VIATRA/GRASS: Graph Transformation-based Stochastic Simulation

(1) Paolo Torrini, Reiko Heckel, (2) Istvan Rath

pt95, reiko@mcs.le.ac.uk, rath@mit.bmi.hu

(1) University of Leicester,
(2) Budapest University of Technology and Economics
Stochastic graph transformation

- Modelling: graph transformation
- Validation: stochastic simulation
  - runs depends on random numbers
  - useful when models are too large for model checking
- Generalised stochastic graph transformation: events associated with general probability distributions
- Discrete event system semantics
- Implementation based on incremental pattern matching
Some history

- 2004-07: Heckel, Lajios, Menge — seminal work on stochastic graph transformation
  rules associated with exponential distributions, translated to labelled transition systems, analysed as Markov chains (probabilistic model checking)

- 2007: Heckel: application to P2P networks

- 2007: Lajos, Kosiuczenko — outline extension, rules with general distributions, semi-Markov processes, unfolding semantics (global name-space needed)
... and more recently

- 2009: Heckel, Torrini — rule matches with general distributions, concrete semantics (numbered graphs) and extensions
- Torrini, Rath — implementation based on VIATRA
- Ajab Kahn — application to Skype (PhD topic)
- Kahn, Torrini, Heckel — ICGT 2008 Doctoral Symposium
- Torrini, Heckel, Rath — paper submitted at FASE 2010
GTS with probability

- Stochastic Graph Transformation: rules associated with exponential probability distributions

- Generalised Stochastic Graph Transformation: rule matches associated with general probability distributions

- Rule matches as equivalence classes — identity through transformation — cardinality restrictions to ensure they are a proper set (numbered graphs)

- Probabilistic rather than indeterministic actions

- Continuous time: waiting times as independent random variables — no parallelism
Probability distributions

- Possible rule application associated with expected delay $D$

- $D$ (waiting time) — random variable associated to a probability distribution function $F_D(x) = P(D \leq x)$
  $F_D$ determines the probability of the delay being less than $x$

- Markov property — process depends on present state only,
  $P(D > x + z|D > z) = P(D > x)$

- Semi-Markov process: next state may depend on time spent in current state

- Exponential distribution: determined by a rate, can express how “fast” is a Markov process.

- Normal distribution: mean and variance — process with meaningful average value and deviation
Discrete event systems semantics — stochastic models based on Generalised Semi-Markov Schemes (more general than Markov chains, general distributions)

\[ GSMS = \langle \text{States}, \text{Events}, \text{ActiveEvents}, \text{Transition}, \text{DistrAssign}, \text{InitState} \rangle \]

where \( \text{Distribution} = \mathcal{R} \rightarrow [1, 0] \)
A Generalised Stochastic GTS defines a GSMS, where $\Delta(\langle r, m \rangle)(d)$ is the probability that the waiting time for rule $r$ at match $m$ is less than $d$.

\[
\text{GSGTS} = \langle \text{ReachGraphs}
\text{RuleMatches} \quad (\text{equivalence classes})
\text{EnabledMatches} : \text{ReachGraph} \rightarrow \wp\text{RuleMatch}
\text{GraphTrans} : \\
\quad \text{ReachGraph} \times \text{RuleMatch} \rightarrow \text{ReachGraph}
\Delta : \text{RuleMatch} \rightarrow (\mathcal{R} \rightarrow [1, 0])
\text{InitialGraph} : \text{ReachGraph} \quad \rangle
\]
GSMS-based simulation

- GSMS execution based on event scheduling scheme
- At each step
  - the event with the shortest waiting time is executed
  - the simulation time is updated
  - enabled matches are computed
  - scheduling times of new matches are determined by random number generator given $\Delta$
  - waiting times of old matches decrease
Computational aspect

- Substantial problem: computing all matches at each step, needed because
  - not enough to know that a rule is enabled — number of matches makes difference in actual probability of rule application
  - waiting times may depend on local values of attributes
- Moreover, we need to retain identity of matches — so we cannot recompute the matches at each step
Incremental Pattern Matching

- Incremental approach (RETE algorithm): pattern-matching problem constant in model size, polynomial in rule number — after initialisation phase, which can be hard (subgraph homomorphism problem known to be NP-complete)

- Standard approach: update constant in model size and rule number

- IPM useful when rules have complex LHS and when all matches are needed

- VIATRA (Eclipse plugin) — graph transformation engine that implements IPM
Architecture of the tool

- Graph transformation engine
  - computes matches
  - executes selected rule match

- Simulation engine
  - determines waiting time, relying on SSJ random number generation
  - manages waiting times
  - selects rule matches for execution
  - extracts statistics, relying on SSJ tally classes
  - controls textual and visual output
VIATRA/GRASS

- Stochastic simulation implemented in Java on top of VIATRA — uses Java-SSJ libraries
- inputs: \( G \) (GTS), \( \Delta \) (distribution assignment)
- \( G \): loaded in VIATRA — model-space (vpml) and rules (vtcl)
- \( \Delta \): case-defined in an XML file, automatically translated to a model-space entity
- probe rules: extract information for stochastic analysis — collected in tally class objects, displayed as textual output
- additional simulation parameters: e.g. number of runs, max depth of each run (steps or time)
- visualisation for staged execution
The other ones add redundant connection in a smarter way, by checking that the two nodes are not already pathwise connected through
more than one.
At the moment we assume that the two nodes must be pathwise connected
through another one, that comes from the paper but might have to be
dropped later.

```c
pattern two(N1) =
{
    SN(N1);
    find connected(N1,N2);
    find connected(N1,N3);
    find connected(N1,N4);
}
```

