Sierpinski Triangles in MOMENT2-GT

Artur Boronat and Reiko Heckel

Computer Science Deparment, University of Leicester, UK.
ab373@le.ac.uk reiko@mcs.le.ac.uk

1 Introduction

MOMENT2 is an algebraic model management framework that permits manipulating models in the Eclipse Modeling Framework (EMF) [1]. Our goal consists in using OMG standards, such as Meta-Object Facility (MOF) [2], Object Constraint Language [3] and Query/View/Transformation (QVT) [4], as a clean interface between formal methods and model-based industrial tools that permits taking the best benefit from both at the lowest cost. In particular, we deal with Rewriting Logic (RL for short) [5] and Graph Transformations [6] on the one hand, and with Model-Driven Technology on the other hand.

MOMENT2-GT constitutes an extension of MOMENT2 that provides support for graph-based transformations. The syntax for defining graph patterns is directly based on the QVT Relations language. We present the solution of a case study of the AGTIVE contest by using it: the construction of a Sierpinski Triangle.

The paper is structured as follows: Section 2 introduces the main features of MOMENT2-GT; Section 3 describes the solution to AGTIVE case study 2; Section 4 provides a brief discussion of the experience; and Section 5 finally summarizes the main outcomes.

2 MOMENT2-GT Features

MOMENT2 constitutes an efficient framework that enhances the experimentation of practical graph-based model transformation language features: MOF-based concepts, MOF introspection and structural reflection, graph pattern matching, negative application conditions, conditional production rules, OCL support, textual concrete syntax of transformation definition languages; and formal verification techniques: checking invariants through reachability analysis, model checking, and theorem proving among others. Indeed, MOMENT2 constitutes an abstraction on top of RL that permits dealing with MOF-based models and their consistent manipulation (keeping the graph structure, for example). The framework is entirely written in Maude [7], an efficient term rewriting system that provides support for RL. MOMENT2 has been designed in order to take advantage of Maude's formal analysis techniques. MOMENT2-GT reuses its core functionality for graph transformation purposes:

- MOMENT2-GT provides a textual QVT-based language for defining production rules, where graph patterns are defined as in the QVT Relations language [4] and attribute values can be queried and manipulated by means of OCL expressions.
- Support for textual concrete syntax. The grammar of the transformation language itself is defined as an algebraic signature. A transformation definition is a term, which is compiled into a Maude module. Therefore, our framework can be extended with new concepts and syntax very easily.
- Definition of production rules either as equations, to specify functional behavior, or as rewrite rules, to specify non-deterministic and possibly concurrent behavior. A MOMENT2-GT transformation definition can be viewed as a rewrite theory.
- EMF is reused as a metamodeling front-end so that graphical editors and persistence can be customized easily. At the time of writing this paper, the EMF concepts that are supported are: class definition with attributes and references; abstract classes; class specialization; multiplicity features on references (but only one-bounded attributes), including ordering and uniqueness; opposite references; and containment references. Since MOMENT2-GT is a self-contained framework in Maude, adopting other metamodeling front-ends such as Microsoft DSL Tools [8] is straightforward.
- In MOMENT2-GT, edges are updated automatically when they are defined as opposite references in the metamodel. On the one hand, if one of two opposite references is removed, the other one is also removed. On the other hand, if one of them is updated, the other one is also updated. In addition, an object in an EMF model can be the root of a subtree (taking into account containment references). When an object of this kind is deleted, all objects in the subtree are deleted and dangling edges are removed as well. These actions are carried out automatically by MOMENT2-GT in what we call *consistency checking*. MOMENT2-GT permits disabling it when there are no dangling edges. We study the performance of the Sierpinski Triangle transformation by taking into account both possibilities.

3 AGTIVE Case study 3: Sierpinski Triangle

In this Section, we describe three solutions for the case study. For each of them we take into account two variants: (1) when the transformation can produce dangling edges, and (2) when the transformation is designed to avoid such dangling edges so that MOMENT2-GT can be used without consistency checking. For each solution, we present both the metamodel that is used to represent Sierpinski triangles and the transformation definition.

