A new graphical calculus of proofs

Sandra Alves

DCC - Faculty of Science & LIACC
University of Porto
R. do Campo Alegre 1021/55,
4169-007, Porto, Portugal

Maribel Fernández

Department of Informatics
King's College London
Strand, London, WC2R 2LS U.K

Ian Mackie

LIX, CNRS UMR 7161
École Polytechnique
91128 Palaiseau Cedex, France

We offer a simple graphical representation for proofs of intuitionistic logic, which is inspired by proof nets and interaction nets (two formalisms originating in linear logic). This graphical calculus of proofs inherits good features from each, but is not constrained by them. By the Curry-Howard isomorphism, the representation applies equally to the lambda calculus, offering an alternative diagrammatic representation of functional computations.

Keywords: Intuitionistic logic, lambda-calculus, visual representation, proof theory

1 Introduction

There are many different ways to write proofs in a given logic, for instance, natural deduction, sequent calculus and Hilbert systems are well-known proof systems (we refer the reader to [10] for details). Each syntax has advantages and disadvantages. For example, classical logic works well in sequent calculus because it allows the symmetry of the connectives to be seen; a natural deduction presentation of classical logic is considered artificial since rules are needed which do not correspond to the introduction or elimination of a connective. Intuitionistic logic, which links exceptionally well with computation, works better in natural deduction, where proofs correspond to programs and there is a notion of canonical proof (which is not the case in sequent calculus).

In this paper we focus on intuitionistic logic, and we aim at designing a syntax that can both

- facilitate the visualisation and understanding of proofs, and

- serve as a basis for the implementation of the simplification rules in the logic.

We choose intuitionistic logic as a basis for this work because of the computational significance of the logic through the Curry-Howard isomorphism: proofs in this logic correspond to functional programs; logic formulas correspond to types of programs; and proof normalisation corresponds to computation. Thus we are not looking for a visual representation of *truth*, but rather that of a *proof*. Furthermore, we are not interested in just representing logics, but also in studying the reduction process (normalisation) which corresponds to computation through the Curry-Howard isomorphism.

Linear logic [8] comes equipped with a graphical syntax called proof nets. One of the motivations for the adoption of a graphical syntax is that traditional syntaxes for logic, such as the sequent calculus, have a lot of constraints to do with the formalism—and not with the logic. To illustrate this, we borrow a well-known example from Girard [9]. Let (r) and (s) be two logical rules, and consider a cut working on auxiliary formulas (not the main formulas of the rules r and s):

$$\cfrac{\cfrac{\vdash \Gamma, A}{\vdash \Gamma', \mathbf{A}}\,(r) \qquad \cfrac{\vdash \neg A, \Delta}{\vdash \neg \mathbf{A}, \Delta'}\,(s)}{\vdash \Gamma', \Delta'}\,(\mathsf{Cut})$$

If we permute the cut rule up through r and s, then there are two possible choices depending on whether we first permute through s or through r. These two choices are represented by:

$$\cfrac{\cfrac{\cfrac{\cfrac{\vdash \Gamma, \mathbf{A} \qquad \vdash \neg\mathbf{A}, \Delta}{\vdash \Gamma, \Delta}\,(\mathsf{Cut})}{\vdash \Gamma', \Delta}\,(r)}{\vdash \Gamma', \Delta'}\,(s)}{}
\qquad\qquad
\cfrac{\cfrac{\cfrac{\cfrac{\vdash \Gamma, \mathbf{A} \qquad \vdash \neg\mathbf{A}, \Delta}{\vdash \Gamma, \Delta}\,(\mathsf{Cut})}{\vdash \Gamma, \Delta'}\,(s)}{\vdash \Gamma', \Delta'}\,(r)}{}$$

This demonstrates that permutation of rules is inherent in the syntax. In addition, there is no canonical representation of proofs, even for cut-free proofs. For example, consider the following two alternative proofs (in classical logic), of the formula: $\neg A, C, A \wedge \neg B, B \wedge \neg C$, which differ only by the order in which axioms are combined:

$$\cfrac{\cfrac{}{\vdash \neg A, A}\,(\mathsf{Ax}) \qquad \cfrac{\cfrac{}{\vdash \neg B, B}\,(\mathsf{Ax}) \qquad \cfrac{}{\vdash \neg C, C}\,(\mathsf{Ax})}{\vdash C, \neg B, B \wedge \neg C}\,(\wedge)}{\vdash \neg A, C, A \wedge \neg B, B \wedge \neg C}\,(\wedge)$$

$$\cfrac{\cfrac{\cfrac{}{\vdash \neg A, A}\,(\mathsf{Ax}) \qquad \cfrac{}{\vdash \neg B, B}\,(\mathsf{Ax})}{\vdash \neg A, B, A \wedge \neg B}\,(\wedge) \qquad \cfrac{}{\vdash \neg C, C}\,(\mathsf{Ax})}{\vdash \neg A, C, A \wedge \neg B, B \wedge \neg C}\,(\wedge)$$

A graphical syntax will free us from these inessential permutations. To illustrate the point, the previous two proofs are given by the following proof net:

$$\neg A \qquad A \wedge \neg B \qquad B \wedge \neg C \qquad C$$

In this visual representation, the nodes represent the connectives and the edges are labelled by formulas. Unlike the sequent calculus, we do not need to impose an order on the introduction of $\wedge$. Moreover, many of the permutation equivalences become identities in proof nets.

In recent years, advances in this area have produced a better understanding of the notion of proof, and a pleasing mathematical theory has been developed. However, it is not clear that this is the best way of visualising proofs or mechanising proof transformations. Proof nets work very well for some fragments of the logic, but less well for others where extra structure is required (boxes for instance). Related systems, such as interaction nets [12] have been used to provide implementations of proof nets. Indeed, they can be seen as a generalisation of proof nets. However, the implementation nature of interaction nets often leads to cluttering proofs with low-level details, which do not aid the understanding of proofs.

In this paper we propose to take features from both proof nets and interaction nets to build a hybrid system that takes some of the good features of each, but is not limited by either of them. We put the case forward for intuitionistic logic, and by the Curry-Howard isomorphism, we also get results for the λ-calculus. This hybrid notation allows us to choose the level of detail that we want to include in the proof. We can give either a high level, visual description of the dynamics of proof simplification, or a low level description which is suited for the implementation or for fine-grained analyses of the cut elimination process.

This graphical notation can be formally defined as a system of port graphs [1, 3]. The visualisation tools available for port graphs (see [2]) can then be used to study proofs and proof transformations.

Summarising, the main contributions of this paper are:

- A new graphical notation for proofs in intuitionistic logic where one can choose different levels of abstraction depending on the kind of analysis that will be done on proofs.

