Function-Constructor Nets and their Semantics

Marc Thatcher
University of Sussex
Falmer, U.K.
m.thatcher@sussex.ac.uk

1 Introduction

Interaction nets [9] are a form of graph rewriting, generalising the proof structures of multiplicative linear logic [7, 10]. Introduced as "a new kind of programming language", they have rather been primarily applied in the *optimal* [8] and *efficient* [12] reduction of λ-calculus terms. More recently, the availability of multicore processors has renewed interest in interaction nets as a model of asynchronous parallel computation with emerging industrial implementations [2,19] and active research languages [17, 21].

Under their standard operational semantics, interaction nets do not have notions of input or output, and the evaluation result, if any, is its normal form. This creates a number of difficulties in formalising their use as a programming language:

- Results may be *inaccessible*, in the sense that no observable interface exposes them (Figure 1).
- Results may be accessible but *deadlocked*, unable to propagate information (Figure 2).
- Divergent computations are uniformly treated as non-results, even when they exhibit observable, productive behaviour such as *streams* (Figure 3).

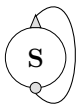

Figure 1: An inaccessible net.

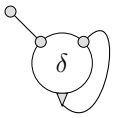

Figure 2: A deadlocked net.

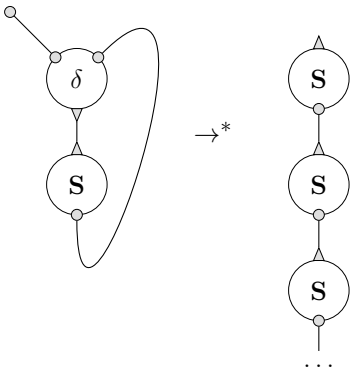

Figure 3: A stream reduction sequence.

We reconsider interaction nets from Lafont's original programming language perspective by adopting an observational view of semantics. Our contributions are:

- We refine the definition of interaction nets by distinguishing *function* symbols from *constructors* and distinguishing inputs from outputs.

- An operational semantics is given, which replaces normal forms as the result of evaluation with *observational output*, orthogonal to termination.
- We develop a denotational semantics for function-constructor nets based on constructor trees, analogous to Böhm trees.
- We sketch a proof of the full abstraction of this semantics.

2 Interaction nets

Interaction nets are finite, undirected labelled graphs. They may be cyclic and/or disconnected. Labels are drawn from a signature Σ, where each symbol $\alpha \in \Sigma$ is equipped with an arity $ar(\alpha) \in \mathbb{N}$. Each vertex (an *agent*) has a number of *ports* determined by its label: an agent labelled α has $ar(\alpha)$ *auxiliary* ports and one *principal* port. Wires connect ports pairwise, with at most one wire per port. Ports not connected to a wire are *free*.

Computation proceeds as for term graph rewriting [16] but without automatic garbage collection. A rewrite rule is a pair of nets, written $LHS \Rightarrow RHS$, where LHS consists of two agents connected via their principal ports (an *active pair*), and RHS is a net with the same set of free ports (*interface*) as LHS. In any such set of rules, each LHS must be unique. Figure 4 gives an example in which the active pair of the `:`-labelled (list constructor) agent and `hd`-labelled agent is defined to rewrite to a net in which the h(ead) and r(esult) ports are connected by a wire, and the t(ail) is explicitly discarded via an ϵ (erasing) agent, which ensures interface preservation.

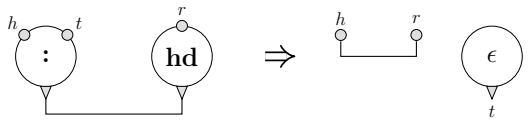

Figure 4: Example interaction rule.

A pair of a net and a set of rewrite rules is an *interaction net system*. As rules are *local* and *deterministic*, interaction net reduction sequences are *permutation equivalent*[1]—the order of reduction of active pairs does not affect the resulting net [9, Prop. 1]. This justifies *parallel reduction steps*: at each step, all active pairs are reduced simultaneously.

