Design and Analysis of Distributed Interacting Systems

Lecture 13 – Graph Transformation Systems and Hints on the Exam

Prof. Dr. Joel Greenyer

July 11, 2013
• Many objects with relationships that change over time
• How do we describe how the structure of these systems change over time?
  – their reconfiguration behavior

from http://www fhwa dot gov/advancedresearch/pubs/12033/004 cfm
Last Time: Describe Structural Changes

• Most children understand this way of describing structural changes:
• Idea: View the system as a graph
  – objects are nodes
  – edges are relationships
• Describe the context of the change and the changes in a rule

the intuition is clear (at least for this example), but what does this mean exactly?
Last Time: Models and Graphs

- **Class Model vs. Object Model**

  ![Class Model vs. Object Model Diagram]

  - `:RailCab` isOn `:Track`
  - `:Track` 1..1 `isOn` `:RailCab`
  - `:Track` 0..1 `next` `:Switch`
  - `:Switch` 0..1 `turnNext` `:Track`

  Instance of:

  - `:RailCab` `isOn`
  - `:Track` `turnNext` `isOn`
Last Time: Models and Graphs

- Type Graph vs. Typed Graph

Type Graph

Typed Graph
(edges are typed, too)
A graph grammar consists of
- a set of *graph grammar rules*
- a *start graph* (also called *host graph*)
- a *type graph*

A graph grammar describes a (possibly infinite) set of graphs
- that can be constructed from the start graph by applying the
  graph grammar rules

**Synonym:** Graph Rewriting- or Graph Transformation System

---

**Start Graph**

- `:RailCab` `isOn` `:Track`
- `:Track` `next` `:Track`
- `:Track` `next` `:Track`

**Move**

- `:RailCab` `isOn`
- `:Track` `next` `:Track`
- `:Track` `next` `:Track`
Agenda

• Introduction to Graph Transformations (✓)
• Models and Graphs (✓)
• Graph Grammars (✓)
• Graph Grammar Rules
• Rule Application
• Advanced Concepts
  – Negative Application Conditions
  – Attribute Constraints
• Analysis of Graph Transformation Systems
A graph grammar rule consists of two typed graphs—called left-hand side (LHS) and right-hand side (RHS).
Graph Grammar Rule Application

host graph:

1. Match LHS in host graph
   (find isomorph subgraph)

\[
\text{move} \\
\text{:RailCab} \quad v_1 \\
isOn \\
\text{:Track} \quad v_2 \quad \text{next} \\
\text{:Track} \quad v_3
\]

(LHS)

\[
\text{::=} \\
\text{:RailCab} \quad v_1 \\
isOn \\
\text{:Track} \quad v_2 \quad \text{next} \\
\text{:Track} \quad v_3
\]

(RHS)
Graph Grammar Rule Application

host graph:

