nodes; the result of such a reduction process is a reduced OBDD. In an OBDD, the minimal diagram for a given order can be computed efficiently, however, the computation of the optimal order is still NP-hard[MT12, p. 139ff.].

We follow the definition of an OBDD given in [MT12] extending the problem to several output variables as mentioned in Section 1 as follows:

- An OBDD with m input and n output variables has at most 2^n leaves, each with a different assignment function $f : V_o \to \{0, 1\}$ with V_o as the set of output variables.
- There is an order $<_\pi$ for input variables, such that on an edge from one node referring to input variable x to a node referring to variable y it holds that $x <_\pi y$.

Conceptually, we split the construction of a reduced ordered BDD in three stages: the computation of a port order, the construction of a perfect tree, and its reduction. Listing 5 shows the attribute that performs this staged process. It is a synthesized attribute defined for a truth table. First, a BDD node is constructed and the relations to the truth table and its variables are established (lines 2–16 of Listing 5). Then, the leaves are

[4] These relations must not be bidirectional, since the direction from the source model to the computed NTA model would violate the rule that attribute computations may not alter the tree — other than adding the result of the computation.

[5] In literature such as [MT12, RK08], ports are called variables.

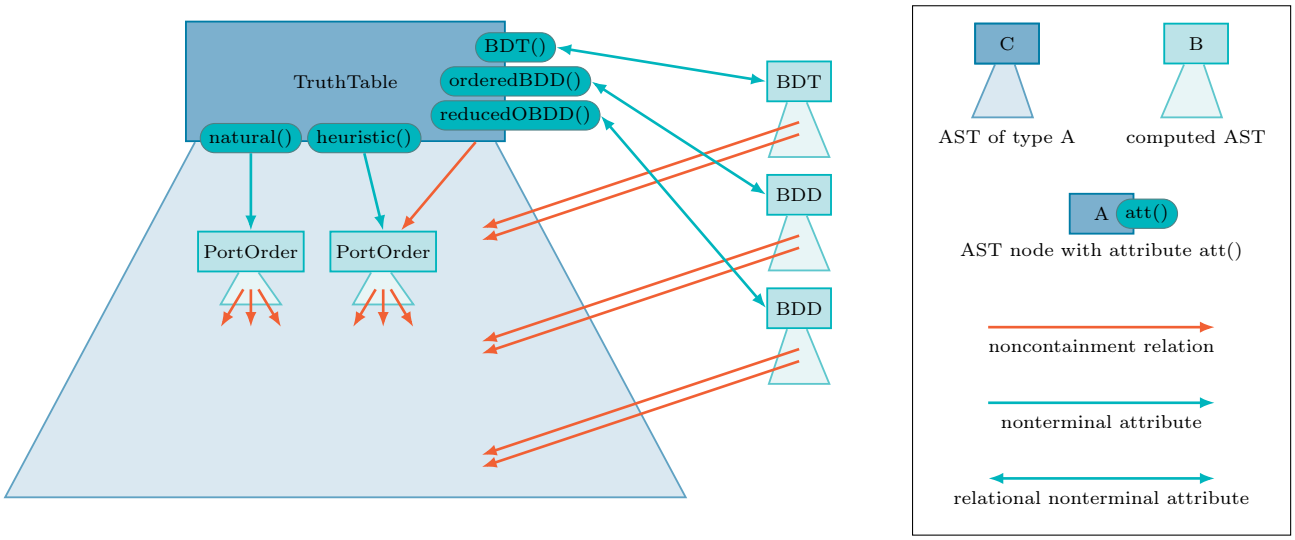


Figure 1: Elements of the transformation and their relations

constructed by a helper function and added to the diagram (line 18). Afterwards, a port order is retrieved by accessing the noncontainment relation to a *PortOrder* pointing to one of the NTAs that computes a port order.

