Extending the Interaction Nets Calculus by Generic Rules

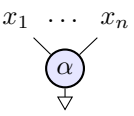
Eugen Jiresch *
jiresch@logic.at

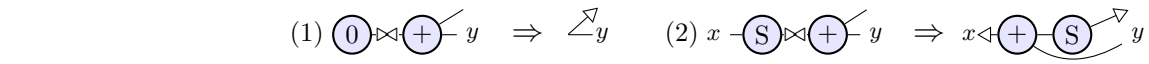
Institute for Computer Languages
Vienna University of Technology

Abstract. We extend the textual calculus for interaction nets by generic rules and propose constraints to preserve uniform confluence.

1 Introduction and Overview

Interaction nets are a model of computation based on graph rewriting. They enjoy several useful properties which makes them a promising candidate for a future functional programming language. *Interaction nets* were first introduced in [5]. A *net* is a graph consisting of *agents* (nodes) and *ports* (edges). Agent labels denote data or function symbols. Computation is modeled by rewriting the graph, which is based on *interaction rules*. These rules apply to two nodes which are connected by their *principal ports* (indicated by the arrows), forming an *active pair*. For example, the following rules model the addition of natural numbers (encoded by 0 and a successor function S):





This simple system allows for parallel evaluation of programs: If several rules are applicable at the same time, they can be applied in parallel without interfering with each other. In addition, reducible expressions (*active pairs*) cannot be duplicated: They are evaluated only once, which allows for sharing of computation.

Overall Goal and Current Contribution Our goal is to promote interaction nets to a practically usable programming language. Unfortunately, the beneficial properties of interaction nets impose strong restrictions on the shape of rules: This makes it hard to express features such as higher-order functions or side effects. In this abstract, we improve this deficiency by extending the textual calculus for interaction nets by *generic rules*. In addition, we define constraints on these rules to preserve uniform confluence, which is the basis for parallel evaluation. Using ideas of [1], we complement our previous work [4], which defined generic rules in the graphical setting of interaction nets.

* The author was supported by the Austrian Academy of Sciences (ÖAW) under grant no. 22932 and the Vienna PhD School of Informatics.

2 Generic Rules for the Lightweight Calculus

Ordinary interaction rules describe the reduction of a pair of two concrete agents (e.g., 0 and + in rule (1) above). *Generic rules* allow one concrete agent to interact with an arbitrary agent. This arbitrary, generic agent corresponds to a function variable, adding a higher-order character to interaction nets. Such rules have already been used in several publications (e.g., [6]), usually to model duplication and deletion of agents, albeit without a formal definition of generic rules. We present such a definition by extending the *lightweight calculus*, which is a textual representation of interaction nets [1].

The lightweight calculus provides a precise semantics for interaction nets. It handles application of rules as well as rewiring and connecting of ports and agents. It uses the following ingredients:

Symbols Σ representing agents, denoted by α, β, γ.
Names N representing ports, denoted by $x, y, z, x_1, y_1, z_1, \ldots$.
Terms T either a name or a symbol with a number of subterms, corresponding to the agent's arity: $t = x \mid \alpha(t_1, \ldots, t_n)$. $\bar{t}$ denotes a sequences of terms.
Equations E denoted by $t = s$ where t, s are terms, representing connections in a net. Δ, Θ denote multisets of equations.
Configurations C representing a net by $\langle \bar{t} \mid \Delta \rangle$. $\bar{t}$ is the interface of the net, i.e., its ports that are not connected. All names in a configuration occur at most twice. Names that occur twice are called *bound*.
Rules R denoted by $\alpha(\bar{t_1}) = \beta(\bar{t_2}) \to \Theta$. α, β is the active pair of the left-hand side (LHS) of the rule and Θ represents the right-hand side (RHS).

Rewriting a net is modeled by applying four *reduction rules* to a configuration:

Communication: $\langle \bar{t} \mid x = t, y = u, \Delta \rangle \overset{com}{\to} \langle \bar{t} \mid t = u, \Delta \rangle$
Substitution: $\langle \bar{t} \mid x = t, u = s, \Delta \rangle \overset{sub}{\to} \langle \bar{t} \mid u[t/x] = s, \Delta \rangle$, where u is not a name.
Collect $\langle \bar{t} \mid x = t, \Delta \rangle \overset{col}{\to} \langle \bar{t}[t/x] \mid \Delta \rangle$, where x occurs in $\bar{t}$.
Interaction $\langle \bar{t} \mid \alpha(\bar{t_1}) = \beta(\bar{t_2}), \Delta \rangle \overset{int}{\to} \langle \bar{t} \mid \Theta', \Delta \rangle$, where $\alpha(\bar{s_1}) = \beta(\bar{u_2}) \to \Theta \in$ R. Θ' denotes Θ where all bound names receive fresh names and $\bar{s}, \bar{u}$ are replaced by $\bar{t_1}, \bar{t_2}$.

