A Tool for Programming with Interaction Nets

José Bacelar Almeida, Jorge Sousa Pinto, Miguel Vilaça
{jba,jsp,jmvilaca}@di.uminho.pt

Techn. Report DI-PURe-06.04.01
2006, April

PURe
Program Understanding and Re-engineering: Calculi and Applications
(Project POSI/ICHS/44304/2002)

Departamento de Informática da Universidade do Minho
Campus de Gualtar — Braga — Portugal

DI-PURe-06.04.01
A Tool for Programming with Interaction Nets by José Bacelar Almeida, Jorge Sousa Pinto, Miguel Vilaça

Abstract

This paper introduces INblobs, a visual editor and interpreter for interaction nets that is presently being developed at Minho. The editor is based on Blobs, a front-end for drawing and editing graph diagrams written with wxHaskell. The tool fills a gap in the community, since all the existing tools for interaction nets take as input textual descriptions of nets. INblobs includes a visual editor that allows users to edit both interaction nets and interaction rules; one then has a choice of reducing the net step by step within the tool, or else export a textual description to be given as input to other tools.

1 Introduction

Interaction Nets are a formalism based on a local and very restricted form of graph-rewriting, introduced by Lafont [2] as a generalization of proof-nets for multiplicative Linear Logic.

The formalism has become popular notably as an *implementation tool* for functional programming languages. For instance, *call-by-need* evaluation of functional programs can be implemented with a (slight variation on an) interaction net system [9]. More sophisticated strategies (that focus on minimizing the overall number of steps required by evaluation) are difficult to specify directly on the programs, and have only been discovered and implemented with interaction nets, see for instance [4].

The interaction nets formalism can also be used as a visual programming language in itself, as shown in the seminal paper by Lafont, but has been less explored in this context. One reason for this, is that although the formalism is based on graphs, there is suprisingly little work on graphical tools to support it.

The tool presented in this paper aims at filling this gap. It encompasses all the basic functionalities that can be expected from a tool for programming with the formalism:

- Visual, point-and-click editing of interaction nets and interaction net systems;
- Automatic conversion of nets and systems to textual notation;
- Interactive reduction of nets.

Related Work. Lippi [3] has produced an interpreter that reads a textual description of a net and displays a visual representation of it, and then animates its reduction (the user can interactively select the next active pair to be reduced). M. de Falco (private communication) is currently developing a tool that is similar to Lippi's but allows the user to edit nets visually.

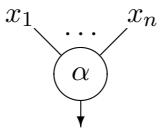
A number of other evaluators exist (see for instance [6]) that do not visualize nets at all: they take as input, and produce as output, textual representations. INblobs will be useful as a complementary editor for these tools.

Organization of the Paper. Section 2 reviews Interaction Nets and their Calculus. Section 3 briefly introduces Blobs, the front-end on which INblobs is based. In Section 4 an overview of INblobs is given; Sections 5, 6 and 7 then describe in detail specific functionalities of the tool – the

generation of textual descriptions of nets and systems, the editing of an interaction net system, and the implementation of reduction. Section 8 concludes the paper.

2 Interaction Nets

An interaction net system [2] is specified by giving a set Σ of symbols, and a set $\mathcal{R}$ 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*, and n *auxiliary ports* labelled $x_1, \ldots, x_n$. This is depicted in the following way:



A net built on Σ is a graph which has agents as nodes. The edges of the graph are connected to *ports* in the agents, such that there is at most one edge connected to every port in the net. Edges may be connected to two ports of the same agent. Principal ports of agents are depicted by some distiguishing sign, such as an arrow.

The ports of agents where there is no edge connected are called the *free ports* of the net. The *interface* of the net is the set of its free ports. There are two special instances of a net: a wiring (a net containing no agents, only edges between free ports), and the empty net (containing no agents and no edges).

Dynamics. An *active pair* is any pair of agents (α, β) in a net, with an edge connecting together their principal ports. An *interaction rule* $((\alpha, \beta) \Longrightarrow N) \in \mathcal{R}$ replaces an occurrence of the active pair (α, β) by the net N. Rules must satisfy two conditions: the interfaces of the left-hand side and right-hand side are equal (this implies that the free ports are preserved during reduction), and there is at most one rule for each pair of agents, so there is no ambiguity regarding which rule to apply.

Interaction nets are an instance of an abstract rewrite system, and as such their properties are studied using standard terminology from that field. If a net does not contain any active pairs then we say that it is in normal form. We use the notation $\Longrightarrow$ for one-step reduction and $\Longrightarrow^*$ for its transitive reflexive closure. Additionally, we write $N \Downarrow N'$ if there is a sequence of interaction steps $N \Longrightarrow^* N'$, such that N' is a net in

normal form. The strong constraints on the definition of interaction rules imply that reduction is strongly confluent (the one-step diamond property holds). Consequently, any normalizing interaction net is strongly normalizing.

