Böhm's Theorem

Stefano Guerrini*   Adolfo Piperno*   Mariangiola Dezani-Ciancaglini†

under the exceptional guidance of Corrado Böhm

April 26, 2008

1   Introduction

The technical significance of Böhm's Theorem [Böhm, 1968] suffices to deserve it a prominent place in any monograph on the theory of the λ-calculus [Barendregt, 1984, Hindley and Seldin, 1986, Hankin, 1995] and makes it a basic result that any researcher working on λ-calculus must know. In addition to its technical content, we think that behind this beautiful result there is something of interest for a much wider audience. The clear thread that starting from his thesis [Böhm, 1954] led Corrado Böhm to the research on λ-calculus and to the quest for an "internal" way to discriminate λ-terms, the deep analysis of the structures of λ-terms required by the proof of the theorem, the so-called Böhm out technique, and the many unexpected consequences and applications of that technique [Lévy, 2001] clearly put Böhm's Theorem on a relevant position in the bookshelf of the main achievements of theoretical computer science. Moreover, as in the case of almost all the relevant results of Mathematics, the interest of Böhm's Theorem is not only in the statement that it asserts, but also, and maybe mainly, in the constructions required by its proof.

As we already mentioned, Böhm's Theorem is one of the main results of the theory of λ-calculus. Looking at the theorem from a computer science perspective, it states that the extensional equivalence of λ-calculus "normal forms" may be defined by means of a syntactic equivalence. More precisely, let us interpret λ-terms as programs, assuming that two programs/λ-terms are equivalent when they behave in the same way on all the inputs (extensional equivalence); then it is a consequence of Böhm's Theorem that two programs/λ-terms in "normal-form" are equivalent if and only if they are written in the same way (syntactic equivalence), apart for some expansions corresponding to the so-called η-equivalence of λ-terms (which is easily decidable). Let us remark that when we say "on all the inputs" we mean "on all the λ-terms", since we are in an untyped setting and any λ-term can be the argument of any other λ-term.

By the way, Böhm's Theorem does not imply at all that in the λ-calculus the equivalence of programs is decidable: the equivalence of two λ-terms (not in normal-form) was the first problem for which undecidability

---

*Dipartimento di Informatica, Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy. Email: guerrini@di.uniroma1.it, piperno@di.uniroma1.it

†Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, 10149 Torino, Italy. Email: dezani@di.unito.it

could be proved [Church, 1936], even before the undecidability of the halting problem [Turing, 1936]. In fact, in the '30s, Church proposed the λ-calculus as a foundational system for mathematical logic [Church, 1932]. Then, while his former student Kleene analyzed the notion of λ-definability showing that every recursive function can be coded (by means of "normal forms") into the λ-calculus [Kleene, 1935, Kleene, 1936], Church related the notion of effective calculability to that of recursive function, and then of λ-definability, proving at the same time that the equivalence of two λ-terms (not in normal-form) is undecidable [Church, 1936]. Immediately after the work of Church, Turing introduced his machine approach to computation and proved the undecidability of the halting problem [Turing, 1936]; then, he also proved the equivalence between his notion of computability and that of λ-definable function [Turing, 1937].

In λ-calculus, programs/λ-terms are constructed in a purely functional way, and there is no distinction between programs and data: every program can be passed as the argument of another program. The evaluation mechanism, the so called β-*rule*, mimics the operation of replacing equally labeled formal place-holders with a specific λ-term, without looking at its actual structure nor if the places are for functions or for arguments. A λ-term is in normal form when it cannot be reduced any further, by the application of the β-rule, implicitly declaring the end of the evaluation process. Being the ending results of the evaluation of a λ-term (when this evaluation terminates), normal forms play the role of *values* and Böhm's Theorem ensures that two values are equal only if they are written in the same way.

In λ-calculus the normal form, if any, of a term is unique; in other words, the result of a computation is independent of the order in which the computational steps are applied. Because of this, normal forms can be seen as the "denotations" of λ-terms, or equivalently, as their primary meaning. Böhm's Theorem ensures that in λ-calculus every denotation of a λ-term (if we limit to programs/λ-terms that terminate) can be written in only one way: two syntactically distinct values/normal forms correspond in fact to two distinct denotations. Such a property makes the λ-calculus an ideal mathematical model in which to interpret programs and in which to study their properties—in particular, their equivalence. Moreover, Böhm's Theorem ensures that the way in which we can separate two distinct λ-terms is internal to the calculus; it suffices to apply the distinct λ-terms to the same suitable sequence of inputs. The construction of such a sequence of inputs requires the determination of a set of combinatory operations on the tree structure of λ-terms that are at the basis the so-called Böhm out technique (see the next section). Proving his theorem, Corrado Böhm not only recognized the basic operations of the Böhm out technique, but also had the great intuition that such combinatory operations could be internalized into the λ-calculus by means of suitable λ-terms. Such a deep understanding of the computational mechanism behind the β-rule was a great breakthrough in the analysis of the basic computational mechanisms of programs and played a central role in the development of the mathematical studies of the semantics of programs (see below the section on the follow-up to Böhm's Theorem).

