Sierpinski Triangle
for the AGTIVE 2007 Tool Contest

Rubino Geiß Christoph Mallon Moritz Kroll

July 15, 2007

1 Introduction

The goal of this case is to construct a Sierpinski triangle. The following is a quotation from Wikipedia:

> The Sierpinski triangle, also called the Sierpinski gasket, is a fractal named after Waclaw Sierpinski who described it in 1915. Originally constructed as a curve, this is one of the basic examples of self-similar sets, i.e. it is a mathematically generated pattern that can be reproducible at any magnification or reduction.
>
> An algorithm for obtaining arbitrarily close approximations to the Sierpinski triangle is as follows:
>
> The evolution of the Sierpinski triangle
>
> 1. Start with any triangle in a plane (any closed, bounded region in the plane will actually work). The canonical Sierpinski triangle uses an equilateral triangle with a base parallel to the horizontal axis.
> 2. Shrink the triangle by $\frac{1}{2}$, make two copies, and position the three shrunken triangles so that each triangle touches the two other triangles at a corner.
> 3. Repeat step 2 with each of the smaller triangles.
>
> Note that this infinite process is not dependent upon the starting shape being a triangle—it is just clearer that way.

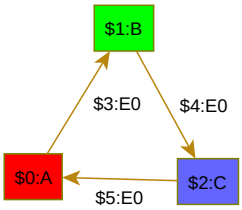


Figure 1: Initial Sierpinski triangle

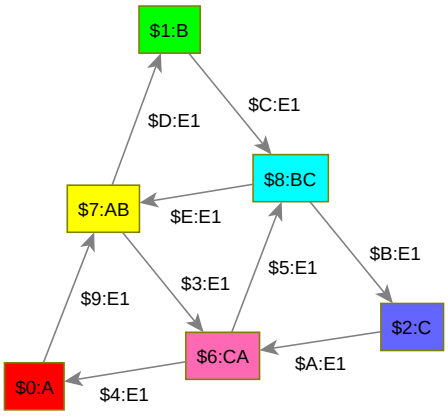


Figure 2: First generation of the Sierpinski triangle

2 Sierpinski Triangle as a Graph

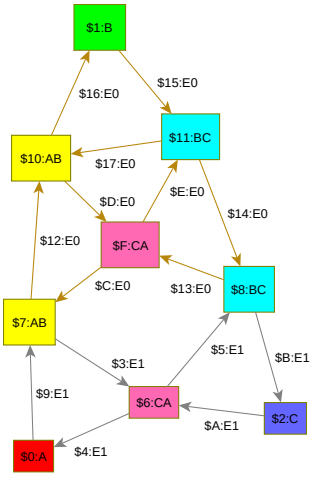
For the proposed benchmark we have to represent the Sierpinski triangle as a (mathematical) graph[1] and restate the rules of construction. As a first step we have to represent a triangle as a graph. We achieve this in the obvious way: Three nodes connected by three edges (see Figure 1).

Next we define an elementary "Sierpinski step": On every edge of a triangle we place some node. These new nodes are connected by edges (see Figure 2 in comparison to 1). This forms a triangle consisting of four smaller triangles. The inner of the four triangles is considered "dead" and no further "Sierpinski steps" will be performed there. The other three triangles are candidates for further steps. If all steps are done for a certain graph (without reconsidering newly created triangles) we call this a generation (Figure 1 shows generation zero, Figure 2 shows generation one, and Figure 3(c) shows generation two). We require that a generation is completed before any transformation for the next generation takes place.

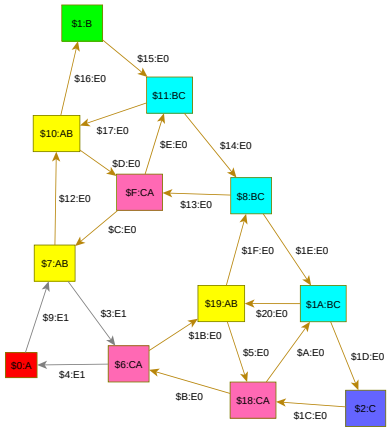
The figures are only for the purpose of illustration. We used our visualization tool yComp together with GrGen.NET to obtain these figures.

[1] A mathematical graph has no immediate representation on a two dimensional plane—though embeddings may be computed.

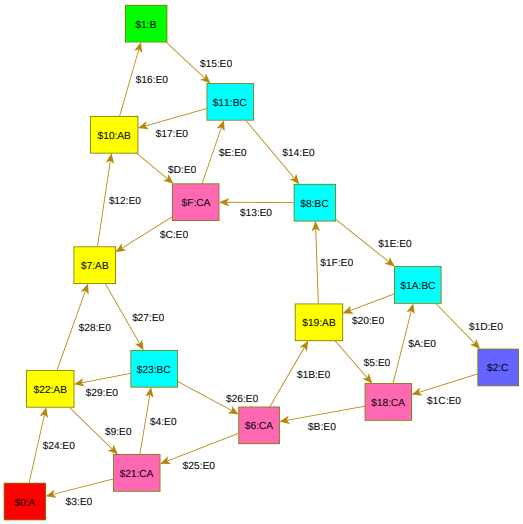
Therefore all nodes and edges have types (identifiers behind the colon) and edges are directed. The edges and nodes have unique identifiers with $-prefix (before the colon). The nodes are colored according to their type. It is not necessary to be conformant to any of these details in your implementation. We merely require you to produce the same graph structure (regardless its element types and edge directions).



(a) First rewrite



(b) Second rewrite



(c) Third rewrite

Figure 3: Second generation of the Sierpinski triangle

3 Goals of the Case

This case is pretty easy to build: It uses only small pattern graphs, simple graph rewrites, and only a few rules. The generated graphs get huge[2] fast. So it tests the ability of a tool to represent large graphs efficiently, and to perform simple rewrites, fast. To have a reference for testing your implementation, Table 1 shows the number of nodes and edges for different generations.

With growing numbers of generations (iterations) we can get samples for memory usage and computation time. Last but not least we can see how the tools are capable of enabling adequate meta models, rule sets, and rule applications.

Table 1: Number of nodes/edges for a generation

generation	# nodes	# edges
0	3	3
1	6	9
2	15	27
3	42	81
4	123	243
5	366	729
6	1,095	2,187
7	3,282	6,561
8	9,843	19,683
9	29,526	59,049
10	88,575	177,147
11	265,722	531,441
12	797,163	1,594,323
13	2,391,486	4,782,969
14	7,174,455	14,348,907

[2]The number of nodes/edges is actually about 3^n with n being the number of generations.