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 GridNodes connected by directed GridEdges 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 GridEdges, 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 GridEdges, 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 corresponding 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 GridNodes with an AntHill in the center, as described in the specification. The initial 8 Ants 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 GridNodes 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:

```plaintext
1 (cur:GridNode)=ReachedEndOfWorld &&
2 (
3 (cur, curOuter:GridNode)=GrowWorldFirstNotAtCorner(cur) ||
4 (cur, curOuter)=GrowWorldFirstAtCorner(cur)
5 ) &&
6 (
7 (cur, curOuter)=GrowWorldNextNotAtCorner(cur, curOuter) ||
8 (cur, curOuter)=GrowWorldNextAtCorner (cur, curOuter)
9 )* &&
10 GrowWorldEnd(cur, curOuter)
```

We also tried an alternative implementation which models the border GridNodes as special type. It sped up checking whether an Ant reached the outer circle at the expense of retyping the special nodes to normal GridNodes 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 Ants is:

```plaintext
1 curAnt:Ant=firstAnt &&
2 (
3  TakeFood(curAnt) | GoHome(curAnt) ||
4  DropFood(curAnt) | ($[SearchAlongPheromones(curAnt)]) ||
5  $[SearchAimless(curAnt)])
6 ) &&
7 (curAnt)=GetNextAnt(curAnt)
8 )*
6 A new Day in the Ant World

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

\[(\text{curAnt}) = \text{Food2Ant}(\text{curAnt}) \times [\text{EvaporateWorld}]\]

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

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

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

A The Graph Model

```
node class GridNode
{
  food: int;
  pheromones: int;
}
node class GridCornerNode extends GridNode;
node class AntHill extends GridNode
{
  foodCountdown: int = 10;
}
node class Ant
{
  hasFood: boolean;
}
node class GammelFix
{
  val: int;
}
node class Zero extends GammelFix;
node class MinusOne extends GammelFix { val = -1; }
node class MinusTwo extends GammelFix { val = -2; }

draw edge class GridEdge connect GridNode[1] <-> GridNode[1];
draw edge class PathToHill extends GridEdge;

draw edge class AntPosition;
draw edge class NextAnt;
```

B The Rule Specification

```
using AntWorld;