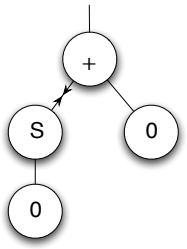


Fig. 1. Example net, representing $\mathsf{S}0 + 0$

An Example. As a very simple example of an interaction net system for natural number arithmetics, consider Σ containing $\{0, S, +\}$, with arity 0, 1, 2 respectively. Figure 1 shows an example of a net built from agents in this system; observe that the principal port of 0 needs not be marked, since it is the single port of this agent; in the remaining agents an arrow is used to single out the principal port.

The 0 and S agents are used as constructors, where the principal port corresponds to the constructed term and the auxiliary ports to the constructor arguments. The + agent on the other hand plays the role of a function, which implements addition by recursion on its first argument (connected to its principal port). The auxiliary ports correspond to the second argument and to the result of the operation, respectively.

We remark that it is the programmer's responsibility to make this system behave according to the intended interpretation – formally, all three agents have the same status and there is no distinction between constructors and functions. Figure 2 shows the two interaction rules that define the behaviour of the + agent.

As a second example of a net in this system, the equation $a = S(0) + (S(0) + b)$ can be represented by the net shown in Figure 3, where a, b correspond to the free ports. This net has two active pairs.

A Textual Representation and a Calculus for Interaction Nets. Interaction nets can be represented textually. Here we adopt a notation in the style introduced by Lafont [2], used in the Calculus for Interaction Nets [1].

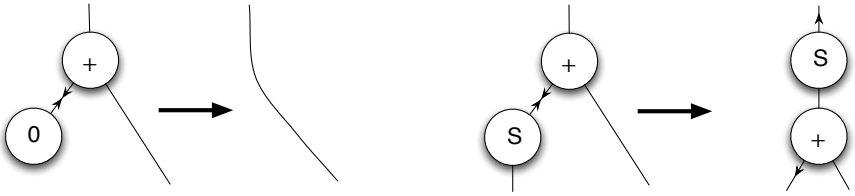


Fig. 2. Interaction rules for natural number arithmetics

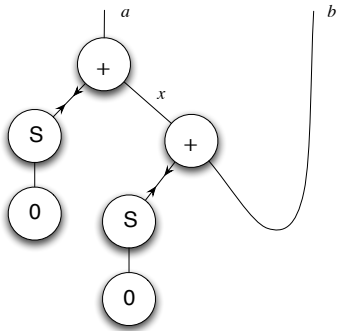


Fig. 3. Example net, representing the equation $a = S(0) + (S(0) + b)$

Nets can be written as sets of equations, involving algebraic terms built with symbols from Σ, used as constructors, and variables taken from a given set. Variables correspond to edges, but not every edge must be represented by a variable. In particular, variables corresponding to edges that connect the principal port of an agent to an auxiliary port of another agent can be dispensed with. An edge linking two auxiliary ports (two leaves) in two such terms (trees) is represented by two occurrences of the same variable. Variables are also allowed as members in equations, to allow for modular descriptions.

In addition to the set of equations, a set of terms must be given, corresponding to the interface of the net. To fully describe nets textually, it then suffices to fix a syntax for representing these two sets. A net will be written as a *configuration* of the form $\langle \mathbf{t} \mid \Delta \rangle$, with $\mathbf{t}$ a sequence of terms (its observable interface) and Δ a sequence representing a multiset of equations. Each variable occurs *exactly twice* in a net.

We give as example a representation of the net in Figure 3:

$$\langle a, b \mid S(0) = +(a, x),\ S(0) = +(x, b) \rangle$$

With respect to the rules of an interaction system, each one may be represented succinctly as a net with one active pair (and empty observable

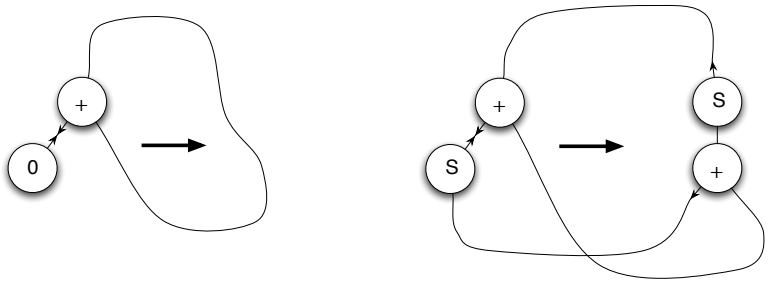


Fig. 4. Our example rules represented as nets: $(+(x,x),0)$ and $(+(S(y),x),S(+(y,x)))$

interface), by adding edges linking together each free port occurring in the left-hand side of the rule and the corresponding port in its right-hand side (see Figure 4). We write rules simply as pairs of terms. Observe that the non-ambiguity requirement for the selection of an interaction rule implies that any rule for reducing a pair of occurrences of the same agent has the form $(\alpha(t_1 \dots t_n), \alpha(t_1 \dots t_n))$ – it is symmetric.

