Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 219 (2008) 83–96

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Tool for Programming with Interaction Nets

José Bacelar Almeida,[1] Jorge Sousa Pinto[2] and Miguel Vilaça[3]

Informatics Department
University of Minho
Braga, Portugal

Abstract

This paper introduces INblobs, a visual tool developed at Minho for integrated development with Interaction Nets. Most of the existing tools take as input interaction nets and interaction rules represented in a textual format. INblobs is first of all a visual editor that allows users to edit interaction systems (both interaction nets and interaction rules) graphically, and to convert them to textual notation. This can then be used as input to other tools that implement reduction of nets. INblobs also allows the user to reduce nets within the tool, and includes a mechanism that automatically selects the next active pair to be reduced, following one of the given reduction strategies. The paper also describes other features of the tool, such as the creation of rules from pre-defined templates.

Keywords: Interaction Nets, editor/interpreter.

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 λ-calculus. Both standard strategies such as call-by-value or call-by-need [9], or more sophisticated and efficient strategies [4] can be implemented.

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 surprisingly little work on graphical tools to support it.

The tool presented in this paper aims at filling this gap. It encompasses the following functionality:

[1] Email:jba@di.uminho.pt

[2] Email:jsp@di.uminho.pt

[3] Email:jmvilaca@di.uminho.pt

doi:10.1016/j.entcs.2008.10.036

- Visual, point-and-click editing of Interaction Nets and interaction rules;
- Automatic conversion of nets and systems to textual notation;
- Interactive reduction of nets, using a choice of strategies;
- Editing the interaction with the aid of *wizards* and rule *templates*.

Related Work.

S. 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) developed a tool that is similar to Lippi's but allows the user to edit nets visually.

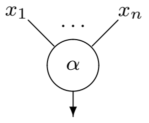
A number of other evaluators exist (see for instance [6]) that do not visualize nets at all: they take as input, and produce as output, textual representations. INblobs can also be used as a complementary editor for this class of tools.

Organization of the Paper.

Section 2 reviews Interaction Nets. In Section 3 an overview of INblobs is given; Sections 4, 5 and 6 then describe in detail specific functionalities of the tool. Section 7 briefly describes the front-end on which INblobs is based. 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 marked by 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 (the free ports are preserved by 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 abstract rewrite system, and their properties are described using standard terminology. If a net does not contain any active pairs it is said to be 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.

An Example.

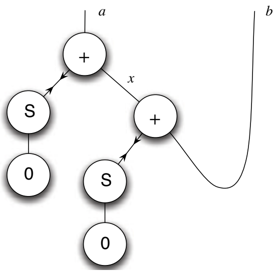


Fig. 1. Example net, representing the equation $a = S(0) + (S(0) + b)$

As a very simple example of an Interaction Net system, 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, that can be seen as representing the equation $a = S(0) + (S(0) + b)$. This net has two active pairs. Observe that the principal port of 0 needs not be marked, since it is the single port of this agent.

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). Auxiliary ports correspond to the second argument and to the result.

Formally, all three agents have the same status and there is no distinction between constructors and functions. It is the programmer's responsibility to make this system behave according to the intended interpretation. Figure 2 (top) shows the two interaction rules that define the behaviour of the $+$ agent.

A Textual Representation for Interaction Nets.

Interaction nets can be represented textually [2]. Here we adopt the notation used in the Calculus for Interaction Nets [1]. 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

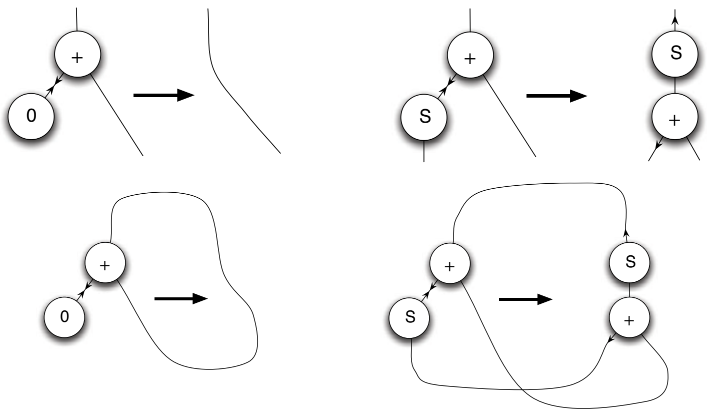


Fig. 2. Interaction rules for natural number arithmetics (top), and the same rules represented as the nets $(+(x,x),0)$ and $(+(S(y),x),S(+(y,x)))$ (bottom)

