Interaction Systems *

Andrea Asperti[1] and Cosimo Laneve[2]

[1] Dip. di Matematica, P.za di Porta S. Donato, 5, 40127 Bologna, Italy.
[2] INRIA Sophia Antipolis, 2004 Route des Lucioles BP 93, 06902 Valbonne, France.

Abstract. A new class of higher order rewriting systems, called Interaction Systems (IS's), was introduced by the author in [2]. Interaction Systems provide a nice integration of the functional paradigm with a rich class of data structures and basic control flow constructs such as conditionals and (primitive or general) recursion.
In this paper, we focus on the relation between Interaction Systems and Intuitionistic Logic, via the Curry-Howard analogy. Pursuing this analogy, we define an encoding of Interaction Systems into Girard's Linear Logic, passing through Danos and Regnier's *pure nets* (IS's are untyped). Using the local implementation of boxes in [9], the translation of IS's into Pure Nets provides the intermediate step towards their *optimal* implementation (in Lamping style [12]) described in [4].

1 Introduction

There are two possible ways to understand Interaction Systems. From one side they are the *intuitionistic* generalization of Lafont's Interaction Nets [11], from which we borrow the logical setting, the bipartition of operators into constructors and destructors, and the principle of binary interaction. From the other side, Interaction Systems (IS's) are the subclass of Klop's CRS [10] where the Curry-Howard (proofs as propositions) analogy still makes sense. This means that we may associate every IS with a suitable "intuitionistic" system: constructors and destructors respectively correspond to right and left introduction rules, interaction is cut, and computation is cut-elimination.

Interaction Systems have their original motivation in the problem of generalizing Lamping's optimal graph reduction technique to supersystems of λ-calculus [2, 4, 13]. From [9], it is clear that Lamping's sharing operators (fan, croissant and bracket), provide a very abstract framework for an optimal implementation of the *structural part* of Intuitionistic Systems (i.e., via the Curry-Howard analogy, the part charged of the management of resources in functional languages), passing through a local implementation of the box of Linear Logic. Therefore it seemed that Lamping-Gonthier's evaluation style could be *smoothly generalized* to a larger class of systems by just replacing the *logical part* of Intuitionistic Logic (namely, abstraction, application and their interaction through β-reduction), with a richer collection of dual operators, binarily interacting via

* Partially supported by the ESPRIT Basic Research Project 6454 - CONFER.

cuts. Note that the duality of forms comes from the logical distinction into right and left introduction rules. From the point of view of (the practice of) optimal reduction, it is important to keep close to the above logical paradigm, in order to a have a simple interface between the logical and the structural part of the system [4].

The main property of an Interaction System as a rewriting system, is its *sequential nature*; in particular, the leftmost outermost reduction only reduces needed redexes. Note that this is another main brick in the theory of optimal reductions [14]: a reduction strategy is optimal if and only if it reduces maximal families of *needed* redexes. For this reason, it is not fair to compare IS's with the full expressive power of CRS's. They can be only reasonably compared with other known sequential subclasses of the latters, such as Left Normal CRS's [10]. With respect to Left Normal CRS's, Interaction Systems have a strong logical status, a deep relation with Lafont's Interaction Nets, interesting applications to the theory of data types (due to the bipartition of operators into constructors and destructors), and nice perspectives in the typing direction. Moreover, we conjecture that every Left Normal CRS has an optimal encoding as an Interaction System.

In this paper we focus on the relation between logics and Interaction Systems. In particular, every Interaction System is associated with a suitable Intuitionistic System, which is turned into a Linear System by the usual refinement of the structural rules via the two modalities "!" (of course) and "?" (why not). Actually, since we work in an untyped setting, we shall directly define a translation of Interaction Systems into (a suitable variant of) Danos and Regnier's pure nets [6, 15]. Then we prove the correctness of the pure-nets implementation by defining a procedure reading IS-terms back from the nets. This readback and the proof of correctness are adaptations to a simple case of the approach in [3]. Indeed, the relationships between the optimal implementation in [4, 3] and the pure-net description are quite strong: the first one can be obtained from the latter by

1. implementing boxes in a "local" way, according to the principles fixed in [9];
2. keeping advantage of the sharing expressed by metavariables in the implementation of rewriting rules (see [4, 3, 13]).

The paper is structured in the following way. In Section 2 we give the formal definition of Interaction Systems, providing some examples. In Section 3 we study the *intuitionistic nature* of IS's, and their relation with Lafont's Interaction Nets. In Sections 4 and 5 we define the pure-net description of every IS. Finally, Section 6 is devoted to an analysis of the main properties of this translation.

2 Interaction Systems

An Interaction System is defined by a *signature* Σ and a set of *rewriting rules* R.

The signature Σ consists of a denumerable set of *variables* and a set of *forms*. The set of forms is partitioned into two disjoint sets Σ^+ and Σ^-, representing *constructors* (ranged over by $\mathbf{c}$) and *destructors* (ranged over by $\mathbf{d}$). Variables will be ranged over by $x, y, z, \cdots$, possibly indexed. Vectors of variables will be denoted by $\mathbf{x}_i$ where i is the length of the vector (often omitted).

Each form can work as a binder. In the arity of the form we must specify not only the number of arguments, but also, for each argument, the number of variables it is supposed to bind. Thus, the *arity* of a form $\mathbf{f}$, is a finite (possibly empty) sequence of naturals. Moreover, we have the constraint that the arity of every destructor $\mathbf{d} \in \Sigma^-$ has a leading 0 (i.e., it cannot bind over its first argument). The reason for this restriction is that, in Lafont's notation [11], at the first argument we find the *principal port* of the destructor, that is the (unique) port where we will have interaction (local sequentiality).

Expressions, ranged over by $t, t_1, \cdots$, are inductively generated as follows:

a. every variable is an expression;
b. if $\mathbf{f}$ is a form of arity $k_1 \cdots k_n$ and $t_1, \cdots, t_n$ are expressions then $\mathbf{f}(\mathbf{x}^1_{k_1}.\, t_1, \cdots, \mathbf{x}^n_{k_n}.\, t_n)$ is an expression.

Free and bound occurrences of variables are defined in the obvious way. As usual, we will identify terms up to renaming of bound variables (α-conversion).

Rewriting rules are described by using schemas or *metaexpressions*. A metaexpression is an expression with *metavariables*, ranged over by $X, Y, \cdots$, possibly indexed (see [1] for more details). Metaexpressions will be denoted by $H, H_1 \cdots$.

A *rewriting rule* is a pair of metaexpressions, written $H_1 \rightarrow H_2$, where H_1 (the *left hand side* of the rule, lhs for short) has the following format

$$\mathbf{d}(\mathbf{c}(\mathbf{x}^1_{k_1}.\, X_1, \cdots, \mathbf{x}^m_{k_m}.\, X_m), \cdots, \mathbf{x}^n_{k_n}.\, X_n)$$

and $i \neq j$ implies $X_i \neq X_j$ (*left linearity*). The arity of $\mathbf{d}$ is $0k_{m+1} \cdots k_n$ and that of $\mathbf{c}$ is $k_1 \cdots k_m$.