These are defined in a separate grammar aspect shown in Listing 4. Besides the default port order defined in line 5 of Listing 4, we provide a heuristic ordering defined in line 6. Since most heuristics in the literature rely on a logical formula as an input, we have chosen to use a simple metric based on the correlation of an input variable to the output vector in the given truth table. Therefore, every input vector of bits is compared to every output vector, and the similarities regarding output variables are summed up for each input variable. Then, the input variables are ordered in descending order of similarity. Using this port order in Listing 5, the tree is constructed iteratively by adding the path for each input row in lines 23–28. Finally, the reduction is performed in line 30. The employed algorithm can be studied in [MT12, p. 96ff.]. For a given truth table, there is exactly one minimal OBDD [MT12, p. 94ff.] which is computed by the algorithm in $\mathcal{O}(d \log d)$ for d decision nodes [MT12, p. 98].

Besides computing and reducing an OBDD, we have also implemented other approaches as shown in Section 5. How to apply those approaches to the given case will be described in the next section.

4 A Dynamic Transformation Toolchain

To perform the transformation, we follow the process outlined in Figure 2 and describe it in the following. In particular, we focus on demonstrating the means of variability that enables reuse and the combination of different transformation approaches. Additionally, we show how relational RAGs support traceability between models and help in analysing these models.

As *JastAdd* is not based on Ecore, the meta model of EMF [SBMP08], we can not directly use the given input models. Instead, we translated the given metamodel into the grammar shown in Listing 1 and built a hand-written XMI parser constructing an AST according to this grammar. At the current development stage of *JastAdd* of this induces implementation overhead. However, relational RAGs provide some mechanisms to simplify this process.

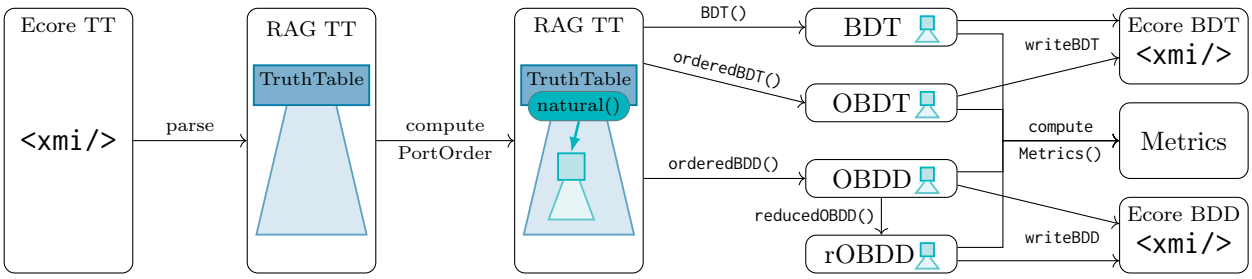


Figure 2: Transformation process and temporary artefacts

```
1  syn BDD TruthTable.reducedOBDD() {
2    BDD bdd = new BDD();                                    // construct an empty BDD
3    bdd.setName(getName());                                 // and copy the name from the truth table
4
5    bdd.setTruthTable(this);                                // create relations to truth table
6    for (Port port: getPortList()) {
7      if (port.isInput()) {                                 // for each port, its type is checked with an attribute
8        BDD_InputPort bddPort = new BDD_InputPort(port.getName());   // and the according BDD variant of the port is created
9        bddPort.setTruthTableInputPort(port.asInput());     // and linked to its truth table counterpart
10       bdd.addPort(bddPort);
11     } else {
12       BDD_OutputPort bddPort = new BDD_OutputPort(port.getName());
13       bddPort.setTruthTableOutputPort(port.asOutput());
14       bdd.addPort(bddPort);
15     }
16   }
17
18   for (BDD_Leaf leaf: constructLeaves()) bdd.addTree(leaf);        // add leaves
19
20   PortOrder portOrder = getPortOrder();                   // obtain the port order by accessing a non-containment relation
21                                                           // to the selected nonterminal attribute
22
23   BDD_Subtree root = new BDD_Subtree();                   // create root node, set its port to the first in the port order,
24   root.setPort(bdd.bddInputPort(portOrder.getPortList().get(0)));   // add it to the BDD, and specify it as the root
25   bdd.addTree(root);
26   bdd.setRoot(root);
27
28   for (Row row : getRowList()) insertRow(bdd, root, row, 0);       // fill the BDD by adding all defined paths using a helper function
29
30   bdd.reduce();                                           // perform the reduction by calling a helper function
31
32   return bdd;
33 }
```