The idea that an interesting computational system should have enough power to be able to speak about itself has played a central role in all the research of Corrado Böhm, not only in his studies on λ-calculus. In fact, in his thesis [Böhm, 1954], Corrado Böhm defined the first compiler that could be described in its own language (see also [Knuth and Pardo, 1980]). Since then, one of the main questions that guided Corrado Böhm in his research on computational systems was "how much of its meta-theory is contained into the system itself?" Therefore, when he started to think at the λ-calculus as a basis for defining programming languages (actually, he believes that the λ-calculus is THE programming language), one of the first questions that he tried to answer was if there was an internal way for studying the equality of λ-terms, and the answer was Böhm's Theorem.

This note is organized as follows: in Section 2 we explain Böhm's Theorem in an informal way using trees, while in Section 3 we introduce the λ-calculus in order to properly formulate Böhm's Theorem. The last section gives an overview of the impressive research activity which originated from Böhm's Theorem.

2 Böhm's Theorem for Trees

Assume that we are given two disjoint sets of labels, namely a set $\mathcal{L} = \{A, B, C, \ldots\}$, and a set $\ell = \{a, b, c, \ldots\}$. We consider the set $\mathcal{T}$ of *trees* labelled with elements from $\mathcal{L} \cup \ell$, with the restriction that, for any $T \in \mathcal{T}$ and for any $x \in \ell$, the label x appears at most once in T.
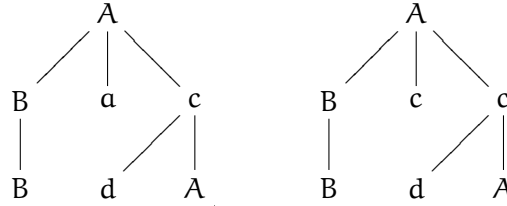
In the example above, the first tree is an element of $\mathcal{T}$, while the second one is not in $\mathcal{T}$, since the label c appears twice in it.

A notion of equivalence is established over elements of $\mathcal{T}$: two trees $T_1, T_2 \in \mathcal{T}$ are *equivalent* if either their are equal or they both can be made equal to a tree $T \in \mathcal{T}$ by adding nodes, labelled in ℓ, as rightmost sons of some nodes. In such a case, we say that T_1 and T_2 *expand* to T. Clearly, both T_1 and T_2 are equivalent to T.

As an example, the first two trees in the following figure are equivalent, since they both expand to the third one.
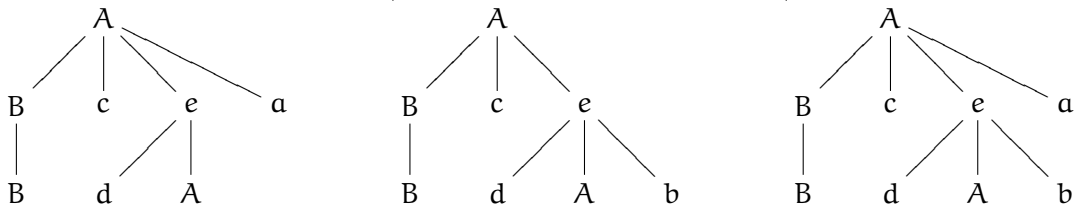

The defined equivalence is relevant since it allows, when comparing trees, to abstract from their structure. In effect, given $T_1, T_2 \in \mathcal{T}$, there always exist $T_1', T_2' \in \mathcal{T}$, having the same tree structure, such that T_i is equivalent to T_i' ($i = 1, 2$).

