Extensions for Interaction Nets

Eugen Jiresch and Bernhard Gramlich (Faculty Mentor)
Institute of Computer Languages
Vienna University of Technology
Vienna, Austria
Email: {jiresch,gramlich}@logic.at

Abstract — Interaction Nets *are a novel model of computation based on graph rewriting. Their main properties are parallel evaluation and sharing of computation, which leads to efficient programs.*
However, Interaction Nets *lack several features that allow for their convenient use as a programming language. In this paper, we describe the implementation of an extension for pattern matching of interaction rules. Furthermore, we show the correctness of the implementation and discuss its complexity.*

I. Introduction

A. Overview

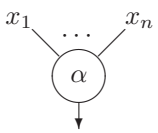
Models of computation are the basis for many programming languages. They allow for reasoning on formal properties of programs such as their correctness and termination (i.e., whether a program eventually halts). An example is the λ-calculus, which is the basis for functional programming languages such as *Haskell*.

Interaction Nets are a relatively new model of computation based on graph rewriting. They enjoy several useful properties that give them great potential for a future programming language. In this paper, we introduce a method for extended pattern matching which allows for the formulation of more powerful interaction rules. We describe our recent contribution, an implementation of the extended pattern matching, and discuss its properties.

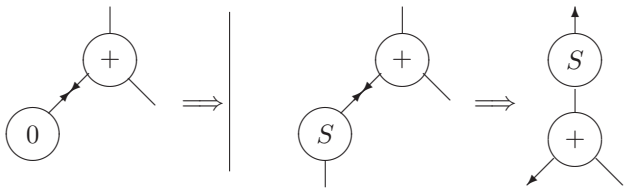
This paper is organised as follows: The next subsection gives a short introduction to *Interaction Nets*. In Section II., we introduce extended pattern matching and outline its implementation. Finally, we present a conclusion and give an outlook on further research.

B. Interaction Nets

Interaction Nets were first introduced in [1]. A *net* is a graph consisting of *agents* (nodes) and *ports* (edges).



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* (denoted by the arrow). For example, the following two 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 more than one interaction rule is applicable at the same time, they can be applied in parallel without interfering with each other. In addition, nets share computation: If an expression appears multiple times in a program, it is evaluated only once.

II. Nested Pattern Matching and its Implementation

The simplicity of interaction rules brings a disadvantage: Only the two nodes that are connected via their principal ports are relevant to the rule. This makes it hard to express functions that depend on more than two non-variable symbols, or in other words, have a more complicated pattern. Consider a function that finds and returns the last element of a list:

```
last Cons(x, Nil) = x
last Cons(x, xs) = last xs
```

The first rule pattern includes three function symbols: $last$, $Cons$ and Nil. Since interaction rules are restricted to two function symbols per rule pattern (the names of the agents), one has to introduce an auxiliary rule to model this function:

```
last Cons(x, xs) = aux x xs
aux x Nil = x
aux x Cons(y, ys) = last Cons(y, ys)
```

To counter this problem, *Interaction Nets* with *nested patterns* were introduced [2]. Nested rules allow for pattern matching of more complicated functions. The beneficial properties of *Interaction Nets* are preserved if the nested rules are of a specific form (i.e., *well-formed*). Moreover, it is possible to transform nested rules into ordinary ones by introducing auxiliary rules, much like in the example above.

The first author contributed to this research by implementing nested pattern matching in the *Interaction Nets* based prototype programming language *inets* [3]. The corresponding theoretical description of this work was presented at the workshop RULE'09 [4]. In the remainder of this section, we outline the main ingredients of the implementation.

The implementation consists of two parts:

1. Verification of the well-formedness of interaction rules
2. Translation of nested rules into ordinary rules

Verification of well-formedness The well-formedness property of a set of interaction rules ensures that there is no overlap between rules: A given pair of agents (with nested arguments) must not match more than one rule. Otherwise, the rules cannot be applied in a deterministic fashion which leads to inconsistent results. The algorithm in *inets* tries to falsify this condition of the rule set: It searches (exhaustively) for two nested patterns that can match the same net. The condition is falsified if and only if the set of rules is not well-formed. A formal proof can be found in [4, 5].

Rule translation We now give an overview of the actual translation of nested rules. This is done as follows: The *inets* compiler reads source code and builds an abstract syntax tree (AST) which is further compiled into byte code and later C source code. Our translation function rewrites ASTs that represent nested rules into ASTs that represent ordinary interaction rules. The back end of the compiler remains unaffected by the translation. Overall, our translation function is similar to the compilation schemes defined in the original paper [2]. We summarise the translation algorithm in the following steps:

1. A rule is found in the AST. If the rule has a nested pattern, its well-formedness is verified.
2. If the rule is well-formed, it is translated: The first nested argument is removed from the rule and an auxiliary rule is generated. This rule is appended to the AST.
3. The remaining nested agents are not (yet) translated. They are resolved by translating the auxiliary rule.
4. The AST is traversed until the next (unprocessed) rule is found.

This algorithm allows for an arbitrary number of nested patterns (i.e., the number of nested agents in a nested rule) and an arbitrary pattern depth.

Properties of the translation We show that the algorithm is terminating: The translation function either fails (due to a non well-formed rule) or yields a set of ordinary interaction rules. The idea behind the formal proof is to show that the number of rules and nested patterns decreases with each call of the translation function.

The time and space complexity of the algorithm can be described as follows: The time complexity of the translation is $O(n^2)$ where n is the sum of the number of rules and nested patterns in the input rule set. This is due to (one part of) the well-formedness check that compares each rule with every other rule ($\frac{n(n-1)}{2}$ checks are performed). Space complexity is linear with the number of input rules. For a more detailed complexity discussion, the reader is referred to [5].

III. Conclusion and Outlook

In this paper, we have given a short introduction to *Interaction Nets* and their extension through nested patterns. We have contributed to this field of research by implementing a translation algorithm for nested patterns in the *Interaction Nets* based programming language *inets*. This algorithm includes verification of the well-formedness of rules. Moreover, the translation handles programming language features that are not part of the original definition of the translation. Some important examples are data values of agents (integers, floats, strings,...), side effects (declaration and manipulation of variables, I/O) and conditions.

Nested pattern matching can serve as a basis for further language extensions for *Interaction Nets*. These extensions are pursued in the context of the PhD project of the first author which is funded by the Austrian Academy of Sciences (ÖAW) and the Vienna PhD School of Informatics.

References

[1] Y. Lafont. Interaction nets. In *Proceedings of the 17th ACM symposium on Principles of programming languages (POPL)*, pages 95–108, 1990.

[2] A. Hassan and S. Sato. Interaction nets with nested pattern matching. *Electr. Notes Theor. Comput. Sci.*, 203(1):79–92, 2008.

[3] The inets project. http://www.interaction-nets.org/.

[4] A. Hassan, E. Jiresch, and S. Sato. Interaction nets with nested patterns: An implementation. Prelim. Proceedings: *10th Int. Workshop on Rule-Based Programming (RULE'09), pp. 14-25, Brasília, Brazil, June 2009*.

[5] A. Hassan, E. Jiresch, and S. Sato. Interaction nets with nested patterns: An implementation. Full version of [4]. In Proceedings of RULE'09: *Electr. Proceed. in Theor. Comp. Sci. (EPTCS)*, 2010, to appear.