Resource-bound quantification for graph transformation

Paolo Torrini
University of Leicester
pt95@mcs.le.ac.uk

Reiko Heckel
University of Leicester
reiko@mcs.le.ac.uk

Graph transformation has been used to model concurrent systems in software engineering, as well as in biochemistry and life sciences. The application of a transformation rule can be characterised algebraically as a construction of a double-pushout (DPO) diagram in the category of graphs. We show how Intuionistic Linear Logic (ILL) can be extended with resource-bound quantification, allowing for an implicit handling of the DPO conditions, and how resource logic can be used to reason about graph transformation systems.

1 Introduction

Graphs can be used to model a variety of systems, not just in computer science, but throughout engineering and life sciences. When systems evolve, we are generally interested in the way they change, to predict, support, or react to evolution. Graph transformations combine the idea of graphs, as a universal modelling paradigm, with a rule-based approach to specify the evolution of systems, which can be regarded as a generalisation of term rewriting. There are several formalisations of graph transformation based on algebraic methods — the double-pushout approach (DPO) is one of the most influential [EEPT06] — and several analysis tools based (mainly) on rewriting systems. Logic-based representation and proof-theoretic methods can be worth investigating though, when we are interested in mechanising the verification of abstract properties, and in the formal development of model-based systems.

Intuitionistic linear logic (ILL) has been applied to the modelling of linear resources in programming languages as well as of concurrent systems [CS06, Abr93, Mil92], the latter through semantics based on Petri nets, transition systems, multiset rewriting and process calculi. In this paper we propose a new approach to the representation of graph transformation systems (GTS) based on DPO in a variant of quantified intuitionistic linear logic (QILL). The DPO approach is arguably the most mature of the mathematically-founded approaches to graph transformation, with a rich theory of concurrency comparable to (and inspired by) those of place-transition Petri nets and term rewriting systems.

ILL can provide a formally neat way to handle the creation/deletion of graph components associated with the application of transformation rules in GTS, once we have fixed a textual encoding for graphs and their transformations. Syntactical presentation of DPO-GTS in terms of graph expressions tend to focus on hypergraphs — a generalisation of graphs allowing for edges that connect more than two nodes [CMR94]. Hyperedges can be intuitively associated to predicates defined on nodes. However, a predicate calculus account of graph transformation can be misleading, insofar as the actual content of the theory is less a static relational structure, than a dynamic system of connected components — those that can be either created, deleted or preserved by each transformation. Graph components include the nodes as much as the edge components, each made of an edge with its attached nodes — something that points indeed toward a second-order account.

Submitted to:
LINEARITY 2009

Intuitively, each graph component can be regarded as linear resource. However, there is a clear distinction between components which are used linearly in a graph expression, i.e. the edge components, and those that represent connections between them, and therefore may have multiple occurrences in the expression, i.e. the nodes. This distinction is reflected in one of the conditions that characterise the definition of valid transformation rule in DPO — the dangling condition, which essentially prohibits deleting a node unless the attached edges are deleted, too. The difference between nodes and edges can be taken into account in at least two different ways, using linear logic. One possibility is to consider a node n as resource shared between distinct edge components, say $c_1(n)$ and $c_2(n)$ for example. The logic expression of this idea would be to represent the sharing as $n \multimap c_1(n) \& c_2(n)$. This approach, bearing similarities to the work in [DP08] based on separation logic, has the possible drawback of complicating the representation of parallel edge components.

An alternative approach, that we are pursuing here, is to express parallel composition in a straightforward way — i.e. in terms of tensor product, on one hand, and on the other hand, to distinguish between a node as graph component that can be transformed, i.e. as linear resource, and the node name that may occur any times in a graph expression — hence clearly a non-linear term. This distinction appears to rely on the possibility to express the referential relation between nodes and their names. When we reason about graphs up to isomorphism, assuming node names are hidden (i.e. restricted, in process calculus parlance) makes it necessary to express this relation modulo α-renaming. Moreover, it is important to make sure that hidden node names match quantitatively and qualitatively (by type) the node components. This correspondence is at the basis of the other constraint to the validity of DPO rules — the identification condition. We present an extension of linear logic that allows keeping track of these relations, in order to represent GTS. In the first-order formulation that we have given in [TH], we define a translation of GTS assuming a restriction to transformation rules that allow only for preservation of nodes. Here we introduce a higher-order version of the logic, needed in order to lift that restriction.