The *right hand side* H_2 (rhs, for short) is every *closed* metaexpression, whose metavariables are already in the lhs and built up by the following syntax

$$H \;::=\; x \;\mid\; \mathbf{f}(\mathbf{x}^1_{a_1}.\, H_1, \cdots, \mathbf{x}^j_{a_j}.\, H_j) \;\mid\; X_i[{}^{H_1}/_{x^i_1}, \cdots, {}^{H_{k_i}}/_{x^i_{k_i}}]$$

The expression $X[{}^{H_1}/_{x_1}, \cdots, {}^{H_n}/_{x_n}]$ denotes a meta-operation of substitution, as in the λ-calculus.

Finally, the set of rewriting rules must be *non-ambiguous*, i.e. there exists at most one rewriting rule for every pair $\mathbf{d}$-$\mathbf{c}$.

Interaction Systems are readily a subsystem of Klop's (Orthogonal) Combinatory Reduction Systems [10, 1]. We just added a bipartition of operators into constructors and destructors, and imposed a suitable constraint on the shape of the lhs of each rule. As a subclass of non ambiguous, left-linear CRS's, Interaction Systems inherit all good properties of the formers (Church Rosser, finite development, ...).

Example 1. (**The λ-calculus**) The application @ is a destructor of arity 00, and λ is a constructor of arity 1. The only rewriting rule is β-reduction:

$$@(\lambda(x.X),\, Y) \;\rightarrow\; X[^{Y}/_{x}].$$

(**Primitive Recursion**) An Interaction System where no form is a binder is called *discrete*. A typical discrete IS is *Primitive Recursion*. There are only two constructors 0 and succ. Composition of two functions $\mathbf{f}(-)$ and $\mathbf{g}(-)$ is obviously expressed as $\mathbf{f}(\mathbf{g}(-))$. The primitive recursion scheme has already the correct IS-shape:

$$\begin{array}{l} \mathbf{d}(0, X) \;\rightarrow\; \mathbf{h}(X) \\ \mathbf{d}(\mathtt{succ}(X), Y) \;\rightarrow \mathbf{f}(X,\, Y,\, \mathrm{g}(X, Y)) \end{array}$$

For instance, we may define

$$\begin{array}{ll} \mathtt{add}(0, X) \;\rightarrow\; X & \mathtt{add}(\mathtt{succ}(X), Y) \;\rightarrow\; \mathtt{succ}(\mathtt{add}(X, Y)) \\ \mathtt{mult}(0, X) \;\rightarrow\; 0 & \mathtt{mult}(\mathtt{succ}(X), Y) \;\rightarrow\; \mathtt{add}(Y, \mathtt{mult}(X, Y)) \end{array}$$

In a similar way, we may define all inductive types (boolean, lists, trees, ...). Let us remark a trivial but important property of Discrete Interaction Systems, that could motivate the bipartition of forms into constructors and destructors form the point of view of the theory of data types.

Proposition 1. *If in a Discrete Interaction System we have a rewriting rule for every pair* **d**-**c***, then all closed terms in normal form may only contain constructors.*

In other words, constructors may be used to define the "abstract data type", whose definition is then unaffected by the introduction of new destructors (provided that the definition of each destructor is complete on the data). Note also that all syntactically different terms only built up by means of constructors are eventually distinct (in the equational theory obtained by reflexive, symmetric and transitive closure of the rewriting relation), since they are normal forms (and IS-rewriting is confluent).

Example 2. (**General Recursion**) The recursion operator $\mu x.M \rightarrow M[^{\mu x.M}/_{x}]$ is a bit problematic, since IS's are based on a principle of *binary* interaction. The obvious idea is to introduce a suitable "dummy" operator interacting with μ. This "dummy" operator may be indifferently regarded as a constructor or a destructor, yielding the two following encodings for μ:

$$\begin{array}{rcl} \mathbf{d}_{\mu}(\mu(x.X)) & \rightarrow & X[^{\mathbf{d}_{\mu}(\mu(x.X))}/_{x}] \\ \mu(\mathbf{c}_{\mu},\, x.X) & \rightarrow & X[^{\mu(\mathbf{c}_{\mu},\, x.X)}/_{x}] \end{array}$$

3 The Intuitionistic Nature of Interaction Systems

In this section we shall investigate the logical, intuitionistic nature of Interaction Systems, and their relation with Lafont's Interaction Nets.

3.1 Statics

An Intuitionistic System, in a *sequent calculus* presentation (*à la* Gentzen) (see [8]), consists of expressions, named *sequents*, whose shape is $A_1, \cdots A_n \vdash B$ where A_i and B are formulas. Inference rules are partitioned into three groups (in order to emphasize the relationships with IS's, we write rules by assigning terms to proofs):

(**Structural Rules**)

$$(\mathit{Exchange}) \quad \frac{\Gamma, x : A, y : B, \Delta \vdash t : C}{\Gamma, y : B, x : A, \Delta \vdash t : C}$$

$$(\mathit{Contraction}) \quad \frac{\Gamma, x : A, y : A \vdash t : C}{\Gamma, z : A, \Delta \vdash t[^z/_x, ^z/_y] : C} \qquad (\mathit{Weakening}) \quad \frac{\Gamma \vdash t : C}{\Gamma, z : A \vdash t : C}$$

(**Identity Group**)

$$(\mathit{Identity}) \quad \frac{}{x : A \vdash x : A} \qquad (\mathit{Cut}) \quad \frac{\Gamma \vdash t : A \qquad \Delta, x : A \vdash t' : B}{\Gamma, \Delta \vdash t'[^t/_x] : B}$$