- By the Curry-Howard isomorphism, the previous point gives a graphical notation for the λ-calculus with the same characteristics.

Related work   In addition to proof nets and interaction nets, several graphical representations of proofs have been proposed in the literature. We can cite for instance the deduction graphs defined by Geuvers and Loeb [7] and Lamping's sharing graphs [13]. Deduction graphs are a generalisation of natural deduction and Fitch style flag deduction; they have both nodes and boxes. The latter are collections of nodes that form a node themselves, and in this sense they are related to Milner's bigraphs [14], where the place graph describes the nesting of nodes. However, deduction graphs do not have an explicit way to represent sharing; they are not intended as a notation for fine-grained analysis of resource management in proof normalisation. Sharing graphs (introduced in [13] and further developed by Asperti and Guerrini [4]) were presented as a solution for the implementation of Lévy's notion of optimal reduction in the λ-calculus, and, as their name suggests, emphasise the sharing of subexpressions: sharing is explicit.

The rest of this paper is organised as follows. In the following section we briefly recall intuitionistic logic, proof nets, interaction nets and the notions of port graph and port graph rewriting. In Sections 3 and 4 we give the visual representation of the logic and the λ-calculus, respectively. In section 5 we discuss the significance of this work, and speculate on future work.

2   Background

In this section we recall (minimal) intuitionistic logic and sketch some of the ideas behind proof nets and interaction nets, on which our work is built upon. To formalise the notation, we recall the notion of port graph and port graph rewriting from [1, 3]. For more details on linear logic and proof nets, we refer the reader to [8]. For details and examples of interaction nets we refer to [12].

Intuitionistic logic   In Figure 1 we give the natural deduction in sequent form presentation of the logic. We give explicitly the structural rules, which helps understanding the graphical notation later. Adding $\vee$ is straightforward, but not included in this paper.

Identity and Structural Group:

$$\frac{}{A \vdash A}\,(\mathsf{Ax}) \qquad \frac{\Gamma,A,B,\Delta \vdash C}{\Gamma,B,A,\Delta \vdash C}\,(X) \qquad \frac{\Gamma \vdash B}{\Gamma,A \vdash B}\,(W) \qquad \frac{\Gamma,A,A \vdash B}{\Gamma,A \vdash B}\,(C)$$

Logical Group:

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma,\Delta \vdash A \wedge B}\,(\wedge\mathscr{I}) \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\,(\wedge\mathscr{E}_1) \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\,(\wedge\mathscr{E}_2)$$

$$\frac{\Gamma,A \vdash B}{\Gamma \vdash A \Rightarrow B}\,(\Rightarrow\mathscr{I}) \qquad \frac{\Gamma \vdash A \Rightarrow B \qquad \Delta \vdash A}{\Gamma,\Delta \vdash B}\,(\Rightarrow\mathscr{E})$$

Figure 1: Intuitionistic Natural Deduction

The main computational interest for us is the normalisation procedure, which transforms proofs by eliminating redundancies (called detours by Prawitz [15]). The main cases are defined below, showing how the introduction of a connective followed by the elimination of that same occurrence of the connective can be transformed into a proof without the two rules. For simplicity, we have ignored details of permutations of rules that might need to be applied so that one of the following rules can be applied.

Definition 1 *(One Step Normalisation)*

- $(\wedge \mathscr{I})$ *followed by* $(\wedge \mathscr{E}_1)$*: below the double line indicates that (W) may be applied zero or many times. There is a similar case for* $(\wedge \mathscr{I})$ *followed by* $(\wedge \mathscr{E}_2)$.

$$
\cfrac{\cfrac{\cfrac{\pi_1}{\Gamma \vdash A} \qquad \cfrac{\pi_2}{\Delta \vdash B}}{\Gamma, \Delta \vdash A \wedge B}(\wedge \mathscr{I})}{\Gamma, \Delta \vdash A}(\wedge \mathscr{E}_1)
\quad becomes \quad
\cfrac{\cfrac{\pi_1}{\Gamma \vdash A}}{\Gamma, \Delta \vdash A}(W)
$$

- $(\Rightarrow \mathscr{I})$ *followed by* $(\Rightarrow \mathscr{E})$*:* π_1' *is the proof* π_1 *where all axioms* $A \vdash A$ *are replaced (substituted) by a proof of* π_2.

$$
\cfrac{\cfrac{\cfrac{\pi_1}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B}(\Rightarrow \mathscr{I}) \qquad \cfrac{\pi_2}{\Delta \vdash A}}{\Gamma, \Delta \vdash B}(\Rightarrow \mathscr{E})
\quad becomes \quad
\cfrac{\pi_1'}{\Gamma, \Delta \vdash B}
$$

In this presentation, the notion of substitution is important, and is quite difficult to understand. An advantage of the visual representation given below is that it clarifies this notion.

Proof Nets and Interaction Nets   Proof nets were introduced as the graphical syntax for linear logic. One of the motivations for the study of graphical presentations is to free us from inessential permutations in proofs, as mentioned in the Introduction. Proof nets work very well for the multiplicative fragment of the logic, but less well for the other fragments. For instance, for the exponentials, more complicated machinery is needed, which takes us away from a uniform visual notation. More precisely, exponentials are represented using boxes to group parts of the graph, shown as a dotted line in the diagram below:
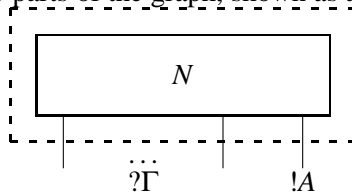
Boxes work at a different level to the other nodes in the graph, leading to a two-level syntax. Interaction nets, on the other-hand, encode the box machinery in the same notation, as shown below.

An interaction net system [12] is specified by giving a set Σ of symbols, and a set $\mathscr{R}$ of interaction rules. Each symbol $\alpha \in \Sigma$ has an associated (fixed) *arity*. An occurrence of a symbol $\alpha \in \Sigma$ will be called an *agent*. If the arity of α is n, then the agent has $n+1$ *ports*: a distinguished one called the *principal port* depicted by an arrow, and *n auxiliary ports* labelled $x_1, \ldots, x_n$ corresponding to the arity of the symbol. Such an agent will be drawn in the following way:
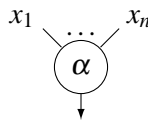
A net N built on Σ is a graph (not necessarily connected) with agents at the vertices. The edges of the graph connect agents together at the ports such that there is only one edge at every port.