We express the referential relation by annotating the type of a node with the name that refers to the node (using the symbol $\downharpoonleft$). We express name hiding by associating the introduction/elimination of hidden non-linear names with the consumption/deallocation of linear resources. This is essentially achieved by $\hat{\exists}$, operationally defined as a resource-bound existential quantifier. $\hat{\exists}$ has a separating character (though in a different sense from the intensional quantifiers in [Pym02]), by implicitly associating each bound variable to a linear resource in the sense of a tensor product. The representation of hiding, in connection to the availability of fresh names, is usually associated with the freshness quantifier of nominal logic [Pit01, CC04]. However, in our case linearity and the assumption that the environment does not contain duplicated labels suffice for the freshness of linear resources. We need to introduce a freshness condition on the instantiating non-linear term when we introduce the binding — which amounts, essentially, to making the introduction rule invertible, in contrast with standard existential quantification.

2 Hypergraphs and their transformations

Graph transformations can be defined on a variety of graph structures. In this paper we prefer typed hypergraphs, their n-ary hyperedges to be presented as predicates in the logic. A hypergraph (V, E, s) consists of a set V of vertices, a set E of hyperedges and a function

$\mathsf{s} : E \to V^*$ assigning each edge a sequence of vertices in V. A morphism of hypergraphs is a pair of functions $\phi_V : V_1 \to V_2$ and $\phi_E : E_1 \to E_2$ that preserve the assignments of nodes, that is, $\phi^*_V \circ \mathsf{s}_1 = \mathsf{s}_2 \circ \phi_E$. By fixing a type hypergraph $TG = (\mathcal{V}, \mathcal{E}, \mathsf{ar})$, we are establishing sets of node types $\mathcal{V}$ and edge types $\mathcal{E}$ as well as defining the arity $\mathsf{ar}(a)$ of each edge type $a \in \mathcal{E}$ as a sequence of node types. A TG-typed hypergraph is a pair $(HG, type)$ of a hypergraph HG and a morphism $type : HG \to TG$. A TG-typed hypergraph morphism $f : (HG_1, type_1) \to (HG_2, type_2)$ is a hypergraph morphism $f : HG_1 \to HG_2$ such that $type_2 \circ f = type_1$.

A *graph transformation rule* is a span of injective hypergraph morphisms $L \xleftarrow{l} K \xrightarrow{r} R$, called a *rule span*. A hypergraph transformation system (GTS) $\mathcal{G} = \langle TG, P, \pi, G_0 \rangle$ consists of a type hypergraph TG, a set P of rule names, a function mapping each rule name p to a rule span $\pi(p)$, and an initial TG-typed hypergraph G_0. A *direct transformation* $G \overset{p,m}{\Longrightarrow} H$ is given by a *double-pushout (DPO) diagram* as shown below, where (1), (2) are pushouts and top and bottom are rule spans. For a GTS $\mathcal{G} = \langle TG, P, \pi, G_0 \rangle$, a derivation $G_0 \Longrightarrow G_n$ in $\mathcal{G}$ is a sequence of direct transformations $G_0 \overset{r_1}{\Longrightarrow} G_1 \overset{r_2}{\Longrightarrow} \cdots \overset{r_n}{\Longrightarrow} G_n$ using the rules in $\mathcal{G}$. An hypergraph G is *reachable* in $\mathcal{G}$ iff there is a a derivation of G from G_0.

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

Intuitively, the left-hand side L contains the structures that must be present for an application of the rule, the right-hand side R those that are present afterwards, and the gluing graph K (the *rule interface*) specifies the "gluing items", i.e., the objects which are read during application, but are not consumed. Operationally speaking, the transformation is performed in two steps. First, we delete all the elements in G that are in the image of $L \setminus l(K)$ leading to the left-hand side pushout (1) and the intermediate graph D. Then, a copy of $L \setminus l(K)$ is added to D, leading to the derived graph H via the pushout (2). The first step (deletion) is only defined if the a built-in application condition, the so-called gluing condition, is satisfied by the match m. This condition, which characterises the existence of pushout (1) above, is usually presented in two parts.