(**Logical Rules**) These are the "peculiar" operations of the systems, introducing new formulae (i.e. new types) in the proof. The unique new formula introduced by each rule is called the *principal* formula of the inference. If the principal formula is in the rhs of the final sequent, the inference rule is called *right*, and *left* otherwise. Right and left introduction rules respectively correspond with *constructors* and *destructors* in IS's. The shape of these rules is:

$$\frac{\Gamma_1, \mathbf{x}^1 : \mathbf{A}^1 \vdash t_1 : B_1 \quad \cdots \quad \Gamma_m, \mathbf{x}^m : \mathbf{A}^m \vdash t_m : B_m \qquad \Delta, z : C \vdash t : D}{\Gamma_1, \cdots, \Gamma_m, \Delta, y : \mathtt{T}_{\mathbf{d}}(\mathbf{A}^1, B_1, \cdots, \mathbf{A}^m, B_m, C) \vdash t[^{\mathbf{d}(y, \mathbf{x}^1 . t_1, \cdots, \mathbf{x}^m . t_m)}/_z] : D}$$

for left introduction rules (destructors), and

$$\frac{\Gamma_1, \mathbf{x}^1 : \mathbf{A}^1 \vdash t_1 : B_1 \quad \cdots \quad \Gamma_n, \mathbf{x}^n : \mathbf{A}^n \vdash t_n : B_n}{\Gamma_1, \cdots, \Gamma_n \vdash \mathbf{c}(\mathbf{x}^1 . t_1, \cdots, \mathbf{x}^n . t_n) : \mathtt{T}_{\mathbf{c}}(\mathbf{A}^1, B_1, \cdots, \mathbf{A}^n, B_n)}$$

for right introduction rules (constructors). Above $\mathtt{T}_{\mathbf{d}}(\mathbf{A}^1, B_1, \cdots, \mathbf{A}^m, B_m, C)$ and $\mathtt{T}_{\mathbf{c}}(\mathbf{A}^1, B_1, \cdots, \mathbf{A}^n, B_n)$ are formulae (types) built up by means of the types they take as arguments and they are equal, provided they correspond to the same logical operator. The contexts Γ_i are pairwise different. We remark that the above rules have the general shape of the so-called multiplicative connectives (implication, conjunction, etc.). As in Interaction Nets, there is no means to directly represent additives (disjunction, etc.) (see Section 3.2.1 in [13]).

A standard example is implication, which gives the expressions of typed λ-calculus:

$$(\rightarrow \mathit{left}) \quad \frac{\Delta \vdash t : A \qquad z : B, \Gamma \vdash t' : C}{\Delta, y : A \rightarrow B, \Gamma \vdash t'[^{@(y,t)}/_z] : C} \qquad (\rightarrow \mathit{right}) \quad \frac{\Delta, x : A \vdash t : B}{\Delta \vdash \lambda(x.t) : A \rightarrow B}$$

An immediate consequence of the above construction is that every proof of an Intuitionistic System may be described by an IS-expression.

Following Lafont's, we may provide a graphical representation of IS-forms. This will explain some important relations between the arity of the forms and the way they link upper sequents in a proof. In the notation of Interaction Nets, a proof of a sequent $A_1, \ldots, A_n \vdash B$ is depicted as a graph with $n+1$ conclusions, n with a negative type (the inputs) and one with a positive type (the output). In particular, an axiom is just a line with one input and one output.

Every logical rule is represented by introducing a new operator in the net (a new labeled node in the graph). The operator has a *main port*, individuated by an arrow, that is associated with the *principal formula* of the logical inference. For instance, the two logical rules for implications are illustrated in Figure 1.

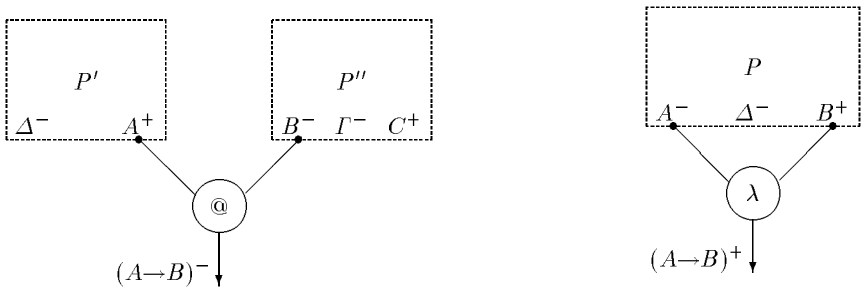


Fig. 1. The graphical representation of @ and λ.

The main port of each operator may be either an input or an output. In the first case it corresponds to a new logical assumption in the left hand side of the sequent (as for @), and the operator is a *destructor*; in the second case it corresponds to the right hand side of the sequent (as for λ), and the operator is a *constructor*. The other ports of the agents are associated with the *auxiliary* formulae of the inference rule, that is the distinguished occurrences of formulae in the upper sequents of the rule. In the two rules above, the auxiliary formulae are A and B.

The auxiliary ports of an agent may be either inputs or outputs, independently from the fact that it is a constructor or a destructor. Actually, in the general theory of Interaction Nets, which is inspired by *classical* (linear) logic, there is a complete symmetry between constructors and destructors, and no restriction at all is imposed on the type of the auxiliary ports (in other words, there is a complete symmetry between inputs and outputs). On the contrary, the fact of limiting ourselves to the intuitionistic case, imposes some obvious "functional" constraints.

Note first that auxiliary formulae may come from different upper sequents or from a single one. For instance, the auxiliary ports of @ are in different upper sequents, while those of λ are in a same one. Lafont expresses this fact by defining *partitions* of auxiliary ports. So, in the case of @, A and B are in different partitions, while in the case of λ they are in the same partition. Moreover, the polarity of an auxiliary port is the opposite of the polarity of the conclusion it has to be matched against. Then, the intuitionistic framework imposes the following constraints:

- In every partition there is at most one negative port. If a negative port exists, we shall call it an *input* partition; otherwise it is an *output* partition.
- Every agent has exactly "one output" (functionality). In particular, if the agent is a constructor, the main port is already an output, and all the partitions must be inputs. Conversely, in the case of destructors, we have exactly one output partition among the auxiliary ports, and this partition has to be a singleton.

If $k_1 \dots k_n$ is the arity of a form, every k_i denotes the number of positive ports in the i-th *input* partition. In the case of a destructor, one input partition must be a singleton (since it corresponds to the main port), so its arity is 0. By convention, in the concrete syntax of IS's, we have supposed that this is always the first partition of the destructor (this assumption is absolutely irrelevant). Summing up, a form with arity $k_1 \dots k_n$ is associated with an operator with $1 + \sum_{i=1}^{n}(k_i + 1)$ ports.

3.2 Dynamics

The main theorem of sequent calculi is that stating the redundancy of cut-rules, (i.e. every proof can be effectively turned into a cut-free one). From the operational point of view, this gives dynamics, since it equips the logical system with a rewriting systems on proofs. The rewritings rules must be specified *a priori*: a proof ending into a cut can be simplified into another one by means of some mechanism that is characteristic of that cut.

In order to ensure the possibility of eliminating *all* cuts, the cut-elimination process (term rewriting) must *terminate*. In general, this property does not hold in IS's. We just have a general correspondence between IS's and systems with an *intuitionistic nature*, but only *a posteriori* we may actually check if a particular system has good "logical" properties (cut-elimination, subformula property, ...).

Obviously, we could proceed the other way round, imposing some sufficient conditions on IS's (typing, first of all) to establish a tighter relation with logic. This is surely an interesting subject, but it is out of the scope of this paper. So, we shall merely focus on the *dynamic* aspects of cut-elimination, without worrying with termination.

