Graph Transformation and Intuitionistic Linear Logic

Paolo Torrini

pt95@mcs.le.ac.uk

University of Leicester
Work on Graph Transformation

- project SENSORIA (with Reiko Heckel), work package on model-driven development
- validation techniques for graph transformation systems — verification and simulation
- modelling of transition systems
  Petri net: markings and transitions
  Graph Transformation: graphs and transformation rules (higher level)
  — may use attributes, types, negative conditions
- Different approaches: algebraic (SPO and DPO), logic-based (monadic 2nd-order logic), operational
- Models of concurrency
Application of GT

- Model-driven development: generation of object-oriented code from models (e.g. UML class diagrams) through model transformation (refinement, refactoring), also automatically (e.g. Fujaba)

- Modelling of discrete event systems by transition rules: concurrent, interactive, reactive systems (e.g. simulation of P2P networks)

- Model properties: shapes in graphs, invariants in unfolding

- Verification of model properties: model-checking (LTL, CTL, CSL, modal logic), theorem-proving (HOL, 1st-order temporal logic), critical pair analysis
Concurrent/reactive systems

Validation of whole systems by model-checking or stochastic simulation

in case of large models, soft targets — e.g. quality of service agreements

Verification of digital components — code satisfying model properties, including low-level ones (e.g. use of memory)

Graph transformation — intuitive, general modelling paradigm
Typed hypergraphs

Hypergraph $G = \langle V, E, s \rangle$
$V$ nodes (vertices), $E$ (hyper)-edges
assignment $s : E \rightarrow V^*$

graph morphism $\langle \phi_V : V_1 \rightarrow V_2, \phi_E : E_1 \rightarrow E_2 \rangle$
assignment-preserving

type h-graph $TG = \langle \mathcal{V}, \mathcal{E}, \text{ar} \rangle$
$\mathcal{V}$ set of node types, $\mathcal{E}$ set of edge types
$\text{ar}(l) : \mathcal{E} \rightarrow \mathcal{V}^*$

$TG$-typed graph $(G, t)$, with $t : G \rightarrow TG$

$TG$-typed graph morphism $f : (G_1, t_1) \rightarrow (G_2, t_2)$
$f : G_1 \rightarrow G_2$ graph morphism, with $t_2 \circ f = t_1$
Graph Transformation

- Double-Pushout approach (DPO)

- Transformation rule $p : L \leftarrow K \rightarrow R$
  span of injective graph morphisms $(l, r)$, matched to a graph $G$ by morphism $d$ up to iso
  $L/K$ deleted, $R/K$ created, $K$ is the interface (read-only)

\[
\begin{array}{c}
L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
\downarrow m & & (1) & & \downarrow m^* \\
G & \xleftarrow{g} & D & \xrightarrow{h} & H
\end{array}
\]

- $m|_{L/K}$ and $m^*|_{R/K}$ are injective
- $\text{img}(d)$ and $\text{img}(m|_{L/K})$ are disjoint
- no dangling edges
Logic translation

- Operational characterisation of DPO-GTS — monoidal structure with restriction over node names
  - node names can be bound by restriction ($\nu$), edges as relations over nodes, parallel composition $\otimes$ ($\mathbf{1}$ for the empty graph)
- Translation to quantified extension of ILL
- easy translation of monoidal operations
- linear implication $\rightarrow$ to represent transformation
- universal quantifier: abstraction of interface elements
- restriction: more problematic — linear quantifier
- dependent-typing approach: linear $\lambda$-proof terms
List reverse

typedef struct node {
    struct node *nxt;
    int data;
} *List

List reverse(List x) {
    List y, t;
    y = NULL;
    while (x!=NULL) {
        t = y;
        y = x;
        x = x->nxt;
        x->nxt = t;
    }
}

ILL representation

Definitions

\[ ptlist(x, l) \rightarrow pt(x, Hd(l)) \otimes list(l) \]

\[ list(h\#l) \rightarrow nx(h, Hd(l)) \otimes list(l) \]

\[ list([]) \rightarrow 1, \quad Hd([]) = null \]

Initial state

\[ ptlist(x, l) \otimes pt(y, null) \]

Final state

\[ ptlist(x, []) \otimes ptlist(y, \text{rev}(l)) \]
ILL representation

- Transformation rules
  \[ \forall b, c. \exists a. pt(x, a) \otimes nx(a, b) \otimes pt(y, c) \rightarrow \]
  \[ \exists a, t. pt(t, c) \otimes pt(x, b) \otimes nx(a, b) \otimes pt(y, a) \]
  \[ \forall b, c. \exists a, t. pt(t, c) \otimes pt(x, b) \otimes nx(a, b) \otimes pt(y, a) \rightarrow \]
  \[ \exists a. pt(x, b) \otimes pt(y, a) \otimes nx(a, c) \]