of equations in a sequence, and $\mathcal{N}$ the function that returns the set of variables that occur 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: $(\alpha(t'_1,\ldots,t'_n),\beta(u'_1,\ldots,u'_m))$ is an interaction rule $\Rightarrow$

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

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

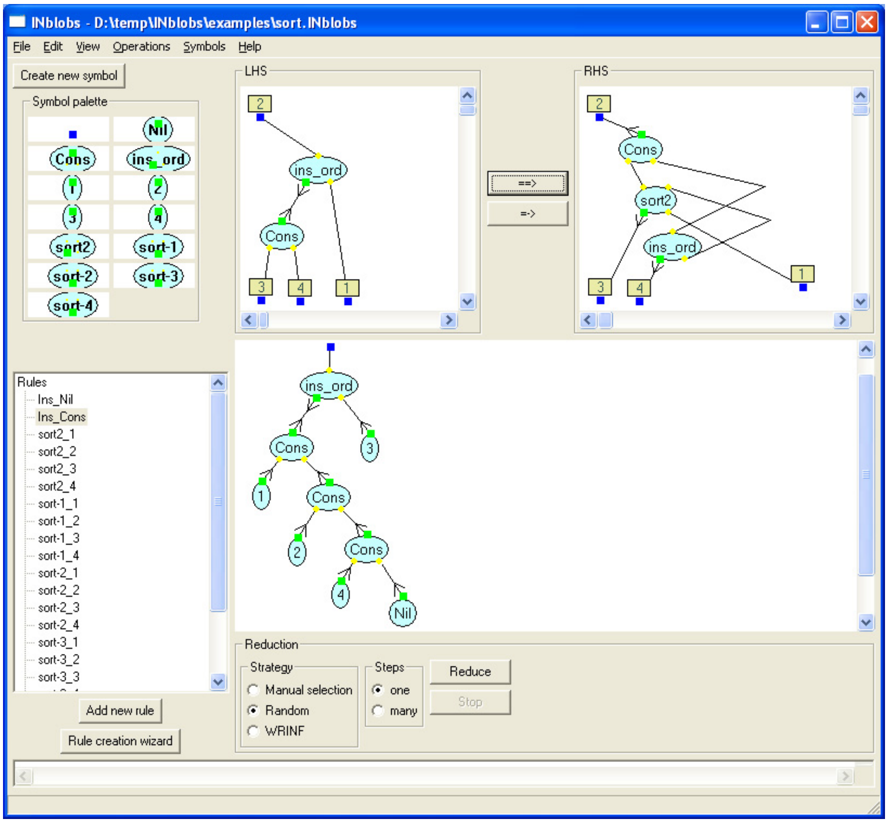
Multiset: $\Theta \rightleftharpoons^* \Theta', \langle \mathbf{t_1} \mid \Theta'\rangle \longrightarrow \langle \mathbf{t_2} \mid \Delta'\rangle, \Delta' \rightleftharpoons^* \Delta \;\Rightarrow\; \langle \mathbf{t_1} \mid \Theta\rangle \longrightarrow \langle \mathbf{t_2} \mid \Delta\rangle$

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 $x\in\mathcal{N}(t)$.

Nets can be written as multisets 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 for edges that connect the principal port of an agent to an auxiliary port of another agent, variables 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 multiset of equations, a multiset 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 multisets. 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 of equations. Each variable occurs *exactly twice* in a net.



