PROGRAMMED
CONTROLLED
COMPOSED GRAPH REWRITING
(ILLUSTRATED IN GROOVE)
Arend Rensink, University of Twente
Graphs as Models, Eindhoven, April 2016
Graph Rewriting Framework

- Main ingredients
  - Graphs $G \in \text{Graph}$: the objects being rewritten
    - (Partial) morphisms $f \in \text{Morph} \subseteq \text{Graph} \times \text{Graph}$
  - Rules $r \in \text{Rule}$: embodiment of types of change to (certain) graphs
  - Matches $m \in \text{Match}$: places in graph where rule can be applied

- Matching function $M: \text{Rule} \times \text{Graph} \rightarrow 2^{\text{Match}}$
  - $M^r(G)$ denotes the set of matches of $r$ in $G$

- Rule application $A: \text{Graph} \times \text{Rule} \times \text{Match} \rightarrow \text{Morph} \times \text{Graph}$
  - $G \Rightarrow^{r,m,f} H$ denotes $A(r,m) = (f,H)$
  - Match resolves non-determinism: $A$ is a (partial) function
  - Defined on $(G,m,r)$ if and only if $m \in M^r(G)$
  - Match $m$ and morphism $f$ often omitted: $G \Rightarrow^{r,m} H$ or $G \Rightarrow^r H$

- Nothing in the above is specific to graphs
  - Other rewriting formalisms: strings, terms, proofs, bigraphs, ...
EXAMPLE: FROG PUZZLE

Demo using GROOVE: http://sf.net/projects/groove
Graph transition system (GTS): tuple \( S = \langle Q, R, \rightarrow, \iota \rangle \)

- States \( q \in Q \), each with associated graph \( G_q \)
- Rules \( R \subseteq \text{Rule} \)
- Transition relation \( \rightarrow \subseteq Q \times R \times \text{Match} \times \text{Morph} \times Q \)
  - \( q \rightarrow^{r,m,f} q' \) only if \( G_q \Rightarrow^{r,m,f} G_{q'} \) (not necessarily if!)
  - Again, we may omit \( m \) and (more often) \( f \)
- Initial state \( \iota \in Q \)

Frequently: uncontrolled (unscheduled) GTS

- \( Q \subseteq \text{Graph} \) and \( G_q = q \)
- Every transformation generates a (unique) transition
  - Here, \( q \rightarrow^{r,mf} q' \) whenever \( G_q \Rightarrow^{r,mf} G_{q'} \)
- \( S \) is completely determined by \( \langle R, \iota \rangle \)
GRAPH GRAMMARS

- Rule system \( R \) with initial graph defines *graph language*
  - Language of a GTS = set of graphs of reachable terminal states
  - For instance: language of trees, 2-coloured graphs, flow graphs
  - Generalises string grammars

- Common technique: every rule consumes a “non-terminal”
  - When all non-terminals are consumed, state is terminal
  - Context-freedom: LHS is *only* a single non-terminal

- Transition systems typically uncontrolled and infinite
GRAPH PRODUCTION SYSTEMS

- Rule system $R$ defines relation $\rho_R$ over graphs
  - $(G, H) \in \rho_R$ iff $H$ is the graph of a reachable state when $G_i = G$
  - $H$ is “produced” from $G$
- Often, $\rho_R$ is meant to be a (partial or total) function
  - Only (or at most) one reachable terminal state for any start graph
  - Which can be found (or its absence confirmed) quickly and reliably
- Transition systems typically finite
  - Infinite paths are very undesirable
  - Schedules can help to find short paths (“evaluation strategies”)
- Examples
  - Normal form computations
    - E.g., functional programming, theorem proving
  - Model transformation
    - E.g., “construct the flow graph from an abstract syntax graph”
GRAPH-BASED BEHAVIOURAL SEMANTICS

- Graph transition system describes evolution of system
  - Either trace set or full transition system is relevant
  - Often, reachable terminal state = deadlock = error

- Transition systems
  - Typically contain cycles
  - Typically are non-deterministic
  - May very well be infinite (though this is often an error)

