Graph Transformation in a Nutshell

Reiko Heckel
University of Leicester, UK

Why it is fun: Programming By Example

StageCast (www.stagecast.com): a visual programming environment for kids (from 8 years on), based on
- behavioral rules associated to graphical objects
- visual pattern matching
- simple control structures (priorities, sequence, choice, ...)
- external keyboard control
- intuitive rule-based behavior modelling

Next: abstract from concrete visual presentation

Rules of the PacMan Game: Graph-Based Presentation, PacMan

Rules of the PacMan Game: Graph-Based Presentation, Ghost

States of the PacMan Game: Graph-Based Presentation

Outline
- Graph Transformation
  - why it is fun
  - how it works
- Applications and Theory
A Basic Formalism: Typed Graphs

Directed graphs
- multiple parallel edges
- undirected edges as pairs of directed ones

Graph homomorphism as mappings preserving source and target

Typed graphs given by
- fixed type graph \( T \)
- instance graphs \( G \)
  typed over \( T \) by homomorphism

Rules

\[ p: L \rightarrow R \] with \( L \cap R \) well-defined, in different presentations
- like above (cf. PacMan example)
- with \( L \cap R \) explicit [DPO]: \( L \leftarrow K \rightarrow R \)

Semantic Questions: Dangling Edges

Transformation Step

1. select rule \( p : L \rightarrow R \); occurrence \( o_1 : L \rightarrow G \)
2. remove from \( G \) the occurrence of \( L \setminus L \)
3. add to result a copy of \( R \setminus L \)

Semantic Questions: Conflicts

- conservative solution: application is forbidden
  - invertible transformations, no side-effects
- radical solution: delete dangling edges
  - more complex behavior, requires explicit control
Advanced Features

Dealing with unknown context
- set-nodes (multi-objects): match all nodes with the required connections
- explicit (negative) context conditions

(turns f1 into a trap by reversing all outgoing edges to Field vertices, but only if there is no Ghost)

Control Structures
- priorities
- programmed transformation

Motivation: Software Development as Integration of Views

Aspects of Requirements Models

Outline
✓ Graph Transformation
  ✓ why it is fun
  ✓ how it works
✓ Applications and Theory
  ✓ Modelling and Analysis of Functional Requirements
  ✓ Model Transformation and Semantics

Structure: Class and Object Diagrams
Graph Transformation in a Nutshell
Reiko Heckel, Univ. of Leicester

**Behaviour: Use Case Description by Structured Text**

- Customer:
  - take shopping cart
  - select items from rack
  - pay required amount
  - collect payment
  - pack and give items to customer

- Clerk:
  - create empty bill for new customer
  - take items out of customer’s cart
  - add them to the bill
  - collect payment
  - take items out of customer’s cart

- Shop:
  - sell items

- Bill:
  - total = x

- CashBox:
  - cash = y

- Item:
  - amount = y+x

- Cart:
  - close bill

- Shop:
  - owns Bill

**Conflicts Between Functional Requirements**

- Customer:
  - customer updates cash
  - customer's cart

- Clerk:
  - clerk updates amount
  - close bill

- Shop:
  - buy items

**Aspects of Requirements Models**

**Model A**
- Static domain model: Agree on vocabulary first!
- Business process model: Which actions are performed in which order?
- Functional model: What happens if an action is performed?

**Model B**
- Static domain model: Agree on vocabulary first!
- Business process model: Which actions are performed in which order?

3. Functional model: What happens if an action is performed?
   - pre/post conditions as logic constraints
   - transformation rules on object diagrams
     (Fusion, Catalysis, Fujaba, formally: graph transformations)

**Function: Transformation Rules on Object Diagrams**

- Customer:
  - pay bill

- CashBox:
  - close bill

- Shop:
  - sell items

- Item:
  - amount = y+x

- Bill:
  - total = x

**Theory: Independence, Causality and Conflicts in Graph Transformation**

- Alternative steps are parallel independent if they do not disable each other. Otherwise they are in conflict.
- Consecutive steps are sequentially independent if they may be swapped without affecting the result. Otherwise they are causally dependent.

**Characterization (EPS73):**
Two (alternative or consecutive) steps are independent iff all commonly accessed items are in read-access only.
Focus and primary artifacts

- Graph transformations can be used to study program evolution.
- They are examples of model transformations.

