A GrGen.NET solution of the AntWorld case for the GraBaTs 2008 Contest

Sebastian Buchwald and Moritz Kroll

July 11, 2008

1 Introduction

The challenge of the AntWorld case is to simulate an expanding ant colony using graph transformation (see [1] for details).

2 What is GrGen.NET?

GrGen.NET is an application domain neutral graph rewrite system developed at the IPD Goos of Universität Karlsruhe (TH), Germany [2]. The feature highlights of GrGen.NET regarding practical relevance are:

Fully Featured Meta Model: GrGen.NET uses attributed and typed multigraphs with multiple inheritance on node/edge types.

Expressive Rules, Fast Execution: The expressive and easy to learn rule specification language allows straightforward formulation of even complex problems while remaining one of the fastest automatic graph rewrite systems known to us (cf. [3]).

Programmed Rule Application: GrGen.NET has a high-level interface to programmed rule application: Graph Rewrite Sequences (GRS).

Graphical Debugging: GrShell, GrGen.NET's command line shell, offers interactive execution of rules, visualising the current graph and the rewrite process. This way you can see what the graph looks like at a given step of a complex transformation and develop the next step accordingly. Or you can debug your rules.

3 Modelling the Ant World

The area grid consists of *GridNode*s connected by directed *GridEdge*s with the special *AntHill GridNode* as the center of each ant's life. With each *GridNode* we associate an integer amount of $food$ and ant *pheromones*. At this point we had to consider, what additional information we would need

later on. Firstly the ants always have to know the direction back to the ant hill. This can be realized by marking the *GridEdge*s, which connect different circles of the grid, as *PathToHill* edges directed towards the hill. Secondly we have to recognize the main axes of the grid in order to expand the grid according to the specification. We can solve this by using the *GridCornerNode* subtype for the nodes on the axes. We fix the remaining *GridEdge*s, i.e. the circle edges, to always use the same circular direction making it easier to build the next circle afterwards. The specification says that every 10th created *GridNode* shall get 100 units of food. We handle this by adding a *foodCountdown* attribute to the unique *AntHill* node type and initializing it with 10.

The ants are modelled as nodes with a boolean attribute *hasFood* indicating whether the ant currently carries food. The current location of each *Ant* is given by an *AntPosition* edge pointing from the *Ant* to the according *GridNode*. We manage the ants in a singly-linked list using *NextAnt* edges to ensure that each ant moves exactly once in a round. The list can be traversed using the *GetNextAnt* rule.

4 Building up the Grid

During initialization of the simulation (**InitWorld** rule) we create a 4x4 grid of empty *GridNode*s with an *AntHill* in the center, as described in the specification. The initial 8 *Ant*s will start swarming from the *AntHill*.

To keep the illusion of an endless world in the ants' minds, we have to make sure they never fall off the edge of the world. So at the end of each round, we check, whether an *Ant* has reached the outer circle (indicated by the **ReachedEndOfWorld** rule). If we find such an *Ant*, we construct a new circle around the grid by extending each node on the current outer circle, starting at the *Ant*'s position. For each step we have to distinguish whether we extend a normal *GridNode* receiving one child (**...NotAtCorner** rules) or a *GridCornerNode* receiving three children (**...AtCorner** rules). The new outer circle is built up in three phases: The first phase just extends the *GridNode* at the *Ant*'s position (**GrowWorldFirst...**), the second phase extends all following *GridNode*s along the old outer circle and connects their children to the corresponding predecessor (**GrowWorldNext...**), and the last phase closes the new circle (**GrowWorldEnd**). For each created child we decrement the *foodCountdown* attribute of the *AntHill* and place 100 food units on the new child, if the counter reaches zero. Inside the **GrowWorld...** rules we test this by calling the **GrowFoodIfEqual** rule, which places food at the given *GridNode* when the *foodCountdown* attribute equals some parameter. Providing this parameter is required because of the corner nodes receiving three child nodes at once, making it necessary to call the **GrowFoodIfEqual** rule three times. Although GrGen.NET already

supports values next to graph elements as parameters, the current beta does not allow integer constants in GRS execution statements (exec). Thus, we had to create three special nodes with an attribute always being 0, -1, and -2, respectively (the *GammelFix* types; "Gammel" can be translated as scruffy).

