Towards Generic Inductive Constructions in Systems of Nets

Stéphane Gimenez

Institute of Computer Science
University of Innsbruck, Austria
stephane.gimenez@uibk.ac.at

Abstract

As an alternative to recursion [2] or cycles [6], which grant Turing-completeness, we sketch a system of nets that is sufficiently expressive to manipulate complex inductive and co-inductive objects, but enforces termination at the same time. We aim at a logically sound framework, close to prior fixed-point logics [1], that can be computationally implemented as a graph-based rewriting system in the style of proof nets. A complete and fully satisfactory formalization might require a refined approach, based on systems of interaction nets that are enriched with a deep-inference typing system as presented in [3].

1998 ACM Subject Classification F.1.2, F.4.1

Keywords and phrases Interaction Nets, Proof Nets, Linear Logic, Fixed Points, Inductive and Co-Inductive Types.

1 Introduction

Inductive types include typically integers, list, trees, or more complex finite objects, possibly simply-typed λ-terms. Co-inductive types typically represent infinite data-structures like streams, however some infinite co-inductive terms also admit finite representations.

For instance, lists of elements of type A are either the empty list or a construct made of an element of type A and another list. An unfolded definition of their type could be written, using some standard functional programming syntax, as follows:

```
Nil | Cons of A * (Nil | Cons of A * (Nil | Cons of A * (...)))
```

With linear logic connectives it writes $1 \oplus A \otimes (1 \oplus A \otimes (1 \oplus A \otimes (\dots)))$, which can also be represented finitely with a μ abstraction: $\mu\alpha.\, 1 \oplus A \otimes \alpha$.

Using instead a ν abstraction, the dual connector, type $\nu\alpha.\, 1 \oplus A \otimes \alpha$ denotes streams, i. e. possibly infinite sequences of elements of A. In particular, streams of natural numbers, for instance, include recurrent sequences such as $[1, 2, 1, 2, 1, 2, \dots[$, or slightly more complex sequences like $[1, 1, 2, 1, 2, 3, 1, 2, 3, 4 \dots[$, which can be defined finitely in most programming languages.

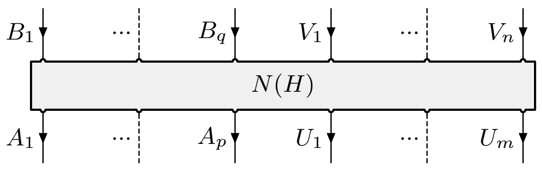
A few specific inductive types like integers, lists or trees are easily represented in systems of nets like interaction nets [5], but co-inductive types are not commonly found. We describe in this paper a generic method to implement any inductive or co-inductive type, from the multiplicative, additive and exponential constructions provided by linear logic. We will moreover show that the exponential constructions are not strictly required.

2 Unfolding Boxes

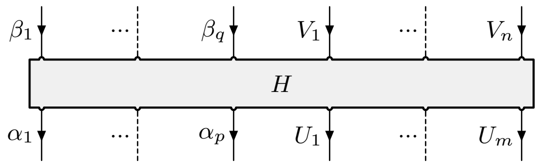
Given the usual system of proof nets for MELL [4], we consider sets $\mathcal{N}_\Gamma\,[P_1 : \Gamma_1, \dots, P_k : \Gamma_k]$ of nets of interface Γ containing open emplacements (also called net variables) P_i of interface

Γ_i. These are called *open nets*. The usual principles behind net reduction are unchanged; open emplacements just do not actively take part in the interaction process.

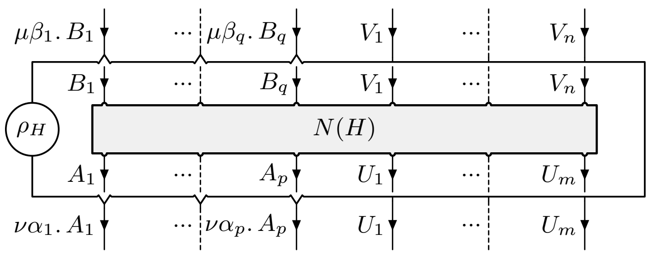
Unfolding boxes (a finite description of a co-inductive object). Let $N(H)$ be an open net with interface:



whose open emplacement H is typed with atomic type variables $\alpha_1, \ldots, \alpha_p$ and $\beta_1, \ldots, \beta_q$ as follows:



Such a net can be enclosed within an *unfolding box* as described hereafter. The open emplacement H and type variables α_i and β_i, which are expected to appear in their respective A_i or B_i and are assumed not to appear in any of the V_i or the U_i, are bound in the process.