2. Remove nodes and edges that are in the LHS, but not in the RHS
Graph Grammar Rule Application

host graph:

```
move
:RailCab v1

(RHS)

:RailCab v1

isOn

:Track v2

next

:Track v3

:::=

(LHS)

:RailCab

isOn

:Track

next

:Track

next

:Track

next

:Switch

2. Create nodes and edges that are in the RHS, but not in the LHS

:::=

isOn

:-track

next

:Track

next

:Track

next

:Switch`
Graph Grammar Rule Application

• When to move which RailCab?
  – non-determinism!

\[
\begin{align*}
& (: \text{RailCab}) \\
& \downarrow \text{isOn} \\
& (: \text{Track}) \quad \text{next} \quad (: \text{Track}) \quad \text{next} \quad (: \text{Track})
\end{align*}
\]

\[
\begin{align*}
\text{move} \\
& (: \text{RailCab}) \quad v1 \\
& \downarrow \text{isOn} \\
& (: \text{Track}) \quad \text{next} \quad (: \text{Track}) \\
& \downarrow v2 \quad v3
\end{align*}
\]

\[
\begin{align*}
& (: \text{RailCab}) \quad v1 \\
& \downarrow \text{isOn} \\
& (: \text{Track}) \quad \text{next} \quad (: \text{Track}) \\
& \downarrow v2 \quad v3
\end{align*}
\]

\[
\begin{align*}
\text{:::=}
\end{align*}
\]
Graph Grammar Rule Application

- Does the RailCab move straight or turn?
  - non-determinism!

Remember: switches are tracks, too. (But that's already an advanced concept)
Agenda

• Introduction to Graph Transformations (✓)
• Models and Graphs (✓)
• Graph Grammars (✓)
• Graph Grammar Rules (✓)
• Rule Application (✓)
• Advanced Concepts
  – Negative Application Conditions
  – Attribute Constraints
• Analysis of Graph Transformation Systems
Negative Application Condition

- Example: A RailCab must not crash into another
A broken RailCab cannot move
Agenda

• Introduction to Graph Transformations (✓)
• Models and Graphs (✓)
• Graph Grammars (✓)
• Graph Grammar Rules (✓)
• Rule Application (✓)
• Advanced Concepts (✓)
  – Negative Application Conditions (✓)
  – Attribute Constraints (✓)
• Analysis of Graph Transformation Systems
Exploring the State Space

- A rule application can be considered a transition in an LTS
  - source state: host graph before the rule application
  - event: rule application
  - target state: host graph after the rule application

![State space explored with Henshin (see last assignment)]
A rule application can be considered a transition in an LTS:

- source state: host graph before the rule application
- event: rule application
- target state: host graph after the rule application

start graph

:RailCab

isOn

:Track

next

:Track

move

:RailCab

isOn

:Track

next

:Track

move

:Track

next

:Track

move

:Track

next

:Track

16 states

two RailCabs – how many states now?
Exploring the State Space

two RailCabs and not a RailCab moving on a track if another RailCab is already on it – how many states?

16-4 states
Exploring the State Space

Rule move

- «preserve» :RailCab
- «create» isn
- «delete» isn

:RailCab isOn :Track

next :Track

next :Track

rule as specified in Henshin
Model Checking Graph Transformation Systems

• If the state space is finite, we can analyze a graph transformation system via model checking
  – can you think of an example where the state space is infinite?

• Henshin supports checking OCL invariants
• Henshin also supports more sophisticated model checking using external tools
  – probabilistic model checking with PRISM

• GROOVE is another tool for model checking GTS
  – temporal properties can be specified using CTL (which we did not cover in the lecture)
  – atomic propositions are graph patterns (true if match exists)
Example: Checking OCL Invariants with Henshin

- Example OCL Invariant:
  - never two RailCabs on the same track

```
self.railCabs
->forAll(rc1, rc2:RailCab | rc1 <> rc2 implies rc1.isOn <> rc2.isOn)
```

OCL property will be evaluated for the root object of the graph (here :TrackSystem)
Design and Analysis of Distributed Interacting Systems

Lecture 13 – Graph Transformation Systems and Hints on the Exam

Prof. Dr. Joel Greenyer

July 11, 2013
Oral Exam

• General:
  – Exam in English or German, 30 Minutes examination time
  – No additional material admitted or required
    • We have enough paper and can provide a pen for you

• Dates and Registration:
  – August 13 and 15 (Tuesday and Thursday):
    • morning or afternoon slots available
  – August 14 and 19 (Wednesday and following week Monday)
    • morning slots available
  – Register by writing your name on forms on G3 floor
    • Preferred dates of some students have already been noted
  – Please send an email five to two days before the exam, then we will tell you the exact time for your exam
Oral Exam

• Contents of the Exam
  – first 5-10 Minutes: Summarize the contents of the lecture
  – Then questions
    • General Questions: Explain ideas of formalisms / approaches / algorithms, Understanding of practical problems
    • Detailed/Technical Questions: Demonstrate knowledge of formalisms / approaches / algorithms, typically by examples
    • Advanced Questions: Understand connections between formalisms / approaches, apply concepts to new problems
Question Examples*

• General Questions:
  – What is model checking?
  – Explain the approach for LTL model-checking that was introduced in the lecture
  – What is the state-space explosion problem? What cause a state-space explosion? Give examples.
  – What is an implementation of an MSD specification?
  – Why is testing important?
  – What is a graph grammar and what does it do?
Question Examples*

• Detailed Questions:
  – Specify LTL property for “If p then eventually q”
  – Give a Büchi Automaton for given language
  – Give a Büchi Automaton for a given universal MSD
  – Is a given MSD specification realizable?
  – Give a Büchi Automaton that accepts the intersection of...
  – Compose two LTSs via handshaking
  – Understand / extend a Promela program (knowledge of language concepts enough, exact syntax not important)
  – What is a cold violation?

* no warranty
Question Examples*

• Advanced Question:
  – Suppose you want to build a model checker for checking universal MSDs, what would you do?
  – ...

* no warranty