Listing 5: Relational non-terminal attribute to create an ordered BDD

While parsing an XMI file is rather straightforward, the resolution of the XMI references is not. In attribute grammars, name analysis is well-supported and frequently used application. Relational RAGs provide an additional method to defer the name resolution after the parsing while still allowing the result of the attribute-computed name resolution to be stored as a non-containment relation.

Once the truth table is parsed, the transformation is performed. The first step is to create an additional subtree with a *PortOrder*, which can later be used to create ordered BDTs and BDDs. This is also the first configurable step of the process, since the algorithm for the computation of the *PortOrder* can be switched.

Afterwards, BDTs or BDTs are created by different attributes. In the case of the OBDD, the reduction can be seen as an additional step of the computation. Since all variants are independent trees, it is possible to create several variants at the same time. Each created variant contains trace links into the truth table model that are created during construction (cf. Listing 3, lines 17 to 20).

Then, the results are validated and different metrics described in Section 5 are computed. One example for a metric is the number of decision nodes in Listing 6.

```
1 syn int BDT_Tree.decisionNodeCount();
2 eq BDT_Subtree.decisionNodeCount() = 1 + getTreeForZero().decisionNodeCount() + getTreeForOne().decisionNodeCount();
3 eq BDT_Leaf.decisionNodeCount() = 0;
```

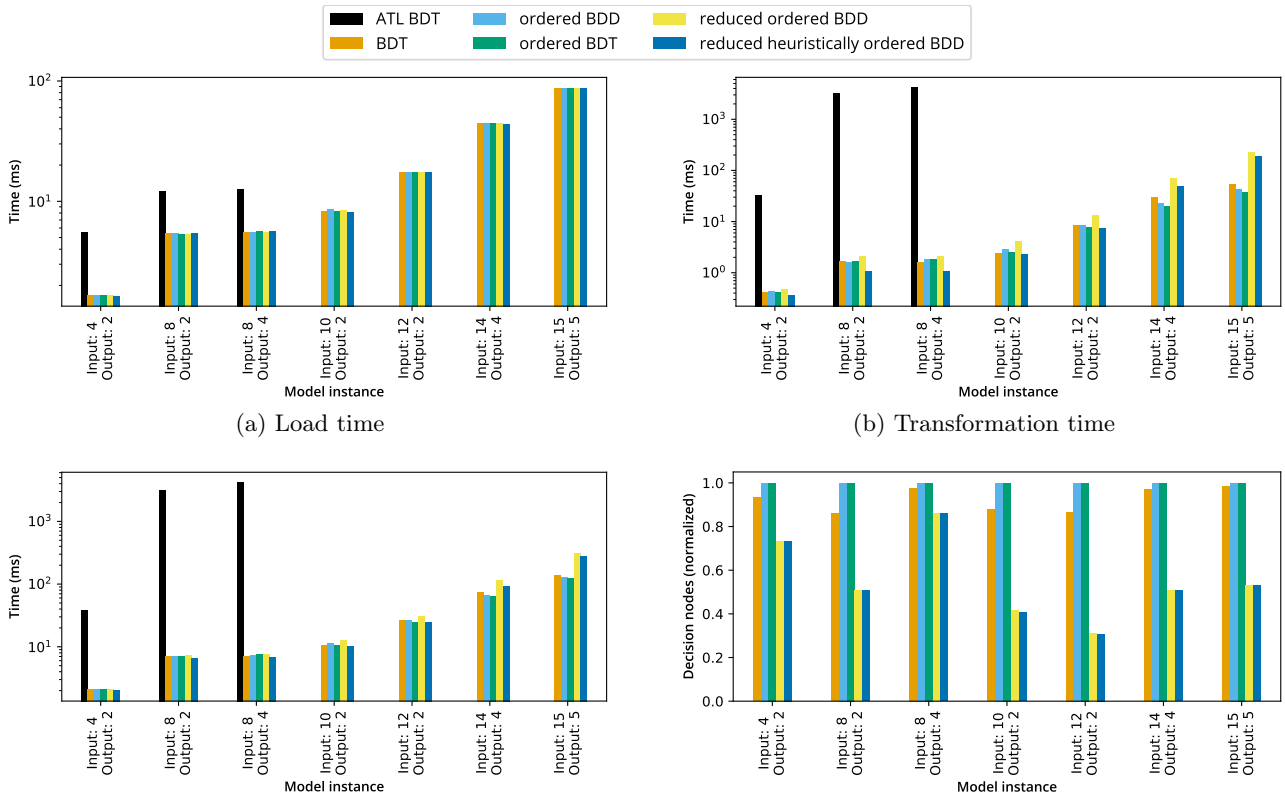
Listing 6: Computation of the number of decision nodes in the BDT

