Truth Tables to Binary Decision Diagrams in Modern ATL

Dennis Wagelaar
Corilus
Vilvoorde, Belgium
dennis.wagelaar@corilus.be

Théo Le Calvar
LERIA, Université d'Angers
Angers, France
theo.lecalvar@univ-angers.fr

Frédéric Jouault
ERIS, ESEO-TECH
Angers, France
frederic.jouault@eseo.fr

Abstract

Model transformation technology has evolved over the last 15 years, notably regarding scalability and performance. The Truth Tables to Binary Decision Diagrams transformation was written for an early version of ATL roughly 13 years ago. At that time, performance was not as much of a concern as it is today. Not only were execution engines slower than now, but they also did not provide as many optimization opportunities. Consequently, in its original form, this transformation does not scale well to large models. It remains slow, even when using EMFTVM, the state-of-the-art ATL virtual machine. In this work, we show that by leveraging the profiler, and carefully optimizing the transformation code, significantly improved performance can be achieved. Our updated solution scales up to the largest generated model ($\sim$ 40 MB), which is transformed in about 23 seconds on a modern desktop (Ryzen 5 1600X with 16 GB RAM running Fedora 29): several hundred times faster than the original code.

1 Introduction

This paper presents the ATL/EMFTVM solution to the offline case of the Transformation Tool Contest[1] 2019: Truth Tables to Binary Decision Diagrams (TT2BDD)[2] [GDH19]. This case was developed from an ATL transformation published in the ATL Transformation Zoo [Sav06] roughly 13 years ago in 2006: TT2BDD, a model transformation initially written for the first ATL virtual machine ever released. Since that time, two generations of ATL virtual machines have been developed: EMFVM around 2007, and EMFTVM[3] [WTCJ11] from 2011 on.

The resources for the case include the original ATL code from 2006, but run it with EMFTVM. This results in better performance than using the virtual machine of the time, but it does not solve the issues that are in the transformation code. Our updated solution[4] is called ATLEMFTVMImproved, and mostly consists of optimizations to the transformation code. These optimizations are described in the following section.

In: Antonio Garcia-Dominguez, Georg Hinkel, and Filip Krikava (eds.): Proceedings of the 12th Transformation Tool Contest, Eindhoven, The Netherlands, 19-07-2019, published at http://ceur-ws.org

[1] `https://www.transformation-tool-contest.eu/`

[2] GitHub repository with accompanying resources: `https://github.com/TransformationToolContest/ttc2019-tt2bdd`

[3] Documented at `https://wiki.eclipse.org/ATL/EMFTVM`.

[4] GitHub repository with our solution: `https://github.com/dwagelaar/ttc2019-tt2bdd`

2 Optimizations

In order to optimize the TT2BDD transformation, we used the EMFTVM built-in profiler. This lead us to the three following optimizations:

- **Leveraging helper attributes caching.** ATL provides two kinds of helpers: helper operations, and helper attributes. Helper attributes are basically similar to parameterless helper operations with a significant performance-related difference: their result is cached. Therefore, multiple accesses do not result in multiple computations. The `getTree` helper operation was thus changed into a `tree` helper attribute. With a lower performance impact, the `getNode` helper operation was also changed into a `node` helper attribute.
- **Applying the object indexing pattern[5].** This pattern uses a Map in order to avoid expensive lookups that can be precomputed. A typical example is the navigation of missing opposite references.
- **Leveraging Maps.** In the `getPartition` helper operations, the original row-accessing code is quadratic: a use of the `exists` iterator in the body of a `select` iterator. We instead compute two Maps: one for each of the possible `true` and `false` values. These Maps are indexed by ports. Moreover, these Maps are computed in helper attributes, and are therefore cached and computed only once per context. Because EMFTVM uses a HashMap to implement Maps, the resulting code has a linear time complexity.

Another optimization was performed in the transformation launcher: module loading code was moved into the initialization phase.