3.1 Solution 1: Many-bounded Reference

Many references should be avoided in graph patterns when using MOMENT2-GT. This solution is intended to study the performance of MOMENT2-GT with many references.

Many-bounded references are represented by means of OCL collections in MOMENT2. The internal axiomatization of OCL collection types in MOMENT2-GT is intended to capture ordering and uniqueness features that are inherent to OCL collection types. However, due to the combination of associativity and commutativity attributes in the axiomatization, generating term-based patterns from graph-based patterns is not straightforward at the moment. We leave this minor disadvantage of our tool for future work. However, in most of the cases where we find a many-bounded reference in the metamodel, we can define an opposite one-bounded reference. Therefore, to avoid this problem, the opposite one-bounded reference should be the one that is taken into account in the graph pattern. In the case of many-to-many associations, there is no efficient solution yet.

The metamodel for defining Sierpinski triangles in this solution is shown in Fig. 1. In a Sierpinski model, there is a root object that contains the rest of the nodes. In this container object, we keep the current number of nodes (*nodesNumber* attribute) and edges (*edgesNumber* attribute) of the triangle in a given generation step, the total number of nodes (*totalStepNodesNumber* attribute) and edges (*totalStepEdgesNumber* attribute) of the triangle in a specific step, the current generation step (*currentStep* attribute), the final generation step in which we should stop (*currentStep* attribute), a boolean flag indicating if have already reached the final generation step (*stop* attribute). Each node may have up to two references to other nodes. In addition, each node is tagged with the step number in which it has been generated (*step* attribute).

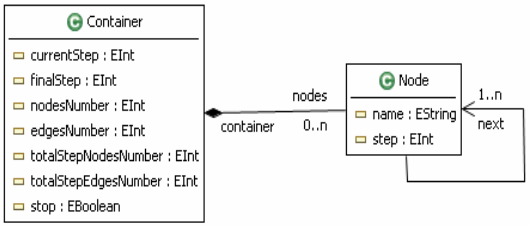

Fig. 1. Sierpinski Metamodel: solution 1.

The transformation definition is constituted by three rules: the first rule performs the first generation step in an adhoc way, the second rule is partially shown in Fig. 2 and is discussed below, and the third rule ensures that a generation step has been completed and starts the next generation step if needed.

In this transformation definition, whenever a triangle is constituted by three nodes, all of them tagged with the same step number, it cannot be rewritten any more. Taking into account the graph pattern of the LHS of the production rule in the figure, this rule can be applied when the following properties hold: the *stop* flag is not activated in the container node, $n < m$ and $m \leqslant k - 1$, where

k is the current step in the container object. Each time this rule is applied, we increase the current number of nodes and edges in the container node.

In the third rule, in order to ensure that a generation step is completed, we check that the current node number in the container node coincides with the total node number for this generation step. Then, we update the total number of nodes for the next generation step and we increase the current generation step in the container node. In case that the next generation step is the final one, the flag *stop* is activated and no more generation steps will be performed.

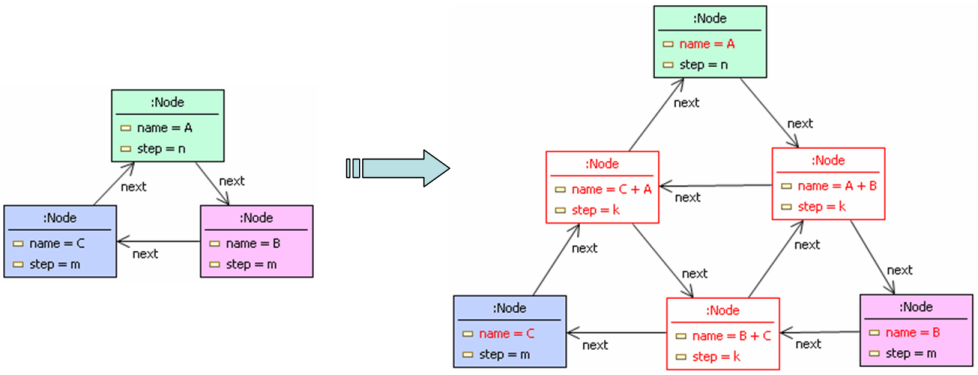

Fig. 2. Solution 1: Part of the Production Rule *split_triangle*.