Semantics. In [1] a calculus for interaction nets is proposed as an operational semantics for the formalism. We review here the untyped version of this calculus. Let $\rightleftharpoons$ be the obvious equivalence relation on sequences of equations, stating that the order of the members in equations is irrelevant, as well as the order of equations in a sequence, and $\mathcal{N}$ the function that returns the set of variables occurring in a term.

The calculus consists of the following conditional rewrite rules on configurations. In the Interaction rule, if a variable occurs simultaneously in a rule and in the net it must first be renamed.

Interaction: $(\alpha(t'_1, \dots, t'_n), \beta(u'_1, \dots, u'_m))$ is an interaction rule $\Rightarrow$

$\langle \mathbf{t} \mid \alpha(t_1, \dots, t_n) = \beta(u_1, \dots, u_m), \Gamma \rangle \longrightarrow \langle \mathbf{t} \mid t_1 = t'_1, \dots, t_n = t'_n, u_1 = u'_1, \dots, u_m = u'_m, \Gamma \rangle$

Indirection: $x \in \mathcal{N}(u) \;\Rightarrow\; \langle \mathbf{t} \mid x = t, u = v, \Gamma \rangle \longrightarrow \langle \mathbf{t} \mid u[t/x] = v, \Gamma \rangle$

Collect: $x \in \mathcal{N}(\mathbf{t}) \;\Rightarrow\; \langle \mathbf{t} \mid x = u, \Delta \rangle \longrightarrow \langle \mathbf{t}[u/x] \mid \Delta \rangle$

Multiset: $\Theta \rightleftharpoons^* \Theta', \langle \mathbf{t_1} \mid \Theta' \rangle \longrightarrow \langle \mathbf{t_2} \mid \Delta' \rangle, \Delta' \rightleftharpoons^* \Delta \;\Rightarrow\; \langle \mathbf{t_1} \mid \Theta \rangle \longrightarrow \langle \mathbf{t_2} \mid \Delta \rangle$

We remark that each variable occurs exactly once in the term (or list) where it is substituted, and no two applications of the indirection or collect rules may perform substitution of the same variable.

Nets in normal form correspond to pairs $\langle \mathbf{t} \mid \varepsilon \rangle$ or $\langle \mathbf{t} \mid \mathcal{C} \rangle$, with $\mathcal{C}$ a list of cycles, which are written in this notation as equations $x = t$, with $x \in \mathcal{N}(t)$.

3 Blobs

Blobs is a (visual) editor for directed graphs, implemented in Haskell using the wxHaskell library. Blobs was produced by a team integrating the authors of Dazzle [8], a Bayesian network editor, who realized that its front-end could be helpful to other completely different Haskell projects. This front-end has been extracted from Dazzle, and made available (with some added features) to the functional programming community.

Quoting the authors, "Blobs is a front-end for drawing and editing graph diagrams." The editor described in the present paper is built on top of Blobs and provides additional functionality to the front-end editor, concerning the editing of interaction nets (in particular the presence of ports in the nodes), and also back end functionalities – editing the interaction system, generating textual descriptions/configurations, and reducing interaction nets.

INblobs inherits all its editing capabilities from Blobs, in particular point-and-click placement of nodes and edges; the possibility of selecting a sub-graph with the mouse and performing actions on it; undo/redo actions; and saving/loading the current application state to/from a file.

Data Structures. INblobs also inherits (and adapts, see below) from Blobs the data structures used to represent graphs. Essentially, a graph is represented as a pair of *integer maps* (intmaps for short). An integer map is a mapping (a partial function) from natural numbers to some other type – the functional equivalent of indexed arrays. All nodes and edges in a graph are indexed by integers. The *node intmap* associates to each node the relevant information about it, in particular its label and shape. The *edge intmap* associates to each edge (the indexes of) its source and destinations nodes. The latter map effectively represents the structure of the graph.

4 The Tool

INblobs has been designed so that an interaction net system and an interaction net can be edited simultaneously. There are natural constraints that result directly from the formalism: the current net can only use