- Refinement 1
  \[ \forall l_1, l_2. \exists h. ptlist(x, h \# l_1) \otimes ptlist(y, l_2) \rightarrow \]
  \[ \exists h, t. pt(y, h) \otimes ptlist(x, l_1) \otimes ptlist(t, l_2) \]
  \[ \forall l_1, l_2. \exists h, t. pt(y, h) \otimes ptlist(x, l_1) \otimes ptlist(t, l_2) \rightarrow \]
  \[ \exists h. ptlist(x, l_1) \otimes ptlist(y, h \# l_2) \]

- Refinement 2
  \[ ptlist(x, h \# l_1) \otimes ptlist(y, l_2) \rightarrow ptlist(x, l_1) \otimes ptlist(y, h \# l_2) \]
General idea

- specification of an imperative program, turned into a more declarative, functional one
- ILL can make it easier to represent declaratively imperative programs
- however, in the example we have assumed there is a binder that can be used to turn variables into local constants (names)
- names can be replaced equivariantly ($\alpha$-renaming), but cannot be identified by instantiation
- $\hat{\exists}$ is neither existential nor universal
- some analogy with freshness quantification
Renaming

- not a question of nominal logic, but of preserving isomorphically a structure of components
- renaming = injective morphisms
  — important, as a rule may be matched by different subgraphs
- components might be identified by complex terms (e.g. a list), hence also complex terms might be local constants
- general criterion: separate name spaces
- different names depend on disjoint (non-empty) subsets of the name space
- introducing new names extends the name space
Linearity

- related to linearity, but not quite the same
- linearity is about system components that occur exactly once — e.g. graph components
- rules — can be used many times, therefore declared as unbounded with !
- each name may occur many times, still is linearly associated to a subset of the name space
- makes little sense to consider the closure ! of a name-space — connection with separation logic more natural than with linear logic
Axioms and RB-quantifier

\[ \Gamma; \cdot; u :: \alpha \vdash u :: \alpha \] \hspace{1cm} \text{LId} \\
\[ \Gamma, x :: \alpha; \cdot \vdash x :: \alpha \] \hspace{1cm} \text{UIId}

Conditions: one-side freshness, name-space separation

\[ FV(D) \cap FV(\Sigma) = \emptyset \] \\
\[ \Gamma_1; \cdot \vdash D :: \beta \] \\
\[ \frac{\Gamma_1; \cdot \vdash D :: \beta \quad \Gamma_1, \Gamma_2; \Sigma; \Delta \vdash M :: \alpha[D/x]}{\Gamma_1, \Gamma_2; \Sigma, n :: \beta \downharpoonright D; \Delta \vdash \hat{\exists}D.M :: \hat{\exists}x : \beta.\alpha} \] \hspace{1cm} \\hat{\exists}R

\[ \Gamma, z :: \beta; \Sigma, n :: \beta \downharpoonright z; \Delta, v :: \alpha \vdash N :: \gamma \] \\
\[ \frac{\Gamma; \Sigma; \Delta, u :: \hat{\exists}z : \beta.\alpha \vdash \text{let } \hat{\exists}z.v = u \text{ in } N :: \gamma}{\hat{\exists}L} \]
Universal quantifier

\[
\Gamma, x :: \beta; \Sigma; \Delta \vdash M :: \alpha \\
\frac{}{\Gamma; \Sigma; \Delta \vdash \lambda x. M :: \forall x : \beta. \alpha} \forall R
\]

\[
\Gamma; \cdot, \cdot \vdash D :: \beta \quad \Gamma; \Sigma; \Delta, v :: \alpha[D/x] \vdash N :: \gamma \\
\frac{}{\Gamma; \Sigma; \Delta, u :: \forall x : \beta. \alpha \vdash \text{let } v = uD \text{ in } N :: \gamma} \forall L
\]
Tensor

\[ \Gamma; \Sigma_{1}; \Delta_{1} \vdash M :: \alpha \quad \Gamma; \Sigma_{2}; \Delta_{2} \vdash N :: \beta \quad FV(\Sigma_{1}) \cap FV(\Sigma_{2}) = \emptyset \]

