Encoding Graph Transformation in Linear Logic

Paolo Torrini
joint work with Reiko Heckel
pt95@mcs.le.ac.uk

University of Leicester
Graph Transformation Systems (GTS) — high-level approach to system modelling, UML, model-driven development, stochastic simulation

Existing formalisations — algebraic-categorical (SPO, DPO), 2nd-order predicate logic

High-level character, strong mathematical foundation

Double-pushout (DPO) — mature approach, based on category theory
Linear Logic

- Linear logic — can handle resources at the propositional level, by dropping Weakening and Contraction.
- Intuitionistic variant (ILL)
- Linearity — each premise used exactly once in a deduction, each argument used once by a function.
- Non-linearity recovered by means of !.
- Interesting proof theory (natural deduction, sequent calculus), various implementations (declarative languages, logical frameworks).
Encoding GT in LL — why?

- LL close to process algebras (Abramsky, Pfenning, Cervesato)
- Parallel composition ($\alpha \otimes \beta$), choice ($\alpha \& \beta$), reachability ($\vdash \alpha \rightarrow \beta$), replication ($!$)
- Semantic motivation: taking closer graph transformation and process algebra
- Existing approach: hyperedge replacement
- What we do: logic-based hyperedge replacement
- Practical motivation: making proofs about GTS easier
**Typed hypergraphs**

- Hypergraph $G = \langle V, E, s \rangle$
  - $V$ set of nodes, $E$ set of hyperedges
  - assignment $s : E \rightarrow V^*$

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

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

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

- $TG$-typed h-graph morphism $f : (G_1, t_1) \rightarrow (G_2, t_2)$
  - is h-morphism $f : G_1 \rightarrow G_2$ with $t_2 \circ f = t_1$
DPO diagram

Graph transformation rule \( p : L \xleftarrow{l} K \xrightarrow{r} R \)
span of typed h-graph morphisms \((l, r)\),
\(K\) interface, \(L/K\) to be deleted, \(R/K\) to be created,
rule application determined by match morphism \(m\),
\(m\) determined up to iso by interface morphism \(d\)

DPO conditions — (1) Identification condition:
(a) \(m\) never identifies distinct \(L/K\) elements
(b) \(m\) never identifies \(L/K\) elements with \(K\) ones
(2) Dangling condition: for each node \(n \in L/K\), all edges connected to \(n\) are in \(L/K\), too

\[
\begin{array}{ccc}
L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
\downarrow m & & \downarrow d & & \downarrow m^* \\
G & \xleftarrow{g} & D & \xrightarrow{h} & H
\end{array}
\]
Graph expressions

- Algebraic characterisation of DPO-GTS:
  - edge as predicates over nodes, empty graph, parallel composition,
  - restriction for nodes

- Graph constituent $C = e(n_1, \ldots, n_k) \mid \text{Nil} \mid C_1 \parallel C_2 \mid vn.C$

- Implicit typing — $n : A, \quad e(n_1, \ldots, n_k) : L(A_1, \ldots, A_k)$

- Graph expression $X \vdash C$
  - $X \subseteq V$ is graph interface — generalisation of rule interface, includes the free nodes of $C$ and free isolated nodes

- closed GE has empty interface
Structural congruence

\[ C \equiv C' \]

\[ \parallel \] — associative, commutative
Nil — neutral element

\[ vn. C \equiv vm. C[m/n], \text{ if } m \text{ does not occur free in } C. \]

\[ vn.vm.C \equiv vm.vn.C \]

\[ vn.(C_1 \parallel C_2) \equiv C_1 \parallel (vn.C_2) \]
if \( n \) does not occur free in \( C_1 \)

\[ X \models C \equiv Y \models C' \text{ iff } X = Y \text{ and } C \equiv C' \]
Transformation

- $E_1 = K \vdash L$ and $E_2 = K \vdash R$
- GEs sharing no free isolated nodes

- $\Lambda \bar{x}.L \xrightarrow{p} R$ rule expression for $p : L \xleftarrow{l} K \xrightarrow{r} R$
- $\bar{x} = x_1, \ldots, x_k$ represents $K$ as sequence of variables

- restriction to node interfaces (no edges in $K$)

- Application of $p$ at match $m$ ($G$ closed GE), schema satisfies DPO conditions

\[
\begin{align*}
\Lambda \bar{x}.L \xrightarrow{p} R & \quad G \equiv \nu \bar{n}.L[\bar{n} \leftarrow^d \bar{x}] \parallel C \\
H \equiv \nu \bar{n}.R[\bar{n} \leftarrow^d \bar{x}] \parallel C \\
\xrightarrow{p,d} \quad G \xrightarrow{p,m} H
\end{align*}
\]
Overall plan

