Meaningfulness and Genericity in a Subsuming Framework

Delia Kesner ∇ ☑
Université Paris Cité – CNRS – IRIF, France

Victor Arrial ∇ ☑
Université Paris Cité – CNRS – IRIF, France

Giulio Guerrieri ∇ ☑
University of Sussex, Department of Informatics, Brighton, United Kingdom

Abstract

This paper studies the notion of meaningfulness for a unifying framework called dBang-calculus, which subsumes both call-by-name (dCBN) and call-by-value (dCBV). We first define meaningfulness in dBang and then characterize it by means of typability and inhabitation in an associated non-idempotent intersection type system previously appearing in the literature. We validate the proposed notion of meaningfulness by showing two properties: (1) consistency of the smallest theory, called H, equating all meaningless terms, and (2) genericity, stating that meaningless subterms have no bearing on the significance of meaningful terms. The theory H is also shown to have a unique consistent and maximal extension H*, which coincides with a well-known notion of observational equivalence. Last but not least, we show that the notions of meaningfulness and genericity in the literature for dCBN and dCBV are subsumed by the corresponding ones proposed here for the dBang-calculus.

2012 ACM Subject Classification Theory of computation → Operational semantics

Keywords and phrases Lambda calculus, Solvability, Meaningfulness, Inhabitation, Genericity

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.1

Category Invited Talk


1 Introduction

A common line of research in logic and theoretical computer science is to find unifying frameworks that subsume different paradigms, systems or calculi. Examples are call-by-push-value [54, 55], polarized system LU [45], linear calculi [57, 58, 72], bang-calculus [38, 39, 23, 24], system L [62, 35], ecumenical systems [68], monadic calculus [60, 61], and others [71, 40, 73].

The relevance of these unifying frameworks lies in the range of properties and models they encompass. Finding unifying and simple primitives, tools and techniques to reason about properties of different systems is challenging, and provides a deeper and more abstract understanding of these properties. The advantages of this kind of approach are numerous, for instance the several-for-one deal: study a property in a unifying framework gives appropriate intuitions and hints for free for all the subsumed systems. The aim of this paper is to go beyond the state of the art in a framework subsuming the call-by-name and call-by-value evaluation mechanisms, by unifying their notions of meaningful (and meaningless) programs.

Call-by-name and call-by-value. Every programming language implements a particular evaluation strategy, specifying when and how parameters are evaluated during function calls. For example, in call-by-value (CBV), the argument is evaluated before being passed to the function, while in call-by-name (CBN) the argument is passed immediately to the function.
Meaningfulness and Genericity in a Subsuming Framework

body, so that it may never be evaluated, or may be re-evaluated several times. These models of computation serve as the basis for many theoretical and practical studies in programming languages and proof assistants, such as OCaml, Haskell, Coq, Isabelle, etc.

The CBN strategy has garnered significant attention in the literature on theoretical studies and is generally perceived as well-established. In contrast, the CBV strategy has received limited attention. Despite their similarities, CBN and CBV strategies have predominantly been studied independently, leading to a fragmented research. This approach not only duplicates research efforts – once for CBN and once again for CBV – but also generally results in ad-hoc methods for dealing with the CBV case that are naively adapted from the CBN one.

Understanding the (logical) duality between CBN and CBV (e.g. [34]) marked a significant step towards properly unifying these models. It paved the way for the emergence of Call-by-Push-Value (CBPV), a unifying framework introduced by P.B. Levy [54, 55] which subsumes, among others, CBN and CBV denotational and operational semantics thanks to the distinction between computations and values, according to the slogan “a value is, a computation does”. This framework attracts growing attention: proving advanced properties of a single unifying paradigm, and subsequently instantiating them for a wide range of computational models.

The distant Bang-calculus. Drawing inspiration from Girard’s Linear Logic (LL) [44] and the interpretation of CBPV into LL [38], Ehrhard and Guerrieri [39] introduced an (untyped) restriction of CBPV, named Bang-calculus, already capable of subsuming both CBN and CBV. It is obtained by enriching the λ-calculus with two modalities ! and its dual der. The modality ! actually plays a twofold role: it freezes the evaluation of subterms (called thunk in CBPV), and it marks what can be duplicated or erased during evaluation (i.e. copied an arbitrary number of times, including zero). The modality der annihilates the effect of !, effectively restoring computation and eliminating duplicability. Embedding CBN or CBV into the Bang-calculus via Girard’s translations simply consists in decorating λ-terms with ! and der, thereby forcing one model of computation or the other one. Thanks to these elementary modalities and embeddings, the Bang-calculus eases the identification of shared behaviors and properties of CBN and CBV, encompassing both syntactic and semantic aspects of them.