Foremost, there are several kinds of cuts. The most interesting case is the *logical cut*, i.e. a cut between two dual (left-right) introduction rules for the same principal formula. In all the other cases (structural cuts), Intuitionistic Systems "eliminate" the cut by lifting it in the premise(s) of one of the rules preceding the cut (that becomes the last rule of the new proof). As in the case of λ-calculus, these kind of cuts are essentially "unobservable" in IS's (they are dealt with in the metalevel definition of substitution). So, let us concentrate on logical cuts only.

Let us consider the case of implication. Here is the typical situation:

$$\dfrac{\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda(x.t) : A \to B} \qquad \dfrac{\Delta \vdash t' : A \qquad y : B, \Theta \vdash t'' : C}{\Delta, z : A \to B, \Theta \vdash t''[{}^{@(z,t')}/_y] : C}}{\Gamma, \Delta, \Theta \vdash t''[{}^{@(z,t')}/_y][{}^{\lambda(x.t)}/_z] : C}$$

The elimination of the above cut consists in introducing two cuts of lesser grade (see [8]). The rewritten proof is:

$$\dfrac{\Delta \vdash t' : A \qquad \dfrac{\Gamma, x : A \vdash t : B \qquad y : B, \Theta \vdash t'' : C}{\Gamma, x : A, \Theta \vdash t''[{}^{t}/_y] : C}}{\Gamma, \Delta, \Theta \vdash t''[{}^{t}/_y][{}^{t'}/_x] : C}$$

Note that this meta-operation on proofs induces a rewriting rule in the underlying IS, which is, in this case, β-reduction. Indeed, the equation

$$t''[{}^{@(z,t')}/_y][{}^{\lambda(x.t)}/_z] = t''[{}^{t}/_y][{}^{t'}/_x]$$

implies β-reduction.

Let us come to the general case. Let L and R be the left and right sequent in the cut-rule, respectively. During the process of cut-elimination, the proofs ending into the premises of L and R must be considered as "black boxes" (only their "interfaces", that is their final sequents are known). During cut-elimination, one can build new proofs out of these black boxes. The only constraint is the *prohibition of having new hypotheses* in the final sequent of rewritten proof. This has two implications:

1. the variables bound by L or R (i.e. the auxiliary formulae of the rules) must be suitably filled in (typically with cuts or introducing *new* rules binding them);
2. if a new axiom is introduced by the rewriting, then the hypothesis in the lhs must be consumed (with a cut or by binding it via a logical rule) inside the rewritten proof.

According to statics, a cut in the syntax of IS's corresponds to a term of the following kind:

$$\mathbf{d}(\mathbf{c}(\mathbf{x}^1.X_1, \cdots, \mathbf{x}^m.X_m), \cdots, \mathbf{x}^n.X_n).$$

The X_i represent the proofs ending into the upper sequents of L and R (the "black boxes" above). Following the previous discussion, in the right hand sides of a rewriting rule, we may either

– introduce new variables with axioms or with weakenings (denoted by x),

or, starting from proofs $H_1, \cdots, H_n$ that have been already built up, we may

– exploit proofs X_i, provided we fill the bound hypothesis (notation $X[{}^{H_1}/_{x_1}, \cdots, {}^{H_n}/_{x_n}]$)
– or introduce a new logical rule (denoted as $\mathbf{f}(\mathbf{x}^1.\,H_1, \cdots, \mathbf{x}^n.\,H_n)$).

The other logical rules (contractions, cuts) are visible in the concrete syntax of IS's under copying or interactions. Finally, the syntactical constraint reflecting the logical absence of new hypotheses is: right hand sides of rules must be closed expressions.

Example 3. **(Naturals)** We have two constructors `0` and `succ`, respectively associated with the following right introduction rules:

$$(nat, right_0) \quad \vdash \mathtt{0} : nat \qquad\qquad (nat, right_S) \quad \frac{\Delta, \vdash n : nat}{\Delta \vdash \mathtt{succ}(n) : nat}$$

A typical destructor is `add`.

$$(nat, left_{add}) \quad \frac{\Delta \vdash p : nat \qquad \Gamma, y : nat \vdash t : A}{\Delta, \Gamma,\ x : nat \vdash t[{}^{\mathtt{add}(x,p)}/_y] : A}$$

where A can be any type. The following is an example of cut:

$$\frac{\vdash \mathtt{0} : nat \qquad \dfrac{\Delta \vdash p : nat \qquad y : nat, \Theta \vdash t : A}{\Delta, x : nat, \Theta \vdash t[{}^{\mathtt{add}(x,p)}/_y] : A}}{\Delta, \Theta \vdash t[{}^{\mathtt{add}(x,p)}/_y][{}^{\mathtt{0}}/_x] : A}$$

that is simplified into:

$$\frac{\Delta \vdash p : nat \qquad\qquad y : nat, \Theta \vdash t : A}{\Delta, \Theta \vdash t[{}^{p}/_y] : A}$$

The above elimination induces the IS-rule $\mathtt{add}(\mathtt{0}, X) \to X$, corresponding to the equation $t[{}^{\mathtt{add}(x,p)}/_y][{}^{\mathtt{0}}/_x] = t[{}^{p}/_y]$.

Example 4. **(Lists)** Lists are defined by means of two constructors `cons` and `nil` of arity 00 and ε, respectively. The typical destructors are `hd` and `tl` of arity 0. In the case of lists of integers, we may write the following introduction rules for the type *natlist*:

$$(natlist, right_{nil}) \vdash \mathtt{nil} : natlist$$

$$(natlist, right_{cons}) \ \frac{\Delta \vdash n : nat \qquad \Gamma \vdash l : natlist}{\Delta,\ \Gamma \vdash \mathtt{cons}(n, l) : natlist}$$

$$(natlist, left_{hd}) \ \frac{\Gamma, y : nat \vdash t : A}{\Gamma,\ x : natlist \vdash t[{}^{\mathtt{hd}(x)}/_y] : A}$$

$$(natlist, left_{tl}) \ \frac{\Gamma, y : natlist \vdash t : A}{\Gamma,\ x : natlist \vdash t[{}^{\mathtt{tl}(x)}/_y] : A}$$

A typical cut is:

$$\dfrac{\dfrac{\Delta \vdash n : nat \qquad \Gamma \vdash l : natlist}{\Delta, \Gamma \vdash \texttt{cons}(n, l) : natlist} \qquad \dfrac{\Theta, y : Nat \vdash t : A}{\Theta,\, x : natlist \vdash t[^{\texttt{hd}(x)}/_y] : A}}{\Delta, \Gamma, \Theta \vdash t[^{\texttt{hd}(x)}/_y][^{\texttt{cons}(n,l)}/_x]}$$

and the obvious cut elimination rule gives:

$$\dfrac{\Delta \vdash n : nat \qquad \Theta, y : nat \vdash t : A}{\Delta, \Theta \vdash t[^n/_y] : A}$$

Again, by the reduction rule $\texttt{hd}(\texttt{cons}(X, Y)) \to X$, we have $t[^{\texttt{hd}(l)}/_y][^{\texttt{cons}(n,l)}/_x] = t[^n/_y]$. As an exercise the reader can provide the cut between $(natlist, right_{cons})$ and $(natlist, left_{tl})$ and verify that it induces the following rewriting:

$$\texttt{tl}(\texttt{cons}(X, L)) \to L$$

The following proposition sums up the previous informal discussion.