A pair of agents $(\alpha,\beta) \in \Sigma \times \Sigma$ connected together on their principal ports is called an *active pair*; the interaction net analog of a redex. An interaction rule $((\alpha,\beta) \Longrightarrow N) \in \mathscr{R}$ replaces an occurrence of the active pair (α,β) by a net N. The rule must satisfy two conditions: all free ports are preserved during reduction, and there is at most one rule for each pair of agents.

Boxes are encoded in interaction nets using extra nodes, as shown below:
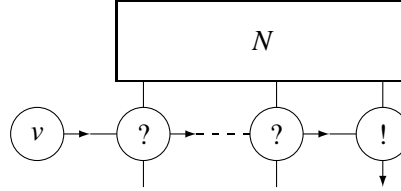
Due to the constraints in the rules, interaction nets are easy to implement, but extra rules are needed for management: the details of copying and erasing for instance must be given in full detail. In addition, each node in an interaction net has a unique principal port, and this is fixed for each kind of agent, which means that the reduction system is fixed as part of the encoding.

Our approach in this paper is to put forward a hybrid notation between proof nets and interaction nets to get the best from each. We will be able to use interactive tools developed for interaction nets, such as PORGY [2] in order to visualise the encodings of proofs. In fact, PORGY deals with port graphs, a class of graphs that is more general than interaction nets, and which can be used to formalise our hybrid notation. We recall port graphs below.

Port graphs   A port graph [1] is a graph where nodes have explicit connection points for edges, called *ports*. Port graphs were first identified as an abstract view of proteins and molecular complexes.

Let $\mathscr{N}$ and $\mathscr{P}$ be two disjoint sets of node names and port names respectively. A *p-signature* over $\mathscr{N}$ and $\mathscr{P}$ is a mapping $\nabla : \mathscr{N} \to 2^{\mathscr{P}}$ which associates a finite set of port names to a node name. A p-signature can be extended with variables: $\nabla^{\mathscr{X}} : \mathscr{N} \cup \mathscr{X}_{\mathscr{N}} \to 2^{\mathscr{P} \cup \mathscr{X}_{\mathscr{P}}}$, where $\mathscr{X}_{\mathscr{P}}$ and $\mathscr{X}_{\mathscr{N}}$ are two disjoint sets of port name variables and node name variables respectively. A *labelled port graph* over a *p-signature* $\nabla^{\mathscr{X}}$ is a tuple $G = \langle V_G, lv_G, E_G, le_G \rangle$ where: V_G is a finite set of nodes; $lv_G : V_G \to \mathscr{N} \cup \mathscr{X}_{\mathscr{N}}$ is an injective labelling function for nodes; $E_G \subseteq \{\langle (v_1,p_1),(v_2,p_2)\rangle \mid v_i \in V_G, p_i \in \nabla(lv_G(v_i)) \cup \mathscr{X}_{\mathscr{P}}\}$ is a finite multiset of edges; $le_G : E_G \to (\mathscr{P} \cup \mathscr{X}_{\mathscr{P}}) \times (\mathscr{P} \cup \mathscr{X}_{\mathscr{P}})$ is an injective labelling function for edges such that $le_G(\langle (v_1,p_1),(v_2,p_2)\rangle) = (p_1,p_2)$. A port may be associated to a state (for instance, active/inactive or principal/auxiliary); this is formalised using a mapping from ports to port states. Similarly, nodes can also have associated properties (like colour or shape that can be used for visualisation purposes).

Let G and H be two port graphs over the same p-signature $\nabla^{\mathscr{X}}$. A *port graph morphism* $f : G \to H$ maps elements of G to elements of H preserving sources and targets of edges, constant node names and associated port name sets, up to variable renaming. We say that G and H are *isomorphic* if $f : V_G \times \nabla(lv_G(V_G)) \to V_H \times \nabla(lv_H(V_H))$ is bijective.

A *port graph rewrite rule* $L \Rightarrow R$ is itself represented as a port graph consisting of two port graphs L and R over the same p-signature and one special node $\Rightarrow$, called *arrow node* connecting them. L and R are called the *left-* and *right-hand side* respectively. The arrow node is used to represent the interface of the rule; it has the following characteristics: for each port p in L, to which corresponds a non-empty set of ports $\{p_1,\ldots,p_n\}$ in R, the arrow node has a unique port r and the incident directed edges (p,r) and (r,p_i), for all $i = 1,\ldots,n$; all ports from L that are deleted in R are connected to the *black hole* port

of the arrow node. When the correspondence between ports in the left- and right-hand side of the rule is obvious we omit the ports and edges involving the arrow node. In this way, we avoid dangling edges after rewriting (for more details on graph rewriting we refer the reader to [11, 6]).

Let $L \Rightarrow R$ be a port graph rewrite rule and G a port graph such that there is an injective port graph morphism g from L to G; hence $g(L)$ is a subgraph of G. A *rewriting step* on G using $L \Rightarrow R$, written $G \rightarrow_{L \Rightarrow R} G'$, transforms G into a new graph G' obtained from G by replacing the subgraph $g(L)$ of G by $g(R)$, and connecting $g(R)$ to the rest of the graph as specified in the arrow node. We call $g(L)$ a redex. If there is no such injective morphism, we say that G is *irreducible* by $L \Rightarrow R$. Given a finite set $\mathscr{R}$ of rules, a port graph G *rewrites* to G', denoted by $G \Rightarrow_{\mathscr{R}} G'$, if there is a rule r in $\mathscr{R}$ such that $G \Rightarrow_r G'$. This induces a transitive relation on port graphs, denoted by $\Rightarrow_{\mathscr{R}}^*$. A port graph on which no rule is applicable is in *normal form*.

3   Graphs from Proofs

In this section we give a graphical representation of proofs in intuitionistic logic. The general idea is to interpret a proof π of $\Gamma \vdash A$ as a port graph $\mathscr{G}(\pi)$ with edges representing formulas in the following way (the second alternative borrows a notation from electronic circuits):
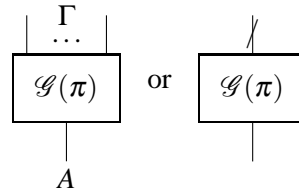
The nodes of the graph represent rules in the logic, edges will be attached to ports which are optionally labelled with formulas. We will explicitly distinguish the conclusion port when it is relevant. Node names will be introduced on demand in the translation below. Later we will see that we need some additional control nodes, so that there will not be a 1-1 correspondence between logical rules and node names. We give a translation inductively over the structure of the proof, and refer to Figure 1 for the rules that we are translating.