$\overset{com}{\to}$ and $\overset{sub}{\to}$ replace names by terms: this explicitly resolves connections between agents which are generated by interaction rules. $\overset{col}{\to}$ also replaces names, but only for the interface. Naturally, $\overset{int}{\to}$ models the application of interaction rules: an equation corresponding to a LHS is replaced by the equations of the RHS.
We extend the calculus by generic rules as follows: We introduce additional symbols for generic agents and define generic rules. We then modify the $\overset{int}{\to}$ reduction rule to support generic agents.

Generic Names V representing generic agents, denoted by ϕ, ψ, ρ.
Generic Rules GR denoted by $\alpha(\bar{t_1}) = \phi(\bar{t_2}) \to \Theta$. Θ does not contain any generic names other than ϕ.

Generic Interaction $\langle\ \bar{t}\ \mid\ \alpha(\bar{t_1}) = \beta(\bar{t_2}), \Delta\rangle \overset{int}{\to} \langle\ \bar{t}\ \mid\ \Theta', \Delta\rangle$, where $\alpha(\bar{s_1}) = \beta(\bar{u_2}) \to \Theta\ \in \mathrm{R}$ or $\alpha(\bar{s_1}) = \phi(\bar{u_2}) \to \Theta\ \in \mathrm{GR}$ if β and ϕ have the same *arity* (number of ports). In the latter case, Θ' equals Θ where all occurrences of ϕ are replaced by β (in addition to using fresh names and replacing $\bar{s}, \bar{u}$).

The above definition gives a precise semantics for the application of generic rules. Note that this only covers generic rules where the generic agent has a *fixed* arity. Our approach can be extended to generic rules with agents of arbitrary arity.

Unfortunately, generic rules introduce *ambiguity* or *overlaps* to rule application: One equation could possibly be reduced by more than interaction rule. This generally destroys the nice properties of interaction nets (in particular, parallel evaluation). Therefore, we impose constraints on the application of generic rules:

Default Priority Constraint (DPC) An equation $\alpha(\bar{t_1}) = \beta(\bar{t_2})$ can only be reduced using a generic rule if no matching ordinary rule exists, i.e., $\alpha(\bar{s_1}) = \beta(\bar{u_2}) \to \Theta\ \notin \mathrm{R}$.

Generic Rule Constraint (GRC) If there is more than one generic rule that can be applied to a given equation $\alpha(\bar{t_1}) = \beta(\bar{t_2})$, there must exist an ordinary rule that can be applied aswell.

The DPC restricts the behavior of $\overset{int}{\to}$: ordinary rules always have priority over generic rules. The GRC restricts the set of generic rules GR. We can show that the combination of these constraints prevents overlaps: with the DPC, a generic rule corresponds to a set of ordinary rules without ambiguity. The GRC eliminiates a few obvious cases of rule overlaps.

Discussion In this abstract, we presented a glimpse of how to extend the interaction nets calculus by generic rules. Our previous work [4] did not consider this textual calculus. Due to space constraints, we did not discuss generic rules with arbitrary arity. Generic rules allow us to conveniently express higher-order functions: we used generic rules to model side effects via monads in interaction nets [3, 4]. In addition, we are involved in the implementation of an interaction nets based programming language [2], and already extended it by generic rules. We are currently investigating an implementation on parallel hardware (GPUs).

References

1. Hassan, A., Mackie, I., Sato, S.: A lightweight abstract machine for interaction nets. ECEASST 29 (2010)
2. The inets project. `http://www.interaction-nets.org`
3. Jiresch, E.: Realizing impure functions in interaction nets. `http://www.logic.at/people/jiresch/pub/jiresch_rifins.pdf` (2011), *ECEASST 38, to appear*
4. Jiresch, E., Gramlich, B.: Realizing monads in interaction nets via generic typed rules (2011), submitted to *Int. Conference of Functional Programming (ICFP'11)*
5. Lafont, Y.: Interaction nets. Proceedings, 17th ACM Symposium on Principles of Programming Languages (POPL'90) pp. 95–108 (1990)
6. Mackie, I.: YALE: yet another lambda evaluator based on interaction nets. Internation Conference on Functional Programming (ICFP'98) pp. 117–128 (1998)