A *tree transformation* is an operator which, when substituted to a label, rearranges its sons according to some rule. We will use tree transformations of three different kinds:

Replacement: REPLACE(k, S), where $S \in \mathcal{T}$, is the operator which takes k subtrees, discards them and
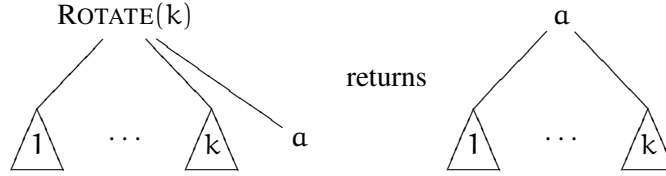
3

returns the tree S:

$$\textsc{Replace}(k, S) \quad \text{returns} \quad S$$
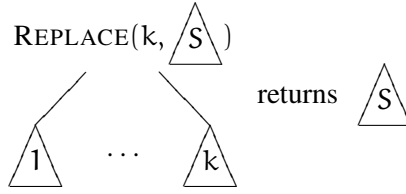
Selection: $\textsc{Select}(k, i)$ is the operator which takes k subtrees and returns the i-th one:

$$\textsc{Select}(k, i) \quad \text{returns} \quad i$$
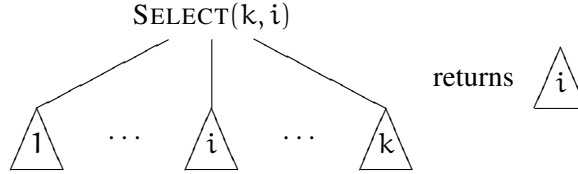
Rotation: $\textsc{Rotate}(k)$ is the operator which takes $k+1$ subtrees, where the $k+1$-th one is a leaf and its label is from ℓ, and moves the $k+1$-th up to the root:

$$\textsc{Rotate}(k) \quad \text{returns} \quad a$$

Böhm Theorem (rephrased)   Let $S_1, S_2 \in \mathcal{T}$ be arbitrary trees. Then for every pair of non-equivalent trees $T_1, T_2 \in \mathcal{T}$ there exist $T_1', T_2' \in \mathcal{T}$ such that:

(i) T_1' is equivalent to T_1 and T_2' is equivalent to T_2;

(ii) there exist tree operations which *discriminate* T_1' from T_2', transforming any tree with the same structure of T_1' and equivalent to T_1 into S_1, and any tree with the same structure of T_2' and equivalent to T_2 into S_2, respectively.

We proceed modulo the previously defined equivalence relation over trees, thus assuming that the trees to be discriminated have the same structure. The proof of the theorem is split into three cases:

A simple case. Let $T_1, T_2 \in \mathcal{T}$ be such that their roots have different labels, say A and B, respectively, as in the following example:
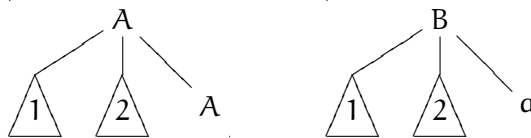
then the substitution $A \leftarrow \textsc{Replace}(3, S_1), B \quad \textsc{Replace}(3, S_2)$ transforms T_1 into S_1 and T_2 into S_2.

4

Information extraction. In general, the difference between the two trees to be discriminated is not imme-
diately visible at the roots. Such information must be extracted from some deeper level by means of
selection operators. As an example, if $T_1, T_2 \in \mathcal{T}$ are the following trees:
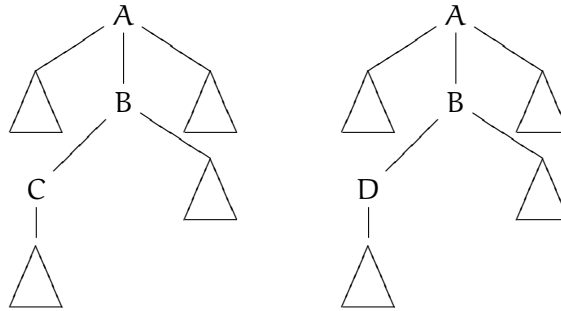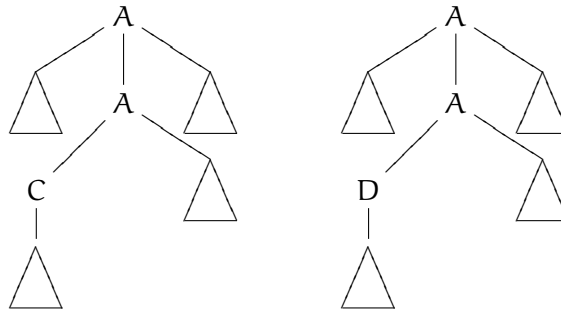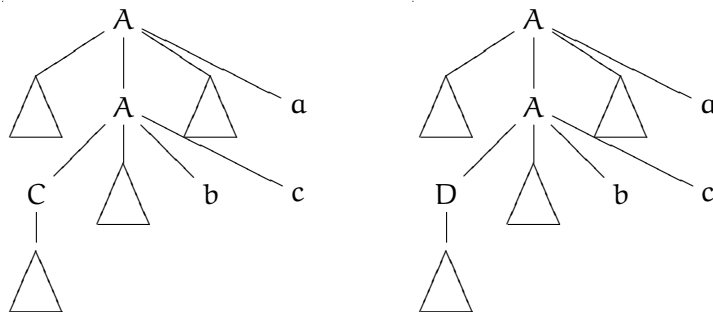
then the substitution

$$A \quad \text{SELECT}(3,2), B \quad \text{SELECT}(2,1), C \quad \text{REPLACE}(1, S_1), D \leftarrow \text{REPLACE}(1, S_2)$$

transforms T_1 into S_1 and T_2 into S_2.