The original Bang-calculus [39] uses some permutation rules, similar to the ones used in [70, 30], that unveil hidden redexes and unblock reductions that otherwise would be stuck. These permutation rules make the calculus adequate, preventing some normal forms from being observationally equivalent to non-terminating terms. A major drawback is that the resulting combined reduction is not confluent (Page 6 in [39]). The distant Bang-calculus (dBang) [23, 24] was proposed as an adequate and confluent alternative. This is achieved by enriching the syntax with explicit substitutions, in the vein of Accattoli and Kesner’s linear substitution calculus [7, 9, 1, 2] (generalizing in turn Milner’s calculus [59, 51]), thanks to rewrite rules that act at a distance, so that permutation rules are no longer needed.

In this paper, we focus on dBang, and its relations with dCBN [9, 1] and dCBV [11], which are distant adequate variants of the CBN and CBV λ-calculi. This unifying framework is fruitful, subsuming numerous dCBN and dCBV properties through their associated embedding, as for instance big step semantics: evaluating the result from the dCBN/dCBV embedding of a given program t with the dBang model actually corresponds to the embedding of the result of evaluating the original program t with the dCBN/dCBV model. In other words, dBang is a language that breaks down the dCBN and dCBV paradigms into elementary primitives.

Let us now review the state of the art by discussing some advanced properties of programming languages that have been studied in the literature by using the unifying approach dBang. Some of these results, including this work, strongly rely on semantical tools such as quantitative types. To ensure clarity regarding the state of the art, let us briefly discuss in first place the main ideas behind quantitative types.
Quantitative Type Systems. Intersection type systems \[31, 32\] increase the typability power on \(\lambda\)-terms with respect to simple types by introducing a new intersection type constructor \(\land\) that is associative, commutative and idempotent \((i.e., \sigma \land \sigma = \sigma)\). Intersection types allow terms to have different types simultaneously, e.g. a term has type \(\sigma \land \tau\) whenever it has both types \(\sigma\) and \(\tau\). They constitute a powerful tool to reason about qualitative properties of programs. For example, different notions of normalization can be characterized using intersection types \[67, 33\], in that a term \(t\) is typable in a given system if and only if \(t\) is normalizing (as a consequence, typability in these systems is undecidable). An alternative version of intersection type systems for the \(\lambda\)-calculus, called non-idempotent \[43, 36\], is obtained by dropping idempotence. In such a setting, a term of type \(\sigma \land \sigma \land \tau\) can be seen as a resource used exactly once as a data of type \(\tau\) and twice as a data of type \(\sigma\).

Interestingly, such type systems provide not only qualitative characterizations of different operational properties, but also quantitative ones: e.g. a term \(t\) is still typable if and only if \(t\) is normalizing; moreover any type derivation of \(t\) gives an upper bound to the execution time for \(t\) (the number of steps to reach a normal form) \[37\]. These upper bounds can be further refined into exact measure using tight non-idempotent typing systems, as pioneered in \[4\].

State of the Art. This paper contributes to a broader initiative aimed at consolidating the theory of dCBN and dCBV, by unifying them into dBang. Several results have already been factorized and generalized in this framework, we now revisit some of them.

In \[46\], it is shown that the interpretation of a term \(t\) in any denotational model of CBN/CBV obtained from LL is included in the interpretation of the CBN/CBV translation of \(t\) in any denotational model of Bang obtained from LL. The reverse inclusion also holds for CBN but not for CBV. In particular, these results apply to typability in non-idempotent intersection type systems inspired by LL. Indeed, typing is preserved by Girard’s translations, meaning that if a term is typable in the CBN/CBV type system, then its CBN/CBV translation is typable in the type system \(B\) for Bang, using the same types. The converse holds for CBN but not for CBV. In \[23, 24\], the CBV typing system is modified so that the reverse implication also holds. Moreover, an extension of Girard’s CBN translation to dCBN and a new CBV translation for dCBV are proposed. Similar typing preservation results have been obtained in \[52\] for the translations in \[23, 24\], but for the more precise notion of tight typing introduced in \[4\].