Finally, the resulting BDT or BDD is serialized to XMI. Again, this step requires attributes to compute the references within the file, e.g., paths specified in an XMI.

This whole process is embedded into the provided benchmarking framework and evaluated in the next section.

5 Evaluation

The evaluation of the presented approach is split in two parts. First, the properties of the transformation are discussed with a focus on conciseness, modularity, and reuse. Then, the runtime performance and the quality of the obtained models are evaluated.



(a) Load time

(b) Transformation time

(c) Total time

(d) Number of decision nodes (normalized)

Figure 3: Evaluation result

5.1 Properties of the Transformation

Using relational RAGs, the presented models can be specified concisely. As shown in Listings 1, 3 and 4, the grammars for the truth table and the BDT comprise few lines of code. The specification of the transformation using the attributes of *JastAdd* proves to be a good combination of imperative code (which is beneficial for such complex transformation algorithms as, e.g., for the OBDD reduction) and efficient attribute-supported tree navigation. Additionally, attributes can help by checking both the *correctness* and different *metrics* of the obtained model.

An important aspect of the presented solution is modularity and the opportunities for reuse that stem from it. Even though, technically, all models are combined in one large grammar, this grammar can be used in independent modules consisting of grammar fragments and accompanying attributes. The example of the *PortOrder* extension shows how additional analysis modules can simply be added to the grammar. Using relations, entire new models can be integrated and used from the existing models. On the attribute level the definition of attributes in *aspects* also helps to combine and reuse separate parts of the transformation system.

Furthermore, relational RAGs allow traceability. Relations between models can be established that, e.g., show which rows have taken part in the creation of a *Leaf* in a BDD.

5.2 Performance and Quality

To evaluate the runtime performance, we used the provided benchmark script with ten runs and a timeout of ten minutes for each model to execute it on a Intel i7-8700 workstation with 64 gigabytes of memory using Fedora Linux 29 running on kernel 4.18, OpenJDK version 1.8 and JastAdd version 2.3.3.

In addition to the suggested time measurements, we added a number of quality metrics computed be *JastAdd* attributes, namely the number of nodes of the two different types and the minimum, maximum, and average path length through the result diagrams.

Figure 3 shows our evaluation results for the seven provided input models depicting load time, transformation time, and total runtime as well as the number of decision nodes, i.e., instances of type *SubTree*, normalized to

the greatest number among all variants.[6] For the measurement, we included the provided ATL solution and the following variants of our approach: The algorithm presented in the case description (*BDT*), an ordered BDT variant generating the tree iteratively (*ordered BDT*), an ordered BDD generated iteratively (*ordered BDD*) and the algorithm described in Section 3.3 (*reduced ordered BDD*). For the last variant, we used two different port orders: the order given in the truth table, and an order determined by a heuristic based on the correlations between input and output variables.

The reference solution ATL is only able to compute a BDD for the smallest four input models, and was much slower than our approaches. The improvement over the ATL performance is caused both by reduced load times shown in Figure 3a and a shorter transformation time shown in Figure 3b. Additionally, relational RAGs do not require any initialization other than loading the required Java classes, resulting in a good overall performance shown in Figure 3c. Comparing the execution times of our approaches, all perform almost equally well, except for the reduced OBDD, which takes up 2.6 times more time for the largest model with 15 inputs and five outputs. This is due to the final reduction step.