The hard case. In the previous example, the labels C and D have been easily extracted from the trees using
suitable selectors. This is not immediately feasible in the following example:

since the same selector, when substituted to different occurrences of A, is now required to have two
different behaviors; namely, in the topmost occurrence, it must select the second of three subtrees, in
the second occurrence it must select the first of two subtrees. To solve this case, we first consider the
trees $T_1', T_2' \in \mathcal{T}$, equivalent to T_1 and T_2, respectively:

We then apply the substitution $A \leftarrow \text{ROTATE}(3)$, thus obtaining the following trees, which can be
discriminated as in the previous case.
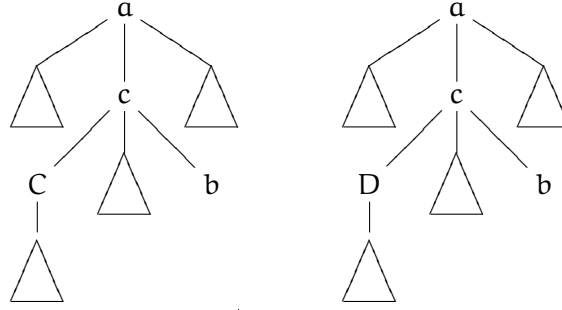
In fact the substitution

$$\mathsf{a} \quad \textsc{Select}(3,2), \mathsf{c} \quad \textsc{Select}(3,1), \mathsf{C} \quad \textsc{Replace}(1, \mathsf{S}_1), \mathsf{D} \leftarrow \textsc{Replace}(1, \mathsf{S}_2),$$

transforms T_1 into S_1 and T_2 into S_2.

3 Böhm's Theorem for λ-calculus

The λ-calculus, sometimes also referred to as the calculus of λ-notation, was introduced by Alonzo Church in the 1930s [Church, 1932]. In the attempt to give a complete system for the foundations of mathematics, Church took as primitive the notion of function instead of that of set. Even if the foundational project failed, because of the fact that the Russell's paradox of "the set of all sets that do not contain themselves as members" can be reformulated in the λ-calculus, Church used the λ-calculus to start the study of computability. In particular, by proving that the equivalence of two λ-terms is undecidable [Church, 1936], Church gave the first problem for which undecidability could be proved, even before the halting problem [Turing, 1936]. Since then the λ-calculus has played a relevant role in the development of theoretical computer science, in particular, it has inspired programming languages like LISP [McCarthy, 1960, Seibel, 2005] and ML [Gordon et al., 1978, Milner, 1984, Milner et al., 1997] and has proved to be a fundamental tool in the analysis of the semantics of programming languages [Tennent, 1976, Winskel, 1993, Stoy, 1997, Pierce, 2002].

The main idea of the λ-calculus is that every expression of the calculus, i.e. every λ-term, stands for a function. λ-terms are built from variables, the basic elements of the calculus, in two ways:

application given two λ-terms T and S, the composition TS represents the application of the function T to the argument S;

abstraction given a λ-term T and a variable x, the abstraction $\lambda\mathsf{x}.\mathsf{T}$ represents the function defined by the λ-term T viewed as an expression parametric in x.

In an abstraction $\lambda\mathsf{x}.\mathsf{T}$, the name x of the variable is no longer relevant, but just a way to denote the places in which the parameter of the function built by abstraction occurs. Therefore the name x could be replaced by any other name y, provided that this would not cause that some occurrences of y in T be improperly associated to the renamed abstraction (e.g., in $\lambda\mathsf{x}.\mathsf{xy}$, the x can be renamed by z getting the equivalent λ-term $\lambda\mathsf{z}.\mathsf{zy}$, but it cannot be renamed by y, that would lead to $\lambda\mathsf{y}.\mathsf{yy}$). The equivalence induced by variable renaming is the so-called α-congruence.

A λ-term is *closed* if all its variables are abstracted.

