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 $G = \langle V, E, \mathsf{s} \rangle$
  V nodes (vertices), E (hyper)-edges
  assignment $\mathsf{s} : E \to V^*$
- graph morphism — $\langle \phi_V : V_1 \to V_2,\ \phi_E : E_1 \to E_2 \rangle$
  assignment-preserving
- type h-graph $TG = \langle \mathcal{V}, \mathcal{E}, \mathsf{ar} \rangle$
  $\mathcal{V}$ set of node types, $\mathcal{E}$ set of edge types
  $\mathsf{ar}(l) : \mathcal{E} \to \mathcal{V}^*$
- TG-typed graph (G, t), with $\mathsf{t} : G \to TG$
- TG-typed graph morphism $f : (G_1, \mathsf{t}_1) \to (G_2, \mathsf{t}_2)$
  $f : G_1 \to G_2$ graph morphism, with $\mathsf{t}_2 \circ f = \mathsf{t}_1$

Graph Transformation

- Double-Pushout approach (DPO)
- Transformation rule $p : L \xleftarrow{l} K \xrightarrow{r} R$
  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)

$$\begin{tikzcd} L \arrow[d, "m"'] \arrow[dr, phantom, "(1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "d"] \arrow[dr, phantom, "(2)"] & R \arrow[d, "m^*"] \\ G & D \arrow[l, "g"] \arrow[r, "h"'] & H \end{tikzcd}$$

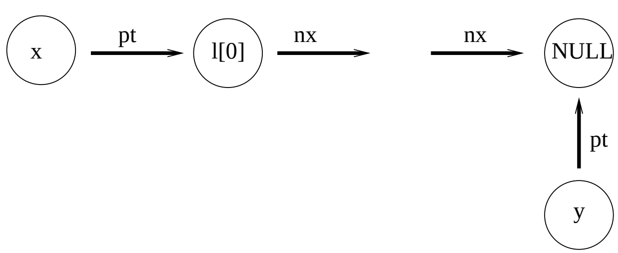
- $m|_{L/K}$ and $m^*|_{R/K}$ are injective
- $img(d)$ and $img(m|_{L/K})$ are disjoint
- no dangling edges

Logic translation

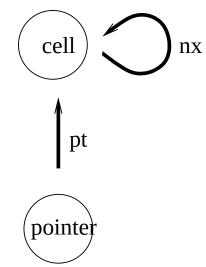
- Operational characterisation of DPO-GTS — monoidal structure with restriction over node names
- node names can be bound by restriction (ν),
  edges as relations over nodes,
  parallel composition $\otimes$ ($\mathbf{1}$ for the empty graph)
- Translation to quantified extension of ILL
- easy translation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