The graph rewrite sequence for the grid extension is:

```
(cur:GridNode)=ReachedEndOfWorld &&
(
    (cur, curOuter:GridNode)=GrowWorldFirstNotAtCorner(cur) ||
    (cur, curOuter         )=GrowWorldFirstAtCorner(cur)
) &&
(
    (cur, curOuter)=GrowWorldNextNotAtCorner(cur, curOuter) ||
    (cur, curOuter)=GrowWorldNextAtCorner   (cur, curOuter)
)* &&
GrowWorldEnd(cur, curOuter)
```

We also tried an alternative implementation which models the border *GridNode*s as special type. It sped up checking whether an *Ant* reached the outer circle at the expense of retyping the special nodes to normal *GridNode*s while expanding the grid. Unfortunately, our empirical studies showed, that the running time was slightly higher.

5 Controlling the Ants

Initially the eight ants search for food choosing their direction randomly (**SearchAimless**). To provide fair random selection we had to add a mechanism to GrGen.NET which randomly selects a given number of matches for a given rule. Otherwise it would have been necessary to fall back to the API making development and debugging less convenient.

When an *Ant* finds food, it takes one unit (**TakeFood**) and starts moving home dropping pheromones on its way (**GoHome**). If the *Ant* reaches the *AntHill*, it drops the food (**DropFood**) and follows a random pheromone trail back to an assumed food supply (**SearchAlongPheromones**).

The graph sequence handling all *Ant*s is:

```
curAnt:Ant=firstAnt &&
(
    (
        TakeFood(curAnt) | GoHome(curAnt) ||
        DropFood(curAnt) | ($[SearchAlongPheromones(curAnt)] ||
                            $[SearchAimless(curAnt)])
    ) &&
    (curAnt)=GetNextAnt(curAnt)
)*
```

6 A new Day in the Ant World

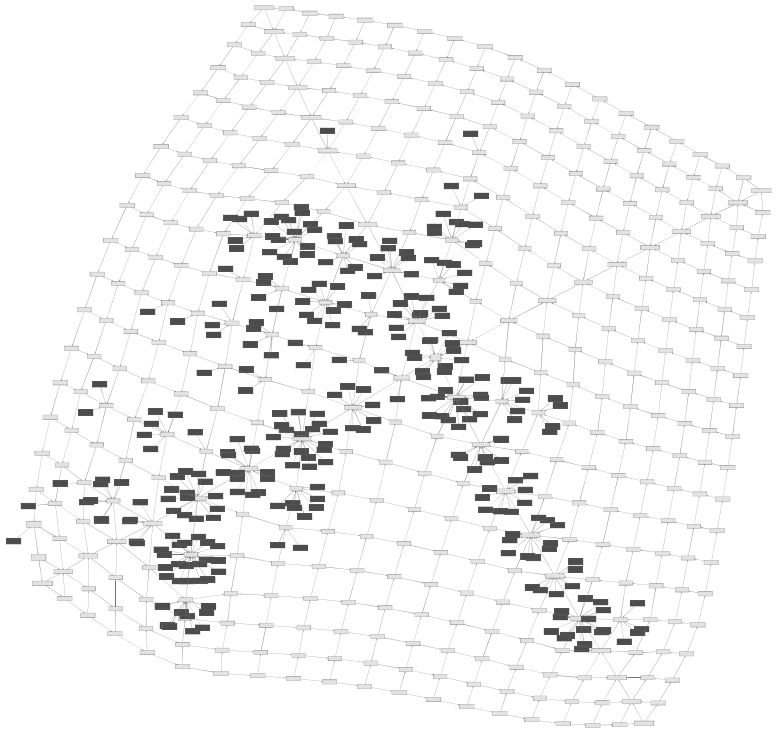
After each round the *AntHill* produces one new *Ant* for each food unit dropped by the *Ant*s (**Food2Ant**) and the pheromones evaporate a bit (**EvaporateWorld**). We implemented this by transforming food into *Ant*s as long as there is food left in the *AntHill* and by multiplying the *pheromones* attribute of each *GridNode* by 0.95:

```
1 (curAnt)=Food2Ant(curAnt)* | [EvaporateWorld]
```