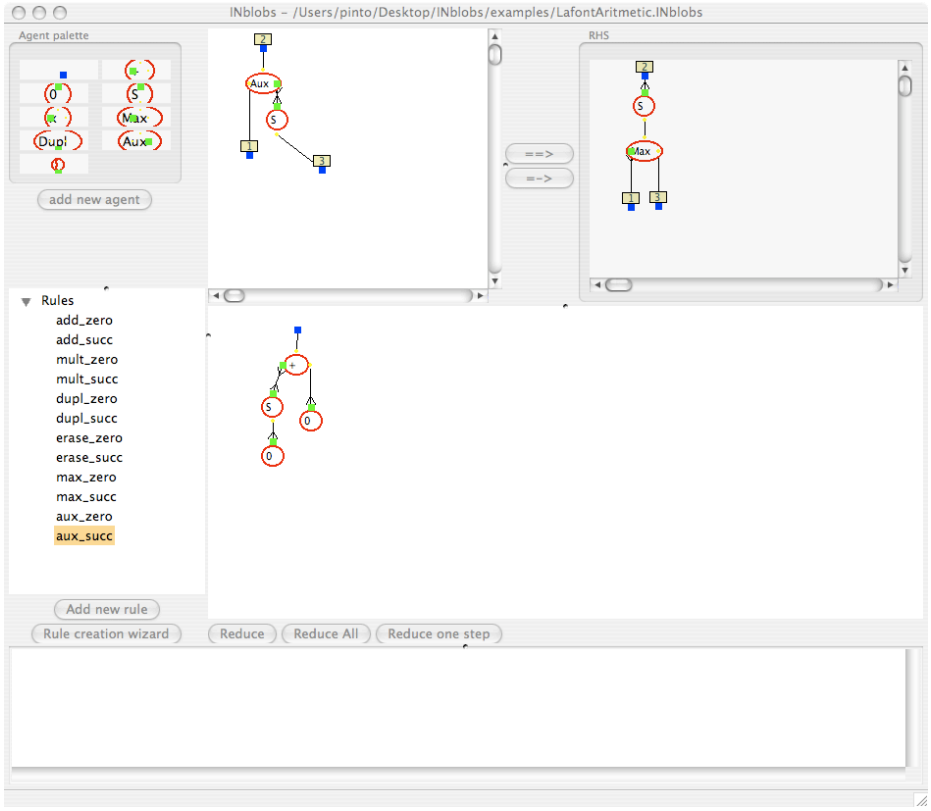


Fig. 5. Main tool window

agents that are defined in the current interaction system, and in order for an active pair to be reduced, a matching interaction rule must be included in the system.

The current state of the tool can be saved to be loaded later; this encompasses the interaction net system (symbols and rules), together with the current (possibly empty) interaction net.

Figure 5 shows the main window of the tool. On the left the symbol palette can be found, together with the list of interaction rules. The bottom area on the right shows the current interaction net, and on top the currently selected interaction rule is displayed.

Data Structures. INblobs inherits from Blobs the representation of graphs as pairs of intmaps, which are adapted to contain information specific to interaction nets. The *agent intmap* associates to each node, apart from the same information as in Blobs, information relative to its ports (both

symbolic, such as whether the port is the principal port or not, and geometric, such as the placement of the ports in the node). The *edge intmap* also associates to each edge some new information, namely the ports to which it is connected in each agent.

Basic Editing of Nets. Interaction nets can be edited in the bottom right pane of the main tool window, and in the top pane as part of an interaction rule.

Recall that the interface of a net consists of all its free ports, i.e. ports with no edge connected. The tool uses a slightly different (but equivalent) formulation of interaction nets:

- There is *exactly one* edge connected to *every* port of every agent;
- A special 0-ary *interface* symbol is introduced as part of every interaction net system; where one had a free port, one now has a port that is connected by an edge to an interface agent.
- The interface of a net consists of a set of interface agents.

This approach allows for a clear and explicit identification of the interface. We remark that the most common way of representing free ports visualy (see Figs. 1 and 3) cannot be used since there is no way to draw a dangling edge in Blobs (it also does not make much sense graph-theoretically).

Agents (including interface agents) are created from the current symbol palette (see Section 6 for information on creating new symbols); edges are then added using simple point-and-click operations. All the editing functionality is inherited from Blobs, except for the existence of ports in agents, which is not a standard notion for directed graphs.

Ports in the agents are depicted by small coloured shapes; *principal* ports are distinguished in two ways: a different shape and colour is used; and when an edge is connected to this port, an *arrow* appears in the edge, as in the standard representation used in all figures of this paper.

Adding an edge to a net implies successively *selecting* the source and destination ports of the edge. Nodes and edges may also be selected (a contextual menu shows editing operations for each such element of a net). Selected ports are shown in a different colour, and selected nodes and edges are shown thicker.

See the short User's Guide in Appendix A for more details.

5 From Visual Nets to Configurations

There is a one-to-many correspondence between Interaction Nets and configurations of the calculus for interaction nets. In particular, there is

a minimal number of equations to represent a particular net, which is given by the number of active pairs in the net, but other representations can be obtained by introducing additional variables; moreover configurations are considered modulo alfa-conversion (variable names can always be substituted by other fresh names).