Core activities include:
- Maintaining consistency
- Translation
- Execution of models
- These are examples of model transformations

Model-driven Development

- A math. foundation is needed for studying
  - Expressiveness and complexity
  - Execution and optimisation
  - Well-definedness
  - Semantic correctness of transformations
- Graph transformations can be one such foundation

Outline

- Graph Transformation
  - why it is fun
  - how it works
- Applications and Theory
  - Modelling and Analysis of Functional Requirements
  - Model Transformation and Semantics

Outline

- Model transformation
  - denotational semantics
    - analysis → synthesis
  - operational semantics
  - refactoring

Visual Modeling Techniques

- Petri nets
- Function Block Diagrams
- Class Diagrams (UML)

Context-Free Graph Grammar

Concrete Syntax of Well-Formed Activity Diagrams

Production in EBNF-like notation:
Graph Transformation in a Nutshell
Reiko Heckel, Univ. of Leicester

Is this Good Enough?
✓ Visual
  • abstract syntax or concrete syntax templates
✓ Bi-directional
  • swap source and target grammars
✓ Declarative
  • context-free graph languages only
× Expressive?
  • through naming conventions
× Traceable?
  • NP complete parsing problem
× Efficient?
  ...

→ Triple Graph Grammars

Outline
✓ Model transformation
  • denotational semantics
  • operational semantics
  • refactoring

Example: Executable Business Process
✓ refactoring of business processes, replacing centralised by distributed execution
✓ How to demonstrate preservation of behaviour?
  1. specify operational semantics of processes
  2. define transformations
  3. show that transformations preserve semantics

Synthesis
Proc(A0)
Proc(A1) → Proc(A2)
Proc(A3) → Proc(A4) →
if product available then Proc(A5) else Proc(A6)
... receive order → check availability →
if product available then calculate prize else notify client

Analysis
0
receive order
check availability
1
[product available]
2
notify client
send receipt
3
[product not available]
4
calculate prize
5
send receipt

Pair Grammar

Receive order
Check availability
Notify client
Calculate prize
Send receipt

Source: well-structured activity diagrams
Target: CSP

Proc(A) ::= Proc(A1) / Proc(A2)
if [c] then Proc(A1)
else Proc(A2)
do something

Proc(A0)
Proc(A1) → Proc(A2)
Proc(A3) → Proc(A4) →
if product available then Proc(A5) else Proc(A6)
... receive order → check availability →
if product available then calculate prize else notify client
Operational Semantics: Idea

- diagram syntax plus runtime state
- GT rules to model state transitions

Operational Semantics: Formally

\[
GTS = (TG, P) \text{ with start graph } G_0
\]
defines transition system

\[
LTS(GTS, G_0) = (S, L, \rightarrow)
\]

taking
- as states \( S \) all graphs reachable from \( G_0 \)
- observations on rules as labels
- transformations as transitions

Rules: Invoke another Service

Rules: Answer the Invocation

Rules: Receive the Response

Rules: Receive the Response

Type Graph: Metamodel

with runtime state
**Graph Transformation in a Nutshell**

Reiko Heckel, Univ. of Leicester

SegraVis School, Leicester, 8-11/09/2006

---

### Simulation

<n1:Orch> to <n2:Orch> via <inv>:<reply>

- *request*: to <n1:Orch>
- *response*: from <n1:Orch>
- *invoke*: to <n2:Orch>
- *reply*: from <n2:Orch>

**Observations:** req(i.id, m1.id); reply(r.id, m1.id, m2.id); resp(i.id, m2.id)

---

### Refactoring Executable Business Processes

- *Orch 1* to *Orch 2*
- *Orch 1* to *Orch 2*

- *delegate*

- *← invoke* → *Orch 2.op*

- *→ receive* → *op*

- *← reply* → *op*

- **replace local control flow by message passing**

---

### Outline

- **Model transformation**
  - denotational semantics
  - operational semantics
  - refactoring

- **Operational Semantics**

- **Denotational Semantics**

---

### Preservation of Semantics

Show for each refactoring P → P' that P' simulates P, i.e.

- P ↠obs Q implies P' ↠obs Q'
- Q' simulates Q

and vice versa.

**Approach:**
- mixed (local) confluence
- critical pair analysis

---

### Example: Executable Business Process

- refactoring of business processes, replacing centralised by distributed execution

- How to demonstrate preservation of behaviour?
  - specify operational semantics of processes
  - define transformations
  - show that transformations preserve semantics

---

### Theory: Sources of Inspirations

- **Chomsky Grammars**
- **Term Rewriting**
- **Petri Nets**

**Graph Transformation and Graph Grammars**

- Formal language theory of graphs;
- Diagram compiler generators

**Well-definedness**
- Termination
- Confluence
- Semantics of process calculi

**Concurrency theory**
- Causality and conflict
- Processes, unfoldings
- Event-structures