Identification condition: Elements of L that are meant to be deleted are not shared with any other elements, i.e., for all $x \in L \setminus l(K)$, $m(x) = m(y)$ implies $x = y$.

Dangling condition: Nodes that are to be deleted must not be connected to edges in G, unless they already occur in L, i.e., for all $v \in G_V$, if there exists $e \in G_E$ such that $\mathsf{s}(e) = v_1 \ldots v \ldots v_n$, then $e \in m_E(L_E)$.

The first condition guarantees two intuitively separate properties of the approach: First, nodes and edges that are deleted by the rule are treated as resources, i.e., m is injective on $L \setminus l(K)$. Second, there must not be conflicts between deletion and preservation, i.e., $m(L \setminus l(K))$ and $m(l(K)$ are disjoint. The second condition ensures that after the deletion of nodes, the remaining structure is still a graph and does not contain edges short of a node. It is particularly the first condition which makes linear logic so attractive for graph transformation. Crucially, it is also reflected in the notion of concurrency of the approach, where items that are deleted cannot be shared between concurrent transformations.

3 Linear λ-calculus for GTS

We extend ILL with typed quantification, building on a notion of linear λ-calculus that comes to us from an unpublished [Pfe02]. In contrast with [TH], here we do not restrict quantification to be first-order. Our primitive logic formulas are $\alpha = A \mid L(N_1,\ldots,N_n) \mid \mathbf{1} \mid \alpha_1 \otimes \alpha_2 \mid \alpha_1 \multimap \alpha_2 \mid !\alpha_1 \mid \top \mid \alpha_1 \& \alpha_2 \mid \forall x : \beta.\alpha \mid \hat{\exists} x : \beta.\alpha \mid \alpha \mid \alpha \lfloor N \mid \alpha = \alpha$, where we assume A to represent a primitive node type, and $L(N_1,\ldots,N_n)$ to represent the type of an edge component. We also define

$$\alpha \hat{\equiv} \beta \;=_{df}\; (\alpha \multimap \beta)\&(\beta \multimap \alpha) \qquad\qquad \alpha\#(x,N) \;=_{df}\; (\alpha[N/x])[x/N] = \alpha$$

Primitive expressions are $M = x \mid u \mid \mathsf{nil} \mid N_1 \otimes N_2 \mid \hat{\varepsilon}(N_1|N_2).N_3 \mid \lambda x.N \mid \hat{\lambda} u.N \mid N_1\hat{\ }N_2 \mid N_1 N_2 \mid \langle N_1,N_2\rangle \mid \langle\rangle \mid \mathsf{fst}\ N \mid \mathsf{snd}\ N$. We define $(\mathsf{let}\ P = N_1\ \mathsf{in}\ N_2) \;=_{df}\; (\lambda P.N_2)N_1$, where $P = x \mid u \mid \mathsf{nil} \mid N_1 \otimes N_2 \mid \hat{\varepsilon}(N_1|N_2).N_3 \mid \langle N_1,N_2\rangle \mid \langle\rangle \mid \mathsf{fst}\ N \mid \mathsf{snd}\ N$ is a pattern.

We use two-entry sequents of form $\Gamma;\Delta \vdash N :: \alpha$, where Δ is a multiset of typed linear variables, denoted $u :: \alpha, v :: \alpha, \ldots$, Γ is a multiset of typed non-linear variables, denoted $x :: \alpha, y :: \alpha, \ldots$, $\vdash$ represents derivability, and $N :: \alpha$ is a typed expression. We assume that variables can occur at most once in a context. For multisets, we use sequence notation — modulo permutation and associativity, and $\cdot$ for the empty multiset. We consider a subset of linear variables as node names $(m,n,\ldots)$ and another subset as edge components $(c,d,\ldots)$. We consider a subset of non-linear variables as transformation rule names $(p,q,\ldots)$. When we "forget" about proof terms we are left with logic formulas and a consequence relation, which we denote by $\Vdash$. We use a notion of syntactic equality $=$ over types, stronger than linear equivalence $\hat{\equiv}$, to express a freshness constraint in $\hat{\exists} I$. We use $\hat{\lambda}$ to denote linear abstraction (and $\hat{\ }$ for linear application), in order to distinguish it from the non-linear abstraction (λ) — though this distinction is only meant to improve readability, since the difference between the two is determined by whether the abstraction variable is linear or not. The sequent calculus system is given by the axioms, the structural rules *Copy* (in the form of [Pfe94]), *Cut* and *Cut'*, and the operational rules rules. Cut elimination should be provable, along the lines of [Pfe94] — the only difference being the $\hat{\exists}$ rules — however it might not be straightforward.