The only "computational rule" of the λ-calculus is the β-rule that, given the application of an abstraction λx.T to S, replaces S to every occurrence of the variable x in T (this may also require some variable renaming in order to avoid that the variables in S not associated to any abstraction might be "captured" by some abstraction in T). A λ-term is in *normal form* when it cannot be transformed (reduced) by means of the β-rule.

The β-equivalence is the equivalence relation induced on λ-terms by the β-reduction, assuming that two λ-terms are equivalent when they are the same λ-term or when they can be β-reduced to equivalent λ-terms (equivalently, the β-equivalence is the congruence generated by the reflexive, symmetric and transitive closure of the β-rule).

The η-equivalence is the equivalence obtained by assuming that given a λ-term T, if we apply it to a fresh variable x and we construct then the abstraction λx.Tx, we obtain a function that is equivalent to T. In fact, the two functions are extensionally equivalent, since the application of λx.Tx to any λ-term S, immediately reduces to TS by using the β-rule, namely $(\lambda x.Tx)S$ β-reduces to T.

Böhm's Theorem says that the equational theory induced by the β-reduction is complete for its normal forms. In fact, trying to equate any pair of non η-equivalent normal λ-terms would correspond to equating the whole set of the normal λ-terms, forcing the collapse of the whole set of λ-terms into one point.

Normal forms have a structure which is similar to that of the trees defined in Section 2. The η-equivalence corresponds to the equivalence defined over those trees and a representation of normal forms can be obtained by adding abstractions in suitable positions: the λ-calculus expert can recognize the so called *Böhm trees* (see Section 4.4). The tree operations in Section 2 can be represented by λ-terms, so that the discrimination algorithm for closed normal forms can be *internalized*: it can be performed by objects of the calculus itself. For instance, the operators REPLACE(k, S), SELECT(k, i) and ROTATE(k) correspond to the λ-terms

$$\lambda x_1 \dots x_k.S, \qquad \lambda x_1 \dots x_k.x_i \quad \text{and} \quad \lambda x_1 \dots x_{k+1}.x_{k+1}x_1 \dots x_k,$$

respectively. We are now able to state Böhm's Theorem in its original form:

Theorem 1 (Böhm, 1968) *Let Λ_N^0 be the set of closed normal forms, and let S_1 and S_2 be arbitrary λ-terms. For any non η-equivalent terms $T_1, T_2 \in \Lambda_N^0$ there exists a λ-term Δ such that the application of Δ to T_1 evaluates to S_1 and the application of Δ to T_2 evaluates to S_2.*

4   Follow-up to Böhm's Theorem

The semantics of a programming language gives meanings to programs. This can be done in two different ways: *operationally*, providing a way in which programs are evaluated; *denotationally*, defining an interpretation of programs into a *model*, a mathematical structure which is constructed in order to be able to describe some desired computational properties. A huge amount of research has derived from the result and from the technique of Böhm's Theorem, characterizing relevant properties of λ-terms, from both the operational and the denotational perspectives.

4.1 Böhm's Work on Böhm's Theorem

Corrado Böhm himself, together with some of his collaborators, has continued investigating discriminability of λ-terms, essentially from an operational perspective. In [Böhm et al., 1978] a finite set of closed normal forms pair-wise non η-equivalent are discriminated. In [Böhm and Piperno, 1987] the proof of the theorem is revisited according to some restrictions on the shape of the discriminating solution. The notion of X-*separability* has been introduced in [Böhm and Tronci, 1987] and then characterized in [Böhm and Piperno, 1988]. In some sense, X-separability avoids the use of rotation operators at the outer level of λ-terms, introducing the set X of variables to be substituted by operators. The notion of X-separability has interesting relationships with invertibility of λ-terms. The Böhm-out technique is at the basis of the implementation, presented in [Böhm et al., 1994], of the CuCh-machine, a λ-calculus interpreter introduced by Böhm and Gross in [Böhm and Gross, 1966].

4.2 Generalizations of Böhm's Theorem

The first generalizations of Böhm's Theorem considered the pure λ-calculus. Wadsworth in [Wadsworth, 1976] extended Böhm's Theorem to two arbitrary λ-terms which are different in Scott's D_∞ model [Scott, 1972]. We already mentioned [Böhm et al., 1978] in previous subsection. Finally in [Coppo et al., 1978] the discriminability of a finite set of arbitrary λ-terms is characterised. The original Böhm's Theorem and this last generalization are essentially the content of Section 10.4 in [Barendregt, 1984]. The discrimination of infinite sets of λ-terms has been studied in [Ronchi della Rocca, 1981, Piperno, 1999, Statman and Barendregt, 2005].