- If π is an Axiom $A \vdash A$, then $\mathscr{G}(\pi)$ is simply a node Ax with two ports, both with label A. in the diagrams we omit this node and draw simply a line as it is often done in proof nets.
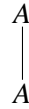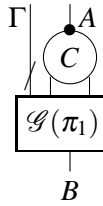
- Exchange. If π_1 is a proof ending in $\Gamma, A, B, \Delta \vdash C$, then we can build a proof π of $\Gamma, B, A, \Delta \vdash C$, using the exchange rule, and a graph $\mathscr{G}(\pi)$ where the exchange rule is encoded by exchanging two edges:
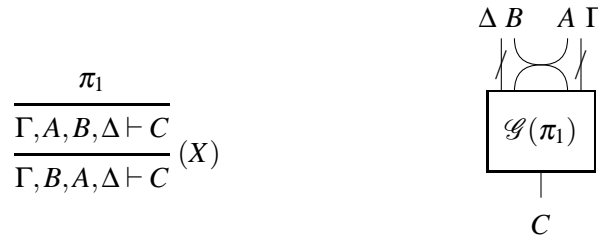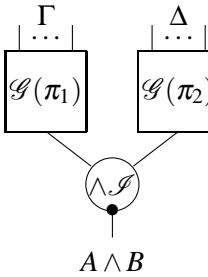
- Weakening. If π_1 is a proof ending in $\Gamma \vdash B$, then we can build a proof π of $\Gamma, A \vdash B$ using the weakening rule, and a graph $\mathscr{G}(\pi)$ as follows, where we explicitly mark the erasing port in the node W:

$$\frac{\dfrac{\pi_1}{\Gamma \vdash B}}{\Gamma, A \vdash B} (W)$$
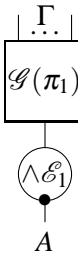
- Contraction. If π_1 is a proof ending in $\Gamma, A, A \vdash B$, then we can build a proof π of $\Gamma, A \vdash B$ using the contraction rule, and a graph $\mathscr{G}(\pi)$, where we explicitly mark the copying port in the node C:

$$\frac{\dfrac{\pi_1}{\Gamma, A, A \vdash B}}{\Gamma, A \vdash B} (C)$$

- If π_1 is a proof ending in $\Gamma \vdash A$ and π_2 is a proof ending in $\Delta \vdash B$, then we can build a proof π ending with $\Gamma, \Delta \vdash A \wedge B$ using the $\wedge \mathscr{I}$ rule, and a graph $\mathscr{G}(\pi)$, where we introduce a new node $\wedge \mathscr{I}$ corresponding to the rule, and explicitly mark its conclusion port:

$$\frac{\dfrac{\pi_1}{\Gamma \vdash A} \quad \dfrac{\pi_2}{\Delta \vdash B}}{\Gamma, \Delta \vdash A \wedge B} (\wedge \mathscr{I})$$

- If π_1 is a proof ending in $\Gamma \vdash A \wedge B$, then we can build a proof π of $\Gamma \vdash A$ using the $\wedge \mathscr{E}_1$ rule, and a graph $\mathscr{G}(\pi)$, where we introduce a new node $\wedge \mathscr{E}_1$:

$$\frac{\dfrac{\pi_1}{\Gamma \vdash A \wedge B}}{\Gamma \vdash A} (\wedge \mathscr{E}_1)$$

- If π_1 is a proof ending in $\Gamma \vdash A \wedge B$, then we can build a proof π of $\Gamma \vdash B$ using the $\wedge \mathscr{E}_2$ rule, and a graph $\mathscr{G}(\pi)$ where we introduce a new node $\wedge \mathscr{E}_2$. The graph is similar to the previous case, except that we use a node $\wedge \mathscr{E}_2$ instead of $\wedge \mathscr{E}_1$, and conclude B instead of A.

- If π_1 is a proof ending in $\Gamma, A \vdash B$ then we can build a proof π of $\Gamma \vdash A \Rightarrow B$ using the $\Rightarrow \mathscr{I}$ rule, and a graph $\mathscr{G}(\pi)$:

$$\frac{\dfrac{\pi_1}{\Gamma,A \vdash B}}{\Gamma \vdash A \Rightarrow B}\ (\Rightarrow \mathscr{I})$$

$$A \Rightarrow B$$
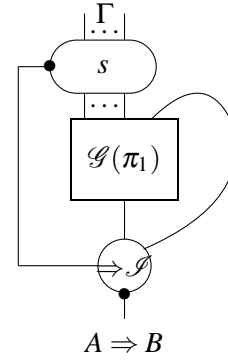
where we have introduced a new node $\Rightarrow \mathscr{I}$ corresponding to the rule, and a second node s with variable arity (formally, a family of nodes) that represents the scope of the rule: when the assumption A is discharged we mark all the other assumptions. This structure plays a role to visually represent scope, but more importantly, it will play a crucial role in the dynamics of cut-elimination that we give later. We remark that if Γ is empty (i.e. the proof is closed), then we do not need this extra node, and will just draw the $\Rightarrow \mathscr{I}$ node. Note also that the node s does not correspond to a connective.

- If π_1 is a proof ending in $\Gamma \vdash A \Rightarrow B$ and π_2 is a proof ending in $\Delta \vdash A$, then we can build a proof π of $\Gamma, \Delta \vdash B$ using the $\Rightarrow \mathscr{E}$ rule, and a graph $\mathscr{G}(\pi)$, where we introduce a new node $\Rightarrow \mathscr{E}$:

$$\frac{\dfrac{\pi_1}{\Gamma \vdash A \Rightarrow B} \qquad \dfrac{\pi_2}{\Delta \vdash A}}{\Gamma,\Delta \vdash B}\ (\Rightarrow \mathscr{E})$$

$$B$$
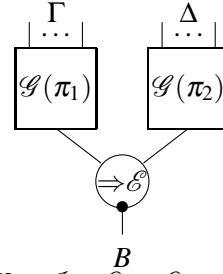
The set of node names introduced for the logic is $\mathscr{N}_{\mathscr{L}} = \{C, W, \wedge \mathscr{I}, \wedge \mathscr{E}_1, \wedge \mathscr{E}_2, \Rightarrow \mathscr{I}, \Rightarrow \mathscr{E}, s\}$.

3.1   Example

To illustrate the translation, we give here an example proof and the corresponding graph. The proof is of $A \wedge B \vdash B \wedge A$, which shows that $\wedge$ is commutative, and in the graph the formulae on the edges can be read from the proof:

$$\frac{\dfrac{\dfrac{\overline{A \wedge B \vdash A \wedge B}\ (\mathsf{Ax})}{A \wedge B \vdash B}\ (\wedge \mathscr{E}_2) \qquad \dfrac{\overline{A \wedge B \vdash A \wedge B}\ (\mathsf{Ax})}{A \wedge B \vdash A}\ (\wedge \mathscr{E}_1)}{A \wedge B, A \wedge B \vdash B \wedge A}\ (\wedge \mathscr{I})}{A \wedge B \vdash B \wedge A}\ (C)$$
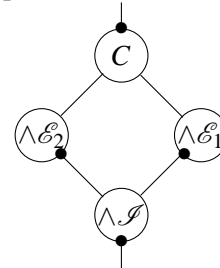
In this example, we can see the true structure of the underlying proof. Contraction was the last rule used in the derivation and is the last rule in the diagram, at the top since the structural rules work on the left of the $\vdash$ symbol and at the top of the diagrams (see the translation above).