\[ \Gamma; \Sigma_{1}, \Sigma_{2}; \Delta_{1}, \Delta_{2} \vdash M \otimes N :: \alpha \otimes \beta \]

\[ \Gamma; \Sigma; \Delta, u :: \alpha, v :: \beta \vdash N :: \gamma \]

\[ \Gamma; \Sigma; \Delta, w :: \alpha \otimes \beta \vdash \text{let } u \otimes v = w \text{ in } N :: \gamma \]
Linear implication

\[
\begin{align*}
\Gamma; \Sigma; \Delta, u :: \alpha & \vdash M :: \beta \\
\Gamma; \Sigma; \Delta & \vdash \lambda u : \alpha. M :: \alpha \rightarrow \beta \quad \rightarrow \ R
\end{align*}
\]

\[
\begin{align*}
\Gamma; \Sigma_1; \Delta_1 & \vdash M :: \alpha \\
\Gamma; \Sigma_2; \Delta_2, u :: \beta & \vdash N :: \gamma \\
FV(\Sigma_1) \cap FV(\Sigma_2) & = \emptyset \\
\Gamma; \Sigma_1, \Sigma_2; \Delta_1, \Delta_2, v :: \alpha \rightarrow \beta & \vdash \mathbf{let} \ u = v^M \ \mathbf{in} \ N :: \gamma \quad \rightarrow \ L
\end{align*}
\]
GTS translation

- (closed) h-graph as (closed) formula
  \[ \exists x : A. \gamma \]

- Sequence of typed variables, either \( \gamma = 1 \) or \( \gamma = L_1 (x_1) \otimes \ldots \otimes L_k (x_k) \)

- Adequacy of h-graph representation

- Transformation rule as closed formula
  \[ \forall x : A. \alpha \rightarrow \beta \]

  with \( \alpha, \beta \) graph formulas
Transformation rules

Consequence relation as transformation:

∀ for interface nodes,

∃̣ for deleted/created nodes (matches injective morphisms components)

\[ \vdash \alpha_G \circ \alpha_{G'} = \exists y. \alpha_L[y \leftarrow^d x] \otimes \alpha_C \]

\[ \vdash \alpha_H \circ \alpha_{H'} = \exists y. \alpha_R[y \leftarrow^d x] \otimes \alpha_C \]

\[ \sqrt{x}. \alpha_L \circ \alpha_R \vdash \alpha_G \circ \alpha_H \]

\[ p,m \rightarrow \]
Quantifier and congruence

\( \exists \) satisfies properties of renaming, exchange and distribution over \( \otimes \)

\[ \vdash (\exists x : \alpha.\beta(x)) \leftrightarrow (\exists y : \alpha.\beta(y)) \]

\[ \vdash (\exists xy : \alpha.\gamma) \leftrightarrow (\exists yx : \alpha.\gamma) \]

\[ \vdash (\exists x : \alpha.\beta \otimes \gamma(x)) \leftrightarrow (\beta \otimes \exists x : \alpha.\gamma(x)) \quad (x \text{ not in } \alpha) \]

Equivalence between \( \alpha \) and \( \exists x. \alpha \) generally fails in both directions, even when \( x \) does not occur free in \( \alpha \)
Incorrect DPO matches — examples

L

K

L

L

CISA – p. 22
... duly falsified

\[ \forall x : \beta. (\exists y : \beta. \alpha(y, y)) \rightarrow \exists y : \beta. \alpha(y, x) \]

\[ y \text{ and } x \text{ should be instantiated with the same term — against the freshness condition in } \exists \text{ introduction} \]

\[ \forall x : \beta. (\exists x : \beta. \alpha_1(x) \otimes \alpha_2(x)) \rightarrow (\exists x : \beta. \alpha_1(x)) \otimes \exists x : \beta. \alpha_2(x) \]

\[ \text{the two bound variables in the consequence require distinct resources and refer to distinct occurrences} \]
Reachability

Transformation — $G_0, G_1$ closed h-graphs, $G_0$ initial, $P_1, \ldots, P_k$ rules

- $G_1$ reachable from by some application of the rules

\[ !P_1, \ldots, !P_k, G_0 \vdash G_1 \]

- $G_1$ reachable by applying each rule once

\[ P_1, \ldots, P_k, G_0 \vdash G_1 \]

- Translation complete with respect to reachability (sequent provable if graph reachable)
Conclusion and further work

- Proof theory-driven approach to GT
- uses resource logic
- resource-bound quantifier to deal with restriction
- theorem proving: work in progress on shallow embedding in higher-order logic (HOL, CIC)