Successively the λ-calculus has been extended or immersed in other languages in order to obtain finer observations on the behaviour of λ-terms. Sangiorgi in [Sangiorgi, 1994] considers the encoding of λ-calculus in the π-calculus, a calculus of mobile processes, and the addition of a unary non deterministic operator. A notion of resource is the extension considered in [Boudol and Laneve, 1996], while [Dezani-Ciancaglini et al., 1996] and [Dezani-Ciancaglini et al., 1999] add a binary parallel operator and a non deterministic choice. All the above mentioned extensions are equivalent from the point of view of discriminability. A weaker discriminability result is obtained by adding to the λ-calculus a binary non deterministic choice and a numeral system in [Dezani-Ciancaglini et al., 1998]. A finer discriminability is presented in [Dezani-Ciancaglini et al., 2003] by means of two suitable projection operators.

4.3 Theories and models of λ-calculus

One immediate consequence of Böhm's Theorem is that the theory of η-equivalence for closed normal forms is Hilbert-Post complete, i.e. given two arbitrary λ-terms $T_1, T_2 \in \Lambda_N^0$, either they are η-equivalent or the theory obtained by adding the equality $T_1 = T_2$ is inconsistent (see Corollary 10.4.3 of [Barendregt, 1984]).

Therefore no consistent model of λ-calculus can equate non η-equivalent closed normal forms.

Similarly, the generalization of Böhm's Theorem of [Wadsworth, 1976] (already mentioned in Section 4.2), implies that the theory of Scott's D_∞ model [Scott, 1972] turns out to be maximal [Wadsworth, 1976].

4.4 Böhm Trees and Böhm Out-technique

The paramount historical importance of Böhm's Theorem lies in the fact, already stressed by the author in the original paper and afterwards pointed out by various researchers, that its proof is constructive; an elegant implementation in CAML is given in [Huet, 1993].

Exactly the original proof of Böhm's Theorem has inspired a representation of normal forms as trees, similar to the representation discussed in Section 2, which was first introduced in [Böhm and Dezani-Ciancaglini, 1974] and then discussed in [Curry, 1979]. Barendregt [Barendregt, 1984] extended this representation to arbitrary λ-terms and called Böhm trees the so obtained trees. Barendregt called also Böhm out-technique essentially the tree operators on trees that we introduced in Section 2. Other trees have been proposed to represent λ-terms: a recent survey can be found in [Ketema, 2006], where Böhm trees for λ-term rewriting systems are studied. The representation of closed normal forms of [Böhm and Dezani-Ciancaglini, 1974] has been later used in [Piperno, 1999] in order to express Böhm's Theorem as a non-equality predicate over the algebra of normal forms.

4.5 Observational Equivalence

In the same year as Böhm's Theorem (1968), Morris [Morris, 1968] for the first time defined a notion of observational or contextual equivalence, which was going to have such important developments in more recent years, particularly in the domain of interactive concurrent computing: two λ-terms were defined equivalent if, whenever they are put in a same context, either they both make it reducible to a normal form or they both make it divergent. Böhm's Theorem can then be viewed as stating that such an observational equivalence coincides, for normal forms, with η-equivalence. A survey on the relations between Böhm's Theorem and observational equivalence is [Dezani-Ciancaglini and Giovannetti, 2001].

References

[Barendregt, 1984] Barendregt, H. (1984). *The Lambda Calculus, Its Syntax and Semantics*. North-Holland Publishing Co., Amsterdam, NL, revised edition.

[Böhm, 1954] Böhm, C. (1954). Calculatrices Digitales. Du d'Echiffrage des Formules Mathématiques par la Machine Même dans la Conception du Programme. *Annali di Matematica Pura e Applicata*, IV(XXXVII):1–51.

[Böhm, 1968] Böhm, C. (1968). Alcune Proprietà delle Forme Normali nel **K**-calcolo. Technical Report 696, INAC, Roma, Italy.

[Böhm and Dezani-Ciancaglini, 1974] Böhm, C. and Dezani-Ciancaglini, M. (1974). Combinatorial Problems, Combinator Equations and Normal Forms. In Loeckx, J., editor, *ICALP'74*, volume 14 of *Lecture Notes in Computer Science*, pages 185–199, Berlin, Germany. Springer-Verlag.

[Böhm et al., 1978] Böhm, C., Dezani-Ciancaglini, M., Peretti, P., and Ronchi della Rocca, S. (1978). A Discrimination Algorithm inside Lambda-calculus. *Theoretical Computer Science*, 8(3):271–291.

[Böhm and Gross, 1966] Böhm, C. and Gross, W. (1966). Introduction to the CUCH. In R.Caianiello, E., editor, *Automata Theory*, pages 35–65, London, UK. Academic Press.