3.2 Solution 2: One-bounded Reference

This solution is equivalent to the previous one but avoids the many-bounded reference *next*. The *Node* class in the metamodel, shown in Fig. 3, has two one-bounded references, *nextInner* and *nextOuter*. In this solution, a triangle that is constituted by three *nextInner* edges is an inner triangle and it cannot be rewritten any more. However, we have kept the same conditions of the first solution in order to enhance the performance comparison between both approaches.

In this tranformation, a triangle can be constituted by *nextInner* and *nextOuter* edges. The case where a triangle is constituted only by *nextOuter* edges is already taken into account in the initial production rule. A triangle that is constituted by *nextInner* edges should not be rewritten and, therefore, a production rule is not needed. We need two rules for considering the other cases: (1) a triangle is constituted by one *nextInner* edge and by two *nextOuter* edges (the production rule is partially shown in Fig. 4); and (2) a triangle is constituted by two *nextInner* edges and by one *nextOuter* edge (the production rule is partially shown in Fig. 5).

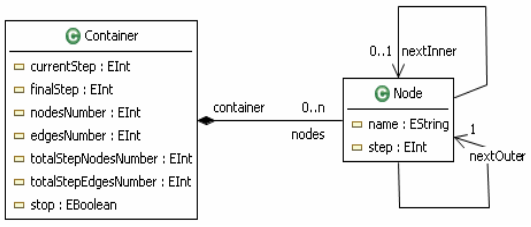

Fig. 3. Sierpinski Metamodel: solution 2.

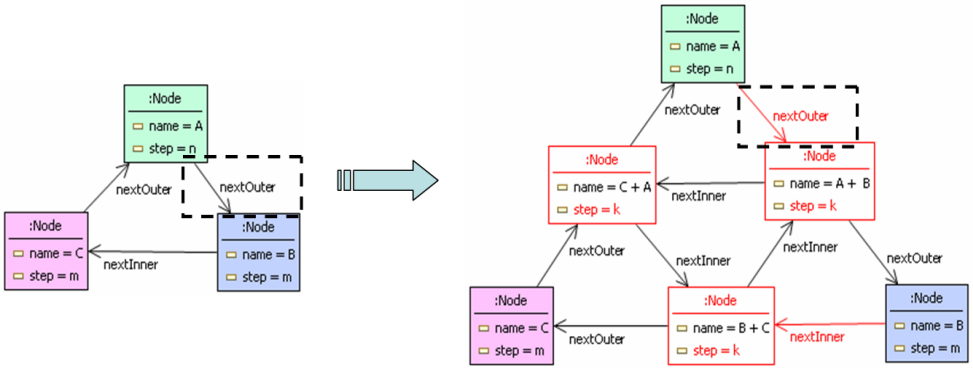

Fig. 4. Solution 2: Part of the Production Rule *split_triangleOuter2Inner1*.

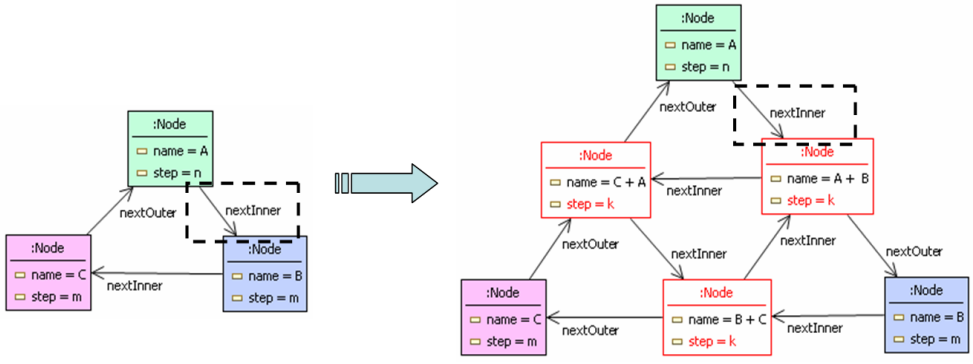

Fig. 5. Solution 2: Part of the Production Rule *split_triangleOuter1Inner2*.