From a logical viewpoint, the content of an unfolding box ρ_H is the body of an inductive proof in which the open emplacement H is used as induction hypothesis. Ports typed $\nu\alpha_i.\,A_i$ or $\mu\beta_i.\,B_i$ are principal ports, and ports typed V_i or U_i are auxiliary ports which do not interact immediately.

Moreover, two atomic variables among the α_i or the β_i can be chosen equal when their respective A_i or B_i (i. e. the types they mask) are equal.

Reduction of unfolding boxes Any logical operator which interacts through a principal port of an unfolding box will open, or "unfold", this box. Type-wise, type variables α_i and β_i are not made free, but are respectively substituted with type $\nu\alpha_i.\,A_i$ or type $\mu\beta_i.\,B_i$. Bounded open emplacements which materialize inductive calls are substituted with copies of the full original unfolding box.

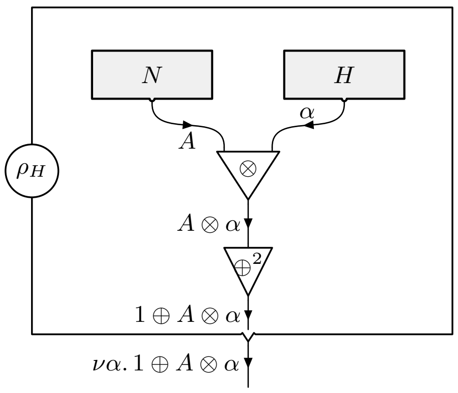
The typing system ensures that recursive calls are made deep inside the inductive structure of the interacting object and prevents non-termination.

Self-reduction Open emplacements may anytime be substituted with copies of the contents of the whole box which binds them. It optimizes later reductions of the net and could be triggered once after each normal unfolding step (again, to avoid non-termination).

3 Examples

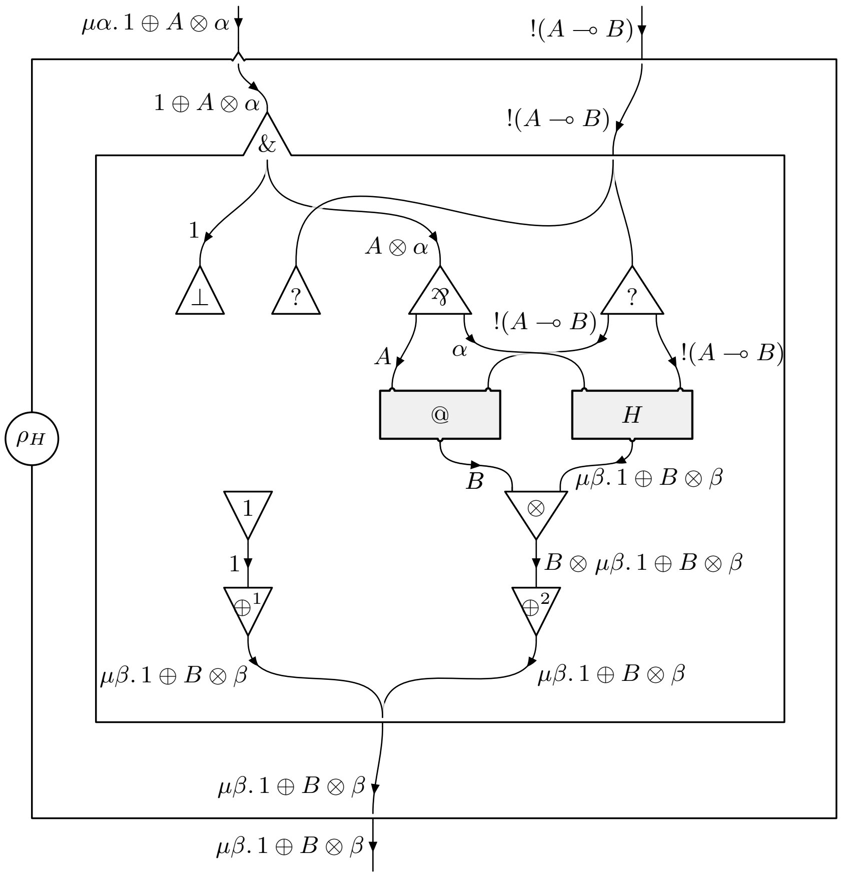
Our system allows one to write any operation on inductive or co-inductive data types, like usual integer addition, multiplication, exponentiation, etc. or list concatenation, mapping, folding, flattening, etc. or stream intermixing, mapping, etc.