The optimizations were applied after repeated measurements with the ATL/EMFTVM profiler. This profiler can be enabled by checking the ”Display Profiling Data” box in the Eclipse Run Configuration dialog. Listing 6 in Appendix shows the output of running the ATL reference solution (before any optimization) against the “GeneratedI8O2Seed68.ttmodel”. Typically, the maximum gains can be achieved by addressing the top lines of the profiler output, in this case the `getPartition` and `getTree` helper operations. The top line of the profiler output amounts to 21.45% of the total measured runtime of the transformation. It represents a qualified portion of the `getPartition` helper operation: the “@0@0” qualifier designates the first closure within the first closure of the `getPartition` helper body, which is invoked 41811968 times.

Listing 1 shows the `getPartition` helper operation. The first closure within the first closure is the body of the `exists` iterator within the `select` iterator (line 9). In this case, the fact that so much runtime is spent here is due to the high amount of invocations.

```
 1 helper def:
 2    getPartition(rows : Sequence(TT!Row), port : TT!Port)
 3       : TupleType( zeroPart : Sequence(TT!Row) , onePart : Sequence(TT!Row) ) =
 4
 5       -- Select the rows for which the port is false
 6       let _zeroPart : Sequence(TT!Row) =
 7          rows->select(r |
 8             r.cells->exists(c |
 9                c.port = port and c.value = false
10             )
11          ) in
12
13       -- Select the rows for which the port is true
14       let _onePart : Sequence(TT!Row) =
15          rows->select(r |
16             r.cells->exists(c |
17                c.port = port and c.value = true
18             )
19          ) in
20
21       -- Build the resulting tuple
22       Tuple{
23          zeroPart = _zeroPart,
24          onePart = _onePart
25       };
```

Listing 1: The `getPartition` helper operation

Instead of trying to optimize directly at this level, we first trace back where the `getPartition` helper is invoked: it is invoked only from within the `getTree` helper operation. The `getTree` helper operation is also

[5] https://wiki.eclipse.org/ATL/Design_Patterns#Object_indexing

responsible for a large portion of the transformation runtime: the main helper body is responsible for 8.50% of the measured runtime, and is invoked 718080 times (line 7 of Listing 6). There are two `getTree` helper operations: one with input parameters, and one without. The one with input parameters invokes itself recursively, and the one without input parameters only invokes the one with parameters.

Listing 2 shows the `getTree` helper without input parameters. This helper operation is invoked on the `TT!TruthTable` context, of which there is always one instance in an input model. Yet, in 6, line 16, it shows up with 2816 invocations. If we convert this helper operation into a helper attribute, its body will be executed only once for each `TT!TruthTable` instance, and any subsequent invocations will be retrieved from cache.

```
helper context TT!TruthTable def:
   getTree()
      : TupleType( cell : TT!Cell , zeroSubtree : OclAny , oneSubtree : OclAny ) =
      thisModule.getTree(self.rows, self.ports->select(p | p.oclIsKindOf(TT!InputPort)));
```

Listing 2: The `getTree` helper operation

Commit c1208ae[6] contains the first performance optimization: convert the parameterless `getTree` helper operation into the `tree` helper attribute. Listing 7 shows the profiler output after applying this optimization. There is only one invocation of the `tree` helper attribute body on line 21. As a result, the amount of runtime spent in the remaining `getTree` helper operation and the `getPartition` helper operation has also been drastically reduced. The total runtime has been reduced from 69.832018 seconds to 0.703755 seconds.

Now, the top line in the profiler output points to the `findCell` helper operation. `findCell` is invoked by the `getNode` helper operation. `getNode` is also a parameterless helper operation, which can easily be converted to a helper attribute to reduce the amount of times its body is invoked. Commit dd9b976[7] does just that, and Listing 8 shows the profiler output after applying this optimization. The 2815 invocations of `getNode` on line 10 of Listing 7 have become 2560 invocations of the `node` helper attribute on line 10 of Listing 8. Not a significant improvement this time, but it has been achieved with very little effort.