Proposition 2. *Every intuitionistic system* $\mathbf{L}$ *can be encoded by an IS* $\mathbf{S}$*. Namely the proofs of* $\mathbf{L}$ *are expressions of* $\mathbf{S}$ *and cut-eliminations are rewriting rules of* $\mathbf{S}$*. The encoding is correct up to structural cuts. That is let* π *be a proof in* $\mathbf{L}$ *which can be rewritten into* π' *through an elimination of a logical cut* κ*. Let* t *be the expression encoding* π*. Then* $t \to t'$ *is an instance of the rewriting encoding* κ *and* t' *encodes a proof equivalent to* π' *modulo structural cuts.*

In order to describe an IS as an intuitionistic system (that is the opposite of what we did so far), we must face the problem of the lack of a type discipline (as in the case of the pure λ-calculus). By adding types to IS's, we can prove their logical foundation by reverting the previous paradigm. Another possibility is to generalize intuitionistic systems by dropping any type discipline. Instead of investigating here this (straightforward) generalization, we shall pursue an alternative approach, which is more useful in the perspective of optimal implementations. We are going to implement any IS into *pure nets*, a type-free variant of Girard's proof nets for Linear Logic, which have been already introduced in [6, 15] with the aim of encoding pure λ-calculus. The implementation, on one hand, should confirm the logical foundation of IS's; on the other hand, it provides the intermediate step towards the *optimal* implementation in [4, 13].

4 The system of pure nets

In [7] Girard proposed a refinement of Intuitionistic Logic called *Linear Logic*. In Linear Logic the structural rules of weakening and contraction are confined to a realm of exponential modalities, ! (*of course*) and ? (*why not*), as described by the following rules:

$$\text{(dereliction left)}\ \dfrac{A, \Delta \vdash B}{!A, \Delta \vdash B} \qquad \dfrac{\Delta \vdash A}{\Delta \vdash ?A}\ \text{(dereliction right)}$$

$$\text{(weakening)}\ \frac{\Delta \vdash B}{!A,\ \Delta \vdash B} \qquad \frac{\Delta,\ !A,\ !A \vdash B}{\Delta,\ !A \vdash B}\ \text{(contraction)}$$

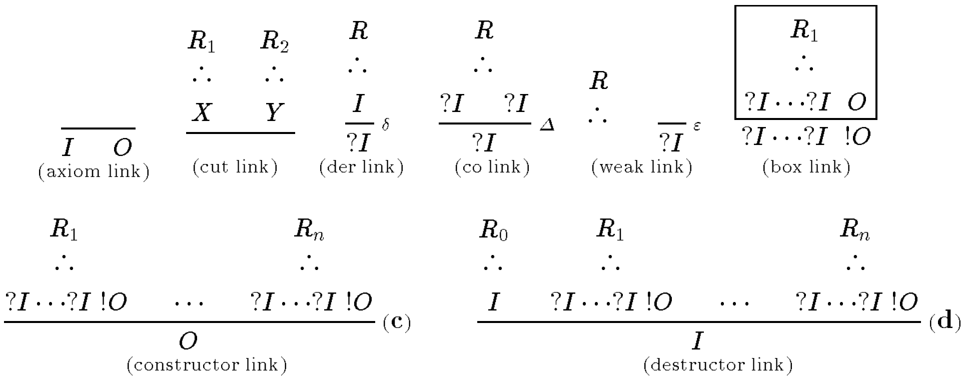
$$\text{(? left)}\ \frac{!\Delta,\ A \vdash ?B}{!\Delta,\ ?A \vdash ?B} \qquad \frac{!\Delta \vdash A}{!\Delta \vdash !A}\ \text{(! right)}$$

Since Linear Logic essentially concerns the structural part of logical systems, we may intuitively define a linear logic version of an arbitrary intuitionistic system, by relying on the usual encoding of Intuitionistic Logic.

The correspondence between IS's and Linear Logic will be the subject of this section. We shall not consider the sequent-like formalization of Linear Logic, but its "natural deduction" version, based on *proof nets* [7]. Actually, since IS's are untyped, we shall work with Danos and Regnier's [6, 15] generalization of proof nets, named *pure nets* (pure nets are essentially obtained by proof nets by just removing types or, if you like, by using exactly one recursive type $D \equiv !D \multimap D$).

In particular, we shall define a direct encoding of Interaction Systems into pure nets. From this encoding, and by adopting the "local" implementation of boxes described in [9], one may easily derive the *optimal* implementation of Interaction Systems (in Lamping style [12]) described in [4].

Definition 3. The *formulas* are I (like *input*), O (like *output*) and $?I$ and $!O$. We say that I and O (resp. $?I$ and $!O$) are dual formulas. The *pure nets* are inductively defined over formulas by the following picture:



where, in the case of the cut link, X and Y are dual formulas. For constructors and destructor links, the formulas in the premises are partitioned, according to the pure net R_i they come from. In every case, except the box link, the *conclusion* of the resulting pure net is the juxtaposition of the *free* conclusions of every pure net in the premise (i.e. those conclusions not involved in the premise of the rule) plus the conclusion of the rule. In the case of box link, *every* conclusion of the pure net R must be involved. Finally, if the nets R_i and R_j $(i \neq j)$ are in the premises of a rule, the two nets are supposed to be disjoint.

The names $(\mathbf{c})$ and $(\mathbf{d})$ in the constructor and destructor links will be replaced by the actual names of constructors and destructors.

The rewriting system for pure nets implementing IS's consists of two parts: one is common for every IS implementation (*control rules*); the other is characteristic of the particular IS under investigation (*proper rules*). We will provide a mechanism for defining the proper rules associated with a generic IS. The *control rules* are illustrated in Figure 2.