3.3 Solution 3: Hyperedge Replacement

In this case study, we have based on the transformation that is provided in [9], where only two rules are needed: one for creating triangles and another one for checking the completion of a generation step.

In the metamodel of this solution, shown in Fig. 6, there are two types of nodes: *CentralNode* and *Node*. In the LHS of the production rule, shown in Fig. 7, a triangle involves one more node that constitutes its center (*CentralNode* instance) and that has an edge to each one of the other three nodes (*Node* instances). The triangle that is produced by this rule can be viewed as four triangles, where the inner one cannot be rewritten any more.

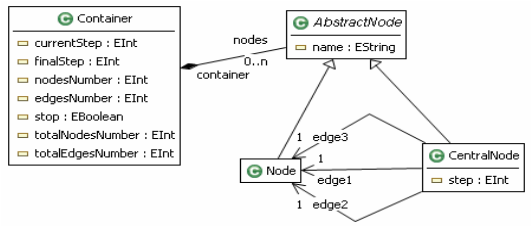

Fig. 6. Sierpinski Metamodel: solution 3.

This solution is also interesting because the transformation definition has only two rules but two more nodes are generated in each rule application. Therefore, we study how Maude's associativity-commutativity matching algorithm behaves in this scenario.

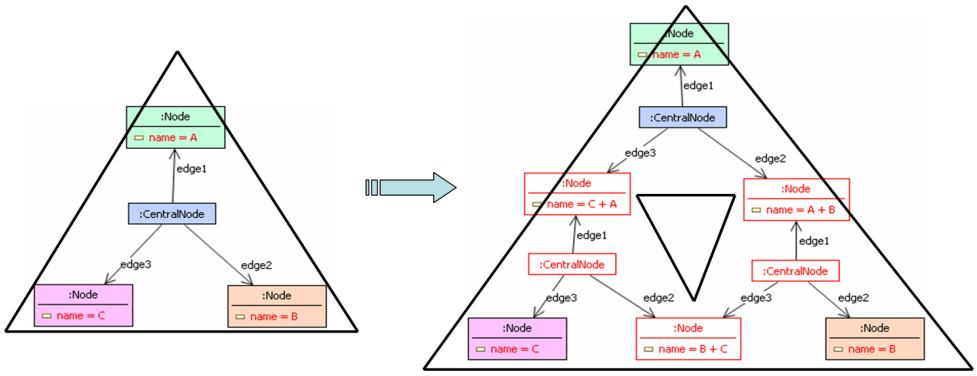

Fig. 7. Solution 3: Part of the Production Rule *split_step*.

4 Discussion

Three solutions for the Sierpinski Triangle case study, which have been discussed above, have been tested by following two variants: using the consistency checking that MOMENT2-GT provides and without using it. In the second variant, we have slightly modified the corresponding transformations in order to avoid dangling edges. The experiments have been performed on a Core DUO 2Ghz with 2Gb RAM, using Ubuntu 7.04.

In Table 1, we provide the resulting time measurements (in ms) of the experiments. These time measurements indicate the actual time that Maude has taken to rewrite the input model (already represented as a term) into its canonical form (resulting output term).

generation step	solution 1 (consistency)	solution 1	solution 2 (consistency)	solution 2	solution 3 (consistency)	solution 3
1	83.8	45.0	72.7	34.4	112.6	42.1
2	334.9	162.0	273.8	120.4	423.6	144.9
3	1,833.4	1,041.3	846.9	386.4	1,362.8	453.3
4	96,757.0	88,606.5	2,900.1	1,292.2	4,667.2	1,475.7
5	-	-	14,020.8	6,768.0	28,578.0	4,928.7
6	-	-	138,953.0	81,892.0	276,148.2	22,284.0
7	-	-	-	-	-	172,868.0
8	-	-	-	-	-	1,511,668.0

Table 1. Experiment results.