That leaves the `findCell` helper at the top of the list with 620415 invocations, responsible for 87.77% of the measured runtime. Listing 3 shows this `findCell` helper. Its purpose is to find the corresponding tree node for a given TT cell. Ideally, it is invoked once per cell, but the representation of the in-memory tree structure makes that we cannot simply convert this operation into an attribute. We therefore resort to the object indexing pattern, which computes a Map of cells to their tree nodes.

```
helper def:
   findCell(cell : TT!Cell, tree : TupleType( cell : TT!Cell, zeroSubtree : OclAny, oneSubtree : OclAny ))
      : TupleType( cell : TT!Cell , zeroSubtree : OclAny , oneSubtree : OclAny ) =

      if tree.cell = cell then
         tree
      else if tree.zeroSubtree.oclIsKindOf(TT!Row) then
         if tree.oneSubtree.oclIsKindOf(TT!Row) then
            -- Both subtrees are leaf nodes
            OclUndefined
         else
            -- Only the subtree 1 is not a leaf
            thisModule.findCell(cell,tree.oneSubtree)
         endif
      else
         let tryInZero : OclAny = thisModule.findCell(cell,tree.zeroSubtree) in
         if tree.oneSubtree.oclIsKindOf(TT!Row) then
            -- Only the subtree 0 is not a leaf
            tryInZero
         else if tryInZero.oclIsUndefined() then
            -- Both subtrees are non-leaves, but subtree 0 did not produce any results
            thisModule.findCell(cell,tree.oneSubtree)
         else
            -- Both subtrees are non-leaves, and subtree 0 has produced results
            tryInZero
         endif endif
      endif endif;
```

Listing 3: The `findCell` helper operation

[6] https://github.com/dwagelaar/ttc2019-tt2bdd/commit/c1208aebc2e3a89601411b66c83446f555dd9fe6

[7] https://github.com/dwagelaar/ttc2019-tt2bdd/commit/dd9b976fb79d95bb11748bb96d5e8326fa43feae

Listing 4 shows the new `nodesByCell` helper attribute, with its companion `collectAllNodes` helper operation. `nodesByCell` uses the `mappedBySingle` built-in helper operation introduced by EMFTVM to convert a list of tree nodes into a map of cells to nodes. Whereas it is trivial to find the accompanying cell for a given tree node, there is no direct way to find the tree node for a given cell. `mappedBySingle` enables one to quickly reverse navigate a given EMF `EReference`. The `collectAllNodes` helper operation serves to flatten the tree of nodes into a `Sequence` of nodes, such that it can be consumed by `mappedBySingle`.

```
helper context TT!TruthTable def:
        nodesByCell
                : Map(TT!Cell, TupleType( cell : TT!Cell, zeroSubtree : OclAny, oneSubtree : OclAny )) =

                thisModule.collectAllNodes(self.tree)
                        ->mappedBySingle(node | node.cell);

helper def:
        collectAllNodes(tree : TupleType( cell : TT!Cell, zeroSubtree : OclAny, oneSubtree : OclAny ))
                : Sequence(TupleType( cell : TT!Cell , zeroSubtree : OclAny , oneSubtree : OclAny )) =

                if tree.zeroSubtree.oclIsKindOf(TT!Row) then
                        if tree.oneSubtree.oclIsKindOf(TT!Row) then
                -- Both subtrees are leaf nodes
                Sequence{}
                else
                -- Only the subtree 1 is not a leaf
                thisModule.collectAllNodes(tree.oneSubtree)
                        endif
                else
                        if tree.oneSubtree.oclIsKindOf(TT!Row) then
                -- Only the subtree 0 is not a leaf
                thisModule.collectAllNodes(tree.zeroSubtree)
                        else
                -- Both subtrees are non-leaves
                                thisModule.collectAllNodes(tree.zeroSubtree)->union(
                                        thisModule.collectAllNodes(tree.oneSubtree))
                        endif
                endif
                ->prepend(tree);
```

Listing 4: The `findCell` helper operation

Listing 9 shows the profiler output after applying the object indexing pattern: total measured runtime has been reduced from 0.670487 seconds to 0.086927 seconds, which is significant. The remaining top entries in the profiling output are `getPartition` and `getTree` (and contained closures thereof). We will focus on the top entry (line 3), which is the first closure within the first closure of `getPartition`. Listing 1 shows the `getPartition` helper operation, our first example, of which line 9 represents the first closure within the first closure. Along with line 5 of the profiler output – the first closure within the second closure (Listing 1 line 17) – this closure stands out through its high amount of invocations (14848), and resulting percentage of the total measured runtime (12.37%).