3.1 Graphs in QILL

According the main lines of the formalisation we have presented in [TH], graphs can be generally associated to expressions in a sublanguage $G = n \mid c \mid \mathsf{nil} \mid G_1 \otimes G_2 \mid \hat{\varepsilon}(n|x).G$, and can be represented as derivations of form $\Gamma;\Delta \vdash G :: \gamma$, where Γ may contain all the typed non-linear variables that are not bound in the sequent, $\Delta = gc(G)$ contains all and only the typed linear variables that occur in G, and γ (the *graph formula*) is a formula in the $\mathbf{1},\otimes,\hat{\exists}$ fragment of the logic. The elements of Δ can represent the *ground constituents* — i.e. the nodes and the edge components of the graph. Untyped expressions (i.e. labels) represent component identity, whereas graph formulas contain the typing and connectivity information. The notion of graph up to isomorphism turns out to be captured by that of graph formula modulo linear equivalence, corresponding semantically to the set of graph formulas that can be derived from the ground constituent of the graph. Graphs can be represented schematically as $\hat{\exists}\overline{x : A}.L_1\ (\overline{x}_1) \otimes \ldots \otimes L_k\ (\overline{x}_k)$ where $\overline{x : A}$ is a sequence $x_1 : A_1, \ldots, x_j : A_j$ of typed variables — this counts as normal form, and it is closed, i.e. $\overline{x}_1, \ldots, \overline{x}_k \subseteq \overline{x}$, whenever all nodes are hidden. As an example, a hypergraph with nodes $n_1 : A_1, \ldots, n_4 : A_4$ and edge components $c_1 : L_1(n_1,n_2), c_2 : L_2(n_1,n_3,n_4), c_3 : L_3(n_2)$, can be represented with hidden nodes, in normal form, by a formula $\hat{\exists} x_1 : A_1, \ldots, x_4 : A_4.\ L_1(x_1,x_2) \otimes L_2(x_1,x_3,x_4) \otimes L_3(x_2)$.

3.2 Resource-bound existential quantifier

In order to type hiding, we need an existential-like quantifier (i.e. distributing over the tensor) that, in contrast with standard ones, ensures distinct bound variables in a formula cannot be instantiated with the same node — we will also say that the binder behaves injectively, i.e. that instantiations of multiple bound variables are always injective mappings. With $\hat{\exists}$, the instantiation of two distinct variables requires two linear resources, therefore cannot be derived from the instantiation of one — hence multiple instantiations behave injectively (Obs. 1(1)).

$$\frac{\Gamma;\Delta \vdash M :: \alpha[N_\Delta/x] \quad \Gamma;\cdot \vdash N_\Delta :: \beta \quad \Gamma;\Delta' \vdash n :: \beta \downharpoonright N_\Delta \quad \Gamma, x :: \beta; \cdot \vdash \mathsf{nil} :: \alpha\#(x, N_\Delta)}{\Gamma;\Delta,\Delta' \vdash \hat{\varepsilon}(n|N_\Delta).M :: \hat{\exists}x : \beta.\alpha}\ \hat{\exists}R$$

$$\frac{\Gamma, x :: \beta; \Delta, n :: \beta \downharpoonright x, v :: \alpha \vdash N :: \gamma}{\Gamma;\Delta, w :: \hat{\exists}x : \beta.\alpha \vdash \mathsf{let}\ \hat{\varepsilon}(n|x).v = w\ \mathsf{in}\ N :: \gamma}\ \hat{\exists}L$$