The tool presented in this paper converts nets to configurations of the calculus using the following straightforward algorithm:

1. Label each *edge* of the net with a fresh name.
2. For each *agent* of arity n in the net, write an equation of the form $y = \alpha(x_1, \ldots, x_n)$, where y is the label of the edge connected to the principal port of α, and $x_1, \ldots, x_n$ are the labels of the edges connected to its auxiliary ports 1 to n. Let Δ be the multiset consisting of these equations.
3. Take the appropriate subset of names for the interface $\mathbf{t}$ of the net. One thus obtains an initial configuration $\langle \mathbf{t} \mid \Delta \rangle$.
4. While there exists in Δ an equation of the form $x = u$, with x a variable and $x \in \mathcal{N}(\Delta \backslash \{x = u\})$, apply the **Indirection** rule of the calculus to eliminate that equation.
5. While there exists in Δ an equation of the form $x = u$, with x a variable and $x \in \mathcal{N}(\mathbf{t})$, apply the **Collect** rule of the calculus to eliminate that equation.

The resulting configuration is minimal in the sense that it contains exactly one equation for each active pair in the net. Alternatively the tool can also output the configuration $\langle \mathbf{t} \mid \Delta \rangle$ obtained after step 3 of the algorithm.

The order in which the equations are generated (step 2) is given by the indexes of the agents, which in turn depend on the order in which the agents were placed in the net. Geometric operations do not modify the indexes or the ordering of the equations.

As an example, the net in Figure 3 would give rise, after step 3, to the configuration

$$\langle a, b \mid y = 0, z = S(y), z = +(a, x), u = 0, v = S(u), v = +(x, b) \rangle$$

which would then be simplified in steps 4 and 5 to

$$\langle a, b \mid S(0) = +(a, x),\ S(0) = +(x, b) \rangle$$

A slightly modified version of the same algorithm is used to convert interaction rules to their textual description. In this case it is compulsory to use the simplified version obtained by successively applying the indirection rule, since rules are written as single equations.

```
agents
     Z       0;
     S       1;
     A       2;

rules
     A(x,x)    >< Z;
     A(x,S(y)) >< S(A(x,y));

net
     S(Z) = A(a,x);
     S(Z) = A(x,b);

interface
     a;
     b;

end
```

Table 1. Textual description generated by INblobs

The usefulness of the conversion of nets to textual configurations lies of course in the possibility of using the tool as a visual editor for interaction nets, which may then be exported in text files to other tools. The tool uses the concrete syntax described in [6]. This is very close to the configurations described above, but allows for the interaction net and interaction net system to be described in the same file. The syntax can easily be modified to match other concrete representations.

The tool generates textual descriptions directly in the editor window, or alternatively writes them to a file specified by the user. Both options are accessible from the "Operations" menu. A choice is offered of generating a joint description of the net and system, or else a description of the system, or just the net configuration. Table 1 shows a file generated by the tool, containing our example net configuration, together with the description of the interaction net system for addition of natural numbers. This illustrates the concrete syntax generated by INblobs.

6 Editing the Interaction Net System

Defining new symbols is straightforward. The user must give a name and a list of ports. This operation has effects at the level of the interaction system (corresponding to including in it a new symbol declaration), but also at the geometric level, since the coordinates of the ports in the corresponding agents must also be given. The representation of the agent is an oval object whose length extends to accommodate its name inside. The

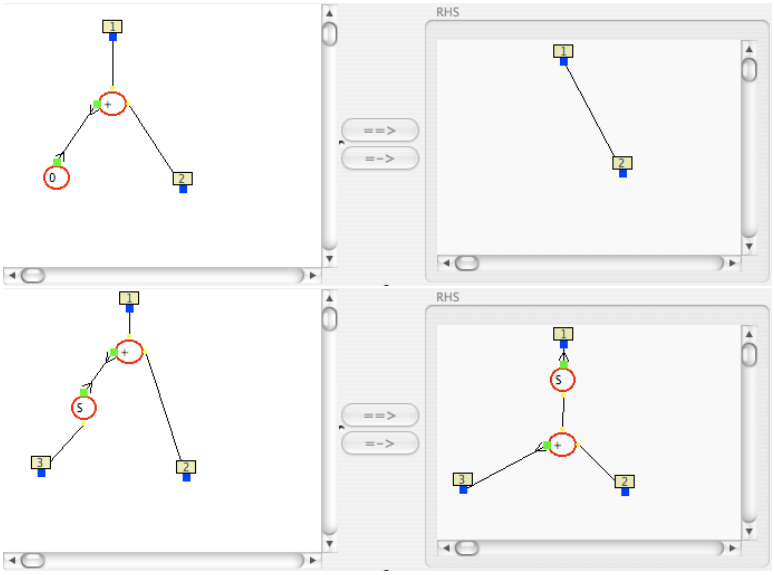


Fig. 6. Editing interaction rules in INblobs

user can also load a different shape palette, allowing for an alternative visual representation of agents (palettes can be designed and programmed with wxHaskell code).