In commit 6fcd025[8], we again apply the object indexing pattern to quickly retrieve cells by their port, also prefiltered by their value (true or false). Listing 5 shows the improved version of the `getPartition` helper. The double nesting of closures has been eliminated, and instead two Maps are created for each row, containing the `true` cells mapped by their port and the `false` cells by their port.

Listing 10 shows the profiler output after this optimization. The remaining entries in this output show little opportunity for further optimization: either their percentage in the total measured runtime is very low, or the number of invocations is very low (i.e. not higher than the amount of model elements). As such, we have decided not to optimize further at this point. Overall, we have achieved a speedup of 912 times.

3 Conclusion

The state of the art EMFTVM was able to run an old ATL transformation. Although, it cannot automatically optimize it, its built-in profiler makes it possible to quickly spot performance bottlenecks and fix them by making relatively simple changes such as turning parameterless helper operations into helper attributes, or applying well-documented patterns such as the object indexing pattern. This paper has also illustrated how to interpret the profiler output, and how to use it in order to apply performance optimizations.

[8] `https://github.com/dwagelaar/ttc2019-tt2bdd/commit/6fcd025952451461a2affb415a75c54f90f3562b`

```
helper def:
   getPartition(rows : Sequence(TT!Row), port : TT!Port)
      : TupleType( zeroPart : Sequence(TT!Row) , onePart : Sequence(TT!Row) ) =

      -- Select the rows for which the port is false
      let _zeroPart : Sequence(TT!Row) =
         rows->reject(r |
            r.falseCellsByPort.get(port).oclIsUndefined()
         ) in

      -- Select the rows for which the port is true
      let _onePart : Sequence(TT!Row) =
         rows->reject(r |
            r.trueCellsByPort.get(port).oclIsUndefined()
         ) in

      -- Build the resulting tuple
      Tuple{
         zeroPart = _zeroPart,
         onePart = _onePart
      };

helper context TT!Row def: trueCellsByPort : Map(TT!Port, Set(TT!Cell)) =
        self.cells
                ->select(c | c.value)
                ->mappedBy(c | c.port);

helper context TT!Row def: falseCellsByPort : Map(TT!Port, Set(TT!Cell)) =
        self.cells
                ->reject(c | c.value)
                ->mappedBy(c | c.port);
```

Listing 5: The improved `getPartition` helper operation

References

[GDH19] Antonio Garcia-Dominguez and Georg Hinkel. Truth Tables to Binary Decision Diagrams. In Antonio Garcia-Dominguez, Georg Hinkel, and Filip Krikava, editors, *Proceedings of the 12th Transformation Tool Contest, a part of the Software Technologies: Applications and Foundations (STAF 2019) federation of conferences*, CEUR Workshop Proceedings. CEUR-WS.org, July 2019.

[Sav06] Guillaume Savaton. Truth Tables to Binary Decision Diagrams. ATL Transformations, `https://www.eclipse.org/atl/atlTransformations/#TT2BDD`, February 2006. Last accessed on 2019-05-14. Archived on `http://archive.is/HdoHM`.

[WTCJ11] Dennis Wagelaar, Massimo Tisi, Jordi Cabot, and Frédéric Jouault. Towards a General Composition Semantics for Rule-based Model Transformation. In *Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems*, MODELS'11, pages 623–637, Berlin, Heidelberg, 2011. Springer-Verlag.