3 Function-constructor nets

To address the issues enumerated in Section 1, we refine nets by distinguishing *input* and *output* ports. Agents whose principal port is an input are *functions*; those with a principal port as an output are *constructors*. The signature Σ is therefore partitioned into disjoint sets of function symbols, $\mathcal{F}$, and constructor symbols, $\mathcal{C}$. Constructors are further restricted so that all auxiliary ports are inputs, reflecting the intended interpretation of constructors as atomic data elements: once consumed, nothing remains. Nets satisfying these constraints are *function-constructor nets*. Such nets' interfaces split into distinct input and output interfaces. We shall consider only nets with empty input interfaces, i.e. where all inputs are instantiated. We additionally require every agent port to be connected to a wire so each free port is at the end of a wire; this is important in our semantics.

Rewrite rules are redefined so that each active pair consists of one function and one constructor, and the right-hand side preserves both the input and output interfaces.

Despite these restrictions, the model retains interaction nets' Turing completeness: systems such as the SK-calculus [6, §5.1] and Lafont's interaction combinators [11] can be encoded as function-constructor nets.

[1] Called *strongly confluent* in [9, Prop. 1] and elsewhere in the literature. For a discussion of the choice of terminology, see [20, §2.5.1].

Prior work: Splitting the signature into functions—often called *destructors* in the literature—and constructors was first proposed in [9, §2] and occurs also in [1,4,5] but these do not require constructor outputs to be unique, nor are the implications for interfaces being split into input and output interfaces, and the necessity to restrict interaction rules to preserve them, considered.

3.1 Operational semantics

Operationally, evaluation of function-constructor nets proceeds exactly as for ordinary interaction nets; the only change is our notion of the evaluation's *result*. Rather than identifying this with a normal form, we define it as the *observational output*, $\mathsf{Obs}(N)$, of the net, given by:

- An erasing function, ϵ, is attached to each output port of the net.
- These, and their descendents, interact as per the rules shown (for arities up to 2) in Figure 5. (Note that the leftmost rule, erasing a nullary constructor, rewrites to the empty net.)

  As these erasure functions propagate through the net, each wire connection and erased constructor's label is recorded.
- The result at net N's port p at depth n is the finite constructor approximation $d_n^p(N)$ obtained by restricting erasing propagation to n steps.
- The recursively generated constructor structure at an output port is its *output tree*, obtained as the unique limit of this process, written $\mathsf{Obs}^p(N)$.
- The observational output, $\mathsf{Obs}(N)$, is the tuple of output trees over all output ports of net N.

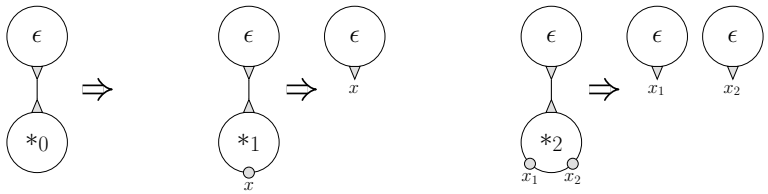

Figure 5: Interaction rules for ϵ.

Note that this definition does not require the net to reach a normal form. Net observational equivalence, $N_1 \approx N_2$, is defined as $\mathsf{Obs}(N_1) = \mathsf{Obs}(N_2)$.

We remark that (i) for nets without an output interface—such as the empty net—the observational output is the empty tuple, and (ii) if a port does not yield any observable constructor structure under any finite observation depth, its output tree is an unattached wire.

Prior work: The most similar prior work is the definition of *interface normal form* of [4]. This does not include explicit input and output ports and therefore identifies evaluation with normal form reduction.

3.2 Denotational semantics

Define the semantic domain of *constructor trees*, $\mathcal{T}_\perp$, coinductively as the greatest set generated by:

$$t ::= \perp \mid \wr(t) \mid \wr C(\wr(t_1), \ldots, \wr(t_n))$$

where $\perp$ represents a net without an output interface, $\wr(t)$ is a wire connected to tree t and $\wr(C(\wr(t_1), \ldots, \wr(t_n)))$ is a constructor agent $C \in \mathcal{C}$ with $t_1, \ldots, t_n$ inputs. Wires are closed under composition; we identify sequential wire connections with a single wire.

