ABSTRACT MACHINES FOR DIALOGUE GAMES

Pierre-Louis Curien & Hugo Herbelin

Panoramas et Synthèses

Numéro 27

2009

SOCIÉTÉ MATHÉMATIQUE DE FRANCE
Publié avec le concours du Centre national de la recherche scientifique

Panoramas & Synthèses
27, 2009, p. 233–277

ABSTRACT MACHINES FOR DIALOGUE GAMES

by

Pierre-Louis Curien & Hugo Herbelin

Abstract. – The notion of abstract Böhm tree has arisen as an operationally-oriented distillation of works on game semantics, and has been investigated in two previous papers. This paper revisits the notion, providing more syntactic support and more examples (like call-by-value evaluation) illustrating the generality of the underlying computing device. Precise correspondences between various formulations of the evaluation mechanism of abstract Böhm trees are established.

***Résumé* (Machines abstraites pour jeux de dialogue).** – La notion d'arbre de Böhm abstrait, introduite et étudiée dans deux articles précédents, est une distillation de travaux en sémantique des jeux, issue d'une volonté d'en expliciter la nature calculatoire. Cet article réexamine cette notion, en fournissant un support syntaxique plus conséquent ainsi que des exemples plus nombreux (comme l'évaluation en appel par valeur), et illustre ainsi la généralité du dispositif de calcul sous-jacent.

1. Introduction

This paper is a contribution to the paradigm of computation as interaction, by which we mean that a computation is described in terms of a game between two players, one representing the expression to be computed, the other its context, containing information such as the values of its free variables, or where the result should be returned to. This line of work has been pursued in different, but related perspectives, giving rise to rich theories and applications.

– The theory of sequential algorithms of Berry and Curien [**4, 5**] arose in the investigation of the full abstraction problem for PCF, a famous problem in the semantics of programming languages. PCF is a core pure functional programming language [**33**], and a fully abstract model is a model capturing the observable differences between programs exactly. Sequential algorithms are mathematical

2010 Mathematics Subject Classification. – 03B40, 68N18, 68N20.
Key words and phrases. – Abstract machines, game semantics, programming languages, lambda-calculus.

objects that in addition to input-output behavior record some information about the order of computation. They turned out to provide a fully abstract model, not of PCF, but of a natural extension of PCF with a non-local control operator [**8**].
- Linear logic, and one of its models in particular – the geometry of interaction – originated in a fine-grained analysis of the cut-elimination process in proof theory, and has brought a wealth of new insights, such as formulas as resources, or proof nets [**18, 19**]. (See Melliès' article in this volume.)
- Game semantics [**9, 25, 1**] was triggered by these previous works, and has allowed to give a neat account of a variety of programming features, such as control, non-determinism, references...

In this paper, we adopt a type-free, operationally-oriented view. Our work takes inspiration mostly from the works of Coquand [**9**] and of the second author [**22, 23**], and from those of Hyland, Ong, and Nickau [**25, 30**]. Our key object is the notion of abstract Böhm tree, which is a generalization of that of Böhm tree (see the introduction to this volume). Böhm trees are (potentially infinite) normal forms, and play an important role in the theory of the λ-calculus [**3**]. The main benefit of the generalization is that it offers the right level of generality for explaining the mechanism of computation at hand in the λ-calculus and similar sequential languages. Abstract Böhm trees have been defined and studied in the two articles [**10, 14**]. Here, we revisit the notion: we provide more syntactic support and more examples (like call-by-value evaluation) illustrating the generality of the underlying computing device. Precise statements on the correspondences between various formulations of the evaluation mechanism of abstract Böhm trees are established.

The paper is meant as rather self-contained, and is organized as follows. Abstract Böhm trees are defined in section 2, where we also introduce our computational engine, called the Geometric Abstract Machine (GAM). A concrete term notation with bound variables, in the style of the λ-calculus, is introduced at the end of this section. Section 3 is devoted to examples, that cover λ-calculus (both normal and non-normal forms), and extensions: PCF and classical PCF; call-by-value evaluation is also treated, and we show finally how Girard's ludics [**20**] fits in our framework. A remarkable feature of our framework is that the computing device need not be extended or adjusted: only the compilation of the different source languages varies, and the machinery of abstract Böhm trees works as a "universal" device.

Sections 4 and 5 propose equivalent formulations of the GAM: the View Abstract Machine (VAM) highlights the important notion of view (basic to the works of Coquand [**9**], and of Hyland and Ong [**25**]), while the Environment Abstract Machine is a straightforward generalization of (a stack-free version) of Krivine Abstract Machine [**27**]. In the appendix, we establish precise correspondences between these machines.

In section 6, we show how to formalize a lazy, stream-like computational loop calling the GAM again and again in order to produce the full result of a composition; each call of the GAM gets us to the (abstract Böhm tree version of the) next head

variable of the composition along a given exploration path. In section 7, we show how to extend the formalism of abstract Böhm trees and the GAM to "non-normal forms".

In section 8, we discuss η-expansion, which is needed to evaluate (the compilation of) untyped λ-terms. In this section, we also discuss the property of separation, which is the ability of observing differences through execution against a fixed counter-strategy.

Finally, in section 9, we discuss some perspectives of further work.