rule InitWorld : (Ant)
{
  modify {
    // Create all grid nodes
```
h1: AntHill; a1: GridCornerNode; a2: GridNode; a3: GridNode; a4: GridCornerNode;
b1: GridNode; b2: GridCornerNode; b3: GridCornerNode; b4: GridNode;
c1: GridNode; c2: GridCornerNode; c3: GridCornerNode; c4: GridNode;
d1: GridCornerNode; d2: GridNode; d3: GridNode; d4: GridCornerNode;

// Connect first circle

h1 <-: PathToHill- b2;
h1 <-: PathToHill- b3;
h1 <-: PathToHill- c3;
h1 <-: PathToHill- c2;

b2 -: GridEdge-> b3 -: GridEdge-> c3 -: GridEdge-> c2 -: GridEdge -> b2;

// Connect second circle

b2 <-: PathToHill- b1;
b2 <-: PathToHill- a1;
b2 <-: PathToHill- a2;

b3 <-: PathToHill- a3;
b3 <-: PathToHill- a4;
b3 <-: PathToHill- b4;

c3 <-: PathToHill- c4;
c3 <-: PathToHill- d4;
c3 <-: PathToHill- d3;

c2 <-: PathToHill- d2;
c2 <-: PathToHill- d1;
c2 <-: PathToHill- c1;

a1 -: GridEdge-> a2 -: GridEdge-> a3 -: GridEdge-> a4;
a4 -: GridEdge-> b4 -: GridEdge-> c4 -: GridEdge-> d4;
d4 -: GridEdge-> d3 -: GridEdge-> d2 -: GridEdge-> d1;
d1 -: GridEdge-> c1 -: GridEdge-> b1 -: GridEdge-> a1;

// Create ants

queen: Ant -: AntPosition-> hill;
atta: Ant -: AntPosition-> hill;
flick: Ant -: AntPosition-> hill;
chuck: Ant -: AntPosition-> hill;
the: Ant -: AntPosition-> hill;
plant: Ant -: AntPosition-> hill;
chewap: Ant -: AntPosition-> hill;
cici: Ant -: AntPosition-> hill;
queen -:NextAnt-> atta -:NextAnt-> flick -:NextAnt-> chuck -:NextAnt-> the
-:NextAnt-> plant -:NextAnt-> chewap -:NextAnt-> cici;

// The ultimate GAMMEL FIX(tm)!!!!
:Zero; :MinusOne; :MinusTwo;

return (queen);

rule TakeFood(curAnt:Ant)
{
curAnt -:AntPosition-> n:GridNode\AntHill;
if { !curAnt.hasFood && n.food > 0; }
modify {
    eval {
        curAnt.hasFood = true;
        n.food = n.food - 1;
    }
}
}

rule GoHome(curAnt:Ant)
{
if { curAnt.hasFood; }
curAnt -oldPos:AntPosition-> old:GridNode -:PathToHill-> new:GridNode;
modify {
    eval {
        old.pheromones = old.pheromones + 1024;
    }
    delete(oldPos);
    curAnt -:AntPosition-> new;
}
}

rule DropFood(curAnt:Ant)
{
if { curAnt.hasFood; }
curAnt -:AntPosition-> hill:AntHill;
modify {
    eval {
        curAnt.hasFood = false;
        hill.food = hill.food + 1;
    }
}
}
rule SearchAlongPheromones(curAnt:Ant)
{
  curAnt -oldPos:AntPosition-> old:GridNode <-:PathToHill- new:GridNode;
  if { new.pheromones > 9; }
  modify {
    delete(oldPos);
    curAnt -:AntPosition-> new;
  }
}

rule SearchAimless(curAnt:Ant)
{
  curAnt -oldPos:AntPosition-> old:GridNode <-:GridEdge-> new:GridNode\AntHill;
  modify {
    delete(oldPos);
    curAnt -:AntPosition-> new;
  }
}

test ReachedEndOfWorld : (GridNode)
{
  :Ant -:AntPosition-> n[prio=10000]:GridNode\AntHill;
  negative { n <-:PathToHill-; }
  return (n);
}

rule GrowFoodIfEqual(n:GridNode, val:GammelFix)
{
  hill:AntHill;
  if { hill.foodCountdown == val.val; }
  modify {
    eval {
      n.food = n.food + 100;
      hill.foodCountdown = hill.foodCountdown + 10;
    }
  }
}

rule GrowWorldFirstAtCorner(cur:GridCornerNode) : (GridNode, GridNode)
{
  cur -:GridEdge\PathToHill-> next:GridNode;
  hill:AntHill;
  zero:Zero;
  minusOne:MinusOne;
  minusTwo:MinusTwo;
}
modify {
    cur <- PathToHill - outer1:GridNode;
    cur <- PathToHill - outer2:GridCornerNode;
    cur <- PathToHill - outer3:GridNode;
    outer1 -:GridEdge-> outer2 -:GridEdge-> outer3;
    eval {
        hill.foodCountdown = hill.foodCountdown - 3;
    }
    return (next, outer3);
    exec( GrowFoodIfEqual(outer1, minusTwo)
         || GrowFoodIfEqual(outer2, minusOne)
         || GrowFoodIfEqual(outer3, zero));
}
}

rule GrowWorldFirstNotAtCorner(cur:GridNode\GridCornerNode) : (GridNode, GridNode)
{
    cur -:GridEdge\PathToHill-> next:GridNode;
    hill:AntHill;
    zero:Zero;
    modify {
        cur <- PathToHill - outer:GridNode;
        eval {
            hill.foodCountdown = hill.foodCountdown - 1;
        }
        return (next, outer);
        exec(GrowFoodIfEqual(outer, zero));
    }
}

rule GrowWorldNextAtCorner(cur:GridCornerNode, curOuter:GridNode) : (GridNode, GridNode)
{
    cur -:GridEdge\PathToHill-> next:GridNode;
    hill:AntHill;
    zero:Zero;
    minusOne:MinusOne;
    minusTwo:MinusTwo;
modify 
  cur <-:PathToHill- outer1:GridNode;
cur <-:PathToHill- outer2:GridCornerNode;
cur <-:PathToHill- outer3:GridNode;
curOuter -:GridEdge-> outer1 -:GridEdge-> outer2 -:GridEdge-> outer3;

eval 
  hill.foodCountdown = hill.foodCountdown - 3;
}

return (next, outer3);
exec( GrowFoodIfEqual(outer1, minusTwo)
  || GrowFoodIfEqual(outer2, minusOne)
  || GrowFoodIfEqual(outer3, zero));
}
}

rule GrowWorldNextNotAtCorner(cur:GridNode\GridCornerNode, curOuter:GridNode)
  : (GridNode, GridNode)
{
  cur -:GridEdge\PathToHill-> next:GridNode;
negative ( cur <-:PathToHill-; )
hill:AntHill;
zero:Zero;

modify 
  cur <-:PathToHill- outer:GridNode;
curOuter -:GridEdge-> outer;

eval 
  hill.foodCountdown = hill.foodCountdown - 1;
}

return (next, outer);
exec(GrowFoodIfEqual(outer, zero));
}

rule GrowWorldEnd(cur:GridNode, curOuter:GridNode)
{
  cur <-:PathToHill- nextOuter:GridNode;
modify 
  curOuter -:GridEdge-> nextOuter;
}

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)])  
    ) && (cur:GridNode)=ReachedEndOfWorld  
    | ((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]);  
} }
new graph AntWorld

#debug set layout Organic
#dump set node Ant color red
#dump add node Ant infotag hasFood
#dump add edge NextAnt exclude
#dump add node GammelFix exclude

randomseed 42

xgrs (firstAnt)=InitWorld

xgrs doAntWorld(firstAnt)

#debug xgrs (curAnt=firstAnt && \
#  (( \n#    TakeFood(curAnt) | GoHome(curAnt) || \n#    DropFood(curAnt) | ($[SearchAlongPheromones(curAnt)] || \n#     $[SearchAimless(curAnt)]) \n#   ) && (curAnt)=GetNextAnt(curAnt)) && \
#   ((cur)=ReachedEndOfWorld \
#    || (cur, curOuter)=GrowWorldFirstNotAtCorner(cur) \n#    || (cur, curOuter)=GrowWorldFirstAtCorner(cur)) \n#    && (cur, curOuter)=GrowWorldNextNotAtCorner(cur, curOuter) \n#    || (cur, curOuter)=GrowWorldNextAtCorner(cur, curOuter) \n#    && GrowWorldEnd(cur, curOuter)) \n#    || (curAnt)=Food2Ant(curAnt) \n#    || [EvaporateWorld] \n#  ) [250]

show num nodes GridNode
show num nodes Ant