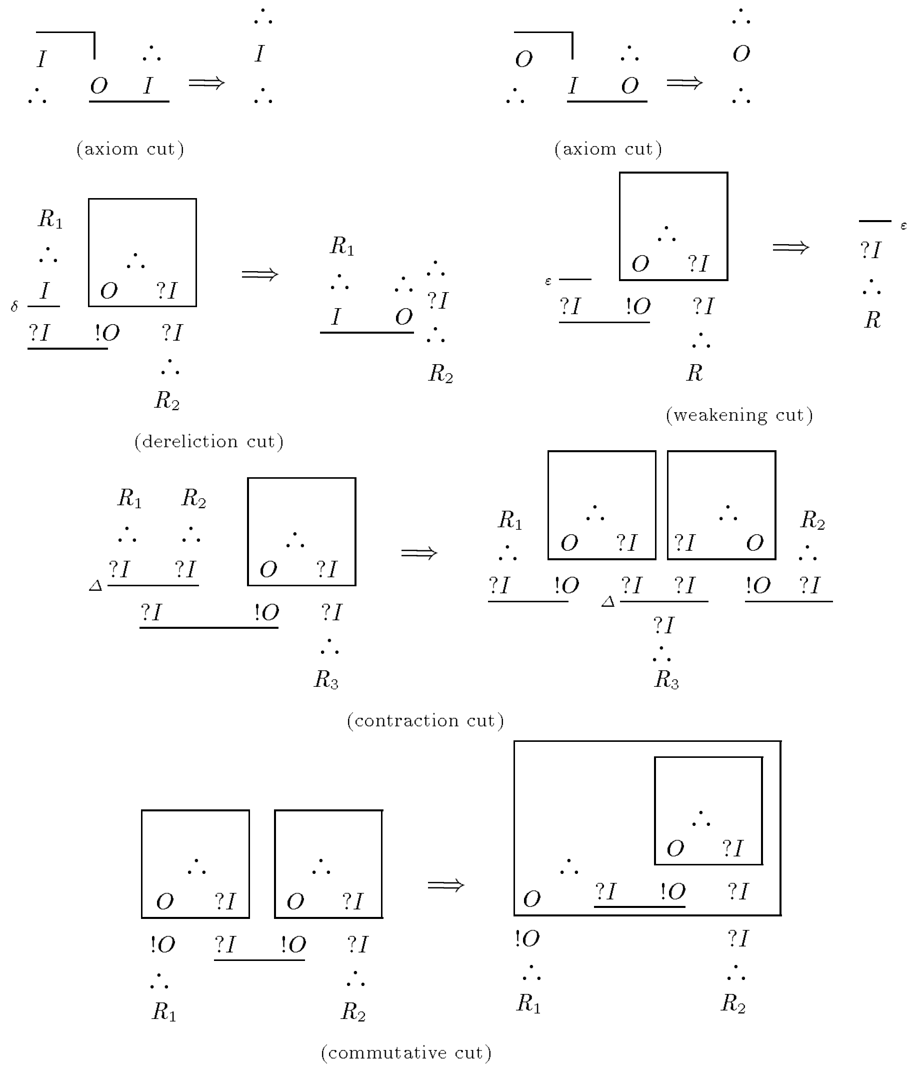


Fig. 2. Control rules

5 The implementation

In order to implement a generic IS into a pure net we must translate IS-expressions and encode IS-rewriting rules through pure net rewritings. Below we shall consider the paradigmatic case when constructors and destructors have respectively arity 1 and 01. The other cases are easily derived.

5.1 Translating the expressions

The translation $\mathcal{T}$ defined in Figure 3 gives, for any IS-expression t with n free variables, a pure net R with $n+1$ conclusions where n have the form $?I$ (the *inputs*) and one the form O (the *output*). For simplicity, in Figure 3, we have described the case when the variable bound by the destructor occurs in the body. The other case is similar to the corresponding one shown for the constructor.

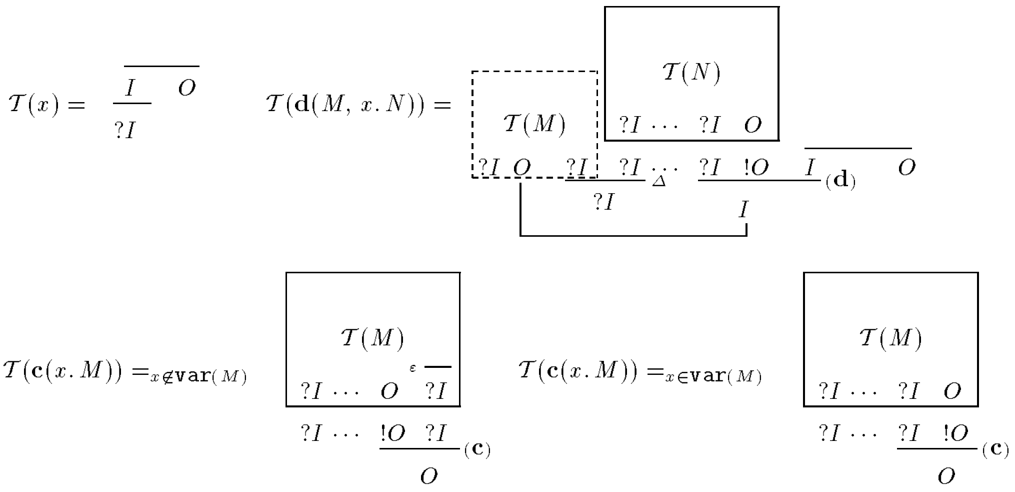


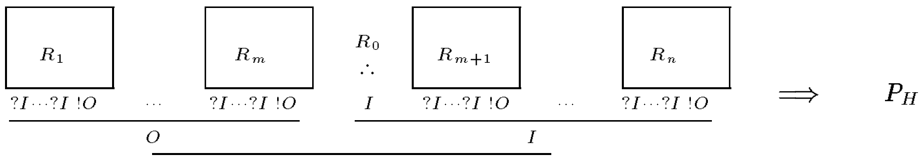
Fig. 3. The translation function $\mathcal{T}$

Observe that in the translation of the destructor the free variables in M and those in N (except x) with the same name are contracted. For simplicity we have illustrated just one contraction.

Proper Rules The proper rules describe the interactions between destructors and constructors of the IS's. An IS-rule

$$\mathbf{d}(\mathbf{c}(\mathbf{x}^1.\,X_1, \cdots, \mathbf{x}^m.\,X_m), \mathbf{x}^{m+1}.\,X_{m+1}, \cdots, \mathbf{x}^n.\,X_n) \to H$$

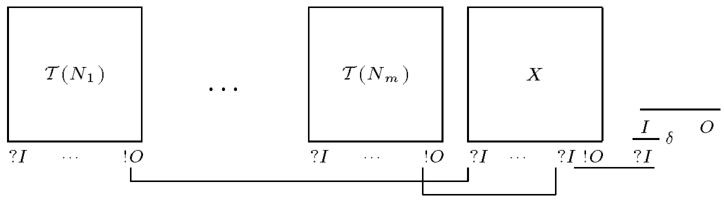
is translated in Pure nets into a rewriting rule of the following shape:



Every box R_i in the previous picture corresponds to the metavariable X_i of the IS-rule. Note that arguments of constructors/destructors are always in boxes, according to the translation. Until the redex is fired, these boxes can be only modified along the reduction by operations at their "free" inputs (inputs which are not bound by the form). In particular, they can only absorb other boxes by commutative cuts. So, by taking the box relative to the output !O of R_i, we are sure to capture the full interface of the metavariable with the form (and possibly other "free" variables, of course).