2. The Geometric Abstract Machine

In this section, we present the ingredients of our theory, starting with moves, positions, strategies and counter-strategies (section 2.1), and continuing with our computing device governing the interaction strategy / counter-strategy: the Geometrical Abstract Machine (section 2.3). To this effect, we introduce the notions of multiplexed position, multiplexed strategy, multiplexed counter-strategy (section 2.2), which accommodate the process of duplication in the course of computation (when a function calls its argument several times). The termination cases of the machine are spelled out (section 2.4). A new contribution of this paper is section 2.5, where we provide a term notation for abstract Böhm trees.

We recommend to the reader to freely jump between this section and the next one, where many illustrations of the technical definitions are given.

2.1. Positions and strategies. – We suppose given an alphabet A of move names, containing a special symbol $\bullet$, which is the initial move. A position p is a sequence of moves with backward pointers for Player's moves (that is, moves occurring at even places in the position). We choose to represent pointers by numbers which count the number of Opponent's moves between the pointing Player's move and the pointed Opponent's move. These pointers may be used to relate the bound occurrences to their binders, or to relate values to their return address – i.e., to the root of the subexpression of which they are a (possible) value. Both of these kinds of pointing structure are present in the language PCF (see section 3.2). Pointers always go from a Player's move in the sequence to an earlier Opponent's move in the sequence. Also, the sequence of moves is alternating between Player and Opponent ($\bullet$ is considered Opponent).

An even position, or *response*, i.e., a position of even length, is a sequence of the form:

$$a_1[a_2, \overset{i_1}{\hookleftarrow}] \dots a_{2n-1}[a_{2n}, \overset{i_n}{\hookleftarrow}]$$

where $a_j \in A$ for all $j \leq 2n$ and $i_l \in \omega \cup \{\_\}$ for all $l \leq n$. An odd position, or *query*, is defined in the same way, but ends with an Opponent's move a_{2n-1}. In a position $p\,[a, \overset{i}{\hookleftarrow}]$, with $i \in \omega$, the intention is that the move $[a, \overset{i}{\hookleftarrow}]$ points to the move b of p which is at distance $2i+1$ from $[a, \overset{i}{\hookleftarrow}]$. For instance, if $i = 0$ and $p = p_1 b$, then $[a, \overset{i}{\hookleftarrow}]$ points to b. The number i has to be small enough to guarantee that the

corresponding b exists. We *will always assume this*, and it will be an (easy) invariant of all the abstract machines presented in this paper that these pointers never become dangling while execution progresses. We use _ to designate free occurrences of Player's moves. (For readers familiar with de Bruijn notation in the λ-calculus, this amounts to use de Bruijn indices for bound variables, but names for free variables.)

We shall let q and r range over queries and responses, respectively. We shall use $[a, \overset{\kappa}{\hookleftarrow}]$ to designate either $[a, \overset{i}{\hookleftarrow}]$ or $[a, \overset{-}{\hookleftarrow}]$.

A *strategy* is a set ϕ of positions sucht that:

- all positions of ϕ are of even length, and of the form $\bullet p$, where $\bullet$ does not occur in p,
- ϕ is closed under prefix,
- if $q[a_1, \overset{\kappa_1}{\hookleftarrow}], q[a_2, \overset{\kappa_2}{\hookleftarrow}] \in \phi$, then $[a_1, \overset{\kappa_1}{\hookleftarrow}] = [a_2, \overset{\kappa_2}{\hookleftarrow}]$.

The last property ensures that a "query", that is, a position of odd length, is uniquely answered in a strategy. Another presentation of a strategy is as a partial function, also written ϕ, from queries (whose Player's moves are irrelevant) to Player's moves. We write $dom(\phi)$ to denote the domain of definition of this partial function. We shall freely use either of the two presentations.

A counter-strategy is a forest of strategies, where the roots are renamed so as to hook-up with the free moves of the strategy against which they are placed to play with. The renaming is defined as follows:

$$[a \leftarrow \phi] = \{ar \mid \bullet r \in \phi\}$$

Hence $[a \leftarrow \phi]$'s root is labelled by a. A counter-strategy ψ is a union of renamed strategies $[a_1 \leftarrow \phi_1], \ldots, [a_n \leftarrow \phi_n]$ (with all the a_i's distinct and $\neq \bullet$). We write ψ as:

$$\psi = [a_1 \leftarrow \phi_1, \ldots, a_n \leftarrow \phi_n]$$

Note that by the first condition in the definition of strategy, $\bullet$ does not occur in ψ. (This convention about $\bullet$ applies everywhere in the paper except in section 3.4, where $\bullet$ will be a "real" move expressing the convergence of a function in the weak sense, i.e. the presence of a head λ.)

Nothing prevents us from having infinite horizontal branching after Player's moves, although in most examples branching will only be finite. Nothing prevents us either from having infinite positions and infinite depth strategies.

2.2. Multiplexing. – Next we introduce multiplexed strategies, which will serve to trace dialogues between strategies and counter-strategies. The idea is that during the course of evaluation, nodes may be visited several times, whence the idea of "opening new copies". A *multiplexed* even position is a sequence $\mathbf{p}$ of the form:

$$\langle a_1, \mathbf{j_1} \rangle [a_2, \overset{i_1}{\hookleftarrow}], \ldots, \langle a_{2n-1}, \mathbf{j_n} \rangle [a_{2n}, \overset{i_n}{\hookleftarrow}]$$

where the a's and the i's are as for positions, and where $\mathbf{j_1}, \ldots, \mathbf{j_n} \in \omega$ encode the multiplexing of Opponent's moves. In **[14]**, we used the terminology "dynamic" for