Appendix

```
Duration (sec.) Duration (%)    Invocations     Operation
14,974607    21,45  41811968    static EMFTVM!ExecEnv::getPartition(...) : Tuple@0@0
14,942565    21,40  41811968    static EMFTVM!ExecEnv::getPartition(...) : Tuple@1@0
 6,402944     9,17   5767168    static EMFTVM!ExecEnv::getPartition(...) : Tuple@0
 6,379739     9,14   5767168    static EMFTVM!ExecEnv::getPartition(...) : Tuple@1
 5,932174     8,50    718080    static EMFTVM!ExecEnv::getTree(...) : Tuple
 4,628028     6,63   5767168    static EMFTVM!ExecEnv::getTree(...) : Tuple@0@0
 4,183250     5,99  25952256    static EMFTVM!ExecEnv::getTree(...) : Tuple@0@0@0
 1,597109     2,29   5049088    static EMFTVM!ExecEnv::getTree(...) : Tuple@1
 1,380862     1,98   5049088    TT!TruthTable::getTree() : Tuple@0
 0,732114     1,05    653055    static EMFTVM!ExecEnv::findCell(...) : Tuple
 0,707898     1,01    718080    static EMFTVM!ExecEnv::getTree(...) : Tuple@0
 0,701314     1,00    718080    static EMFTVM!ExecEnv::getPartition(...) : Tuple
 0,010936     0,02      2815    TT!Cell::getNode() : Tuple
 0,008518     0,01      2816    TT!TruthTable::getTree() : Tuple
 0,005499     0,01       255    rule Cell2Subtree@applier
 0,004568     0,01      2560    rule Cell2Subtree@matcher
 0,003421     0,00      2560    rule Cell2Assignment@matcher
 0,002911     0,00       256    rule Row2Leaf@applier
 0,002905     0,00      2560    rule Row2Leaf@applier@0
 0,001704     0,00       512    rule Cell2Assignment@applier
 0,000131     0,00         1    rule TruthTable2BDD@applier
 0,000015     0,00         8    rule InputPort2InputPort@applier
 0,000004     0,00         2    rule OutputPort2OutputPort@applier
 0,000001     0,00         1    static EMFTVM!ExecEnv::init() : Object
 0,000000     0,00         1    static EMFTVM!ExecEnv::main() : Object
Timing data:
        Loading finished at 0,008558 seconds (duration: 0,008558 seconds)
        Matching finished at 63,503997 seconds (duration: 63,495439 seconds)
        Applying finished at 69,830402 seconds (duration: 6,326405 seconds)
        Post-applying finished at 69,830517 seconds (duration: 0,000115 seconds)
        Recursive stage finished at 69,830529 seconds (duration: 0,000012 seconds)
        Execution finished at 69,832018 seconds (duration: 0,001501 seconds)
```

Listing 6: ATL/EMFTVM Profiler output 1

```
Duration (sec.) Duration (%)    Invocations     Operation
0,613147   88,17 653055 static EMFTVM!ExecEnv::findCell(cell: TT!Cell, tree: Tuple) : Tuple
0,010024    1,44  14848 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@0@0
0,009428    1,36    255 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple
0,009250    1,33  14848 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@1@0
0,004373    0,63   2048 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@1
0,004244    0,61   2048 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@0
0,003874    0,56    255 rule Cell2Subtree@applier
0,003748    0,54   2815 TT!Cell::getNode() : Tuple
0,003501    0,50   2048 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@0@0
0,003118    0,45   2560 rule Cell2Assignment@matcher
0,002839    0,41    256 rule Row2Leaf@applier
0,002664    0,38   2560 rule Row2Leaf@applier@0
0,002371    0,34   2560 rule Cell2Subtree@matcher
0,002108    0,30   9216 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@0@0@0
0,001689    0,24    255 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple
0,001606    0,23   1793 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@1
0,001535    0,22    512 rule Cell2Assignment@applier
0,001383    0,20    255 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@0
0,001256    0,18   1793 TT!TruthTable::tree: Tuple@0
0,000070    0,01      1 rule TruthTable2BDD@applier
0,000028    0,00      1 TT!TruthTable::tree: Tuple
0,000015    0,00      8 rule InputPort2InputPort@applier
0,000004    0,00      2 rule OutputPort2OutputPort@applier
0,000000    0,00      1 static EMFTVM!ExecEnv::init() : Object
0,000000    0,00      1 static EMFTVM!ExecEnv::main() : Object
Timing data:
        Loading finished at 0,007563 seconds (duration: 0,007563 seconds)
        Matching finished at 0,660266 seconds (duration: 0,652703 seconds)
        Applying finished at 0,702967 seconds (duration: 0,042700 seconds)
        Post-applying finished at 0,703010 seconds (duration: 0,000043 seconds)
        Recursive stage finished at 0,703021 seconds (duration: 0,000011 seconds)
        Execution finished at 0,703755 seconds (duration: 0,000745 seconds)
```