- Algebraic characterisation of DPO-GTS — hyperedge replacement-style (difference: isolated nodes)
- Translation to a quantified extension of ILL
- up to iso (typing, connectivity): edge expressions unvaried, Nil as $\mathbf{1}$, $\parallel$ as $\otimes$, $\nu$ as $\hat{F}$, $\implies$ as $\circlearrowleft$, $\Lambda$ as $\forall$
- Nodes: occur as non-linear terms in edge expressions, but need linear treatment to meet DPO conditions
- full translation maps expressions to derivations, and involves proof terms (linear $\lambda$-calculus)
- terms represent identity of nodes and edges
- We translate individual graphs, then forget about terms and reason up to isomorphism
Normal forms

(closed) h-graph as (closed) formula

\[ \exists x : A. \gamma \]

Sequence of typed variables, either \( \gamma = 1 \) or \( \gamma = L_1 (\bar{x}_1) \otimes \ldots \otimes L_k (\bar{x}_k) \)

Adequacy of h-graph representation

Transformation rule as closed formula

\[ \forall x : A. \alpha \rightarrow \beta \]

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

- Soundness — work in progress, general idea — logically valid implications are “read-only” transformations
QILL

- ILL extended with 1st-order quantification
- Labels attached to premises (identity of occurrences)
- Double-entry sequents — linear premises ($\Delta$) and non-linear ones ($\Gamma$, equivalent to $!\Gamma$)

$$\Gamma = x :: (\alpha : \text{term}), \ldots, p :: (\beta : \text{form}), \ldots$$

$$\Delta = u :: (\alpha : \text{form}), \ldots$$

- Proof-terms based on linear $\lambda$-calculus
- Sequents representing derivations

$$\Gamma;\Delta \vdash N :: (\alpha : \tau)$$
Proof system — language

\[ \alpha = A : \text{term} \mid L(N_1, \ldots, N_n) \mid 1 \mid \alpha_1 \otimes \alpha_2 \mid \alpha_1 \rightarrow \alpha_2 \mid !\alpha_1 \mid \alpha_1 \& \alpha_2 \mid \forall x : \beta.\alpha \mid \exists x : \beta.\alpha \mid \alpha \downarrow N \mid \alpha = \alpha \]

\[ M = x \mid p \mid u \mid \text{nil} \mid N_1 \otimes N_2 \mid \lambda x.N \mid \lambda u.N \mid N_1 \hat{\rightarrow} N_2 \mid N_1 N_2 \mid M \mid \langle N_1, N_2 \rangle \mid \text{fst} N \mid \text{snd} N \mid \text{id}_\alpha \]

\[ \alpha \hat{=} \beta =_df (\alpha \rightarrow \beta) \& (\beta \rightarrow \alpha) \]

\[ \alpha \#(x, N) =_df (\alpha[N/x])[x/N] = \alpha \]

meaning \( N \) does not occur free in \( \exists x.\alpha \)

let \( P = N_1 \) in \( N_2 \) =_df \( (\lambda P.N_2)N_1 \)

where \( P \) is a term pattern (does not contain abstractions)

\[ \hat{\varepsilon}(N_1|N_2).N_3 =_df N_1 \otimes! N_2 \otimes N_3 \]
Application schema

Γ; Δ ⊢ ∀x : A_x.α_L → α_R
Γ; · ⊢ α_G ≡ α_{G}'
Γ; · ⊢ α_H ≡ α_{H}'

α_{G}' = \exists z : A_z.α_L[z : A_z \leftarrow^d x : A_x] ⊗ α_C
α_{H}' = \exists z : A_z.α_R[z : A_z \leftarrow^d x : A_x] ⊗ α_C

Γ; Δ ⊢ α_G → α_H

⟨p,m⟩ →
Embedding h-graphs

- H-graphs: edge components, empty graph Nil (1) and parallel composition $||$ ($\otimes$) — straightforward
- restriction $\nu$ — more problematic
- standard quantification ($\forall, \exists$) in ILL deals with non-linear terms
- at first sight — OK, nodes may have multiple occurrences in edge expressions, all we need is to handle edges linearly edge
  - we could map $\nu$ to $\exists$
    — after all, $\nu$ distributes over $||$, $\exists$ over $\otimes$
  - not enough to meet DPO conditions
Quantifier and DPO conditions

- \( \kappa (\exists x: \beta. \alpha(x, x)) \rightarrow \exists y: \beta. \alpha(x, y) \)
  the resource for \( x \) cannot suffice for \( x \) and \( y \).