// Trigger
```c
grule randomConnect() =

    precondition pattern Lhs(N0,N5,N6) =
    {
        SN(N0);
        SN(N5);
        SN(N6);
        find connected(N0,N5);
        find connected(N0,N6);
        find connected(N5,N6);
    }

    action {
        let 0v1 = undef in new(SN,overlay(0v1,N5,N6));
        let 0v2 = undef in new(SN,overlay(0v2,N6,N5));
        println(["RandomConnect added connection between: "+for(N5=1 and +for(N6=1)])
    }
```

pattern twoConnected(N1,N2) =
{
    SN(N1);
    SN(N2);
    find connected(N1,N2);
    find connected(N2,N3);
    find connected(N1,N3);
```
VIATRA2 - stochastic_glibbasicP2P.xvl - Eclipse Platform

VIATRA2 Textual Output 22

NullNode: {default=exp} -> null
NewNodes: {default=exp} -> null
RandomConnect: {default=exp} -> null
Probing predicate: disconnected
Main input rate: 1000.0
Number of runs: 3
Run maximum depth: 20

Report on time: REPORT on Tally stat. collector -> null
num. obs. min max average standard dev.
3 0.223 1.335 0.843 0.573

Run sim-time values: [0.2233671755151142, 0.9499430262106562, 1.3549671065677006]

Report on depth: REPORT on Tally stat. collector -> null
num. obs. min max average standard dev.
3 5.000 20.000 15.333 6.351

Run depth values: [5.0, 20.0, 20.01]

Report on binary prob: REPORT on Tally stat. collector -> null
num. obs. min max average standard dev.
3 0.385 0.626 0.405 0.189

Run avg binary prob values: [0.625, 0.250, 0.3090000000000000]

Report on max node number: REPORT on Tally stat. collector -> null
num. obs. min max average standard dev.
3 5.000 8.000 6.333 1.528

Run max node numbers: [5.0, 8.0, 6.0]

Report on probe value (weighted w.r.t nodes): REPORT on Tally stat. collector -> null
num. obs. min max average standard dev.
3 0.079 0.207 0.144 0.065

Run avg probe values: [0.2058750000000000, 0.0778888888888888, 0.1477222222222222]

Confidence level: 0.95

Elapsed run-time: 190

END REPORT
Case study: P2P

- Simulation of P2P network
- Basic example — rules with exponential distributions
- Two behavioural rules: create node, kill node
- Two alternative reconfiguration rules: randomly/smarterly create redundant connection
- Stochastic analysis based on number of pathwise disconnected nodes at each step, on varying the rate of reconfiguration
- Small model, few seconds execution time
Type Graph

P = peer
R = registry

disconnected
pattern pathEx(N1,N2) = {
    SN(N1);
    SN(N2);
    find connected(N1,N2); 
}
or {
    SN(N1);
    SN(N2);
    SN(N0);
    find connected(N1,N0);
    find pathEx(N0,N2); }

pattern noPathEx(N1,N2) = {
    SN(N1);
    SN(N2);
    neg find pathEx(N1, N2); }

gtrule disconnected() = {
    precondition pattern lhs(N1, N2) = {
        SN(N1);
        SN(N2);
        find noPathEx(N1, N2); }
    action {println("..."+fqn(N1)+fqn(N2));}}
Stochastic analysis

- Program can execute given number of simulation runs up to given depth value, expressed either as sim-time or step number.
- Returns average number of probe matches, max number of nodes, number of steps, and simulation time for each run.
- Average, min, max and deviation over all the runs.
- More specific — wrt to the P2P probe: statistics on $M/N^2$ with $M$ number of probe matches, $N$ number of nodes.
- Basic method: run many simulations long enough, until similar results for probe matches are obtained.
- Hypothesis on behaviour tested by changing reconfiguration rates, comparing models, etc.
## Sample results

<table>
<thead>
<tr>
<th>Model: P2P</th>
<th>Disconnected</th>
<th>Number of steps</th>
<th>Max number of peers</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>random:1</td>
<td>0.46</td>
<td>33</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>random:10</td>
<td>0.62</td>
<td>71</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>random:100</td>
<td>0.55</td>
<td>86</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>random:1000</td>
<td>0.89</td>
<td>284</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>random:10,000</td>
<td>0.46</td>
<td>116</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>smart:1</td>
<td>1.33</td>
<td>18</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>smart:10</td>
<td>0.01</td>
<td>90</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>smart:100</td>
<td>0.00</td>
<td>3561</td>
<td>48</td>
<td>10</td>
</tr>
<tr>
<td>smart:1000</td>
<td>0.00</td>
<td>998</td>
<td>24</td>
<td>10</td>
</tr>
<tr>
<td>smart:10,000</td>
<td>0.00</td>
<td>62</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>
Extending the model

- Distributions may depend on attributes of match elements
- Global variables — e.g. simulation time
- Derived attributes, computed when needed — depending on global variables, depending on local information — incoming/outgoing edges
- Distributions depending on derived attributes
- Spatial dependencies — matches “in the same region”
- Distributions depending on attributes of nearby matches
- Trying to take advantage of incremental pattern matching
Further work

- Handling general distributions (beyond exponential and normal ones)
- Refining stochastic analysis
- Improving scalability
- Modelling VoIP networks (with Ajab Khan)
- Synchronisation of textual and visual output
- Comparison with other tools/approaches