$\hat{\exists}$ left introduction is similar to the standard rule, except for the linear premise $n :: \beta \downharpoonright x$. $\hat{\exists}$ right introduction satisfies two non-standard constraints. The first one — $\Gamma;\Delta' \vdash n :: \beta \downharpoonright N_\Delta$ — means that N_Δ is the unique name (non-linear resource) for linear resource n. We need to force a dependence of the non-linear term on the linear context Δ — and here we do this by an index — lacking this, we might run into trouble with rule *Cut*. $\downharpoonright$ represents an injective mapping from linear resources to non-linear ones, ensuring that the node n cannot be associated to two different names, and hence used twice. The mapping is not generally invertible — since N is arbitrary, it can be associated to different linear resources, however this should not be a problem, as long as we do not allow for the same term to be used in different spatial contexts (that is why we need to force dependence on the linear context Δ). We do not give any proper introduction rule for $\downharpoonright$ (no associated constructor), and therefore $n :: \beta \downharpoonright N$ can only be introduced by axiom, under the assumption N is well-typed, i.e.

$$\frac{\Gamma;\cdot \vdash N :: \beta}{\Gamma; n :: \beta \downharpoonright N \vdash n :: \beta \downharpoonright N}\ \downharpoonright I$$

Given the definition of #, the second constraint is $\Gamma, x :: \beta; \cdot \vdash \mathsf{nil} :: (\alpha[N/x])[x/N] = \alpha$, which means that N does not occur in $\alpha[N/x]$ other than in place of x, i.e. we use type equality and substitution to formalise the requirement that N does not occur free in $\hat{\exists}x.\alpha$. We do not give rules for type equality here, but they are standard ones. With this constraint, the formula $\hat{\exists}x.\alpha$ is determined by the instance $\alpha[N/x]$ modulo renaming of bound variables — since *all* the occurrences of N must be replaced by x. Therefore applications of $\hat{\exists}R$ force a bijection between unbounded resources (node names) and variables that actually occur in α. Moreover, the rule consumes resources associated to $n :: \beta \downharpoonright N$, and therefore forces a linear dependence of variables on resources (nodes).

The hiding operator $\hat{\varepsilon}$ can be defined along the lines of the linear interpretation of the existential quantifier [CP02], closely associated to the standard intuitionistic one, i.e.

$$\hat{\varepsilon}(n|N).M :: \hat{\exists}x : \alpha.\beta \ =_{df}\ n \otimes !N \otimes M$$

with the proviso of the non-occurrence of either N or x in α. N depends only on the global context, hence it can be replaced with $!N$. The linearity of $n :: \alpha$ ensures injectivity.

It is not difficult to see that the following properties hold.

Prop. 1 (1) $\not\Vdash (\hat{\exists} x : \beta.\ \alpha(x,x)) \multimap \hat{\exists} xy : \beta.\ \alpha(x,y)$

the resource associated to x cannot suffice for x and y.

(2) $\not\Vdash \forall x : \beta.\ \beta \downharpoonleft x \otimes \alpha(x,x) \multimap \hat{\exists} y : \beta.\alpha(y,x)$

y and x should be instantiated with the same term — but this is prevented by the freshness condition in $\hat{\exists}$ introduction

(3) $\not\Vdash (\hat{\exists} yx : \beta.\ \alpha_1(x) \otimes \alpha_2(x)) \multimap (\hat{\exists} x : \beta.\alpha_1(x)) \otimes \hat{\exists} x : \beta.\alpha_2(x)$

the two bound variables in the consequence require distinct resources and refer to distinct occurrences

Prop. 2 $\hat{\exists}$ satisfies properties of α-renaming, exchange and distribution over $\otimes$, i.e.

$$\Vdash (\hat{\exists} x : \alpha.\beta(x)) \hat{\equiv} (\hat{\exists} y : \alpha.\beta(y))$$
$$\Vdash (\hat{\exists} xy : \alpha.\gamma) \hat{\equiv} (\hat{\exists} yx.\gamma)$$
$$\Vdash (\hat{\exists} x : \alpha.\beta \otimes \gamma(x)) \hat{\equiv} (\beta \otimes \hat{\exists} x : \alpha.\gamma(x)) \qquad (x \text{ not in } \alpha)$$

In general $\hat{\exists}$ does not satisfy logical η-equivalence, i.e. it cannot be proved that α is equivalent to $\hat{\exists} x.\ \alpha$ when x does not occur free in α (neither sense of linear implication holds). This may come handy though, to represent graphs with isolated nodes.