   Note that we have chosen to use the multiplicative presentation of intuitionistic logic in the sequent calculus, additive is an alternative, and this would lead to a different graphical representation. An essential issue is that the graphical representation is faithful to this choice. In this example, we invite the reader to apply one more rule ($\Rightarrow \mathscr{I}$) to give a proof of $\vdash A \wedge B \Rightarrow B \wedge A$.
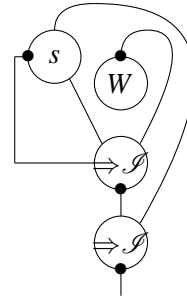
A second example illustrates one of the axioms of intuitionistic logic: $A \Rightarrow B \Rightarrow A$. The proof and the graph are the following, where we demonstrate the two different translations of $\Rightarrow \mathscr{I}$, one closed using just an $\Rightarrow \mathscr{I}$ node, and one with free variables using an *s* node.

$$\cfrac{\cfrac{\cfrac{\overline{A \vdash A} \ (\mathsf{Ax})}{A, B \vdash A} \ (W)}{A \vdash B \Rightarrow A} \ (\Rightarrow \mathscr{I})}{\vdash A \Rightarrow B \Rightarrow A} \ (\Rightarrow \mathscr{I})$$

3.2   Normalisation

Next we turn to the normalisation process in this graphical setting. Our aim is to show graph transformations, formalised as port graph rewrite rules, for each of the normalisation steps given in Definition 1.

- $(\wedge \mathscr{I})$ followed by $(\wedge \mathscr{E}_1)$: we attach the weakening node W to the graph representing π_2. There is a symmetric case for $(\wedge \mathscr{I})$ followed by $(\wedge \mathscr{E}_2)$.
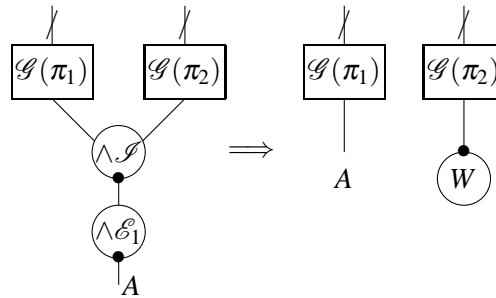
- $(\Rightarrow \mathscr{I})$ followed by $(\Rightarrow \mathscr{E})$:
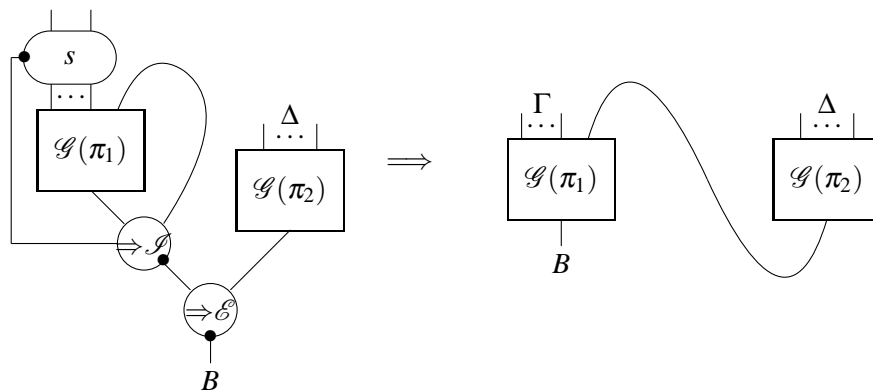
This rule performs a substitution. One of the most beautiful aspects of the notation is the clear explication of the substitution process, which can be seen as replacing an axiom in the proof π_1 with the proof π_2.

We will also consider the following simplification rule, optimising proofs when C is followed by W on the same formula. Thus, the following proof:

$$\frac{\dfrac{\Gamma,A \vdash B}{\Gamma,A,A \vdash B}\ (W)}{\Gamma,A \vdash B}\ (C)$$

will be represented and simplified by the following:
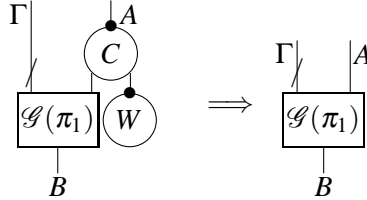
3.3   Copy and Erase

The copy and erase nodes, which correspond to the contraction and weakening rules, require specific rules in order to copy or erase proofs. This is one of the points where interaction net and proof net representations differ. In interaction nets, the copy and erasing processes are performed by traversing the net after it has been normalised (in this way, redexes cannot be copied, which ensures a more efficient implementation). On the other hand, the low level details of the copy and erase rules obscure the understanding of the logic. In the syntax described above, we are free to give global rules for copy and erase as in proof nets (but the price to pay for this is a more involved notion of graph rewriting, with an expensive matching algorithm). We are also free to choose low level, interaction net style rules, which have a simple matching algorithm and are better if we need to analyse the cost of the normalisation process.

Erasing   Here we show that the nets arising from the translation function can be erased, either by global steps on W nodes or using the ε agent to erase locally, thus showing that the weakening cut elimination step above can be fully simulated.

- Erasing a node $\alpha \in \mathcal{N_{L}}$, $\alpha \neq \Rightarrow\mathcal{I}$:
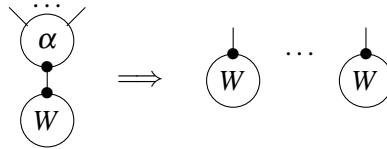
- Erasing $\Rightarrow\mathcal{I}$:
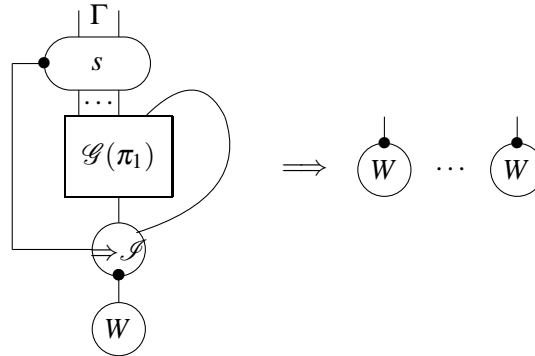
This transformation is a global reduction step: it requires that we identify the graph for π_1, which in turn relies on a notion of pattern matching that is not easy to implement (cf. subgraph isomorphism [16]).