Listing 7: ATL/EMFTVM Profiler output 2

```
Duration (sec.) Duration (%)    Invocations     Operation
0,580876   87,77 620415 static EMFTVM!ExecEnv::findCell(cell: TT!Cell, tree: Tuple) : Tuple
0,009810    1,48  14848 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@0@0
0,009739    1,47  14848 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@1@0
0,009731    1,47    255 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple
0,004357    0,66   2048 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@0
0,004235    0,64   2048 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@1
0,003370    0,51   2048 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@0@0
0,003306    0,50   2560 TT!Cell::node: Tuple
0,003206    0,48   2560 rule Cell2Subtree@matcher
0,003112    0,47    255 rule Cell2Subtree@applier
0,002909    0,44   2560 rule Row2Leaf@applier@0
0,002857    0,43    256 rule Row2Leaf@applier
0,002754    0,42   2560 rule Cell2Assignment@matcher
0,002142    0,32   9216 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@0@0@0
0,001729    0,26    255 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple
0,001663    0,25    512 rule Cell2Assignment@applier
0,001626    0,25   1793 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@1
0,001321    0,20   1793 TT!TruthTable::tree: Tuple@0
0,001057    0,16    255 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@0
0,000074    0,01      1 rule TruthTable2BDD@applier
0,000024    0,00      1 TT!TruthTable::tree: Tuple
0,000017    0,00      2 rule OutputPort2OutputPort@applier
0,000015    0,00      8 rule InputPort2InputPort@applier
0,000000    0,00      1 static EMFTVM!ExecEnv::init() : Object
0,000000    0,00      1 static EMFTVM!ExecEnv::main() : Object
Timing data:
        Loading finished at 0,007919 seconds (duration: 0,007919 seconds)
        Matching finished at 0,657334 seconds (duration: 0,649415 seconds)
        Applying finished at 0,669723 seconds (duration: 0,012390 seconds)
        Post-applying finished at 0,669771 seconds (duration: 0,000048 seconds)
        Recursive stage finished at 0,669782 seconds (duration: 0,000011 seconds)
        Execution finished at 0,670487 seconds (duration: 0,000716 seconds)
```

Listing 8: ATL/EMFTVM Profiler output 3

```
Duration (sec.) Duration (%)    Invocations     Operation
0,009697   12,37  14848 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@0@0
0,009125   11,64    255 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple
0,008714   11,12  14848 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@1@0
0,004266    5,44   2048 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@0
0,004040    5,15   2048 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@1
0,003325    4,24   2048 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@0@0
0,003175    4,05    255 rule Cell2Subtree@applier
0,003014    3,84   2560 rule Cell2Assignment@matcher
0,002862    3,65    256 rule Row2Leaf@applier
0,002754    3,51   2560 rule Row2Leaf@applier@0
0,002372    3,03   2560 TT!Cell::node: Tuple
0,002005    2,56   9216 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@0@0@0
0,001887    2,41    255 static EMFTVM!ExecEnv::collectAllNodes(tree: Tuple) : Sequence
0,001874    2,39   2560 rule Cell2Subtree@matcher
0,001747    2,23    512 rule Cell2Assignment@applier
0,001722    2,20    255 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple
0,001419    1,81   1793 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@1
0,001202    1,53   1793 TT!TruthTable::tree: Tuple@0
0,000994    1,27    255 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@0
0,000444    0,57      1 TT!TruthTable::nodesByCell: Map
0,000103    0,13    255 TT!TruthTable::nodesByCell: Map@0
0,000056    0,07      1 rule TruthTable2BDD@applier
0,000024    0,03      1 TT!TruthTable::tree: Tuple
0,000014    0,02      8 rule InputPort2InputPort@applier
0,000003    0,00      2 rule OutputPort2OutputPort@applier
0,000000    0,00      1 static EMFTVM!ExecEnv::init() : Object
0,000000    0,00      1 static EMFTVM!ExecEnv::main() : Object
Timing data:
        Loading finished at 0,007728 seconds (duration: 0,007728 seconds)
        Matching finished at 0,072620 seconds (duration: 0,064892 seconds)
        Applying finished at 0,086084 seconds (duration: 0,013464 seconds)
        Post-applying finished at 0,086153 seconds (duration: 0,000068 seconds)
        Recursive stage finished at 0,086164 seconds (duration: 0,000011 seconds)
        Execution finished at 0,086927 seconds (duration: 0,000775 seconds)
```