First, we provide the very basic example of a net producing (on demand) an infinite stream $\nu\alpha. 1 \oplus A \otimes \alpha$ of a given element of type A produced by a net N.



N
H
A
α
ρ_H
$\otimes$
$A \otimes \alpha$
$\oplus^2$
$1 \oplus A \otimes \alpha$
$\nu\alpha. 1 \oplus A \otimes \alpha$

Follows another example, which takes as parameters a promoted function $!(A \multimap B)$, that maps elements of type A to elements of type B, and a list $\mu\alpha. 1 \oplus A \otimes \alpha$ with elements of the former type, and returns a list $\mu\beta. 1 \oplus B \otimes \beta$ with elements the latter type:



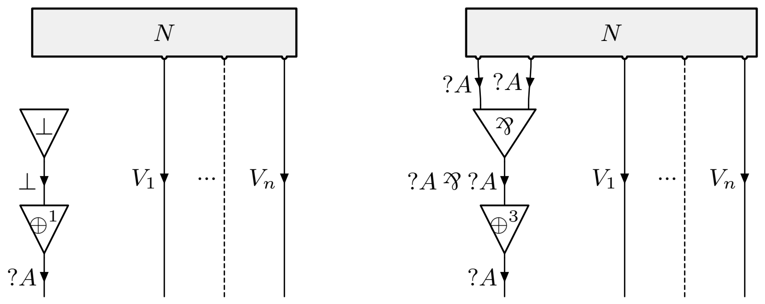
$\mu\alpha. 1 \oplus A \otimes \alpha$
$!(A \multimap B)$
$1 \oplus A \otimes \alpha$
$!(A \multimap B)$
$\&$
1
$A \otimes \alpha$
$\perp$
$?$
$⅋$
$?$
$!(A \multimap B)$
A
α
$!(A \multimap B)$
ρ_H
@
H
B
$\mu\beta. 1 \oplus B \otimes \beta$
1
$\otimes$
1
$B \otimes \mu\beta. 1 \oplus B \otimes \beta$
$\oplus^1$
$\oplus^2$
$\mu\beta. 1 \oplus B \otimes \beta$
$\mu\beta. 1 \oplus B \otimes \beta$
$\mu\beta. 1 \oplus B \otimes \beta$
$\mu\beta. 1 \oplus B \otimes \beta$

This unfolding box will unfold as soon as the top-level constructor of the $\mu\alpha.\,1 \oplus A \otimes \alpha$ list will interact through its principal port. This constructor will then be matched against the additive box. Only one of the two additive slices will be kept. If the input list is not empty, the first element is extracted and a copy of the function is applied to it by the net labeled @. In parallel, an inductive call is made to deal with the tail of the list. Resulting objects are combined to produce the output list. When the input list is eventually depleted, notice that no inductive call is made, since the filled H emplacement is garbage-collected with the entire right slice of the additive box.