In the table, the third solution is the most efficient, despite being the one that grows faster in size. This is due to the fact that there is only one production rule to split the triangle, which, in addition, has no conditions. On the other hand, the variant where consistency checking is disabled in MOMENT2-GT is always faster than its respective variant with consistency checking.

In addition to the time measurements provided in Table 1, the global time that is used to transform an input model includes the serialization of the input model and the metamodel as terms, the compilation of the transformation definition into a Maude module, and the definition of the output EMF model from the resulting term. With these experiments, we have detected that the last process is also very time-consuming. For example, for parsing the Sierpinski triangle that is produced in the seventh generation step (more than 3,282 nodes in the third solution) takes 228608 ms to be parsed. This is due to the fact that we are using regular expressions with backtracking to parse the output term. We have to improve this process in future versions. As discussed above, many-bounded references is another pending task to be improved in future work.

5 Conclusions

We have provided three solutions for the Sierpinski Triangle case study of the AGTIVE contest by using MOMENT2-GT. MOMENT2-GT constitutes an efficient graph-rewriting engine that is powered by Maude, that is integrated into the EMF and that follows OMG standards: MOF, OCL and QVT.

In the current version of MOMENT2-GT we are not making extensive use of Maude features to improve efficiency, like memoization and strategies, yet. However, these experiments show that MOMENT2-GT already constitutes a realistic approach for graph rewriting. This is due to the efficient AC-matching algorithm that is implemented in Maude [10]. In addition, MOMENT2-GT also benefits from formal analysis techniques as discussed in the UML2CSP case study.

Acknowledgments. This work has been partially sponsored by the Spanish Government under the National Program for Research, Development and Innovation, DYNAMICA Project TIC 2003-07804-C05-01 and META Project TIN2006-15175-C05, and by the project SENSORIA, IST-2005-016004.

We are grateful to prof. José Meseguer for his help, advice and corrections in the development of the MOMENT2 kernel.

References

1. Eclipse Organization: The eclipse modeling framework (2007) `http://www.eclipse.org/emf/`.
2. Object Management Group: Meta Object Facility (MOF) 2.0 Core Specification (ptc/06-01-01) (2006) `http://www.omg.org/cgi-bin/doc?formal/2006-01-01`.
3. Object Management Group: OCL 2.0 Specification (2006) `http://www.omg.org/cgi-bin/doc?formal/2006-05-01`.
4. Object Management Group: MOF 2.0 QVT final adopted specification (ptc/05-11-01) (2005) `http://www.omg.org/cgi-bin/doc?ptc/2005-11-01`.
5. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science **96**(1) (1992) 73–155
6. Rozenberg, G., ed.: The Handbook of Graph Grammars, volume 3. World Scientific (1999)
7. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude: a High-Performance Logical Framework. Springer (2007)
8. Microsoft Corp.: Domain-specific language tools (2007) `http://msdn2.microsoft.com/en-us/vstudio/aa718368.aspx`.
9. Kreowski, H.J., Klempien-Hinrichs, R., Kuske, S.: Some essentials of graph transformation. In Esik, Z., Martin-Vide, C., Mitrana, V., eds.: Recent Advances in Formal Languages and Applications. Vol. 25 of Studies in Computational Intelligence, Berlin Heidelberg New York, USA, Springer (2006) 229–254
10. Eker, S.: Associative-commutative rewriting on large terms. In Nieuwenhuis, R., ed.: Rewriting Techniques and Applications (RTA 2003). Number 2706 in Lecture Notes in Computer Science, Springer-Verlag (2003) 14–29