Alternatively, one can use ε nodes that perform small-step erasing, in which case the pattern matching algorithm is trivial. In this case, the previous rule has a left-hand side consisting of just W, $\Rightarrow\mathscr{I}$ and s, and we have a reduction to:
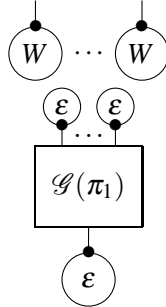

In this way, we can provide a low level definition of weakening which is better adapted for fine grained analysis of the erasing process. The rules for ε, with $\alpha \in \mathscr{N}_{\mathscr{L}} \cup \{\varepsilon\}$, are below (note that if α is ε the right hand side is an empty graph):
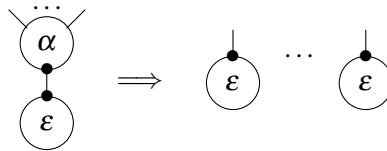
In this case the cost of erasing the graph (i.e., the number of rewriting steps involved) depends on its size.

Lemma 1 (Erasing) *Let $N = \mathscr{G}(\pi)$, for any proof π of $A_1,\ldots,A_{n-1} \vdash A_n$. Then using n ε nodes there is a sequence of rewriting steps that erase N, as shown in the following diagram, where the right-hand side is the empty graph:*
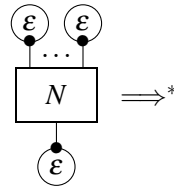
Proof: By induction on π. The proof follows a similar structure to the Duplication Lemma (see Lemma 2 below) that we shall give later. (The case for duplication is slightly more interesting.) $\square$

Duplication   Next we address the issue of duplication: specifically, we show that the graphs arising from the translation function can be copied, either by global steps on C nodes or by using the δ agent to copy step-by-step. We first show the rules for the C node.
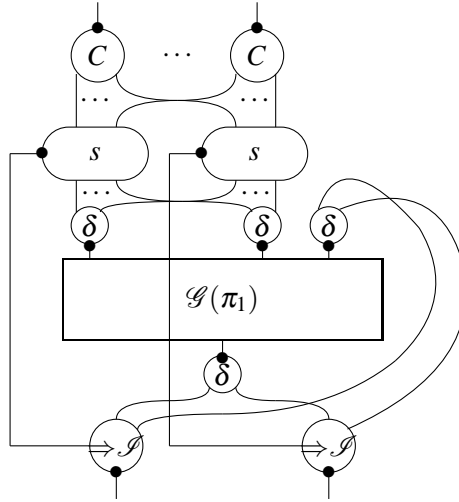
- Copying a node $\alpha \neq \Rightarrow\mathscr{I}$, $\alpha \in \mathscr{N}_{\mathscr{L}}$:
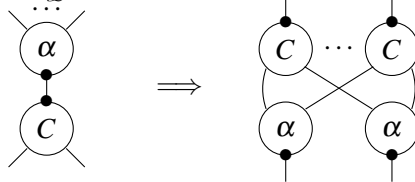

- Copying $\Rightarrow\mathscr{I}$:
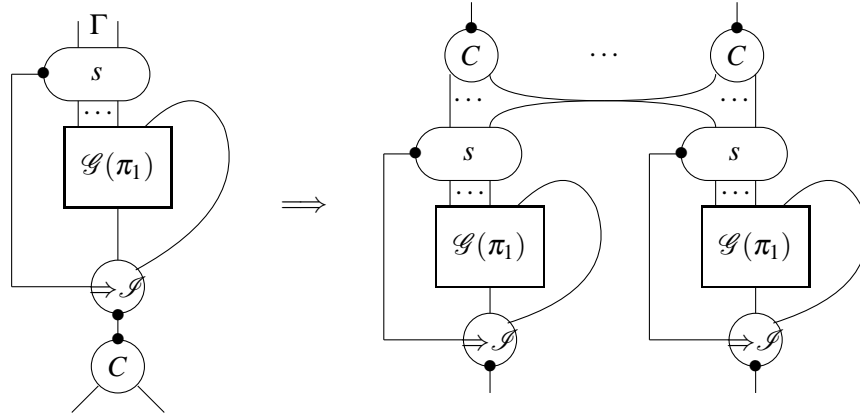
Alternatively, one can use δ nodes to perform small-step copying. In this case the left-hand side of the rule is just C with $\Rightarrow\mathcal{I}$ and s, and we have a reduction to:

We now introduce the rewrite rules for the δ nodes. Let α be any node in $\mathcal{N}_{\mathcal{L}}$. Then δ copies the node and propagates itself to copy the rest of the graph, as shown below. The rule for δ with δ ends the duplication process.
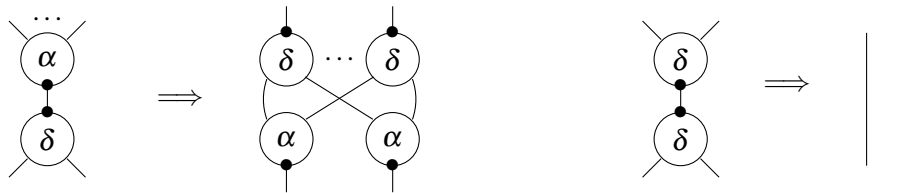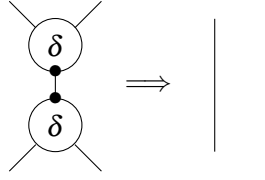