We do not introduce term congruence explicitly, but we assume α-renaming, β- and η-congruence for λ and $\hat{\lambda}$ (with linearity check for the latter), as well as α-renaming, exchange, and distribution over $\otimes$ for $\hat{\varepsilon}$ (to match the type properties in Obs. 2).

3.3 Transformation rules

A DPO transformation rule (we consider rules with interfaces made only of nodes) can be represented as $\forall \overline{x : A}.\alpha \multimap \beta$ where α, β are graph expressions. Transformation rules can be encoded as typed non-linear variables $p :: \forall \overline{x : A}.\gamma_1 \multimap \gamma_2$ where γ_1, γ_2 are graph formulas. The implicit ! closure guarantees unrestricted applicability, universally quantified variables represent the rule interface, and linear implication represents transformation. The application of p to a closed graph formula $\alpha_G = \hat{\exists} \overline{y : A_y}.\beta_G$ determined by morphism m (as shown in the diagram) relies on the fact that the following rule is derivable

$$\frac{\begin{array}{c}\Gamma; \cdot \Vdash \alpha_G \hat{\equiv} \alpha_{G'} \quad \alpha_{G'} = \hat{\exists} \overline{z : A_z}.\alpha_L[\overline{z : A_z} \xleftarrow{d} \overline{x : A_x}] \otimes \alpha_C \\ \Gamma; \cdot \Vdash \alpha_H \hat{\equiv} \alpha_{H'} \quad \alpha_{H'} = \hat{\exists} \overline{z : A_z}.\alpha_R[\overline{z : A_z} \xleftarrow{d} \overline{x : A_x}] \otimes \alpha_C\end{array}}{\Gamma; \forall \overline{x : A_x}.\alpha_L \multimap \alpha_R \Vdash \alpha_G \multimap \alpha_H} \ \overset{p,m}{\Longrightarrow}$$

where the interface morphism d associated with m is represented by the multiple substitution $[\overline{z : A_z} \xleftarrow{d} \overline{x : A_x}]$, with $\overline{z : A_z} \subseteq \overline{y : A_y}$

Prop. 3 The QILL application schema satisfies the DPO conditions.

An informal proof can be built on top of Obs. 2(1,2,3). Injectivity rests on 1 for nodes and on linearity of the consequence relation for edge components, the identification condition rests on 2, the morphism characterisation of instantiations rests on 3, and this, together with the propositional structure of graph expressions, should suffice to ensure that also the dangling edge condition is satisfied.

A sequent $\cdot;G_0,P_1,\ldots,P_k \Vdash G_1$ can express that graph G_1 is reachable from the initial graph G_0 by applying rules $P_1 = \forall\overline{x}_1.\alpha_1 \multimap \beta_1$, $\ldots$, $P_k = \forall\overline{x}_k.\alpha_k \multimap \beta_k$ once each, abstracting from the application order, each application resulting into a transformation step. A sequent $P_1,\ldots,P_k;G_0 \Vdash G_1$ can express that G_1 is reachable from G_0 by the same rules, regardless of whether or how many times they need to be applied. The parallel applicability of rules $\forall\overline{x}_1.\alpha_1 \multimap \beta_1$, $\forall\overline{x}_2.\alpha_2 \multimap \beta_2$ can be represented as applicability of $\forall\overline{x}_1,\overline{x}_2.\alpha_1 \otimes \alpha_2 \multimap \beta_1 \otimes \beta_2$ (true parallellism) or else of $\forall\overline{x}_1,\overline{x}_2.(\alpha_1 \multimap \beta_1)\otimes(\alpha_2 \multimap \beta_2)$.