7 An Optimizing Trick

The GRS execution statements (exec) on the RHS of the rules have a great advantage over the "xgrs" statements in GrShell scripts. The former are compiled while the latter are interpreted. So, by moving the main GRS from the GrShell script into the **doAntWorld** rule, we were able to reduce the running time by 27%. On the downside, it is not possible to set breakpoints at single rules anymore. You can only single step through the whole simulation. For this reason we left an outcommented version of the GRS in the GrShell script to let the interested reader use the full debugging features, which can also be used to animate the ant's life.

Figure 1: An AntWorld before grid extension after 61 rounds



8 Results

This test case needs a random number generator, which has not been specified, thus the comparability of the results is questionable. But our experiments with different initial random seeds suggest that the results are quite stable.

Table 1: Results of different rounds; running time in ms

rounds	circles	grid nodes	food created	ants	running time
125	21	1,765	17,400	3,300	409
250	56	12,545	125,200	12,679	4,609
500	126	63,505	634,800	35,185	34,654
750	198	156,817	1,568,000	61,387	104,060
1,000	275	302,501	3,024,800	87,856	232,005
1,250	353	498,437	4,984,200	116,261	460,956
1,500	430	739,601	7,395,800	144,039	788,603
1,750	515	1,060,901	10,608,800	172,749	1,215,982
2,000	593	1,406,597	14,065,800	203,582	1,758,737

Table 1 shows the number of grid nodes and ants and the computation time of this solution for different number of rounds always using the same initial random seed 42. The results were measured on an AMD Athlon XP 3000+ with 1 GB RAM (Windows XP SP2, .NET 2.0.50727.42, GrGen.NET 2.0 Beta 3). All measurements have been repeated 10-times. The result values are the 0.2 quantile of these values for each number of rounds. Note that the grid nodes also contain the *AntHill* in our model. The GrShell uses 66,320 kiB virtual memory after 250 rounds, and 90,156 kiB after 500 rounds.

9 Conclusion

With GrGen.NET it was possible to create a first running version of this simulation in a few hours starting with reading and understanding the specification, adding the random-selection feature included. The available features of GrGen.NET allowed us to write the graph model and most rules in a very intuitive way. Only the missing support of value parameters in execs was a bit cumbersome, which will be implemented in the official release of GrGen.NET 2.0.

References

[1] Zündorf, A.: AntWorld (2008) Submitted to GraBaTs 2008 Contest.

[2] Geiß, R.: GrGen. http://www.grgen.net (2008)

[3] Schürr, A., Nagl, M., Zündorf, A., eds.: Applications of Graph Transformation with Industrial Relevance, Proceedings of the Thrird International AGTIVE 2007 Symposium, Schlosshotel am Bergpark Wilhelmshöhe, Kassel, Germany. Volume 5088 of Lecture Notes in Computer Science (LNCS)., Heidelberg, Springer Verlag (2008)

A The Graph Model

```
1  node class GridNode
2  {
3      food:int;
4      pheromones:int;
5  }
6  node class GridCornerNode extends GridNode;
7  node class AntHill extends GridNode
8  {
9      foodCountdown:int = 10;
10 }
11 node class Ant
12 {
13     hasFood:boolean;
14 }
15
16 node class GammelFix
17 {
18     val:int;
19 }
20
21 node class Zero       extends GammelFix;
22 node class MinusOne   extends GammelFix { val = -1; }
23 node class MinusTwo   extends GammelFix { val = -2; }
24
25 edge class GridEdge connect GridNode[1] -> GridNode[1];
26 edge class PathToHill extends GridEdge;
27
28 edge class AntPosition;
29 edge class NextAnt;
```

