Motivation: Model-driven Engineering

- Focus and primary artifacts are models instead of programs
- Core activities include
  - maintaining consistency
  - evolution
  - translation
  - execution of models
- These are examples of model transformations
- A math. foundation is needed for studying
  - expressiveness and complexity
  - execution and optimisation
  - well-definedness
  - preservation of semantics of transformations
- Graph transformations can be one such foundation
Outline

- Graph Transformation
  - why it is fun
  - how it works
- Semantics-preserving Model Transformation

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
States of the PacMan Game: Graph-Based Presentation

instance graph
(represent a single state; abstracts from spatial layout)

type graph
(specifies legal instance graphs)

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

f1:Field

pm: PacMan
marbles=m

f2:Field

Marble

collect
pm: PacMan
marbles=m+1

movePM
pm: PacMan
marbles=m

f1:Field

f2:Field

f1:Field

f2:Field
Rules of the PacMan Game: Graph-Based Presentation, Ghost

Outline

- Graph Transformation
  - why it is fun
  - how it works
- Semantics-preserving Model Transformation
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* $TG$
- *instance graphs* $G$ typed over $TG$ by *homomorphism* $g$

---

**Rules**

$p: L \to R$ with $L \cap R$ well-defined, in different presentations
- like above (cf. PacMan example)
- with $L \cap R$ explicit [DPO]: $L \leftarrow K \to R$
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]: \( L \leftarrow K \rightarrow R \)
- with \( L, R \) integrated [UML, Fujaba]: \( L \cup R \) and marking
  - \( L - R \) as destroyed
  - \( R - L \) as new

\[
\text{movePM:} \quad \begin{array}{c}
\text{pm: PacMan} \\
\{\text{destroyed}\} \\
f1: \text{Field} \quad f2: \text{Field}
\end{array}
\]

Transformation Step

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

✓ Graph Transformation
  ✓ why it is fun
  ✓ how it works
✗ Semantics-preserving
Model Transformation
  ■ operational
  ■ denotational

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

Warehouse

Office

Receive order

Undo order

Shipmen
Operational Semantics: Idea

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

Type Graph: Metamodel

with runtime state
Rules: Invoke another Service

Simulation

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

- replace local control flow by message passing

**Semantic Compatibility**

Processes \( P_1 \) and \( P_2 \) are semantically compatible if they are *weakly bisimilar* after hiding labels not in \( \text{alph}(P_1) \cap \text{alph}(P_2) \)

Show for trafo \( P_1 \Rightarrow P_2 \) that \( P_2 \) simulates \( P_1 \), i.e.
- \( P_1 \xrightarrow{\text{obs}} Q_1 \) implies \( P_2 \xrightarrow{\text{obs}} Q_2 \)
- \( Q_2 \) simulates \( Q_1 \) and vice versa.

Approach:
- mixed (local) confluence
- critical pair analysis
Critical Pairs and Local Confluence

- A pair of rules \((p_1, p_2)\) in a *minimal conflict situation*
- Constructed by *overlapping their left-hand sides* so that they *intersect in items to be deleted*
- System is locally confluent if all critical pairs are

![Diagram](image)

Critical Pair Analysis in AGG

delegate vs operational rules
Outline

- Graph Transformation
  - why it is fun
  - how it works
- Semantics-preserving
- Model Transformation
  - operational
  - denotational
Process Improvement

Motivation:
- transform process to increase flexibility, performance, ...
- preserve behaviour!

Aim: rule-level verification

Rule-level Verification

Approach:
- check for each rule \( r \) if \( s(L) \rightarrow s(R) \)
- make sure that \( s(L) \rightarrow s(R) \) implies \( s(G) \rightarrow s(H) \)
Works if ...

- we select semantic domain SD where relation $R$ is compositional
  - trace or failures equivalence or refinement in CSP is closed under context

- we can show that semantic mapping $s$: $AS \rightarrow SD$ is compositional
  - maps union of graphs to composition of CSP processes

→ use GT to define mapping $s$

---

**Context-Free Graph Grammar**

Concrete syntax of *well-structured* activity diagrams

Start graph:

Productions in EBNF-like notation:

- $\text{Act} \rightarrow \text{Act} \text{ Act}$
- $\text{Act} \rightarrow \text{Act} \text{ Act}$
- $\text{Act} \rightarrow \text{Act}$
- $\text{Act} \rightarrow \text{Act}$
- $\text{Act} \rightarrow \text{Act}$

do something
Analysis

1. receive order
2. check availability
3. [product not available]
   - [product available]
   - calculate prize
   - notify client
4. [product not available]
   - send receipt

Pair Grammar

Source: well-structured activity diagrams
Target: CSP

Abstract Syntax

Semantic Domain

denotational semantics

Proc(A)

A:Act::=

A1:Act
A2:Act

[A1:Act, A2:Act]

A:Act::=

Proc(A1) ;
Proc(A2)

if [c] then Proc(A1)
else Proc(A2)

do something
Synthesis

Proc(A0)

Proc(A1) ; Proc(A2)

...

Proc(A3) ;

Proc(A4) ;

if [product available]

then Proc(A5)

else Proc(A8)

...

receive order ;

check availability ;

if [product available]

then calculate prize;

send receipt

else notify client

Is this good enough ... ?

✓ Visual
  - abstract syntax or concrete syntax templates
✓ Bi-directional
  - swap source and target grammars
✓ Declarative

✗ Expressive ?
  - context-free graph languages only
✗ Traceable ?
  - through naming conventions
✗ Efficient ?
  - NP complete parsing problem
  - ...

⇒ Triple Graph Grammars

Challenge: verify compositionality for complex mappings
Relevant Theory

- 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
  - Stochastic concepts
  - Verification, …