- \( \kappa \forall x: \beta. \beta \upharpoonright x \otimes \alpha(x, x) \rightarrow \exists y: \beta. \alpha(y, x) \)
y and \( x \) should be instantiated with the same term — blocked by the freshness condition in \( \exists \) introduction

- \( \kappa (\exists yx: \beta. \alpha_1(x) \otimes \alpha_2(x)) \rightarrow (\exists x: \beta. \alpha_1(x)) \otimes \exists x: \beta. \alpha_2(x) \)
  the two bound variables in the consequence require distinct resources and refer to distinct occurrences
Incorrect matches
RBQ introduction

\[ \Gamma; \Delta \vdash M :: \alpha[N/x] \quad \Gamma; \cdot \vdash N :: \beta \]
\[ \Gamma; \Delta' \vdash n :: \beta \downarrow \!
\]
\[ \Gamma, x :: \beta; \cdot \vdash id_\alpha :: (\alpha[N/x])[x/N] = \alpha \]
\[ \Gamma; \Delta, \Delta' \vdash (!N \otimes n) \otimes M :: \exists x : \beta.\alpha \]

(1) \(\alpha[N/x]\) graph with \(N\) in place of free \(x\)
(2) \(N\) well-typed — enough to restrict \(N\) by \(x\)? No!
(3) to restrict (3) there has to be a node (linear resource) named by \(N\) — \(\downarrow\) denotes lifting of type from term to formula with naming reference to term
(4) moreover (4) \(N\) does not occur in \(\alpha\) (unless \(N = x\)) — a freshness condition, here formalised using type equality and substitution
**RBQ elimination**

\[ \Gamma; \Delta_1 \vdash M : \exists x : \beta. \alpha \quad \Gamma, x : \beta; \Delta_2, n : \beta \vdash x, v : \alpha \vdash N : \gamma \]

\[ \Gamma; \Delta_1, \Delta_2 \vdash \text{let } (\!x \otimes n) \otimes v = M \text{ in } N : \gamma \]

- Standard elimination rule
- since we restrict only introduction, normalisation applies at least as with \( \exists \)
- \( \exists I \) and \( \exists E \) can be used to simulate restriction/unrestriction operationally in the logic as steps in the construction/ destruction of graph expressions

\[ \Gamma; \cdot \vdash N : \alpha \quad \Gamma; n : \alpha \vdash N \vdash n : \alpha \vdash N \]

\[ \Uparrow A \]

GTLL – p. 20
Conclusion and further work

- Proof theory-driven approach to GT
- uses resource logic
- new quantifier to deal with restriction
- two-level embedding approach
- Interest in mechanised theorem proving
- Extension to generalised interfaces
- Stochastic GTS
rules I

\[ \Gamma; u :: \alpha \vdash u :: \alpha \quad \text{Id} \]
\[ \Gamma, p :: \alpha; \cdot \vdash p :: \alpha \quad \text{UId} \]
\[ \Gamma, x :: \alpha; \cdot \vdash x :: \alpha \quad \text{Eq} \]
\[ \Gamma; \cdot \vdash \text{id}_\alpha :: \alpha = \alpha \quad \text{Eq} \]

\[ \begin{align*} & \Gamma; \Delta_1 \vdash M :: \alpha \quad \Gamma; \Delta_2 \vdash N :: \beta \\ & \quad \Gamma; \Delta_1, \Delta_2 \vdash M \otimes N :: \alpha \otimes \beta \quad \otimes \text{I} \end{align*} \]
\[ \begin{align*} & \Gamma; \Delta_1 \vdash M :: \alpha \otimes \beta \quad \Gamma; \Delta_2, u :: \alpha, v :: \beta \vdash N :: \gamma \\ & \quad \Gamma; \Delta_1, \Delta_2 \vdash \text{let } u \otimes v = M \text{ in } N :: \gamma \quad \otimes \text{E} \end{align*} \]

\[ \begin{align*} & \Gamma; \Delta, u :: \alpha \vdash M :: \beta \\ & \quad \Gamma; \Delta \vdash \hat{\lambda}u : \alpha. M :: \alpha \rightarrow \beta \quad \rightarrow \text{I} \end{align*} \]
\[ \begin{align*} & \Gamma; \Delta_1 \vdash M :: \alpha \rightarrow \beta \quad \Gamma; \Delta_2 \vdash N :: \alpha \\ & \quad \Gamma; \Delta_1, \Delta_2 \vdash M \triangleleft N :: \beta \quad \rightarrow \text{E} \end{align*} \]
rules II

\[
\begin{align*}
\Gamma; \cdot \vdash \text{nil} :: 1 \quad & 1I \\
\Gamma; \Delta \vdash M :: \alpha & \quad \Gamma; \Delta \vdash N :: \beta \\
\Gamma; \Delta \vdash \langle M, N \rangle :: \alpha \land \beta & \quad \&I \\
\Gamma; \Delta \vdash \text{fst} \; M :: \alpha & \quad \&E1 \\
\Gamma; \cdot \vdash \text{fst} \; M :: \alpha & \quad \&E2 \\
\Gamma; \cdot \vdash M :: \alpha & \quad !I \\
\Gamma; \Delta_1 \vdash M :: !\alpha & \quad \Gamma, \cdot \vdash M :: \alpha \\
\Gamma; \Delta_1, \Delta_2 \vdash \text{let} \; p = M \; \text{in} \; N :: \beta & \quad !E \\
\Gamma; \Delta \vdash \lambda x. \; M :: \forall x : \beta. \alpha & \quad \forall I \\
\Gamma; \Delta \vdash M :: \forall x : \beta. \alpha & \quad \Gamma; \cdot \vdash N :: \beta \\
\Gamma; \Delta \vdash MN :: \alpha[N/x] & \quad \forall E
\end{align*}
\]
Translation — I