[Böhm and Piperno, 1987] Böhm, C. and Piperno, A. (1987). Surjectivity for Finite Sets of Combinators by Weak Reduction. In Börger, E., Büning, H. K., and Richter, M. M., editors, *CSL'87*, volume 329 of *Lecture Notes in Computer Science*, pages 27–43, Berlin, Germany. Springer-Verlag.

[Böhm and Piperno, 1988] Böhm, C. and Piperno, A. (1988). Characterizing X-Separability and One-Side Invertibility in Lambda-Beta-Omega-Calculus. In Gurevich, Y., editor, *LICS'88*, pages 91–101, New York, NY, USA. IEEE Computer Society Press.

[Böhm et al., 1994] Böhm, C., Piperno, A., and Guerrini, S. (1994). Lambda-Definition of Function(al)s by Normal Forms. In Sannella, D., editor, *ESOP'94*, volume 788 of *Lecture Notes in Computer Science*, pages 135–149, Berlin, Germany. Springer-Verlag.

[Böhm and Tronci, 1987] Böhm, C. and Tronci, E. (1987). X-Separability and Left-Invertibility in Lambda-calculus. In Gries, D., editor, *LICS'87*, pages 320–328, New York, NY, USA. IEEE Computer Society Press.

[Boudol and Laneve, 1996] Boudol, G. and Laneve, C. (1996). The Discriminating Power of Multiplicities in the λ-calculus. *Information and Computation*, 126(1):83–102.

[Church, 1932] Church, A. (1932). A Set of Postulates for the Foundation of Logic. *Annals of Mathematics*, 33:346–366.

[Church, 1936] Church, A. (1936). An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, 58:345–363.

[Coppo et al., 1978] Coppo, M., Dezani-Ciancaglini, M., and Rocca, S. R. D. (1978). (Semi)-separability of Finite Sets of Terms in Scott's D_∞-Models of the lambda-Calculus. In Ausiello, G. and Böhm, C., editors, *ICALP'78*, volume 62 of *Lecture Notes in Computer Science*, pages 142–164, Berlin, Germany. Springer-Verlag.

[Curry, 1979] Curry, H. B. (1979). On a Polynomial Representation of λβ-normal Forms. In Lorenz, K., editor, *Konstruktionen versus Positionen*, pages 94–98, Berlin, Germany. Walter de Gruyter.

[Dezani-Ciancaglini et al., 1996] Dezani-Ciancaglini, M., de'Liguoro, U., and Piperno, A. (1996). Filter Models for Conjunctive-disjunctive λ-calculi. *Theoretical Computer Science*, 170(1-2):83–128.

[Dezani-Ciancaglini and Giovannetti, 2001] Dezani-Ciancaglini, M. and Giovannetti, E. (2001). From Böhm's Theorem to Observational Equivalences: an Informal Account. In Lévy, J.-J., editor, *BOTH'01*, volume 50 of *Electronic Notes in Theoretical Computer Science*, pages 83–116, New York, NY, USA. Elsevier.

[Dezani-Ciancaglini et al., 1998] Dezani-Ciancaglini, M., Intrigila, B., and Venturini-Zilli, M. (1998). Böhm's Theorem for Böhm Trees. In Degano, P. and Vaccaro, U., editors, *ICTCS'98*, pages 1–23, Oxford, UK. World Scientific.

[Dezani-Ciancaglini et al., 2003] Dezani-Ciancaglini, M., Severi, P., and de Vries, F.-J. (2003). Infinitary Lambda Calculus and Discrimination of Berarducci Trees. *Theoretical Computer Science*, 298(2):275–302.

[Dezani-Ciancaglini et al., 1999] Dezani-Ciancaglini, M., Tiuryn, J., and Urzyczyn, P. (1999). Discrimination by Parallel Observers: the Algorithm. *Information and Computation*, 150(2):153–186.

[Gordon et al., 1978] Gordon, M. J. C., Milner, R., Morris, L., Newey, M. C., and Wadsworth, C. P. (1978). A Metalanguage for Interactive Proof in LCF. In Szymanski, T. G., editor, *POPL'78*, pages 119–130, New York, NY, USA. ACM Press.

[Hankin, 1995] Hankin, C. (1995). *Lambda Calculi: A Guide for Computer Scientists*. Oxford University Press, Oxford, UK.

[Hindley and Seldin, 1986] Hindley, J. and Seldin, J. P. (1986). *Introduction to Combinators and λ-calculus*. Cambridge University Press, Cambridge, UK.

[Huet, 1993] Huet, G. (1993). An Analysis of Böhm's Theorem. *Theoretical Computer Science*, 121(1-2):145–167.