List reverse

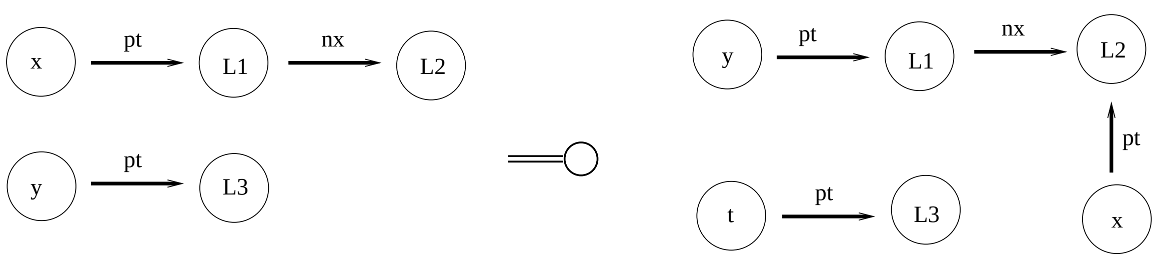
```
typedef struct node {
   struct node *nxt;
   int data;
} *List

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



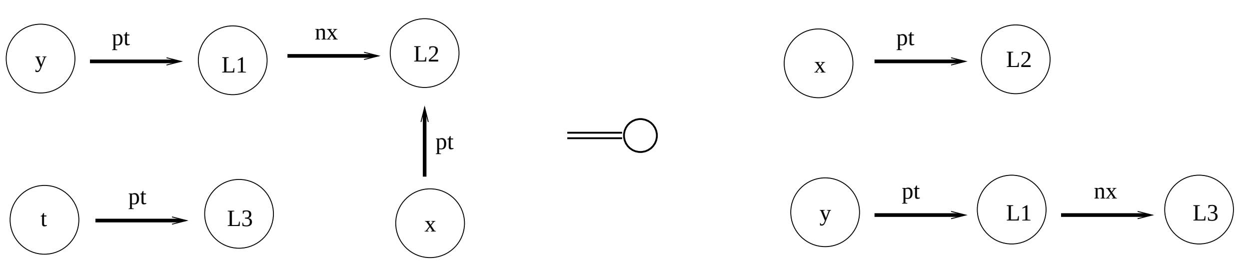
INITIAL STATE



TYPE GRAPH



RULE 1



RULE 2

ILL representation

- Definitions

$$ptlist(x, l) \multimap pt(x, Hd(l)) \otimes list(l)$$

$$list(h\#l) \multimap nx(h, Hd(l)) \otimes list(l)$$

$$list([]) \multimap \mathbf{1}, \qquad Hd([]) = null$$

- Initial state

$$ptlist(x, l) \otimes pt(y, null)$$

- Final state

$$ptlist(x, []) \otimes ptlist(y, rev(l))$$

ILL representation

- Transformation rules
  $\forall b, c.\hat{\exists} a.pt(x,a) \otimes nx(a,b) \otimes pt(y,c) \multimap$
  $\quad \hat{\exists} a, t.pt(t,c) \otimes pt(x,b) \otimes nx(a,b) \otimes pt(y,a)$
  $\forall b, c.\hat{\exists} a, t.pt(t,c) \otimes pt(x,b) \otimes nx(a,b) \otimes pt(y,a) \multimap$
  $\quad \hat{\exists} a.pt(x,b) \otimes pt(y,a) \otimes nx(a,c)$
- Refinement 1
  $\forall l_1, l_2.\hat{\exists} h.ptlist(x, h\#l_1) \otimes ptlist(y, l_2) \multimap$
  $\quad \hat{\exists} h, t.pt(y,h) \otimes ptlist(x, l_1) \otimes ptlist(t, l_2)$
  $\forall l_1, l_2.\hat{\exists} h, t.pt(y,h) \otimes ptlist(x, l_1) \otimes ptlist(t, l_2) \multimap$
  $\quad \hat{\exists} h.ptlist(x, l_1) \otimes ptlist(y, h\#l_2)$
- Refinement 2
  $ptlist(x, h\#l_1) \otimes ptlist(y, l_2) \multimap ptlist(x, l_1) \otimes ptlist(y, h\#l_2)$

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 (α-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

$$\frac{}{\Gamma; \cdot; u :: \alpha \vdash u :: \alpha} \; LId \qquad \frac{}{\Gamma, x :: \alpha; \cdot; \cdot \vdash x :: \alpha} \; UId$$

Conditions: one-side freshness, name-space separation

$$\frac{\begin{array}{ll} FV(D) \cap FV(\Sigma) = \emptyset & \Gamma_2, x :: \beta; \cdot; \cdot \vdash N :: \alpha \multimap \alpha \\ \Gamma_1; \cdot; \cdot \vdash D :: \beta & \Gamma_1, \Gamma_2; \Sigma; \Delta \vdash M :: \alpha[D/x] \end{array}}{\Gamma_1, \Gamma_2; \Sigma, n :: \beta \downharpoonleft D; \Delta \vdash \hat{\varepsilon} D.M :: \hat{\exists} x : \beta.\alpha} \; \hat{\exists} R$$

$$\frac{\Gamma, z :: \beta; \Sigma, n :: \beta \downharpoonleft z; \Delta, v :: \alpha \vdash N :: \gamma}{\Gamma; \Sigma; \Delta, u :: \hat{\exists} z : \beta.\ \alpha \vdash \text{let } \hat{\varepsilon} z.v = u \text{ in } N :: \gamma} \; \hat{\exists} L$$

Universal quantifier

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

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

Tensor

$$\frac{\Gamma;\Sigma_1;\Delta_1 \vdash M :: \alpha \quad \Gamma;\Sigma_2;\Delta_2 \vdash N :: \beta \quad FV(\Sigma_1) \cap FV(\Sigma_2) = \emptyset}{\Gamma;\Sigma_1,\Sigma_2;\Delta_1,\Delta_2 \vdash M \otimes N :: \alpha \otimes \beta} \otimes R$$

$$\frac{\Gamma;\Sigma;\Delta, u :: \alpha, v :: \beta \vdash N :: \gamma}{\Gamma;\Sigma;\Delta, w :: \alpha \otimes \beta \vdash \mathsf{let}\ u \otimes v = w\ \mathsf{in}\ N :: \gamma} \otimes L$$