Constituents

\[
\begin{align*}
\llbracket e_i(m, \ldots, n) : L_i(A_m, \ldots, A_n) \rrbracket &= df \text{ Id } [\Gamma; ; c_i :: L_i(x_m, \ldots, x_n)] \\
\llbracket \text{Nil} \rrbracket &= df \text{ 1I } [\Gamma] \\
\llbracket M \parallel N \rrbracket &= df \otimes I [\llbracket M \rrbracket; ; \llbracket N \rrbracket] \\
\llbracket \nu n : A.N \rrbracket &= df \exists I [\llbracket N \rrbracket; ; \text{UId } [\Gamma; ; x_n :: A]; ; \text{Id } [\Gamma; ; n :: A \downarrow x_n]; ; \Gamma, y :: A; \vdash \text{id : MainType(\llbracket N \rrbracket)[y/x_n]\#(y, x_n)]}
\end{align*}
\]
Translation — II

Graph interfaces

\[
\begin{align*}
\llbracket n : A \rrbracket &= d_f \operatorname{Id} [\Gamma, x :: A]; \ n :: A \downharpoonright x] \\
\llbracket \{n : A\}\rrbracket &= d_f \llbracket n : A \rrbracket \\
\llbracket \{n_1 : A_1\} \cup X \rrbracket &= d_f \otimes I \llbracket \{n_1 : A_1\}\rrbracket; \llbracket X \rrbracket 
\end{align*}
\]

Graph expressions

\[
\llbracket X \vdash C \rrbracket = d_f \otimes I \llbracket [X] \rrbracket; \llbracket C \rrbracket
\]
Graph derivations

- **Graph formulas** — $1, \otimes, \exists, \downarrow$ fragment of the logic containing only primitive graph types (node and edge types)

- **Graph context** — multiset of typed nodes and typed edge components.

- **Graph derivation** — derivable sequent $\Gamma; \Delta \vdash N :: \gamma$, where $\gamma$ is a graph formula, $\Delta$ is a graph context, $\Gamma$ the environment, $N$ a normal derivation.

- Uses only axioms and the introduction rules $1I, \otimes I, \exists I$. 
Quantifier and congruence

\[ \exists \text{ satisfies properties of renaming, exchange and distribution over } \otimes \]

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

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

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

Equivalence between \( \alpha \) and \( \exists x. \alpha \) generally fails in both directions, even when \( x \) does not occur free in \( \alpha \)
Graphs and types — adequacy

- Isomorphism between graph expressions and graph derivations
- Isomorphism between graphs (congruence classes of graph expressions) and graph formulas modulo linear equivalence
- Curry-Howard style correspondence
- Possibility to implement hypergraphs and to reason about them
Graph transformation

- Less interested in component identity, higher-level translation, based on logic formulas
- Linear implication as transformation
- Standard quantifier for interface nodes
- Rule names as non-linear resources (unlimited application)

\[
\begin{align*}
\left[ M \implies N \right]^T &= d_f \left[ M \right]^T \leadsto \left[ N \right]^T \\
\left[ \forall x : A. N \right]^T &= d_f \forall x : A. \left[ N \right]^T \\
\left[ \pi(p) \right] &= d_f \text{FId} \left[ \Gamma ; ; \ p :: \forall x : A_x. \left[ L \right]^T \leadsto \left[ R \right]^T \right]
\end{align*}
\]
Completeness and soundness

Let $\Gamma_P = \Sigma \cup \{\rho | \rho = \llbracket \pi(p) \rrbracket^T, p \in P\}$, then for each reachable h-graph $G$

$$\Gamma_P; \llbracket G_0 \rrbracket^T \vdash \llbracket G \rrbracket^T$$

Let $R$ be a multiset of transformations, $\Delta_R = \{\tau | \tau = \llbracket t \rrbracket^T, t \in R\}$, then for each h-graph $G$ reachable from $G_0$ by executing $R$

$$\Sigma; \llbracket G_0 \rrbracket^T, \Delta_R \vdash \llbracket G \rrbracket^T$$

This is for completeness

Soundness requires more work on the interpretation of linear implication