Retrieving dynamic properties from Bang into CBN and CBV turns out to be a more intricate task, especially in their adequate (distant) variant \[23, 41, 24\].

In \[46\] it is shown that CBN and CBV can be simulated by reduction in Bang through Girard’s original translations. But the CBV translation fails to preserve normal forms, as some CBV normal forms translate to reducible terms in Bang. This issue is solved in dBang \[23, 24\], thanks to the new CBV translation for dCBV previously mentioned. In the end, reductions and normal forms are preserved by both the CBN and the new CBV translations.

Even if dCBN and dCBV can be both simulated by reduction in dBang, the converse, known as reverse simulation, holds for dCBN but fails for dCBV \[24, 14\]: a dBang reduction sequence from a term in the image of the dCBV embedding may not correspond to a valid reduction sequence in dCBV. Yet another new dCBV translation is proposed in \[14\] so that simulation and reverse simulation are now recovered.

Another major contribution concerns the inhabitation problem: given an environment \(\Gamma\) (a type assignment for variables) and a type \(\sigma\), decide whether there is a term \(t\) that can be typed with \(\sigma\) under the environment \(\Gamma\). While inhabitation was shown \[74\] to be undecidable in CBN for idempotent intersection type systems, it turns out to be decidable \[25, 28\] in the non-idempotent setting. Decidability of the inhabitation problem leads to the development
of automatic tools for type-based program synthesis [56, 21], whose goal is to construct a program – the term \( t \) – that satisfies some high-level formal specification, expressed as a type \( \sigma \) with some assumptions described by the environment \( \Gamma \). It has been proved in [13] that the algorithms deciding the inhabitation problem for dBang and dCBV can be inferred from the corresponding one for dBang, thus providing a unified solution to this relevant problem.

### Meaningfulness and Genericity

In this work, we aim to unify the notions of meaningfulness and genericity in dCBN and dCBV so as to derive them from the respective ones in dBang.

A naive approach to set a semantics for the pure untyped \( \lambda \)-calculus is to define the meaning of a \( \beta \)-normalizing \( \lambda \)-term as its normal form, and equating all \( \lambda \)-terms that do not \( \beta \)-normalize. The underlying idea is that, as \( \beta \)-reduction represents evaluation and a normal form stands for its outcome, all non-\( \beta \)-normalizing \( \lambda \)-terms (i.e., diverging programs) are then considered as meaningless. However, this simplistic approach is flawed, as thoroughly discussed in [20]. For example, any \( \lambda \)-theory equating all non-\( \beta \)-normalizing \( \lambda \)-terms is inherently inconsistent – it effectively equates all \( \lambda \)-terms, not just the meaningless ones!

Alternatively, during the 70s, Wadsworth [75, 76] and Barendregt [17, 18, 19, 20] showed that the meaningful (CBN) \( \lambda \)-terms can be identified with the solvable ones. Solvability is defined in a rather technical way: a \( \lambda \)-term \( t \) is solvable if there is a special kind of context, called head context \( H \), sending \( t \) to the identity function \( I = \lambda z . z \), meaning that \( H(t) \) \( \beta \)-reduces to \( I \). Roughly, a solvable \( \lambda \)-term \( t \) may be divergent, but its diverging subterms can be eliminated by supplying the right arguments to \( t \) via an appropriate interaction with a suitable head context \( H \). For instance, in CBN, \( x \Omega \) is divergent but solvable using the head context \( H = (\lambda x . \delta)(\lambda y . I) \). It turns out that unsolvable \( \lambda \)-terms constitutes a strict subset of the non-\( \beta \)-normalizing ones. Moreover, the smallest \( \lambda \)-theory that equates all unsolvable \( \lambda \)-terms is consistent (i.e., it does not equate all terms). In Barendregt’s book [20], these results rely on a keystone property known as (full) genericity, which states that meaningless subterms are computationally irrelevant – in the sense that they do not play any role – in the evaluation of \( \beta \)-normalizing terms. Formally, if \( t \) is unsolvable and \( C(t) \) \( \beta \)-reduces to some \( \beta \)-normal term \( u \) for some context \( C \), then \( C(s) \) \( \beta \)-reduces to \( u \) for every \( \lambda \)-term \( s \). This property stands as a fool guard that the choice of meaningfulness is adequate. A variant of genericity [16], called surface in [15] and tight in [10], states that any meaningless subterm \( t \) is irrelevant in a meaningful term \( C(t) \) in that \( C(s) \) is still meaningful, for every term \( u \).