- Control is often very useful
OUTLINE OF THIS TUTORIAL

- Framework for (graph) transformation
  - Rule+match+tracing morphism-labelled transition systems
  - Usage scenarios: grammars, production systems, semantics

- Composition mechanisms: when simple rules are not enough
  - Amalgamation
    - Multi-nodes
    - Nested rules
  - Parameters
    - Input, output
  - Supervisory control
    - Programmed graph transformation
    - Atomicity
    - Transformation units
  - Strategic control
AMALGAMATION

- Simple rules are limited
  - Effect is local and bounded / rules not generic
  - Example: rewrite (maximal) complete subgraph to star graph
  - Note: limitations can be advantageous!

- Idea: apply one or several rules simultaneously

- Formal interpretation
  - Take multiple matches of one or more rules (in the same graph)
  - Duplicate the rules per match and take their union
  - Apply the composed rule
  - Amalgamated rules may be nested, so $\text{union} \neq \text{disjoint union}$

- This is not always the same as repeatedly applying rules
  - All composed rules are applied to the same graph
  - Conflicts are resolved (or prevent rule application)
  - Matches cannot appear or disappear
GENERALISATION: FAMILIES OF RULES

1. Through amalgamation
   - Copying/gluing subrules arbitrary number of times

2. As the language of a grammar over rules
   - As seen yesterday in Vladimir Zamdzhiev’s presentation

   - I feel the latter is probably strictly more expressive
     - At least to express transformation in 1 rule

   - There are other well-known cases where amalgamation fails
     - Matching/processing all elements of a list
     - Copying a graph of arbitrary structure

   - Copied subrules cannot refer to one another
     - Context-free in some sense?
     - Requires second-order logic
SUPERVISORY CONTROL

- Explicitly determine the order of rule application
  - Programmed graph transformation

- Typical constructs
  - Try a rule, do something else if rule is not applicable
  - Do rules in sequence
  - As long as possible apply a rule/set of rules
RULE PARAMETERS

▪ Output parameters
  ▫ Expose part of the match on the label
  ▫ Primarily for observation

▪ Input parameters
  ▫ Partially determine the match
  ▫ Primarily for control
  ▫ Pragmatic reasons: to avoid “guessing” attribute values

Issue
  ▫ Node type parameters expose node identities
  ▫ Supposed to be internal/unknowable
TRANSACTIONS

- If next rule in a sequence fails, state is terminal
  - This may not be the intended meaning

- Transaction implies:
  - All-or-nothing behaviour
  - Backtrack & abandon path if it leads to terminal state
  - Abandoned part is *not* in the GTS!

- Implicit in the semantics of try/else and alap
  - Body of alap should “fail” on terminal states
  - Not just if first rule is inapplicable
TRANSFORMATION UNITS

- Named control abstractions
  - Signature consisting of (input and output) parameters
  - Control program as body

- Behave as (composed) rules
  - Single transition in GTS
  - Labelled by unit name & tracing morphism
  - Body is executed as transaction (= atomically)

- Groove: Recipes
  - Example: frogs
  - Freak example: fibonacci
STRATEGIC CONTROL

- Often, one does not want to explore entire transition system
  - State space is too large
  - State space known to be confluent

- Exploration strategies
  - Simulation mode
    - Linear exploration
  - Search mode, e.g. for property violations (LTL, invariant)
    - Depth-first rather than breadth-first
  - Optimisation mode: find “good” solution
    - Local rather than global optimum

- Heuristics
  - Decide which path to explore first
  - Problem-dependent vs. problem-independent

- *Supervisory control restricts LTS, strategic control does not!*
EVALUATION

- Why are simple rules not enough?
  - Effect only local, not generic
  - Require to put control elements into graphs
  - Granularity not appropriate for problem at hand
  - Monolithic, no reuse of common elements

- Composition mechanisms
  1. In space: families of rules
  2. In time: supervisory control, transformation units

- Disadvantages
  1. More complex rules: reasoning becomes harder
  2. Loss of declarative nature: reasoning becomes harder

- This is a fake objection!
  - Systems that benefit from composition mechanisms are complex
  - Composition partially relieves this, partially shifts it elsewhere