Nominal Game Semantics

Andrzej S. Murawski
University of Warwick

Nikos Tzevelekos
Queen Mary University of London
Foundations and Trends® in Programming Languages
Volume 2, Issue 4, 2015
Editorial Board

Editor-in-Chief
Mooly Sagiv
Tel Aviv University
Israel

Editors
Martin Abadi
Google & UC Santa Cruz
Anindya Banerjee
IMDEA
Patrick Cousot
ENS Paris & NYU
Oege De Moor
University of Oxford
Matthias Felleisen
Northeastern University
John Field
Google
Cormac Flanagan
UC Santa Cruz
Philippa Gardner
Imperial College
Andrew Gordon
Microsoft Research & University of Edinburgh
Dan Grossman
University of Washington
Robert Harper
CMU
Tim Harris
Oracle
Fritz Henglein
University of Copenhagen
Rupak Majumdar
MPI-SWS & UCLA
Kenneth McMillan
Microsoft Research
J. Eliot B. Moss
UMass, Amherst
Andrew C. Myers
Cornell University
Hanne Riis Nielson
TU Denmark
Peter O'Hearn
UCL
Benjamin C. Pierce
UPenn
Andrea Pitts
University of Cambridge
Ganesan Ramalingam
Microsoft Research
Mooly Sagiv
Tel Aviv University
Davide Sangiorgi
University of Bologna
David Schmidt
Kansas State University
Peter Sewell
University of Cambridge
Scott Stoller
Stony Brook University
Peter Stuckey
University of Melbourne
Jan Vitek
Purdue University
Philip Wadler
University of Edinburgh
David Walker
Princeton University
Stephanie Weirich
UPenn

Full text available at: http://dx.doi.org/10.1561/2500000017
Editorial Scope

Topics

Foundations and Trends® in Programming Languages publishes survey and tutorial articles in the following topics:

- Abstract interpretation
- Compilation and interpretation techniques
- Domain specific languages
- Formal semantics, including lambda calculi, process calculi, and process algebra
- Language paradigms
- Mechanical proof checking
- Memory management
- Partial evaluation
- Program logic
- Programming language implementation
- Programming language security
- Programming languages for concurrency
- Programming languages for parallelism
- Program synthesis
- Program transformations and optimizations
- Program verification
- Runtime techniques for programming languages
- Software model checking
- Static and dynamic program analysis
- Type theory and type systems

Information for Librarians

Foundations and Trends® in Programming Languages, 2015, Volume 2, 4 issues. ISSN paper version 2325-1107, ISSN online version 2325-1131. Also available as a combined paper and online subscription.
Nominal Game Semantics

Andrzej S. Murawski               Nikos Tzevelekos
University of Warwick            Queen Mary University of London
## Contents

1 Introduction 2

2 Elements of Nominal Set Theory 6

3 GroundML 10
   3.1 Syntax 10
   3.2 Operational semantics 12

4 ToyML: A First-Order Language with Integer References 18
   4.1 Types and terms 18
   4.2 Concrete games 19
   4.3 Interpretation of ToyML terms 22

5 Game Model 32
   5.1 Moves, arenas, plays, strategies 32
   5.2 Full abstraction 58
   5.3 Chapter Appendix: deferred proofs 68

6 Conclusions 71
   6.1 Other paradigms: higher-order references 71
   6.2 From references to Java objects 73
   6.3 Algorithmic game semantics 74
Abstract

These tutorial notes present nominal game semantics, a denotational technique for modelling higher-order programs.
Game semantics is a branch of denotational semantics that uses the metaphor of game playing to model computation. The game models of PCF \cite{5, 21, 35} constructed in the 1990s have led to an unprecedented series of full abstraction results for a range of functional/imperative programming languages. A result of this kind characterises contextual equivalence between terms semantically, i.e. equality of denotations coincides with the fact that terms can be used interchangeably in any context. As such, full abstraction results can be said to capture the computational essence of programs.

The fully abstract game models from the 1990s covered a plethora of computational effects, contributing to a general picture referred to as Abramsky’s cube \cite{8}: by selectively weakening the combinatorial conditions on plays of the games, one was able to increase the expressivity of the games and capture desired computational effects.