Meaningfulness was also studied for first order rewriting systems [48] and other strategies of the \( \lambda \)-calculus [71]. Notably, finding the correct notion of meaningfulness for CBV has been a challenge [5, 6, 15]. Similarly, an extension of the dCBN was studied [29, 26] in the framework of a \( \lambda \)-calculus equipped with pattern matching for pairs. The use of different data structures in the language – functions and pairs – makes meaningfulness more challenging. Indeed, it was shown that meaningfulness cannot be characterized only by means of typability alone, as in CBN and CBV, but also requires some additional conditions stated in terms of the inhabitation problem previously mentioned. This result for the \( \lambda \)-calculus with patterns inspired the characterization of meaningfulness for dBang that we provide in this paper. Genericity for dCBN and the more subtle case of dCBV was recently proved in [15].

### Our Contributions

We first define meaningfulness for dBang, for which we provide a characterization by means of typability and inhabitation. As a second contribution, we validate this notion of meaningfulness twofold: meaningless terms enjoy surface genericity, and the smallest \( \lambda dBang \)-theory \( \mathcal{H}_{dBang} \) obtained by equating all the meaningless terms is consistent. Moreover, we show that \( \mathcal{H}_{dBang} \) admits a unique maximal consistent extension
\( H_{dBang} \) and show that it coincides with the well-known notion of observational equivalence. Last but not least, as a third contribution, we show that the notions of meaningfulness in the literature for \( dBang \) and \( dCBV \) are subsumed by the one proposed here for \( dBang \). We also obtain surface genericity for \( dCBN \) and \( dCBV \) as a consequence of the genericity property for \( dBang \), and relate the theories \( H_{dBang} \) and \( H_{dBang}^* \) (in \( dBang \)) to the corresponding ones in \( dCBN \) and \( dCBV \). Detailed proofs of our results can be found in [50].

**Roadmap.** Section 2 recalls \( dBang \) and its quantitative type system \( B \). Section 3 defines meaningfulness for \( dBang \), and characterizes it in terms of typability and inhabitation in the type system \( B \). Section 4 addresses surface genericity and the construction of the theories \( H_{dBang} \) and \( H_{dBang}^* \), while Section 5 establishes a precise relationship between meaningless and genericity in \( dCBN/dCBV \) and their corresponding notions in \( dBang \). Section 6 discusses future and related work and concludes.

## 2 The dBang-Calculus

### 2.1 Syntax and Operational Semantics

We introduce the syntax of the *distant Bang-calculus* (\( dBang \)) [23, 24]. Given a countably infinite set \( X \) of variables \( x, y, z, \ldots \), the set \( \Lambda \) of *terms* is inductively defined as follows:

\[
(\text{Terms}) \quad t, u, s := x \in X \mid tu \mid \lambda x.t \mid t[u] \mid !t \mid \text{der}(t)
\]

The set \( \Lambda \) includes variables \( x \), abstractions \( \lambda x.t \) and applications \( tu \) (as in the \( \lambda \)-calculus), and three other constructors: a closure \( t[x]u \) representing a pending explicit substitution (ES) \([x]u\) on a term \( t \), a bang \( !t \) to freeze the execution of \( t \), and a dereliction \( \text{der}(t) \) to fire again the frozen term \( t \). The argument of an application \( tu \) (resp. a closure \( t[x]u \)) is the subterm \( u \). From now on, we set \( !t := \lambda z. !z \), \( \Delta t := \lambda x. !z \cdot x \), and \( !\Delta := \Delta !t \).