3.4 Extending the logic

In order to deal with transformation rules that preserve edge components, we need higher-order universal quantification. This should ensure though, that while abstracted variables range over non-linear terms, in order to allow for multiple variables to be instantiated with the same term, instantiation takes place only when at least a linear resource of compatible type is available. Here we introduce $\upharpoonright$ in order to extend non-linear contexts with premises that can be dropped when linear ones are available, and a notion of quantification that depends upon it, with the following rules

$$\frac{\Gamma,x :: \beta \upharpoonright;\Delta \vdash N :: \alpha}{\Gamma;\Delta \vdash \lambda x.N :: \hat{\forall} x:\beta.\alpha}\ \hat{\forall}R \qquad \frac{\Gamma;\cdot \vdash M :: \beta \upharpoonright \quad \Gamma;\Delta,v :: \alpha[M/x] \vdash N :: \gamma}{\Gamma;\Delta,u :: \hat{\forall} x:\beta.\alpha \vdash \mathsf{let}\ v = uM\ \mathsf{in}\ N :: \gamma}\ \hat{\forall}L$$

$$\frac{\Gamma;u :: \beta,\Delta \vdash N :: \alpha}{\Gamma,x :: \beta \upharpoonright;u :: \beta,\Delta \vdash N :: \alpha}\ \upharpoonright I$$

We need to assume that $\upharpoonright$ premises cannot be introduced arbitrarily — in fact, restricting weakening. A general DPO transformation rule can be represented as $\forall\overline{x:A}.\hat{\forall}\overline{y:\gamma}.\alpha \multimap \beta$ where α,β,γ are graph expressions. The application of a rule requires that, after the instantiation of the interface nodes, interface edges can be instantiated, only if matching components are available as linear resources in the graph — though this does not involve consuming them.

3.5 Conclusion and further work

We have discussed how to represent DPO-GTS in a higher-order quantified extension of ILL. We have extended the encoding in [TH] by considering edge abstraction. We are interested in a logic that allows us to reason about concurrency and reachability at the abstract level, as well as for the synthesis of proof terms that can represent system runs — hence our interest in constructive logic. While we feel that the representation of contextual dependencies in the operational rules needs more investigation, especially with respect to cut elimination, we are also interested in extending the encoding to stochastic GTS.

References

[Abr93] S. Abramsky. Computational interpretation of linear logic. *Theoretical Computer Science* 111, 1993.

[CC04] L. Caires, L. Cardelli. A Spatial Logic for Concurrency II. *Theoretical Computer Science* 322(3):517–565, 2004.

[CMR94] A. Corradini, U. Montanari, F. Rossi. An abstract machine for concurrent modular systems: CHARM. *Theoretical Computer Science* 122:165–200, 1994.

[CP02] I. Cervesato, F. Pfenning. A linear logical framework. *Information and Computation* 179(1):19–75, 2002.

[CS06] I. Cervesato, A. Scedrov. Relating State-Based and Process-Based Concurrency through Linear Logic. *Electron. Notes Theor. Comput. Sci.* 165:145–176, 2006.

[DP08] M. Dodds, D. Plump. From hyperedge relpacement to separation logic and back. In *ICGT 2008 — Doctoral Symposium*. 2008.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of algebraic graph transformation*. Springer, 2006.

[Mil92] D. Miller. The pi-calculus as a theory in linear logic: preliminary resutls. In *Workshop on Extensions of Logic Programming*. LNCS 660, pp. 242–264. Springer, 1992.

[Pfe94] F. Pfenning. Structural Cut Elimination in Linear Logic. Technical report, Carnagie Mellon University, 1994.

[Pfe02] F. Pfenning. Linear Logic — 2002 Draft. Technical report, Carnagie Mellon University, 2002.

[Pit01] A. M. Pitts. Nominal Logic: A First Order Theory of Names and Binding. In *TACS '01: Proceedings of the 4th International Symposium on Theoretical Aspects of Computer Software*. Pp. 219–242. Springer-Verlag, 2001.

[Pym02] D. J. Pym. *The semantics and proof-theory of the logics of bunched implications*. Applied Logic Series. Kluwer, 2002.