The interpretation of a function-constructor net N at an output port p_i, $[\![N]\!]_{p_i}$, is defined by the equations in Figure 6 applied left to right, yielding an element of $\mathcal{T}_\perp$. In the figure, RHS denotes the

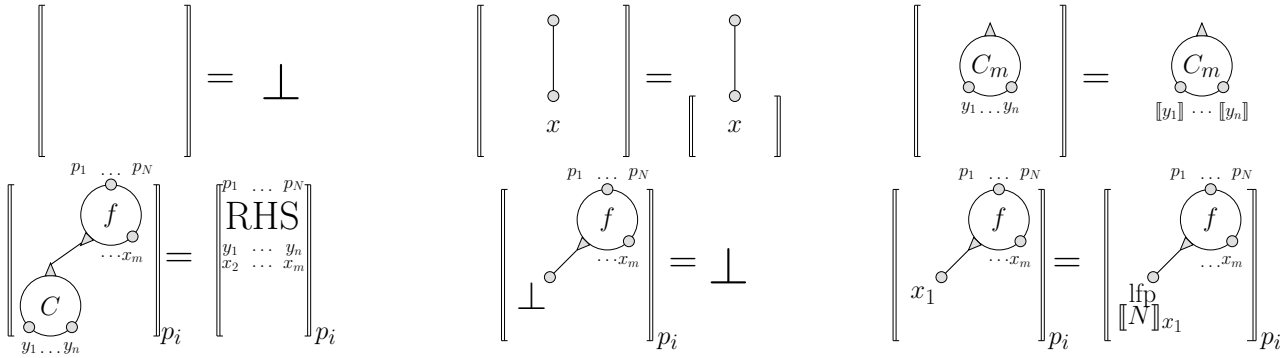

Figure 6: Interpretation function.

right-hand side of the interaction rule whose left-hand side consists of function f and constructor C, and $\mathrm{lfp}[\![N]\!]_x$ is the least fixed point of the interpretation of the net N at port x.

The existence of such a lfp is guaranteed by Kleene's Fixed Point Theorem [18]. Firstly, $(\mathcal{T}_\perp, \sqsubseteq)$ forms a dcpo, where $\sqsubseteq \subseteq \mathcal{T}_\perp \times \mathcal{T}_\perp$ is the least partial order satisfying: $\perp \sqsubseteq \mathfrak{f}(t)$ for $t \in \mathcal{T}_\perp$, $\mathfrak{f}(\perp) \sqsubseteq t$ for $t \in \mathcal{T}_\perp \setminus \{\perp\}$, and $C(\mathfrak{f}(t_1), \ldots, \mathfrak{f}(t_n)) \sqsubseteq C(\mathfrak{f}(u_1), \ldots, \mathfrak{f}(u_n))$ iff $t_i \sqsubseteq u_i$, $1 \leq i \leq n$, and directed suprema exist and are computed inductively over the constructor structure. Secondly, the interpretation function induces a functional F_N on $(\mathcal{T}_\perp, \sqsubseteq)$, which is Scott-continuous since each clause is monotone and depends only on a finite constructor prefix of its arguments.

For a net N with output ports $p_1, \ldots, p_n$, the *denotational output* is given by $[\![N]\!] = ([\![N]\!]_{p_1}, \ldots, [\![N]\!]_{p_n})$.

Prior work Mazza [13] interprets nets of symmetric interaction combinators as subsets of an algebraic domain, restricting consideration to deadlock free nets with normal forms. Perrinel [15] develops a context semantics for interaction nets, representing evaluation as token traversal through the net and proving path stability under reduction. This yields both a weight-based bound on reduction lengths and a sound, fully abstract denotational semantics, though the latter is only guaranteed for a subset of rules which do not connect auxiliary ports of an agent together.

3.3 Full abstraction

A semantics is *fully abstract* [14] if denotational equality and observational equivalence coincide. We sketch an approach to a proof of this.