Abstractions \( \lambda x.t \) and closures \( t[x]u \) bind the variable \( x \) in the term \( t \). Free and bound variables are defined as expected, in particular \( fv(\lambda x.t) := fv(t) \setminus \{x\} \) and \( fv(t[x]u) := fv(u) \cup (fv(t) \setminus \{x\}) \). The usual notion of \( \alpha \)-conversion [20] is extended to \( \Lambda \), and terms are identified up to \( \alpha \)-conversion. We denote by \( t\{x\} \) the usual (capture avoiding) meta-level substitution of the term \( u \) for all free occurrences of the variable \( x \) in the term \( t \).

**List contexts** \( L \), **surface contexts** \( S \) and **full contexts** \( F \), which can be seen as terms containing exactly one hole \( \emptyset \), are inductively defined as follows:

\[
\begin{align*}
\text{(List Contexts)} & \quad L := \emptyset \mid L[x]t \\
\text{(Surface Contexts)} & \quad S := \emptyset \mid S t \mid t S \mid \lambda x.S \mid \text{der}(S) \mid S[x]t \mid t[x]S \\
\text{(Full Contexts)} & \quad F := \emptyset \mid F t \mid t F \mid \lambda x.F \mid \text{der}(F) \mid F[x]t \mid t[x]F \mid !F
\end{align*}
\]

List and surface contexts are special cases of full contexts. The hole can occur anywhere in full contexts, while it is forbidden under \( ! \) in surface contexts. For example, \( y(\lambda x. \emptyset) \) is a surface context hence a full context, while \( (\emptyset)[x]I \) is a full context but not a surface one. We write \( F(t) \) for the term obtained by replacing the hole in \( F \) with the term \( t \).

The following **rewrite rules** are the base components of the reduction system of \( dBang \).

Any term having the shape of the left-hand side of one of these three rules is called a *redex*.

\[
L(\lambda x. t) u \rightarrow_{dB} L(t[x]u) \quad t[x]L(!u) \rightarrow_{!t} L(t[x]u) \quad \text{der}(L(!t)) \rightarrow_{!t} L(t)
\]

Rule \( dB \) (resp. \( !t \)) is assumed to be capture free: no free variable of \( u \) (resp. \( t \)) is captured by the list context \( L \). The rule \( dB \) fires a \( \beta \)-redex and generates an ES. The rule \( !t \) operates a substitution provided its argument is a bang: only bang terms can be erased or duplicated,
and they lose their bang when the substitution is performed. The rule \( \text{d!} \) opens a bang. All these rewrite rules act at a distance \([7, 9, 2]\): the main constructors involved in the rule can be separated by a finite – possibly empty – list context \( L \) of ES. This mechanism unblocks redexes that would otherwise be stuck, e.g. \((\lambda x.x)[y\backslash w] z \rightarrow_{\text{d}B} x[z\backslash y][y\backslash w] \) fires a \( \beta \)-redex where \( L = o[y\backslash w] \) is the list context in between the abstraction \( \lambda x.x \) and the argument \( !z \).

The surface reduction \( \rightarrow_S \) is the surface closure of the three rewrite rules \( \text{dB} \), \( \text{s!} \) and \( \text{d!} \), i.e. \( \rightarrow_S \) only fires redexes in surface contexts (not under bang). Similarly, the full reduction \( \rightarrow_F \) is the full closure of the three rewrite rules \( \text{dB} \), \( \text{s!} \) and \( \text{d!} \), i.e. \( \rightarrow_F \) fires redexes in any full contexts and thus the bang loses its freezing behavior. For example,

\[
(\lambda x.\text{der}(\text{!}x))y \rightarrow_S (\text{!}\text{der}(\text{!}x))[x\backslash y] \rightarrow_S \text{!(der}(\text{!}y)) \rightarrow_F \text{!}y
\]

The first two \( \rightarrow_S \)-steps are \( \rightarrow_F \)-steps too, the last one is not a \( \rightarrow_S \)-step. We denote by \( \rightarrow_S^* \) the reflexive-transitive closure of \( \rightarrow_S \), and similarly for \( \rightarrow_F \). A reduction \( t \rightarrow_R u \) is confluent if for all \( t, u_1, u_2 \) such that \( t \rightarrow_R s \) and \( u_1 \rightarrow_R s \) and \( u_2 \rightarrow_R s \).