Listing 9: ATL/EMFTVM Profiler output 4

```
Duration (sec.) Duration (%)    Invocations     Operation
0,011395   17,15    255 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple
0,003796    5,71   2048 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@0@0
0,003340    5,03    255 rule Cell2Subtree@applier
0,003258    4,90   2560 rule Cell2Assignment@matcher
0,002754    4,15    256 TT!Row::trueCellsByPort: Map
0,002460    3,70   2048 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@1
0,002439    3,67    256 rule Row2Leaf@applier
0,002417    3,64   2560 rule Row2Leaf@applier@0
0,002390    3,60   9216 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@0@0@0
0,002278    3,43   2560 TT!Cell::node: Tuple
0,002226    3,35   2048 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple@0
0,002001    3,01    255 static EMFTVM!ExecEnv::collectAllNodes(tree: Tuple) : Sequence
0,001973    2,97    256 TT!Row::falseCellsByPort: Map
0,001840    2,77    255 static EMFTVM!ExecEnv::getPartition(rows: Sequence, port: TT!Port) : Tuple
0,001804    2,71   2560 rule Cell2Subtree@matcher
0,001580    2,38    512 rule Cell2Assignment@applier
0,001364    2,05   1793 TT!TruthTable::tree: Tuple@0
0,001340    2,02   1793 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@1
0,001212    1,82   2560 TT!Row::trueCellsByPort: Map@0
0,001167    1,76    255 static EMFTVM!ExecEnv::getTree(rows: Sequence, usablePorts: Sequence) : Tuple@0
0,001120    1,69   2560 TT!Row::falseCellsByPort: Map@0
0,000572    0,86   1285 TT!Row::trueCellsByPort: Map@1
0,000502    0,76   1275 TT!Row::falseCellsByPort: Map@1
0,000422    0,63      1 TT!TruthTable::nodesByCell: Map
0,000098    0,15    255 TT!TruthTable::nodesByCell: Map@0
0,000056    0,08      1 rule TruthTable2BDD@applier
0,000028    0,04      1 TT!TruthTable::tree: Tuple
0,000012    0,02      8 rule InputPort2InputPort@applier
0,000004    0,01      2 rule OutputPort2OutputPort@applier
0,000000    0,00      1 static EMFTVM!ExecEnv::init() : Object
0,000000    0,00      1 static EMFTVM!ExecEnv::main() : Object
Timing data:
        Loading finished at 0,008821 seconds (duration: 0,008821 seconds)
        Matching finished at 0,063594 seconds (duration: 0,054773 seconds)
        Applying finished at 0,075310 seconds (duration: 0,011716 seconds)
        Post-applying finished at 0,075353 seconds (duration: 0,000043 seconds)
        Recursive stage finished at 0,075369 seconds (duration: 0,000016 seconds)
        Execution finished at 0,076570 seconds (duration: 0,001217 seconds)
```

Listing 10: ATL/EMFTVM Profiler output 5