A problem that arises when designing a visual editor for interaction rules is the identification of free ports in the left and right-hand sides. One solution would be to make the user draw rules as closed nets with one active pair, as in Figure 4. An alternative solution is used in INblobs, which we find much better from the point of view of usability. Since the interfaces of the nets in both sides of each equation are represented by interface agents (see Section 4), it suffices to associate indexes to these agents when they occur in rules. The geometric placement of the agents is thus irrelevant.

A rule is well-formed if the same interface agents appear in the left and right-hand sides of each rule, and moreover there are no free ports in agents (interface ports must be explicitly connected to interface agents).

The tool provides two ways of creating new interaction rules (both availabe as buttons in the main tool window). The first is to create a blank rule and manually edit the left and right-hand sides. The second is by using a wizard that allows the user to select a pair of agents from the agent palette, and automatically creates the left-hand side of the rule. In either case the user has a choice of copying the interface agents to the

right-hand side, or else copy the entire left-hand side net (this can be useful since the agents in the LHS are often reused in the RHS). Interface agents can of course also be added manually, in which case the user must explicitly match them in both sides.

Once a rule has been created it can be viewed or edited at any time by simply selecting it in the rule list. Example images of the editor window showing the rules of Figure 2 are shown in Figure 6.

The policy regarding the consistency of the interaction net system is as liberal as possible. The tool does not force the existence of a rule for every pair of symbols, and there may be more than one rule for the same pair; moreover the system may contain rules that are not well-formed. All these situations give rise to run-time errors during reduction. The rationale behind this choice is that the editing process may naturally contain inconsistent states that will later be eliminated by the user. Error-checking is thus "lazy": it is performed as needed during reduction.

7 Reducing Interaction Nets

The most obvious way of implementing interaction net reduction is to keep a list of active pairs. Each reduction step corresponds to picking a pair from this list and searching the list of interaction rules for a matching rule. Alternatively, one can keep a list of equations (as in the calculus for interaction nets), which allows for a closer control on the bureaucratic "rewiring" operations. In any case, the appropriate data structure for representing a net is a list of pairs of trees, where each tree corresponds to a term in the calculus. See [5] for details.

In INblobs, the current interaction net can be reduced within the tool. We remark that for the sake of flexibility, it is possible to alternate reduction steps with editing steps (on both the interaction net system and the net being reduced itself). A major consequence of this fact is that reduction must work on the underlying representation of the net used by the visual editor (see Section 4).

We outline the steps involved in the reduction of an active pair. We assume the active pair has been selected either by the user or randomly by the tool; it is identified by an integer index n, corresponding to the edge that connects the pair of agents.

1. Get from the edge intmap (with argument n) the information concerning the agents x, y where the edge n is connected, and also the ports of x, y where it is connected.

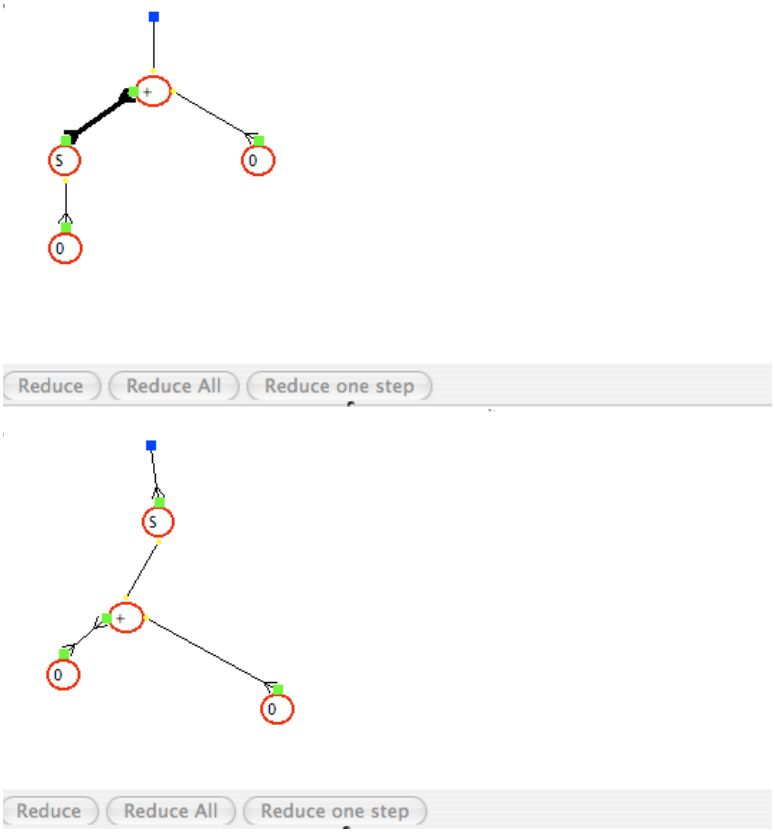


Fig. 7. A reduction step in INblobs

2. Consult the node intmap (with arguments x and y) to check that the two ports are principal; get the corresponding symbols from the agents.
3. Search the list of interaction rules for the rule matching the two relevant symbols. Recall that each side of the rule is represented by a pair of intmaps for its nodes and edges.
4. Integrate in the current net a copy R of the right-hand side of the rule. This is done by adding to the two intmaps the information of the intmaps of R, after updating the indexes in R (all indexes should be fresh with respect to the current net). This copy is for now disconnected from the net.
5. Replace the active pair by R: for each edge that was connected to the active pair in the net (except n), connect it instead to the corresponding port in R. This is a series of operations on the intmaps,