Lemma 2 (Duplication) *Let $N = \mathcal{G}(\pi)$, for any proof π of $A_1, \ldots, A_{n-1} \vdash A_n$. Then using n δ nodes there is a sequence of rewriting steps that duplicates N, as shown in the following diagram:*
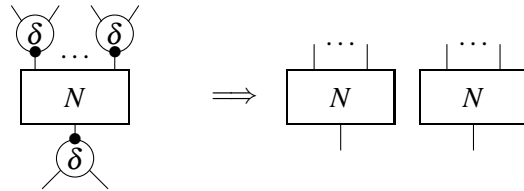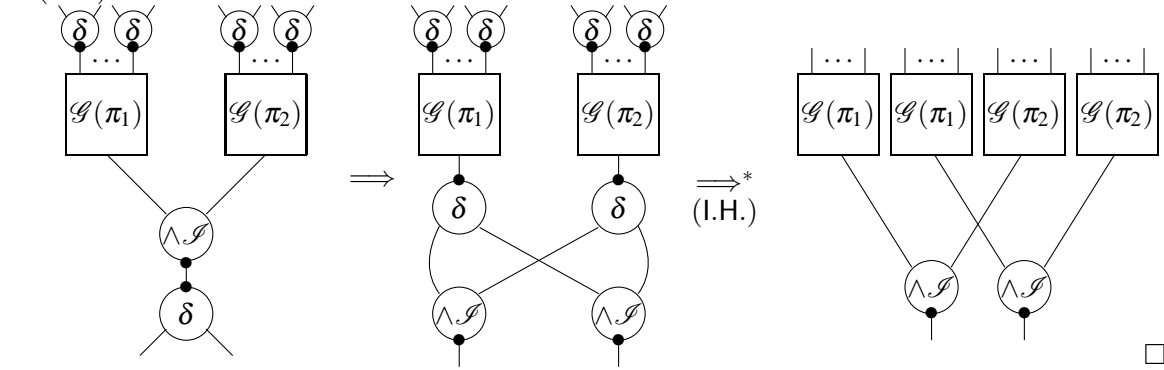

Proof: By induction on the depth of the proof π. We show the cases for Axiom and $(\wedge\mathcal{I})$.

- Axiom:

- $(\wedge\mathscr{I})$:

Proposition 1 *(Correctness) For each normalisation step transforming π to π', there is a transformation from $\mathscr{G}(\pi) \Longrightarrow^* \mathscr{G}(\pi')$.*

Proof: For the first normalisation rule, π is a proof build from π_1 and π_2 using a $(\wedge\mathscr{I})$ followed by $(\wedge\mathscr{E}_1)$ (similar for $(\wedge\mathscr{E}_2)$), and $\mathscr{G}(\pi')$ is a graph consisting of $\mathscr{G}(\pi_1)$ and a W node attached to each hypothesis in Δ (note that Δ are the hypothesis for π_2). Using the normalisation step in Section 3.2, from $\mathscr{G}(\pi)$ one obtains a graph consisting of $\mathscr{G}(\pi_1)$ and a W node attached to the conclusion port of $\mathscr{G}(\pi_2)$. By induction on π_2, and relying on Lemma 1, one can proof that a graph consisting to a W node attached to the conclusion port of $\mathscr{G}(\pi_2)$, reduces to a graph consisting of a W node attached to each hypothesis in Δ. For the second normalisation rule, we rely on Lemma 2 to prove a similar result for C. □

## 4 Graphs for the Linear λ-calculus

Now that we have seen what graph reduction is for the logic, we briefly look at it again, but from a different perspective. There are standard ways of representing the λ-calculus as graphs, and the reduction mechanism as a graph rewriting system. We restrict this section to the linear case which is simpler, and generalises using the structural rules of the previous section. The general form of the translation generating a graph from a term t is the following:
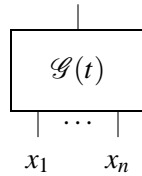
where the free variables of t are $x_1,\ldots,x_n$. The translation $\mathscr{G}(\cdot)$ is given inductively over the structure of the linear term t. We shall often drop the labelling of the edges when there is no ambiguity. For abstractions we assume (without loss of generality) that the (unique occurrence of the) variable x occurs in the leftmost position of the free variables of $\mathscr{G}(u)$. Notice also that, in the case of applications there will not be any common free variables between the graphs for u and v by the linearity constraint.

$$\mathcal{G}(x) = \Big|_{\displaystyle x} \qquad \mathcal{G}(\lambda x.u) = \qquad \mathcal{G}(uv) =$$
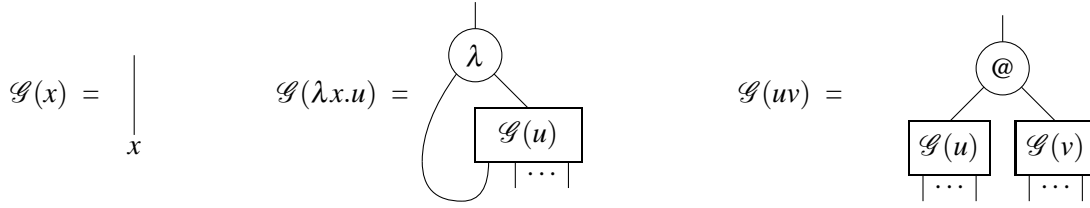
As examples consider the graphs for: $(\lambda x.x)(\lambda x.x)$ and $(\lambda xy.yx)$.
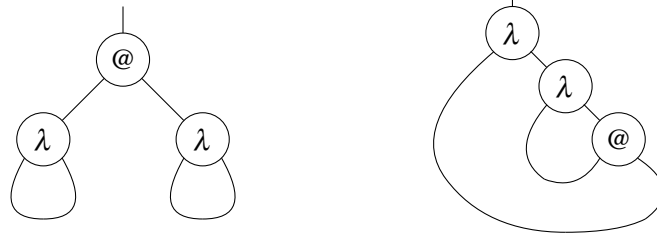
It is not difficult to see that replacing $\Rightarrow\mathcal{I}^1$ by λ and $\Rightarrow\mathcal{E}$ by @, and changing the orientation of these graphs we obtain exactly the same system of graph reduction given for the logic. This all leads to visual confirmation of the Curry-Howard isomorphism, where we can think of graphs corresponding to proofs, types corresponding to formulas, and graph reduction to normalisation.

We now turn to reduction in the linear λ-calculus for these graphs, and set up a notion of *linear graph reduction*. The idea is quite simple: we will draw the graph for the term $(\lambda x.t)u$ and another for $t[u/x]$ and try to deduce the corresponding graph reduction step(s). The required reduction is given by:

$$\Longrightarrow$$
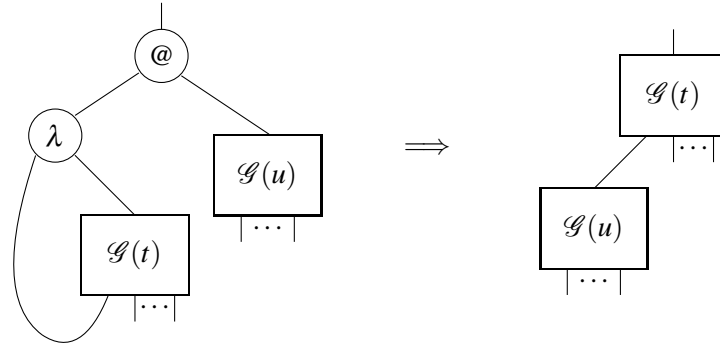
Example 1 *Here are a couple of examples of linear graph reduction. The first example is the identity applied to the identity function. Now $\mathcal{G}((\lambda x.x)(\lambda x.x))$ (which we call N) reduces to $\mathcal{G}(\lambda x.x)$ (which we call N') with one application of the rewrite rule:*