Although those works successfully constructed models of state \cite{7, 6, 4, 9}, the techniques used to interpret reference types did not make them fully compatible with what constitutes the norm in languages such as ML or Java. In particular, references were modelled through a form of indirection originating in the work of Reynolds \cite{39}, namely
by assuming that $\text{ref } \theta = (\theta \rightarrow \text{unit}) \times (\text{unit} \rightarrow \theta)$. The approach led to identification of references with pairs of arbitrary reading ($\text{unit} \rightarrow \theta$) and writing ($\theta \rightarrow \text{unit}$) functions. While this view is elegant and certainly comprises the range of behaviours corresponding to references, it does not enforce a relationship between reading and writing, as witnessed by the presence of the product type. This causes a significant strengthening of the semantic universe used for modelling references and, consequently, many desirable equivalences are not satisfied in the model. For example, the interpretation of $(x := 0; x := 1)$ is different from that of $x := 1$ and, similarly, for $x := !x$ and $(\lambda)$. We list the interpretations below using the terminology of [6].

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x := 0; x := 1$</td>
<td>run write(0) ok write(1) ok done</td>
</tr>
<tr>
<td>$x := 1$</td>
<td>run write(1) ok done</td>
</tr>
<tr>
<td>$x := !x$</td>
<td>run read i write(i) ok done</td>
</tr>
<tr>
<td>$(\lambda)$</td>
<td>run done</td>
</tr>
</tbody>
</table>

Thus, for the first term, the semantic translation treats both updates as observable events and therefore both are recorded in the game play. This immediately distinguishes semantically the first term from the second one, for which only a single update is recorded. On the other hand, the translation of the third term is more verbose, registering calls to both the read and write methods of $x$, even though the computational content of the term is in fact that of the skip command $(\lambda)$ in the modelled language.

To prove full abstraction in this setting, it is then necessary to enrich the syntax with terms that will populate the whole semantic space of references. Such terms are often referred to as bad variables, because they are objects of reference type equipped with potentially unrelated reading and writing methods. These terms, if used by the context, can distinguish the pairs of terms discussed above. For instance, a context that instantiates $x$ to a bad variable with divergent reading and writing capabilities will be able to distinguish $x := !x$ from $(\lambda)$. Nonetheless, that solution is not entirely satisfactory as the bad-variable construct breaks standard expectations for references. Moreover, one would hope to be

---

1In effect, write(0) and write(1) represent calls to the write method of reference $x$, while ok’s correspond to returns of that method.
able to carve the model in such a way that it matches the modelled language, instead of extending the language to match the model.

The bad-variable problem can be seen as the result of modelling a generative effect (the creation and use of references) by equating it with the product of its observable handling methods. Nominal game semantics is a recent branch of game semantics that makes it possible to model generative effects in a more direct manner, by incorporating names (drawn from an infinite set) as atomic objects in its constructions. In particular, it can model reference types without bad variables by using names to interpret references. The names are embedded in moves and also feature in stores that are carried by moves in the game. Intuitively, the stores correspond to the observable part of program memory. For example, the two pairs of terms discussed above can be modelled by the following two nominal plays respectively.

\[ a^{\{(a,i)\}} \otimes \{\{(a,1)\}\} \quad a^{\{(a,i)\}} \otimes \{(a,i)\} \]

Here \( a \) stands for an arbitrary name, i.e. the collection of plays is stable with respect to name permutations. Formally, the objects studied in nominal game semantics (moves, plays, strategies) live in nominal sets [12].

Since 2004, the nominal approach has led to a series of new full abstraction results. The languages covered are the \( \nu \)-calculus [3] (purely functional language with names), \( \lambda \nu \) [25] (a higher-order language with storage of untyped names), Reduced ML [31] (a higher-order language with integer-valued storage), RefML [32] (higher-order references) and Middleweight Java [34]. Nominal game semantics has also been used to model Concurrent ML [26] and exceptions [34].

**Structure of the tutorial**

Our tutorial is meant to complement existing introductory literature to game semantics [1, 8, 19, 16], which highlighted the then new structural components necessary to model higher-order computation, e.g. arenas,

---

[2] Similar issues arise when modelling exceptions in this way, i.e. as products of raise/handle functions [24].
justification pointers, innocence. In contrast, we shall particularly focus on explaining the nominal content of our games. We hope the material has been written in a way that will make it accessible to readers familiar with standard denotational semantics and types, e.g. [10, 17, 40].

We begin our exposition with Chapter 2 covering the basics of nominal sets. In Chapter 3 we introduce the programming language of study, called **GroundML**. **GroundML** is a higher-order language with references capable of storing integers, reference to integers, references to references to integers and so on. In Chapter 5 we shall present the game model of **GroundML** in full detail. Before that, in Chapter 4 we focus on a fragment of **GroundML** that, for the sake of simplicity, features only integer-valued references and restricted higher-order types. Because **ToyML** is simpler, we can give a more direct and elementary presentation of its game semantics, which we hope will help the reader to make a transition to the full-blown model of the following section.
References


References


References