The translation of the right hand side of the rule is essentially similar to the translation of an ordinary term. The main difference is that we also have metavariables, possibly instantiated by a metaoperation of substitution. So we *extend* our previous translation function $\mathcal{T}$ to cover also this case.

(**The encoding of substitutions**) The translation of $X[{}^{N_1}/_{x_1}, \ldots, {}^{N_m}/_{x_m}]$, where the metavariable X is associated by the left hand side of the rule with a box R_i with a distinguished input for each variable x_i, is defined by the following picture



In particular, the box R_i corresponding to the metaterm X must be opened, and substitutions are obviously realized by cuts. Before doing these cuts, the arguments of the substitution must be put inside a box.

Finally, all the free variables of different instancies of a same metavariable must be shared. Note that all bound variables of metavariables are eventually "filled in" by cuts.

Example 5. The encoding of the rhs of the rewriting rule $\mathbf{d}_\mu(\mu(x.X)) \rightarrow X[{}^{\mathbf{d}_\mu(\mu(y.X[{}^{y}/_{x}]))}/_{x}]$ of Example 2 is illustrated in Figure 4.

Remark. For pictorial convenience, in Figure 4 we have omitted to draw the contractions over the *free* variables of the two instancies of the metavariable X. However, there is a subtle problem, here. Indeed, to be formal, we should define a different rewriting scheme for each possible set of free variables (corresponding to the contexts Δ, $\Gamma, \ldots$, in sequent calculi). This is not required in the lambda calculus, since β-reduction is *linear* in its metavariables. In [4] we adopted a

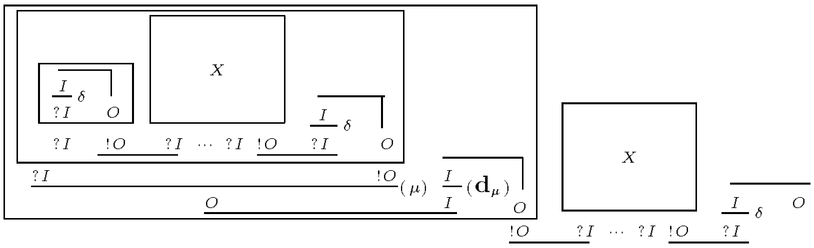


Fig. 4. The implementation of the rhs of the rule for $\mathbf{d}_\mu$-μ

different solution, applying a "linearization" procedure (essentially based on β expansion) to the rhs of each rule.

6 Properties of the implementation

Property 4. *1. The translation of every expression is a pure net;*
2. the class of pure nets is stable under reduction;
3. the rewriting system of pure nets is locally confluent.

Proof. 1. Easy check on the translation $\mathcal{T}$. 2. Examine the rhs of every rewriting rule. In every case the net replacing the redex is a pure net and is interfaced with the context by pure net links. 3. By a case analysis on the pairs of rewriting rules.

Another interesting property is the confluence of pure nets reductions. To this aim we define a reduction relation which is locally confluent, noetherian and whose transitive closure is the standard reduction of pure nets. A standard way consists of taking *developments* as basic steps of such auxiliary reduction relation. Some preliminary notions are needed.

Assume to label cuts of pure nets in order to characterize them uniquely. Then let $R \to R'$ be a pure net reduction. The *residual* of a cut labeled α in R is the set of cut labeled α in R'. A *development* of a set $H = \{\alpha, \beta, \cdots\}$ of cuts in R is a derivation $R \to R_1 \to \cdots \to R_n$ contracting at each step cuts marked by labels in H only. A development is *complete* when no label in H marks cuts in R_n.

Theorem 5. *Every development is finite.*

Proof. (*Hint*) We generalize the proof in Section 8.3 of [6]. The hard part is to prove that developments of contraction, commutation and proper cuts are finite. Both the finiteness of developments of other cuts and the factorization between these two types of development is easy.

Let H be a set of contraction, commutation and proper cuts in R. We must provide a weight for R such that it decreases as the development grows. The first property concerns the *depth* of a pure net R, namely the maximal number of nested boxes in R:

let $\sigma(R)$ be the pure net yielded by the development σ. There exists a natural η_R such that, for every development σ, the depth of $\sigma(R)$ is less or equal to η_R.

Here, w.r.t. the corresponding proof in [6], we have to take into account proper reductions, too. However, this does not causes too much trouble.

The next step consists in defining *duplicable boxes* in $\sigma(R)$ w.r.t. cuts in the family H of R of which σ is a development. This notion is inductively defined by:

- every box in the premise of a contraction cut which is residual of a cut in H after σ is a duplication box;
- every box B connected through the principal port to a duplication box with a commutative cut residual of one in H is duplicable.

Now take the $2\eta_R$-uple defined as follows:

x_{2i} is the number of duplicable boxes at depth i in $\sigma(R)$;
x_{2i+1} is the number of boxes at depth i in $\sigma(R)$.

Then it is easy to prove that, taking the lexicographic order on tuples, the above weight decreases as the development grows. $\square$

Theorem 6. *Pure nets rewritings are confluent.*

Proof. It is not difficult to prove that developments are locally confluent. Then the transitive closure of development-reduction gives the standard reduction. $\square$

The main property about the implementation is the correctness, namely that $t \longrightarrow\!\!\!\!\!\rightarrow t'$ implies $\mathcal{T}(t) \longrightarrow\!\!\!\!\!\rightarrow \mathcal{T}(t')$. This is true up to a suitable equivalence relation over proofs. The problem is due to derelictions of bound variables that do not appear in the body of the forms. During reduction, they create "garbage" which cannot be eliminated by the rewriting system considered previously. A way out is to quotient pure nets through the following axiom:

$$\begin{array}{c} R' \\ \because \\ \dfrac{\overline{?I}^{\,\delta} \quad ?I}{?I}\,\Delta \\ \therefore \\ R \end{array} \quad \Longrightarrow \quad \begin{array}{c} R' \\ \because \\ ?I \\ \therefore \\ R \end{array}$$

Up to this axiom, Danos proves the correctness in the case of λ-calculus [6]. This approach can be generalized in a straightforward way to IS's, since all the problems are related to structural rules.

Another way for proving correctness, close to the approach in [3], relies on the definition of a readback procedure for pure nets.