B The Rule Specification

```
1  using AntWorld;
2
3  rule InitWorld : (Ant)
4  {
5      modify {
6          // Create all grid nodes
7
```

```
8         hill:AntHill;
9         a1:GridCornerNode; a2:GridNode;       a3:GridNode;       a4:GridCornerNode;
10        b1:GridNode;       b2:GridCornerNode; b3:GridCornerNode; b4:GridNode;
11        c1:GridNode;       c2:GridCornerNode; c3:GridCornerNode; c4:GridNode;
12        d1:GridCornerNode; d2:GridNode;       d3:GridNode;       d4:GridCornerNode;
13
14        // Connect first circle
15
16        hill <-:PathToHill- b2;
17        hill <-:PathToHill- b3;
18        hill <-:PathToHill- c3;
19        hill <-:PathToHill- c2;
20
21        b2 -:GridEdge-> b3 -:GridEdge-> c3 -:GridEdge-> c2 -:GridEdge -> b2;
22
23        // Connect second circle
24
25        b2 <-:PathToHill- b1;
26        b2 <-:PathToHill- a1;
27        b2 <-:PathToHill- a2;
28
29        b3 <-:PathToHill- a3;
30        b3 <-:PathToHill- a4;
31        b3 <-:PathToHill- b4;
32
33        c3 <-:PathToHill- c4;
34        c3 <-:PathToHill- d4;
35        c3 <-:PathToHill- d3;
36
37        c2 <-:PathToHill- d2;
38        c2 <-:PathToHill- d1;
39        c2 <-:PathToHill- c1;
40
41        a1 -:GridEdge-> a2 -:GridEdge-> a3 -:GridEdge-> a4;
42        a4 -:GridEdge-> b4 -:GridEdge-> c4 -:GridEdge-> d4;
43        d4 -:GridEdge-> d3 -:GridEdge-> d2 -:GridEdge-> d1;
44        d1 -:GridEdge-> c1 -:GridEdge-> b1 -:GridEdge-> a1;
45
46        // Create ants
47
48        queen:Ant -:AntPosition-> hill;
49        atta:Ant -:AntPosition-> hill;
50        flick:Ant -:AntPosition-> hill;
51        chuck:Ant -:AntPosition-> hill;
52        the:Ant -:AntPosition-> hill;
53        plant:Ant -:AntPosition-> hill;
54        chewap:Ant -:AntPosition-> hill;
55        cici:Ant -:AntPosition-> hill;
56
```

```
57          queen -:NextAnt-> atta -:NextAnt-> flick -:NextAnt-> chuck -:NextAnt-> the
58              -:NextAnt-> plant -:NextAnt-> chewap -:NextAnt-> cici;
59
60          // The ultimate GAMMEL FIX(tm)!!!!
61          :Zero; :MinusOne; :MinusTwo;
62
63          return (queen);
64      }
65  }
66
67  rule TakeFood(curAnt:Ant)
68  {
69      curAnt -:AntPosition-> n:GridNode\AntHill;
70      if { !curAnt.hasFood && n.food > 0; }
71
72      modify {
73          eval {
74              curAnt.hasFood = true;
75              n.food = n.food - 1;
76          }
77      }
78  }
79
80  rule GoHome(curAnt:Ant)
81  {
82      if { curAnt.hasFood; }
83      curAnt -oldPos:AntPosition-> old:GridNode -:PathToHill-> new:GridNode;
84
85      modify {
86          eval {
87              old.pheromones = old.pheromones + 1024;
88          }
89          delete(oldPos);
90          curAnt -:AntPosition-> new;
91      }
92  }
93
94  rule DropFood(curAnt:Ant)
95  {
96      if { curAnt.hasFood; }
97      curAnt -:AntPosition-> hill:AntHill;
98
99      modify {
100         eval {
101             curAnt.hasFood = false;
102             hill.food = hill.food + 1;
103         }
104     }
105 }
```

```
106
107 rule SearchAlongPheromones(curAnt:Ant)
108 {
109     curAnt -oldPos:AntPosition-> old:GridNode <-:PathToHill- new:GridNode;
110     if { new.pheromones > 9; }
111
112     modify {
113         delete(oldPos);
114         curAnt -:AntPosition-> new;
115     }
116 }
117
118 rule SearchAimless(curAnt:Ant)
119 {
120     curAnt -oldPos:AntPosition-> old:GridNode <-:GridEdge-> new:GridNode\AntHill;
121
122     modify {
123         delete(oldPos);
124         curAnt -:AntPosition-> new;
125     }
126 }
127
128 test ReachedEndOfWorld : (GridNode)
129 {
130     :Ant -:AntPosition-> n[prio=10000]:GridNode\AntHill;
131     negative { n <-:PathToHill-; }
132     return (n);
133 }
134
135 rule GrowFoodIfEqual(n:GridNode, val:GammelFix)
136 {
137     hill:AntHill;
138     if { hill.foodCountdown == val.val; }
139     modify {
140         eval {
141             n.food = n.food + 100;
142             hill.foodCountdown = hill.foodCountdown + 10;
143         }
144     }
145 }
146
147 rule GrowWorldFirstAtCorner(cur:GridCornerNode) : (GridNode, GridNode)
148 {
149     cur -:GridEdge\PathToHill-> next:GridNode;