[Ketema, 2006] Ketema, J. (2006). *Böhm-Like Trees for Rewriting*. PhD thesis, Vrije Universiteit Amsterdam, Amsterdam, NL.

[Kleene, 1935] Kleene, S. (1935). A Theory of Positive Integers in Formal Logic. *American Journal of Mathematics*, 57:153–173 and 219–244.

[Kleene, 1936] Kleene, S. (1936). Lambda-Definability and Recursiveness. *Duke Mathematical Journal*, 2:340–353.

[Knuth and Pardo, 1980] Knuth, D. E. and Pardo, L. T. (1980). The Early Development of Programming Languages. In Metropolis, N., Howlett, J., and Rota, G.-C., editors, *A History of Computing in the Twentieth Century*. Academic Press, London, UK.

[Lévy, 2001] Lévy, J.-J. (2001). *BOTH'01: Böhm's Theorem: Applications to Computer Science Theory*, volume 50 of *Electronic Notes in Theoretical Computer Science*. Elsevier, New York, NY, USA.

[McCarthy, 1960] McCarthy, J. (1960). Recursive Functions of Symbolic Expressions and their Computation by Machine, Part I. *Communications of the ACM*, 3(4):184–195.

[Milner, 1984] Milner, R. (1984). A Proposal for Standard ML. In Boyer, R. S., Schneider, E. S., and Guy L. Steele, J., editors, *LISP and Functional Programming*, pages 184–197, New York, NY, USA. ACM Press.

[Milner et al., 1997] Milner, R., Tofte, M., Harper, R., and Macqueen, D. (1997). *The Definition of Standard ML - Revised*. MIT Press, Cambridge, MA, USA.

[Morris, 1968] Morris, J. H. (1968). *Lambda Calculus Models of Programming Languages*. PhD thesis, MIT, Cambridge, MA, USA.

[Pierce, 2002] Pierce, B. C. (2002). *Types and Programming Languages*. MIT Press, Cambridge, MA, USA.

[Piperno, 1999] Piperno, A. (1999). An Algebraic View of the Böhm-Out Technique. *Theoretical Computer Science*, 212(1-2):233–246.

[Ronchi della Rocca, 1981] Ronchi della Rocca, S. (1981). Discriminability of Infinite Sets of Terms in the D_∞-models of the λ-calculus. In Astesiano, E. and Böhm, C., editors, *CAAP'81*, volume 112 of *Lecture Notes in Computer Science*, pages 40–54, Berlin, Germany. Springer-Verlag.

11

[Sangiorgi, 1994] Sangiorgi, D. (1994). The Lazy Lambda Calculus in a Concurrency Scenario. *Information and Computation*, 111(1):120–153.

[Scott, 1972] Scott, D. (1972). Continuous Lattices. In Lawvere, F., editor, *Toposes, algebraic geometry and logic*, volume 274 of *Lecture Notes in Mathematics*, pages 97–136. Springer-Verlag, Berlin, Germany.

[Seibel, 2005] Seibel, P. (2005). *Practical Common Lisp*. Apress, Berkeley, CA, USA.

[Statman and Barendregt, 2005] Statman, R. and Barendregt, H. (2005). Böhm's Theorem, Church's Delta, Numeral Systems, and Ershov Morphisms. In Middeldorp, A., van Oostrom, V., van Raamsdonk, F., and de Vrijer, R., editors, *Processes, Terms and Cycles: Steps on the Road to Infinity*, volume 3838 of *Lecture Notes in Computer Science*, pages 40–54, Berlin, Germany. Springer-Verlag.

[Stoy, 1997] Stoy, J. E. (1997). *Denotational Semantics: The Scott-Strachey Approach to Programming Language Semantics*. MIT Press, Cambridge, MA, USA.

[Tennent, 1976] Tennent, R. D. (1976). The Denotational Semantics of Programming Languages. *Communications of ACM*, 19(8):437–453.

[Turing, 1936] Turing, A. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings London Mathematical Society*, 42(2):230–265. correction ibid. 43, pp 544-546 (1937).

[Turing, 1937] Turing, A. (1937). Computability and λ-definability. *Journal of Symbolic Logic*, 2:153–163.

[Wadsworth, 1976] Wadsworth, C. P. (1976). The Relation between Computational and Denotational Properties for Scott's D_∞-models of the Lambda-calculus. *SIAM Journal on Computing*, 5(3):488–521.

[Winskel, 1993] Winskel, G. (1993). *The Formal Semantics of Programming Languages, an Introduction*. MIT Press, Cambridge, MA, USA.