To this aim, we assume to have a pure net such that the free ports $?I$ keep variable names which are pairwise different. Give new variable names to ports

$?I$ connected to binders such that also these variables are pairwise different. We define the readback building back the terms by traveling through the links of the pure nets. In particular, the readback $\mathcal{R}(R)$ starts from the output O of R and consists of the following steps:

1. let A be the formula under exam; if A is marked by the variable x then the algorithm gives x and stops;
2. if A is connected through an axiom to B then the algorithm continues from B towards the link whose premise is B;
3. if A is connected through a cut with B then the algorithm continues from B towards the link whose conclusion is B;
4. if A is $?I$ and
   a. the link is a weakening then $\mathcal{R}$ stops there with no output;
   b. the link is a contraction then the algorithm continues from the conclusion B of such link;
   c. the link is a box (A is internal to the box) then $\mathcal{R}$ continues from the corresponding auxiliary port;
5. if A is I and the link is a dereliction then $\mathcal{R}$ continues from $?I$;
6. if A is $!O$ and the link is a box then $\mathcal{R}$ continues from O;
7. if A is the conclusion of a constructor-link ($\mathbf{c}$) with n formulas $!O$ in the premise, then the algorithm gives the term $\mathbf{c}(\mathbf{x}^1.t'_1, \cdots, \mathbf{x}^n.t'_n)$, where t_i are obtained by starting the algorithm from the i-th formula $!O$ in the premise and $\mathbf{x}^i$ is the vector of variables bound by the link on that formula;
8. if A is the I-premise of a destructor-link ($\mathbf{d}$) with n formulas $!O$ in the premise, then the algorithm gives the term $\mathbf{d}(t_0, \mathbf{x}^1.t'_1, \cdots, \mathbf{x}^n.t'_n)$, where t_i are obtained by starting the algorithm from the i-th formula $!O$ in the premise and $\mathbf{x}^i$ is the vector of variables bound by the link on that formula; t_0 is the term that is readback starting from the conclusion of ($\mathbf{d}$) toward the cut downstairs.

Let us remark some easy property of the readback.

Proposition 7. *1. (**static correctness**) Let t be an IS-term. Then $\mathcal{R}(\mathcal{T}(t)) = t$;*
*2. (**soundness**) let R be a pure net such that $t = \mathcal{R}(R)$ is an IS-term. If $R \to R'$ then $\mathcal{R}(R) \twoheadrightarrow \mathcal{R}(R')$ (in one or zero steps).*

Proof. 1. Trivial. 2. Boring check on the pure net rewriting rules. Note in particular that all control rules do not affect the readback. □

Note that the previous proposition does not imply the "correctness" of the implementation. In particular, every subsystem (even the empty one) of the rewriting system for pure nets would satisfy Proposition 7.(1). We should also prove that our rewriting system is powerful enough to correctly pursue β-reduction.

Theorem 8. *If R is a pure net yielded by a derivation starting at $\mathcal{T}(t)$ then:*

(A) there exists a IS-term t' such that $\mathcal{R}(R) = t'$ and $t \twoheadrightarrow t'$;

(B) if $t' \rightarrow t''$, then there exist a pure net Q such that $R \longrightarrow\!\!\!\!\rightarrow Q$, and $\mathcal{R}(Q) = t''$
(C) if R is in normal form, than also $\mathcal{R}(R)$ is.

Proof. (*Hint*)
(A) this follows from the Proposition 7.(2);
(B) Normalizing R w.r.t. control rules and let R' be the resulting net. Note that $\mathcal{R}(R) = \mathcal{R}(R') = t'$. Now, it is not difficult to prove that every redex in t must derive, via the readback, from a proper cut in the pure net. By firing this cut, and the soundness of proper rules we get the result.
(C) If R is in normal form, it is cut-free. So we have no way to "read back" a redex. □

In [4] we proved a similar result in the case of the local implementation of "boxes". The proof, there, is much more difficult, due to the great complexity of the readback procedure caused by the presence of control operators.

7 Conclusions

Interaction Systems have been mainly introduced with the aim of generalizing Lamping's optimal graph reduction technique to a supersystem of λ-calculus [2, 4, 3, 13]. In doing this, we heavily relied on the *logical nature* of our Systems that has been discussed in this paper. In particular, the encoding of Interaction Systems into pure nets described in Section 5 provide the intermediate step to the optimal implementation in [3], by adopting the local implementation of boxes of Linear Logic defined in [9].

As a preliminary step to the optimal implementation, we have also generalized Lévy's *theory* of optimal reduction from λ-calculus to IS's (labeling, extraction process, and so on) [2, 13, 5]. These works, that revealed several, somewhat unexpected problems, have provided the first significative progress about the family relation into the realm of Higher Order Rewriting, since Lévy's PhD-thesis in 1978 [14].

Despite of their original motivations, we believe that Interaction Systems have an interest in their own: they tight together Interaction Nets and CRS's, fit well with the Proofs as Proposition analogy, have interesting applications to the theory of data types, and are conceivable of interesting extensions, especially in the typing direction.

References

1. P. Aczel. A general Church-Rosser theorem. Draft, Manchester, 1978.
2. A. Asperti and C. Laneve. Interaction Systems I: the theory of optimal reductions. Technical Report 1748, INRIA-Rocquencourt, September 1992. A revised version will appear on the journal *Mathematical Structures in Computer Science*.
3. A. Asperti and C. Laneve. Interaction Systems II: the practice of optimal reductions. Technical Report UBLCS-93-12, Laboratory for Computer Science, Universitá di Bologna, May 1993. Also Research Report n. 2001, INRIA-Sophia Antipolis.

4. A. Asperti and C. Laneve. Optimal Reductions in Interaction Systems. In *TapSoft '93*, volume 668 of *Lecture Notes in Computer Science*, pages 485 – 500. Springer-Verlag, 1993.
5. A. Asperti and C. Laneve. The family relation in Interaction Systems. to appear on Proc. of Theoretical Aspects of Computer Science 1994, Senday, Japan, 1994.
6. V. Danos. *La Logique Linéare appliquée à l'étude de divers processus de normalisation (principalement du λ-calcul)*. PhD thesis, Université Paris VII, 1990.
7. J. Y. Girard. Linear Logic. *Theoretical Computer Science*, 50, 1986.
8. J. Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1989.
9. G. Gonthier, M. Abadi, and J.J. Lévy. Linear logic without boxes. In *Proceedings 7^{th} Annual Symposium on Logic in Computer Science*, 1992.
10. J. W. Klop. *Combinatory Reduction System*. PhD thesis, Mathematisch Centrum, Amsterdam, 1980.
11. Y. Lafont. Interaction Nets. In *Proceedings 17^{th} ACM Symposium on Principles of Programmining Languages*, pages 95 – 108, 1990.
12. J. Lamping. An algorithm for optimal lambda calculus reductions. In *Proceedings 17^{th} ACM Symposium on Principles of Programmining Languages*, pages 16 – 30, 1990.
13. C. Laneve. *Optimality and Concurrency in Interaction Systems*. PhD thesis, Dip. Informatica, Università di Pisa, March 1993. Technical Report TD – 8/93.
14. J.J. Lévy. *Réductions correctes et optimales dans le lambda calcul*. PhD thesis, Université Paris VII, 1978.
15. L. Regnier. *Lambda Calcul et Réseaux*. PhD thesis, Université Paris VII, 1992.