which involves matching the interface agents on the left- and right-hand sides of the rule.
6. Clean up: remove from the intmaps the information concerning the active pair just reduced, the interface agents in R, and the edges connected to them.

Three modes of reduction are available as buttons in the main window. The user can either reduce the currently selected active pair or else reduce a randomly selected active pair; the third possibility is to loop the latter step until a normal form net is reached (this cannot be interrupted, so the user is responsible for ensuring termination of the net reductions).

Figure 7 shows a net before and after a reduction step. The screenshots in Figure 8 show some steps in the reduction of a net representing the sorted insertion of a number in a list (many steps are omitted).

Net Layout. After reduction the tool does not attempt to draw the net in an intelligent way. Each reduction step results in a net where nodes may be superposed and edges may cross. The user is completely responsible for "tidying up" the output after each reduction step (or a series of steps). Since interaction steps are local, the amount of work involved is minimal for each step. It seems to us that this option is appropriate since the initial net is also drawn by the user, and usually the layout corresponds to some application-dependent interpretation of the net, which only the user is able to restore.

This stands in opposition to the tool reported in [3], where the standard representation (nets as lists of pairs of trees) is used at the visual level, and cannot be modified interactively. This is not always convenient since the "meaning" of a net usually suggests a layout that does not match the standard one.

8 Conclusions and Future Work

The current version of the tool is available for download from

`http://haskell.di.uminho.pt/jmvilaca/INblobs/`

On the technology side, Blobs, and consequently INblobs, is multi-platform to the extent that wxHaskell is. Unfortunately we have found that many graphical objects (such as labelled borders) work perfectly in MS Windows and less so in the Linux and Mac OSX platforms. INblobs runs with limitations on the latter platforms; it must be installed from sources and

requires the previous installation of a few libraries (see details in webpage).

A precompiled file is available for MS Windows, which is the preferred platform for running the tool (full functionality is only available in this version). Although the limitations on the other platforms seem to come directly from the wx toolkit, it is our concern to try to eliminate them as much as possible. We are also working on providing a stand-alone application file for Mac OSX, and an rpm installation for compatible Linux systems.

It would be interesting to incorporate other features in the tool, including:

- The possibility to have multiple active nets. Nets would be selected from a list, in the same way as rules are.
- Predefined reduction strategies (i.e. oracles for selecting the next active pair to be reduced), such as *weak reduction to interface normal form* [1, 7].
- An automatic layout algorithm. This would allow to read net configurations from text files and display them automatically (a possibility is to use the "natural" layout as in [3].
- A layout algorithm for interaction steps. The goal here is to try to minimize the "damage" inflicted by the reduction steps (in terms of agent superpositions and edge crossings) on the visual presentation of the net.
- A mechanism for checking interaction net systems (detecting for instance malformed rules, missing rules, and conflicting pairs of rules).

The editor is being developed as part of a bigger effort, to produce a complete tool for programming with Interaction Nets. In this context, it would also be interesting to include structuring facilities, external to the formalism.

For instance, there could be a macro mechanism, in the form of special "box" nodes. This would consist in a list of box definitions; box nodes could be used inside nets or other box definitions, and they could be expanded at any time according to their definitions, by pushing a button.

References

1. Maribel Fernández and Ian Mackie. A calculus for interaction nets. In G. Nadathur, editor, *Proceedings of the International Conference on Principles and Practice of Declarative Programming (PPDP'99)*, number 1702 in Lecture Notes in Computer Science, pages 170–187. Springer-Verlag, September 1999.