Strategy: Manual selection [✓ Random] WRINF

Steps: [✓ one] many

Fig. 3. Main tool window

For instance one possible representation of the net in Figure 1 is the configuration $\langle a, b \mid S(0) = +(a, x),\ S(0) = +(x, b)\rangle$.

With respect to interaction rules, they may be represented succinctly as nets with one active pair and empty interface (written simply as a pair of terms) 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 2, bottom).

3 The Tool

INblobs has been designed so that an Interaction Net system and an Interaction Net can be edited simultaneously. There are natural editing constraints that result directly from the formalism: the current net can only use agents that are already defined in the 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 state comprises the Interaction Net system (symbols and rules), together with the current (possibly empty) Interaction Net.

Figure 3 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.

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.

Agents (including interface agents) are created from the symbol palette (see Section 5 for information on creating new symbols); edges are then added using simple point-and-click operations.

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 as can be seen in Figure 4.

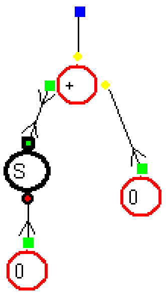


Fig. 4. Different ports and selection

Recall that the interface of a net is defined as the set of all its free ports, i.e. ports with no edge connected. The tool uses a slightly different (but equivalent) formulation of Interaction Nets: A special 0-ary *interface* symbol is introduced as part of every Interaction Net system, and the interface of a net consists of a set of interface agents. Where one had a free port, one now has a port connected by an edge to an interface agent. Thus there should be *exactly one* edge connected to *every* port of every agent.

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

4 From Visual Nets to Configurations

The tool converts nets to configurations of the calculus using the following straightforward algorithm:

(i) Label each *edge* of the net with a fresh name.

(ii) For each *agent* α of arity n in the net, write an equation of the form $y = \alpha(x_1, \dots, x_n)$, where y is the label of the edge connected to the principal port

```
agents                              net
     Z       0;                          S(Z) = A(a,x);
     S       1;                          S(Z) = A(x,b);
     A       2;                     interface
rules                                    a;
     A(x,x)     >< Z;                    b;
     A(S(y),x) >< S(A(y,x));        end
```

Table 1
Textual description generated by INblobs

of the agent, 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.

(iii) Let $\mathbf{t}$ be the multiset of labels associated with edges connected to interface agents of the net. One thus obtains an initial configuration $\langle \mathbf{t} \mid \Delta \rangle$.

(iv) 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 (see Section 2).

(v) 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 (iii) of the algorithm.

As an example, the net in Figure 1 would give rise, after step (iii), to

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

and simplified in steps 4 and 5 to $\langle a, b \mid S(0) = +(a, x),\ S(0) = +(x, b) \rangle$.

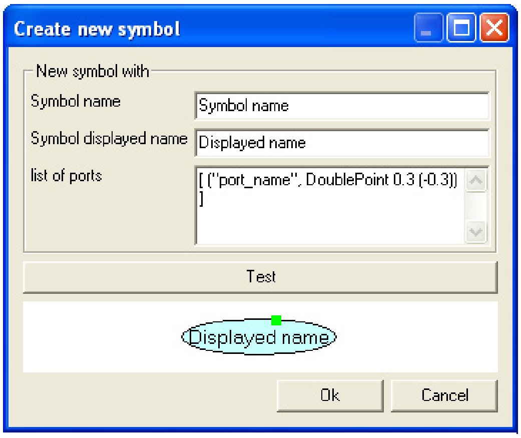
A slightly modified version of the same algorithm is used to convert interaction rules to their textual description. In this case it is compulsory to use the minimal version, since rules are written as single equations.

The usefulness of the conversion of nets to textual configurations lies of course in the possibility of using the tool as a visual Interaction Net editor, which may then be exported in text files to other tools. The tool uses the concrete syntaxs described in [6] and [10], but other concrete representations can be used.

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 just the system or the configuration. Table 1 shows a file generated by the tool, for our example net and system.

5 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



```
[ ("port_name", DoublePoint 0.3 (-0.3))
]
```

Fig. 5. **Create new symbol** wizard

given. The representation of the agent is an oval object whose length extends to accommodate its name inside. All this is done in **Create new symbol wizard** which is accessible from the main window through a button and looks like Figure 5.