However, looking at the number of decisions in the final result, there are three classes of approaches. All non-reduced OBDD variants generate the largest number of decision nodes, as they always produce the full tree. The algorithm used in the case description results in up to 1.16 times fewer nodes. After applying the reduction algorithm, there are only 31% to 86% nodes left compared to the full tree. Unfortunately, using the heuristic port order results in hardly any improvement nodes compared to the natural order; here, more adequate heuristics need to be developed.

The set of variants could be easily extended, e.g., having more heuristics by providing new nonterminal attributes computing them, or applying the elimination part of the reduction algorithm for BDT by partly reusing the existing algorithm.

6 Conclusion and Future Work

We have shown how to apply Relational Reference Attribute Grammars to the problem of transforming truth tables into (ordered) binary decision diagrams. This included defining a suitable grammar for both truth tables and BDDs, parsing the given input models, computing a BDD in different ways while reusing intermediate results, and finally printing the result to the required XMI format. We used the *JastAdd* system to implement our solution and were able to create several configurable transformations with good performance compared to the provided ATL baseline. As the focus of this contribution was not to create new transformation algorithms (we used established ones), we show how the implementation of transformation algorithms, metrics, and tracing relations can be improved with relational RAGs.

The presented approach demonstrates a manual bidirectional transformation from ecore-based models to *JastAdd* ASTs and back. While the grammar as well as the parser and printer were hand-written, there is no obvious reason why this could not be automated. Having an automated transformation from ecore meta-models to grammars and their instances to ASTs would instantly make a huge set of existing models available for efficient attribute-grammar based analysis and is therefore an important direction of research.

Acknowledgements

This work has been funded by the German Research Foundation within the Research Training Group "Role-based Software Infrastructures for continuous-context-sensitive Systems" (GRK 1907), the research project “Rule-Based Invasive Software Composition with Strategic Port-Graph Rewriting” (RISCOS) and by the German Federal Ministry of Education and Research within the project “OpenLicht’.

References

[EH07] Torbjörn Ekman and Görel Hedin. The JastAdd system—modular extensible compiler construction. *Science of Computer Programming*, 69(1):14–26, 2007.

[GD19] Antonio García-Domínguez. The TTC 2019 TT2BDD Case. In *Proceedings of the 12th Transformation Tool Contest, a part of the Software Technologies: Applications and Foundations (STAF 2019)*, CEUR Workshop Proceedings. CEUR-WS.org, 2019.

[Hed00] Görel Hedin. Reference attributed grammars. *Informatica (Slovenia)*, 24(3), 2000.

[6] Since a variant is included that creates a perfect tree, this number is $2^m - 1$ where m is the number of input variables.

[Knu68] Donald E. Knuth. Semantics of context-free languages. *Mathematical systems theory*, 2(2), 1968.

[MSH+18] Johannes Mey, René Schöne, Görel Hedin, Emma Söderberg, Thomas Kühn, Niklas Fors, Jesper Öqvist, and Uwe Aßmann. Continuous Model Validation Using Reference Attribute Grammars. In *Proceedings of the 11th ACM SIGPLAN International Conference on Software Language Engineering*, SLE 2018, pages 70–82. ACM, 2018.

[MT12] Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD-foundations and applications*. Springer Science & Business Media, 2012.

[RK08] Michael Rice and Sanjay Kulhari. A survey of static variable ordering heuristics for efficient bdd/mdd construction. Technical report, 2008.

[SBMP08] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: Eclipse Modeling Framework*. Pearson Education, 2008.

[VSK89] H. H. Vogt, S. D. Swierstra, and M. F. Kuiper. Higher order attribute grammars. In *Proceedings of the ACM SIGPLAN 1989 Conference on Programming Language Design and Implementation*, PLDI '89, pages 131–145, New York, NY, USA, 1989. ACM.