150     hill:AntHill;
151
152     zero:Zero;
153     minusOne:MinusOne;
154     minusTwo:MinusTwo;
```

```
155
156     modify {
157         cur <-:PathToHill- outer1:GridNode;
158         cur <-:PathToHill- outer2:GridCornerNode;
159         cur <-:PathToHill- outer3:GridNode;
160         outer1 -:GridEdge-> outer2 -:GridEdge-> outer3;
161
162         eval {
163             hill.foodCountdown = hill.foodCountdown - 3;
164         }
165
166         return (next, outer3);
167
168         exec(  GrowFoodIfEqual(outer1, minusTwo)
169             || GrowFoodIfEqual(outer2, minusOne)
170             || GrowFoodIfEqual(outer3, zero));
171     }
172 }
173
174 rule GrowWorldFirstNotAtCorner(cur:GridNode\GridCornerNode) : (GridNode, GridNode)
175 {
176     cur -:GridEdge\PathToHill-> next:GridNode;
177     hill:AntHill;
178
179     zero:Zero;
180
181     modify {
182         cur <-:PathToHill- outer:GridNode;
183
184         eval {
185             hill.foodCountdown = hill.foodCountdown - 1;
186         }
187
188         return (next, outer);
189
190         exec(GrowFoodIfEqual(outer, zero));
191     }
192 }
193
194 rule GrowWorldNextAtCorner(cur:GridCornerNode, curOuter:GridNode)
195         : (GridNode, GridNode)
196 {
197     cur -:GridEdge\PathToHill-> next:GridNode;
198     negative { cur <-:PathToHill-; }
199     hill:AntHill;
200
201     zero:Zero;
202     minusOne:MinusOne;
203     minusTwo:MinusTwo;
```

```
204
205     modify {
206         cur <-:PathToHill- outer1:GridNode;
207         cur <-:PathToHill- outer2:GridCornerNode;
208         cur <-:PathToHill- outer3:GridNode;
209         curOuter -:GridEdge-> outer1 -:GridEdge-> outer2 -:GridEdge-> outer3;
210
211         eval {
212             hill.foodCountdown = hill.foodCountdown - 3;
213         }
214
215         return (next, outer3);
216         exec(  GrowFoodIfEqual(outer1, minusTwo)
217             || GrowFoodIfEqual(outer2, minusOne)
218             || GrowFoodIfEqual(outer3, zero));
219     }
220 }
221
222 rule GrowWorldNextNotAtCorner(cur:GridNode\GridCornerNode, curOuter:GridNode)
223         : (GridNode, GridNode)
224 {
225     cur -:GridEdge\PathToHill-> next:GridNode;
226     negative { cur <-:PathToHill-; }
227     hill:AntHill;
228
229     zero:Zero;
230
231     modify {
232         cur <-:PathToHill- outer:GridNode;
233         curOuter -:GridEdge-> outer;
234
235         eval {
236             hill.foodCountdown = hill.foodCountdown - 1;
237         }
238
239         return (next, outer);
240         exec(GrowFoodIfEqual(outer, zero));
241     }
242 }
243
244 rule GrowWorldEnd(cur:GridNode, curOuter:GridNode)
245 {
246     cur <-:PathToHill- nextOuter:GridNode;
247     modify {
248         curOuter -:GridEdge-> nextOuter;
249     }
250 }
251
252 test GetNextAnt(curAnt:Ant) : (Ant)
```

```
{
    curAnt -:NextAnt-> next:Ant;
    return (next);
}

rule Food2Ant(lastAnt:Ant) : (Ant)
{
    hill:AntHill;
    if { hill.food > 0; }

    modify {
        lastAnt -:NextAnt-> newAnt:Ant -:AntPosition-> hill;
        eval {
            hill.food = hill.food - 1;
        }
        return (newAnt);
    }
}

rule EvaporateWorld
{
    n:GridNode\AntHill;
    modify {
        eval {
            n.pheromones = (int) (n.pheromones * 0.95);
        }
    }
}

rule doAntWorld(firstAnt:Ant)
{
    modify {
        exec((curAnt:Ant=firstAnt &&
            ((
                TakeFood(curAnt) | GoHome(curAnt) ||
                DropFood(curAnt) | ($[SearchAlongPheromones(curAnt)] ||
                                   $[SearchAimless(curAnt)])
            ) && (curAnt)=GetNextAnt(curAnt))*
            | ((cur:GridNode)=ReachedEndOfWorld
                && ((cur, curOuter:GridNode)=GrowWorldFirstNotAtCorner(cur)
                    || (cur, curOuter)=GrowWorldFirstAtCorner(cur))
                && ((cur, curOuter)=GrowWorldNextNotAtCorner(cur, curOuter)
                    || (cur, curOuter)=GrowWorldNextAtCorner(cur, curOuter))*
                && GrowWorldEnd(cur, curOuter))
            | (curAnt)=Food2Ant(curAnt)*
            | [EvaporateWorld]
            )[250]);
    }
}
```