The 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).

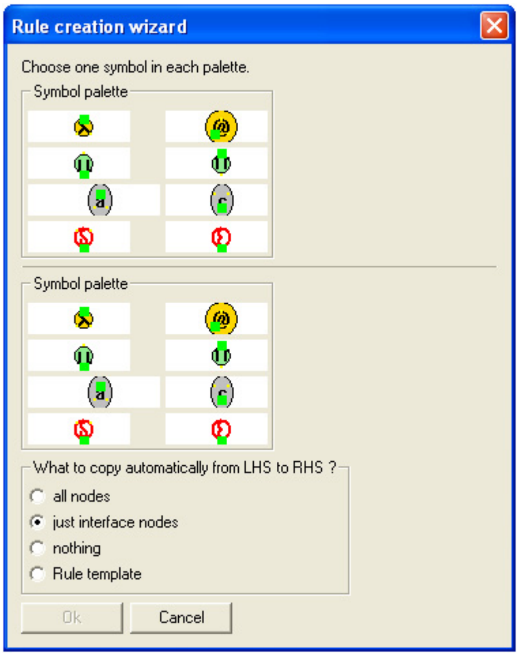
A problem that arises when designing a visual editor for interaction rules is the identification of free ports in the left and right-hand sides. One solution would be to make the user draw rules as closed nets with one active pair, as in Figure 2. An alternative solution is used in INblobs, which we 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 3), 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 sets of indexed interface agents are the same on the left and right-hand sides of each rule, and moreover there are no free ports (interface ports must be explicitly connected to interface agents).

The tool provides two ways of creating new interaction rules (both available 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 (see Figure 6) 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 often some of the agents in the LHS are 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.

Rule Templates.

A very useful feature of the rule creation wizard is the ability to define rules from pre-defined templates. Certain agents have fixed patterns of interaction; examples



Choose one symbol in each palette.

Symbol palette

Symbol palette

What to copy automatically from LHS to RHS ?

( ) all nodes
(●) just interface nodes
( ) nothing
( ) Rule template

Fig. 6. **Rule creation** wizard

of these include the eraser and duplicator agents. For instance, a step of interaction between a duplicator agent and any other agent α with n auxiliary ports generates two agents α and n duplicators. This pattern is encoded as a template that can be used in the wizard: it suffices to select α and the duplicator for the expected rule to be created, whatever the arity of α may be. Figure 7 shows a rule created by the referred template.

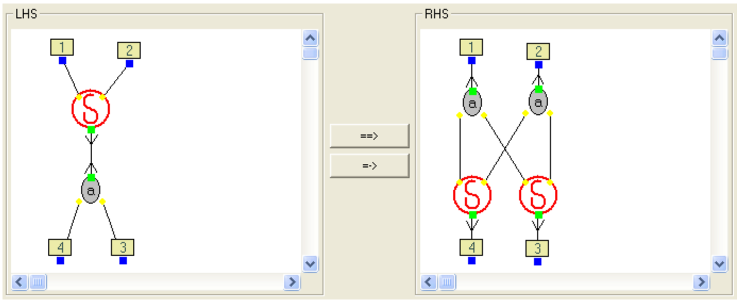


Fig. 7. Rule created by a Rule Template

For now templates are hardwired in the tool and cannot be edited by users.

Once a rule has been created it can be viewed or edited in the top pane of the application window, by simply selecting it in the rule list, as shown in Figure 3.

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. This is the appropriate choice since the focus of the tool is on interactive development of programs consisting of both an interaction system and Interaction Nets. However, the user always has the choice of, whenever wanted, calling check operations (via the "Checks" menu) that will detect most of the consistency violations mentioned above.

6 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.

INblobs is an integrated development environment: it is possible to alternate reduction steps with editing steps (on both the Interaction Net system and the net being reduced itself). This **alternate** is extremely useful for fast prototyping of new Interaction Net systems. To facilitate this, reduction works directly on the underlying representation of the net used by the visual editor (see Section 3).