[TH] P. Torrini, R. Heckel. Towards an embedding of graph transformation systems in intuitionistic linear logic. ICE'09 Workshop.
`www.cs.le.ac.uk/people/pt95/sll69.pdf`

Appendix — other proof rules

$$\frac{}{\Gamma;u::\alpha\vdash u::\alpha}\,Id \qquad \frac{}{\Gamma,x::\alpha;\cdot\vdash x::\alpha}\,UId \qquad \frac{\Gamma,x::\alpha;u::\alpha,\Delta\vdash N::\gamma}{\Gamma,x::\alpha;\Delta\vdash \mathsf{let}\ u=\mathsf{copy}(x)\ \mathsf{in}\ N::\gamma}\,Copy$$

$$\frac{\Gamma;\Delta\vdash N::\alpha \quad \Gamma;u::\alpha,\Delta'\vdash M::\beta}{\Gamma;\Delta,\Delta'\vdash \mathsf{let}\ N=u\ \mathsf{in}\ M::\beta}\,Cut \qquad \frac{\Gamma;\cdot\vdash N::\alpha \quad \Gamma,x::\alpha;\Delta\vdash M::\beta}{\Gamma;\Delta\vdash \mathsf{let}\ N=x\ \mathsf{in}\ M::\beta}\,Cut'$$

$$\frac{\Gamma;\Delta_1\vdash M::\alpha \quad \Gamma;\Delta_2\vdash N::\beta}{\Gamma;\Delta_1,\Delta_2\vdash M\otimes N::\alpha\otimes\beta}\,\otimes R \qquad \frac{\Gamma;\Delta,u::\alpha,v::\beta\vdash N::\gamma}{\Gamma;\Delta,w::\alpha\otimes\beta\vdash \mathsf{let}\ u\otimes v=w\ \mathsf{in}\ N::\gamma}\,\otimes L$$

$$\frac{\Gamma;\Delta,u::\alpha\vdash M::\beta}{\Gamma;\Delta\vdash \hat{\lambda}u:\alpha.\,M::\alpha\multimap\beta}\,\multimap R \qquad \frac{\Gamma;\Delta_1\vdash M::\alpha \quad \Gamma;\Delta_2,u::\beta\vdash N::\gamma}{\Gamma;\Delta_1,\Delta_2,v::\alpha\multimap\beta\vdash \mathsf{let}\ u=v\hat{}N\ \mathsf{in}\ N::\gamma}\,\multimap L$$

$$\frac{}{\Gamma;\cdot\vdash \mathsf{nil}::\mathbf{1}}\,\mathbf{1}R \qquad \frac{\Gamma;\Delta\vdash N::\alpha}{\Gamma;\Delta,u::\mathbf{1}\vdash \mathsf{let}\ \mathsf{nil}=u\ \mathsf{in}\ N::\alpha}\,\mathbf{1}L$$

$$\frac{\Gamma;\Delta\vdash M::\alpha \quad \Gamma;\Delta\vdash N::\beta}{\Gamma;\Delta\vdash\langle M,N\rangle::\alpha\&\beta}\,\&R \qquad \frac{}{\Gamma;\Delta\vdash\langle\rangle::\top}\,\top R$$

$$\frac{\Gamma;\Delta,v::\alpha\vdash N::\gamma}{\Gamma;\Delta,u::\alpha\&\beta\vdash \mathsf{let}\ v=\mathsf{fst}\ u\ \mathsf{in}\ N::\gamma}\,\&L1 \qquad \frac{\Gamma;\Delta,v::\beta\vdash N::\gamma}{\Gamma;\Delta,u::\alpha\&\beta\vdash \mathsf{let}\ v=\mathsf{fst}\ u\ \mathsf{in}\ N::\gamma}\,\&L2$$

$$\frac{\Gamma;\cdot\vdash M::\alpha}{\Gamma;\cdot\vdash !M::!\alpha}\,!R \qquad \frac{\Gamma,p::\alpha;\Delta\vdash N::\beta}{\Gamma;\Delta,u::!\alpha\vdash \mathsf{let}\ p=!u\ \mathsf{in}\ N::\beta}\,!L$$

$$\frac{\Gamma,x::\beta;\Delta\vdash M::\alpha}{\Gamma;\Delta\vdash\lambda x.\,M::\forall x:\beta.\,\alpha}\,\forall R \qquad \frac{\Gamma;\cdot\vdash M::\beta \quad \Gamma;\Delta,v::\alpha[M/x]\vdash N::\gamma}{\Gamma;\Delta,u::\forall x:\beta.\alpha\vdash \mathsf{let}\ v=uM\ \mathsf{in}\ N::\gamma}\,\forall L$$