C The GrShell Script

```
1  new graph AntWorld
2
3  #debug set layout Organic
4  #dump set node Ant color red
5  #dump add node Ant infotag hasFood
6  #dump add edge NextAnt exclude
7  #dump add node GammelFix exclude
8
9  randomseed 42
10
11 xgrs (firstAnt)=InitWorld
12
13 xgrs doAntWorld(firstAnt)
14
15 #debug xgrs (curAnt=firstAnt && \
16 #    (( \
17 #        TakeFood(curAnt) | GoHome(curAnt) || \
18 #        DropFood(curAnt) | ($[SearchAlongPheromones(curAnt)] || \
19 #                            $[SearchAimless(curAnt)]) \
20 #    ) && (curAnt)=GetNextAnt(curAnt))* \
21 #    | ((cur)=ReachedEndOfWorld \
22 #        && ((cur, curOuter)=GrowWorldFirstNotAtCorner(cur) \
23 #            || (cur, curOuter)=GrowWorldFirstAtCorner(cur)) \
24 #        && ((cur, curOuter)=GrowWorldNextNotAtCorner(cur, curOuter) \
25 #            || (cur, curOuter)=GrowWorldNextAtCorner(cur, curOuter))* \
26 #        && GrowWorldEnd(cur, curOuter)) \
27 #    | (curAnt)=Food2Ant(curAnt)* \
28 #    | [EvaporateWorld] \
29 #    )[250]
30
31 show num nodes GridNode
32 show num nodes Ant
```