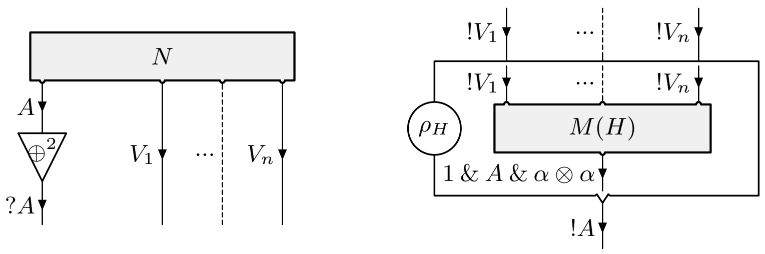
4 A Co-Inductively Defined Exponential

Exponential constructions can be encoded with the provided co-induction scheme, using type $!A := \nu\alpha.\,1 \,\&\, A \,\&\, \alpha \otimes \alpha$, whose relevance was already mentioned in [4]. The implementation of *weakening*, *dereliction*, *contraction* and *promotion* rules is provided below. Obviously, this definition of the exponential is not free. In particular, would we not restrict generation of $!A$ to those four constructions only, we could build non-uniform exponentials.

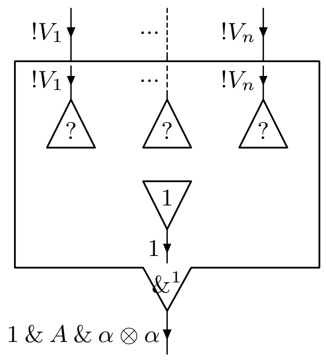
Rules *weakening* and *contraction* write:

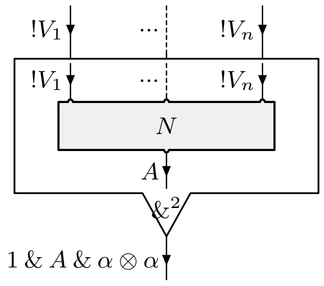


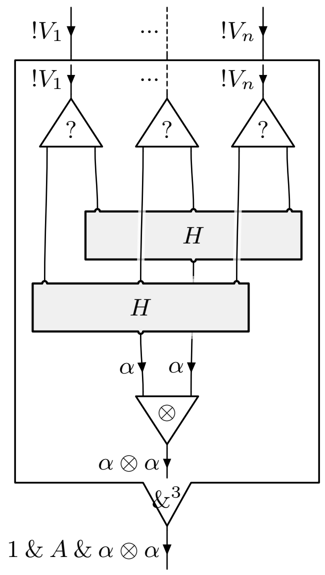
Rules *dereliction* and *promotion* of a net N write:



where $M(H)$ denotes the net obtained by superposing the following additive-box slices. Operators *weakening* and *contraction* used in those slices have already been defined.







Main reductions of exponential constructions are simulated by unfoldings followed by standard reduction steps. As an aside, additive commutation steps are expensive (except in the case of a fully-parallel reduction strategy) and are usually disabled. It is then harmless to allow interaction on upper ports of unfolding boxes that are typed with an exponential type, so that exponential commutations are made available as well.

5 Conclusion

We presented inductive types and sketched a generic method to handle them in the proof net formalism, as an extension of MELL. The exponential fragment can in fact automatically be obtained trough an encoding, although a native handling would certainly be more efficient.

We provided examples that make proper use of the unfolding boxes, but a correctness criterion was not discussed. Deep-inference typing systems such as the one presented in [3] might offer a good framework to properly structure an unfolding construction. In forthcoming works, we aim to extend their reducibility-based strong normalization theorem to inductive and co-inductive types.

References

1 D. Baelde. Least and greatest fixed points in linear logic. *CoRR*, abs/0910.3383, 2009.
2 S. Gimenez. *Programmer, calculer et raisonner avec les réseaux de la logique linéaire*. PhD thesis, Université Paris Diderot, 2009.
3 S. Gimenez and G. Moser. The structure of interaction. In *Computer Science Logic (CSL '13)*, 2013.
4 J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
5 Y. Lafont. Interaction nets. *Principles of Programming Languages (POPL '90)*, pages 95–108, 1990.
6 R. Montelatici. Polarized proof nets with cycles and fixpoints semantics. In *Typed Lambda Calculi and Applications*, volume 2701 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2003.