2. Yves Lafont. Interaction nets. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 95–108. ACM Press, January 1990.
3. S. Lippi. in^2 : A Graphical Interpreter for Interaction Nets. In S. Tison, editor, *Proceedings of the 13th International Conference on Rewriting Techniques and Applications (RTA'02)*, number 2378 in Lecture Notes in Computer Science, pages 380–386. Springer-Verlag, 2002.
4. Ian Mackie. Efficient λ-evaluation with interaction nets. In Vincent van Oostrom, editor, *Proceedings of Rewriting Techniques and Applications: 15th International Conference (RTA 2004)*, volume 3091 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.
5. Jorge Sousa Pinto. Sequential and Concurrent Abstract Machines for Interaction Nets. In Jerzy Tiuryn, editor, *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS)*, number 1784 in Lecture Notes in Computer Science, pages 267–282. Springer-Verlag, 2000.
6. Jorge Sousa Pinto. Parallel Evaluation of Interaction Nets with MPINE. In Aart Middeldorp, editor, *Proceedings of Rewriting Techniques and Applications (RTA)*, number 2051 in Lecture Notes in Computer Science, pages 353–356. Springer-Verlag, 2001.
7. Jorge Sousa Pinto. Weak Reduction and Garbage Collection in Interaction Nets. In B. Gramlich and S. Lucas, editors, *Final Proceedings of the 3rd Int'l Workshop on Reduction Strategies in Rewriting and Programming*, volume 86 of *Electronic Notes in Theoretical Computer Science*, 2003.
8. M.M. Schrage, A. van IJzendoorn, and L.C. van der Gaag. Haskell Ready to Dazzle the Real World. In *Proceedings of the 2005 ACM SIGPLAN workshop on Haskell (Haskell '05)*. ACM Press, 2005.
9. François-Régis Sinot. Token-passing Nets: Call-by-need for Free. In *Proceedings of the 1st. International Workshop on Developments in Computational Models (DCM'05)*, 2005. To appear in ENTCS.

A Short User's Guide

- Select an agent symbol by pressing its button on the left panel (symbol palette).
- Right click (or ctrl-click) on a canvas, node, or edge for a context menu.
- To create a node, first select its symbol from the palette and then shift -click on some blank canvas.
- To create an edge, select (click) the source port, then shift-click the target port.
- To delete a node or edge, select it and press backspace, or else use the context menu.
- To rearrange the diagram, click and drag nodes to where you want them.
- To make an edge look tidier, add a control-point from its context menu, and drag the point to where you want it.
- You can add multiple items into the current selection by meta-clicking the extra nodes and control points. (Meta = Apple key or Alt key.) A multiple selection can be dragged just like a single selection.
- The interface of a net or a rule is explicitly defined by means of special interface agents.
- The net on the bottom is the net to be reduced or converted to a textual configuration.
- The two nets on the top are the left-hand side and right-hand side of the interaction rule currently selected (from the list of rules on the left).
- To add a new rule press button "Add new rule" or else do mouse right-click in Rules (the root of the tree of rules). Then add agents to the canvas on the top.
- Alternatively, use the rule creation wizard. Pressing this button will make a dialog appear where the two agents that will interact can be chosen. The wizard will automatically generate the left-hand side of the rule. It is also possible to choose what is generated in the right-hand side of the rule:
  - a copy of the left-hand side to be manually edited (useful for rules with similar sides);
  - the interface agents from the left-hand side;
  - nothing (generates a blank right-hand side, not recommended).
- Match interface agents in a rule by selecting the desired interface agent in the left-hand side and shift-clicking the corresponding interface

agent in the right-hand side. A box with the same number will appear in both agents.

– Edge and node labels can be made visible by selecting the appropriate command from the View menu. Edge labels are useful since they make the selection of edges easier: to select an edge just click on its label.

B More Screenshots

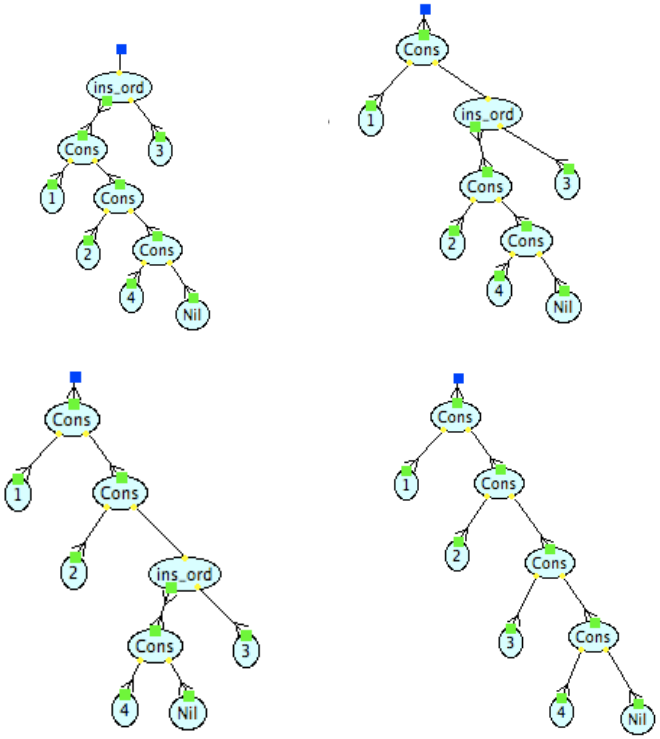


Fig. 8. Screenshots from a sorted insertion example