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

(i) Get from the edge intmap (see Section 7) the information concerning the agents x, y where the edge n is connected, and also the ports of x, y where it is connected.

(ii) Consult the node intmap (with arguments x and y) to check that the two ports are principal; get the corresponding symbols from the agents.

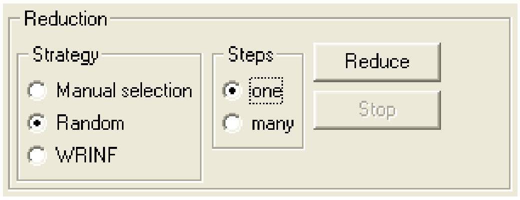
(iii) Search the list of interaction rules for the rule matching the two relevant symbols, represented by a pair of intmaps for its nodes and edges.

(iv) Integrate in the current net N a copy R of the right-hand side of the rule. This is done by adding to the two intmaps of N the information of the intmaps of R, after updating the indexes of the nodes and edges in R (all indexes should be fresh with respect to the current net). This copy is for now disconnected from the net.

(v) 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.

(vi) 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.

Several modes of reduction are available as buttons in the main window (see Figure 8). The user can choose to reduce the currently selected active pair; a random active pair; or else the active pair corresponding to one of the strategies implemented in the tool, in which case, those strategies will be listed in the Reduction area of the main window. For the moment strategies are hard coded in the tool but such coding is relatively easy and so new IN strategies, for example that archive canonical forms faster or keep the nets smaller, can be added to INblobs by request to its maintainers. The *weak reduction to interface normal form* strategy [1,7] is such a case. The user may alternatively reduce the net, accordingly to any of the available strategies, in many steps to normal form, having the possibility of stopping evaluation at any time.



Strategy: Manual selection [✓ Random] WRINF; Steps: [✓ one] many

Fig. 8. Reduction controls

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 a standard representation (nets as lists of pairs of trees) is used at the visual level, and cannot be modified interactively.

7 Implementation Details

Blobs is a (visual) editor for directed graphs, implemented in Haskell using the wx-Haskell 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 concerning the editing of Interaction Nets (in particular the presence of ports in the nodes), the creation of new symbols and interaction rules, rule templates, generation of textual descriptions (configurations), detection of inconsistencies and reduction of Interaction Nets.

INblobs inherits (with modifications) 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.

A graph is represented as a pair of *integer maps* (intmaps for short). An intmap is a mapping (a partial function) from natural numbers to some type – the functional equivalent of indexed arrays. In Blobs 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 destination nodes. The latter map effectively represents the structure of the graph.

The representation of graphs inherited from Blobs is 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 (such as whether the port is the principal port or not, and geometric information regarding the placement of the ports in the node). The *edge intmap* also associates to each edge new information, namely the ports to which it is connected in each agent.

8 Conclusions and Future Work

The current version of the tool is available for download from `http://haskell.di.uminho.pt/jmvilaca/INblobs/`. Blobs, and consequently INblobs, is multiplatform to the extent that wxHaskell is. A precompiled file is available for MS Windows, which is the preferred platform for running the tool. In Mac OSX installation is straightforward. On Linux INblobs must be installed from sources and requires a few libraries (see details on webpage).

Possible future developments include

- The possibility to have multiple active nets. Nets would be selected from a list, in the same way as rules are.
- 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.
- User-editable rule templates.
- Some form of modularity construction (external to the IN formalism) such as a macro mechanism, in the form of special expandable "box" nodes.
- A compilation mechanism for functional programs, that would read a program and encode it as an Interaction Net, using the existing translations.

Acknowledgement

We are thankful to Daniel Mendes for his contribution on the implementation of WRINF reduction strategy, templates and translation of λ-terms to nets.

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] Sylvain 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'04)*, 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'01)*, 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] Martijn M. Schrage and Arjan van IJzendoorn and Linda 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 volume 135 of *Electronic Notes in Theoretical Computer Science*, 2006.

[10] PIN Webpage. http://www.albeity.co.uk/download.html

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.