$$N \equiv \qquad \Longrightarrow \qquad \equiv N'$$
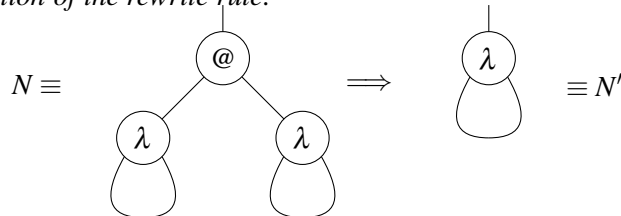
Note that there was only one graph reduction step required here: the β-reduction step, together with the substitution, was captured in a single rewrite. The advantage of this particular system of graph rewriting for the λ-calculus is that substitution is always done for free. As a larger example, consider the term $(\lambda xy.yx)(\lambda x.x)(\lambda x.x)$. As a graph reduction, this simply becomes the following:

---

[1]Simplified to the linear case.
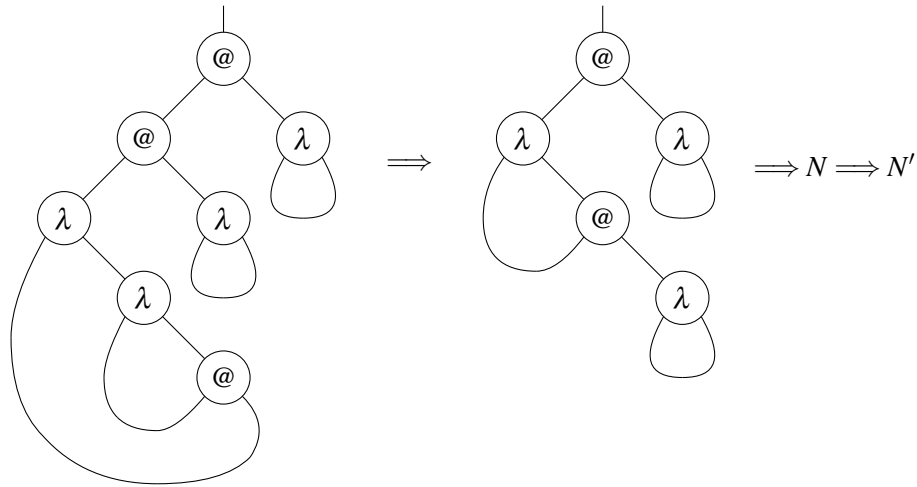
where N and N' are the nets of the previous example. Thus, the total number of graph rewrite steps is just three, corresponding exactly to the number of β-reductions performed at the level of syntax. One can see that there are a lot of benefits from a graphical notation for this simple calculus: the graph rewriting process is local, *meaning that at any time we only rewrite the part of the graph connecting the application and abstraction; the rest of the graph remains unchanged.*

5 Discussion and Conclusion

Our goal was to provide a notation for intuitionistic logic (also for the λ-calculus through the Curry-Howard isomorphism) that shares some of the advantages of previous graphical notions such as proof nets and interaction nets, but also simplifies and alleviates some of the constraints.

- The graphical notation brings out the structure of the proof visually, close to the abstract syntax, and consequently we believe it to be quite natural.

- This notation is preserved under normalisation (computation) which means that we can animate the process. As part of this, we can better understand the process of normalisation and substitution.

- As the examples show, the diagrams also alleviate much of the syntactic clutter which helps to bring out the structure of the underlying proof.

- We have established that normalisation preserves the graphical notation, but we have assumed that proofs always come from a natural deduction proof (i.e., through a translation). We have deliberately avoided the question as to when an arbitrary graph built from the nodes given is a valid proof however. These questions are difficult to solve, and until we have established the usefulness of the notation, we need not invest effort into this. However, it remains a very important question that will need to be addressed.

- For the λ-calculus, our approach and motivation is similar to that of [5]. Our graphs are closer to the abstract syntax trees, and we believe this is easier to relate to the syntax in addition to allowing the process of substitution to be controlled precisely.

- Since the λ-calculus is the foundational calculus underlying functional programming, this gives a starting point for a visual approach for this paradigm.

References

[1] O. Andrei. *A Rewriting Calculus for Graphs: Applications to Biology and Autonomo us Systems*. PhD thesis, Institut National Polytechnique de Lorraine, http://tel.archives-ouvertes.fr/tel-00337558/fr/, 2008.

[2] O. Andrei, M. Fernández, H. Kirchner, G. M. a nd O. Namet, and B. Pinaud. Porgy: Strategy driven inter-active transformation of graphs. In *Proceedings of TERMGRAPH 2011, Saarbrucken*. EPTCS, 2011. In this volume.

[3] O. Andrei and H. Kirchner. A Rewriting Calculus for Multigraphs with Ports. In *Proceedings of RULE'07*, volume 219 of *Electronic Notes in Theoretical Computer Science*, pages 67–82, 2008.

[4] A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages*, volume 45 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

[5] W. Citrin, R. Hall, and B. Zorn. Programming with visual expressions. In *In Proceedings of the 11th IEEE Symposium on Visual Languages*, pages 294–301. IEEE Computer Society Press, 1995.

[6] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation - part i: Basic concepts and double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 163–246. World Scientific, 1997.

[7] H. Geuvers and I. Loeb. Natural deduction via graphs: formal definition and computation rules. *Mathematical Structures in Computer Science*, 17(3):485–526, 2007.

[8] J.-Y. Girard. Linear Logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[9] J.-Y. Girard. Linear logic : its syntax and semantics. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, number 222 in London Mathematical Society Lecture Note Series, pages 1–42. Cambridge University Press, 1995.

[10] J. Goubault-Larrecq and I. Mackie. *Proof Theory and Automated Deduction*, volume 6 of *Applied Logic Series*. Kluwer Academic Publishers, Dordrecht, May 1997.

[11] A. Habel, J. Müller, and D. Plump. Double-pushout graph transformation revisited. *Mathematical Structures in Computer Science*, 11(5):637–688, 2001.

[12] Y. Lafont. Interaction nets. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 95–108. ACM Press, Jan. 1990.

[13] J. Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 16–30. ACM Press, Jan. 1990.

[14] R. Milner. Axioms for bigraphical structure. *Mathematical Structures in Computer Science*, 15(6):1005–1032, 2005.

[15] D. Prawitz. *Natural Deduction, A Proof-Theoretical Study*. Almqvist and Wiskell, Stockholm, 1965.

[16] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23:31–42, January 1976.