To show that observational equivalence implies denotational equality, i.e. $N_1 \approx N_2 \implies [\![N_1]\!] = [\![N_2]\!]$, we first comment that observational equivalence means that for every output port p and depth n, the finite constructor approximations coincide: $d^p_n(N_1) = d^p_n(N_2)$. We can then show that $[\![N]\!]$ coincides with the least upper bound of the finite constructor approximations in $(\mathcal{T}_\perp, \sqsubseteq)$, and since limits in a dcpo are unique, it follows that $[\![N_1]\!] = [\![N_2]\!]$.

To show the reverse direction, note that replacing every occurrence of $\perp$ in $[\![N_1]\!]$ and $[\![N_2]\!]$ with the empty net yields $\mathsf{Obs}(N_1)$ and $\mathsf{Obs}(N_2)$. Therefore $N_1 \approx N_2$.

4 Conclusion

We have defined a Turing-complete refinement of interaction nets equipped with an observational notion of evaluation result. We have given operational and denotational semantics and sketched full abstraction. This aligns function-constructor nets with standard interpretations of programming languages.

This work forms part of a broader effort to develop practical programming languages based on interaction nets, providing a declarative model of parallel computation. Ongoing work includes defining a textual syntax for nets, a type system, and productivity [3] analysis.

References

[1] Andrea Asperti and Cosimo Laneve. Interaction Systems I: The theory of optimal reductions. *Mathematical Structures in Computer Science*, 4(4):457–504, December 1994.

[2] Higher Order Company. HigherOrderCo. `https://higherorderco.com/`, 2025. Accessed: 2025-01-08.

[3] Edsger W. Dijkstra. On the productivity of recursive definitions (EWD749), 1980. EWD794.

[4] Maribel Fernández and Ian Mackie. A Calculus for Interaction Nets. In *Lecture Notes in Computer Science*, pages 170–187. Springer Berlin Heidelberg, 1999.

[5] Maribel Fernández and Ian Mackie. Operational equivalence for interaction nets. *Theoretical Computer Science*, 297(1-3):157–181, March 2003.

[6] Simon Gay. Interaction Nets. Master's thesis, Cambridge, 1991. Diploma in Computer Science thesis.

[7] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[8] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The Geometry of Optimal Llambda Reduction. In *Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '92*. ACM Press, 1992.

[9] Yves Lafont. Interaction Nets. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '90*. ACM Press, 1990.

[10] Yves Lafont. *From proof nets to interaction nets*, page 225–248. London Mathematical Society Lecture Note Series. Cambridge University Press, 1995.

[11] Yves Lafont. Interaction Combinators. *Information and Computation*, 137(1):69–101, August 1997.

[12] Ian Mackie. YALE - Yet Another Lambda Evaluator based on interaction nets. *ACM SIGPLAN Notices*, 34(1):117–128, September 1998.

[13] Damiano Mazza. A denotational semantics for the symmetric interaction combinators. *Mathematical Structures in Computer Science*, 17(3):527–562, June 2007.

[14] Robin Milner. Processes: A mathematical model of computing agents. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 157–173. Elsevier, 1975.

[15] Matthieu Perrinel. On context semantics and interaction nets. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, New York, NY, USA, 2014. Association for Computing Machinery.

[16] Detlef Plump. Term Graph Rewriting. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, pages 3–61. World Scientific, Singapore, 1999.

[17] Shinya Sato. Inpla: Interaction nets as a programming language. `https://github.com/inpla/inpla`, 2025. Accessed: 2025-01-23.

[18] V. Stoltenberg-Hansen, I. Lindström, and E. R. Griffor. *Mathematical Theory of Domains*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1994.

[19] T6. The Vine Programming Language. `https://vine.dev/`, 2025. Accessed: 2025-01-23.

[20] Marc Thatcher. *Designing a High-Level Programming Language for Interaction Nets*. PhD thesis, University of Sussex, 2025. https://sussex.figshare.com/articles/thesis/Designing_a_high-level_programming_language_for_interaction_nets/32024301?file=63759111.

[21] Marc Thatcher. Implementation: Supporting materials for "Designing a High-Level Programming Language for Interaction Nets" PhD thesis. https://github.com/MarcThatcher/Implementation, 2025. Accessed: 2026-05-10.