Linear implication

$$\frac{\Gamma; \Sigma; \Delta, u :: \alpha \vdash M :: \beta}{\Gamma; \Sigma; \Delta \vdash \hat{\lambda} u : \alpha.\, M :: \alpha \multimap \beta} \multimap R$$

$$\frac{\begin{array}{ll}\Gamma; \Sigma_1; \Delta_1 \vdash M :: \alpha & \Gamma; \Sigma_2; \Delta_2, u :: \beta \vdash N :: \gamma \\ FV(\Sigma_1) \cap FV(\Sigma_2) = \emptyset & \end{array}}{\Gamma; \Sigma_1, \Sigma_2; \Delta_1, \Delta_2, v :: \alpha \multimap \beta \vdash \mathsf{let}\ u = v\hat{\ } M\ \mathsf{in}\ N :: \gamma} \multimap L$$

GTS translation

- (closed) h-graph as (closed) formula

$$\hat{\exists}\overline{x : A}.\gamma$$

  $\overline{x : A}$ sequence of typed variables,
  either $\gamma = \mathbf{1}$ or $\gamma = L_1\left(\overline{x}_1\right) \otimes \ldots \otimes L_k\left(\overline{x}_k\right)$

- Adequacy of h-graph representation
- Transformation rule as closed formula

$$\forall\overline{x : A}.\alpha \multimap \beta$$

  with α, β graph formulas

Transformation rules

Consequence relation as transformation:

$\forall$ for interface nodes,

$\hat{\exists}$ for deleted/created nodes (matches injective morphisms components)

$$\frac{\begin{array}{ll} \vdash \alpha_G \multimapboth \alpha_{G'} & \alpha_{G'} = \hat{\exists}\overline{y}.\alpha_L[\overline{y} \xleftarrow{d} \overline{x}] \otimes \alpha_C \\ \vdash \alpha_H \multimapboth \alpha_{H'} & \alpha_{H'} = \hat{\exists}\overline{y}.\alpha_R[\overline{y} \xleftarrow{d} \overline{x}] \otimes \alpha_C \end{array}}{\forall\overline{x}.\alpha_L \multimap \alpha_R \vdash \alpha_G \multimap \alpha_H} \overset{p,m}{\Longrightarrow}$$

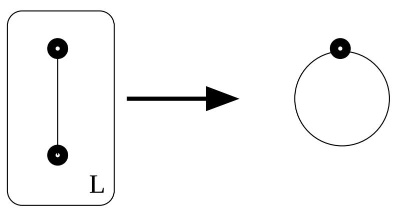
Quantifier and congruence

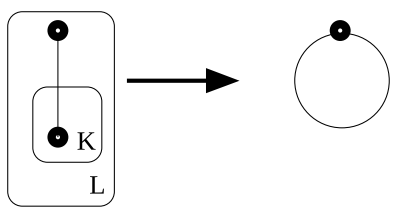
$\hat{\exists}$ satisfies properties of renaming, exchange and distribution over $\otimes$

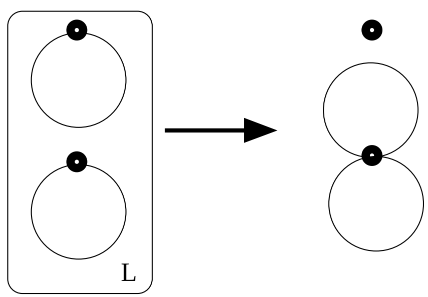
- $\vdash (\hat{\exists} x : \alpha.\beta(x)) \multimapboth (\hat{\exists} y : \alpha.\beta(y))$
- $\vdash (\hat{\exists} xy : \alpha.\gamma) \multimapboth (\hat{\exists} yx : \alpha.\gamma)$
- $\vdash (\hat{\exists} x : \alpha.\beta \otimes \gamma(x)) \multimapboth (\beta \otimes \hat{\exists} x : \alpha.\gamma(x))$ (x not in α)

Equivalence between α and $\hat{\exists} x.\ \alpha$ generally fails in both directions, even when x does not occur free in α

Incorrect DPO matches — examples







... duly falsified

- $\nvdash (\hat{\exists} x : \beta.\ \alpha(x, x)) \multimap \hat{\exists} xy : \beta.\ \alpha(x, y)$
  the resource for x is not enough for x and y.
- $\nvdash \forall x : \beta.\ (\hat{\exists} y : \beta.\ \alpha(y, y)) \multimap \hat{\exists} y : \beta. \alpha(y, x)$
  y and x should be instantiated with the same term — against the freshness condition in $\hat{\exists}$ introduction
- $\nvdash (\hat{\exists} yx : \beta.\ \alpha_1(x) \otimes \alpha_2(x)) \multimap (\hat{\exists} x : \beta. \alpha_1(x)) \otimes \hat{\exists} x : \beta. \alpha_2(x)$
  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)