**Theorem 1.** The reductions \( \rightarrow_S \) and \( \rightarrow_F \) are confluent.

**Proof.** For \( \rightarrow_S \) see \([23]\), for \( \rightarrow_F \) see \([50]\). □

A term \( t \) is a surface (resp. full) normal form if there is no \( u \) such that \( t \rightarrow_S u \) (resp. \( t \rightarrow_F u \)). A term \( t \) is surface (resp. full) normalizing if \( t \rightarrow_S^* u \) (resp. \( t \rightarrow_F^* u \)) for some surface (resp. full) normal form \( u \). Since \( \rightarrow_S \subseteq \rightarrow_F \), some terms may be surface-normalizing but not full-normalizing, e.g. \( \lambda x.\text{d!}(\text{!}x) \).

As a matter of fact, some ill-formed terms are not redexes but neither represent a desired computation result. They are called clashes and have one of the following forms:

\[
L(\text{!}t) = t[x]L(\lambda x.u) \quad \text{der}(L(\lambda x.t)) = t(L(\lambda x.u)) \quad \text{if } t \neq L(\lambda y.s)
\]

This static notion of clash is lifted to a dynamic level. A term \( t \) is surface (resp. full) clash-free if it does not surface (resp. full) reduce to a term with a clash in surface (resp. full) position, i.e. if there are no surface (resp. full) context \( S \) (resp. \( F \)) and clash \( c \) such that \( t \rightarrow_S^* S(c) \) (resp. \( t \rightarrow_F^* F(c) \)). For example, \( x!(y(\lambda z.z)) \) is surface clash-free but not full clash-free as it has a clash \( y(\lambda z.z) \) under a bang. Both notions are stable under reduction.

Finally, some terms contain neither redexes nor clashes. A surface (resp. full) clash-free normal form is a surface (resp. full) normal form which is also surface (resp. full) clash-free, as e.g. the term \( xx \). These are the results of the computation, and they can even be syntactically characterized by the grammar \( \text{no}_S \) below.

\[
\begin{align*}
\text{ne}_S & ::= x \in \mathcal{X} \mid \text{ne}_S \text{ na}_S \mid \text{der}(\text{ne}_S) \mid \text{ne}_S[x\backslash \text{ne}_S] \\
\text{na}_S & ::= \text{!}t \mid \text{ne}_S \mid \text{na}_S[x\backslash \text{ne}_S] \\
\text{nb}_S & ::= \text{ne}_S \mid \lambda x.\text{na}_S \mid \text{nb}_S[x\backslash \text{ne}_S] \mid \text{no}_S \mid \text{nb}_S
\end{align*}
\]

**Lemma 2** ([23]). Let \( t \in \Lambda_1 \), then \( t \in \text{no}_S \) iff \( t \) is a surface clash-free normal form.

### 2.2 Quantitative Typing System

We present the quantitative typing system \( B \) ([23], based on \([43, 36]\), for \( \text{dBang} \). It contains arrow and intersection types. Intersections are associative, commutative but not idempotent, thus an intersection type is represented by a (possibly empty) finite multiset \([\sigma_i]_{i \in I}\). Given a countably infinite set \( TV \) of type variables \( \alpha, \beta, \gamma, \ldots \), we define by mutual induction:

- **(Types)** \( \sigma, \tau, \rho ::= \alpha \in TV \mid M \mid M \Rightarrow \sigma \\
- **(Multitypes)** \( M, N ::= [\sigma_i]_{i \in I} \) where \( I \) is a finite set
A (type) environment, noted $\Gamma$ or $\Delta$, is a function from variables to multitypes, assigning the empty multitype $[]$ to all variables except a finite number (possibly zero). The empty environment, noted $\emptyset$, maps every variable to $[]$. The domain of $\Gamma$ is $\text{dom}(\Gamma) = \{x \in \mathcal{X} \mid \Gamma(x) \neq []\}$, the image of $\Gamma$ is $\text{im}(\Gamma) = \{\Gamma(x) \mid x \in \text{dom}(\Gamma)\}$. Given the environments $\Gamma$ and $\Delta$, $\Gamma + \Delta$ is the environment mapping $x$ to $\Gamma(x) \uplus \Delta(x)$, where $\uplus$ denotes multiset union; and $+_{i \in I}\Delta_i$ (with $I$ finite) is its $n$-ary extension, in particular $+_{i \in I}\Delta_i = \emptyset$ if $I = \emptyset$. An environment $\Gamma$ is denoted by $x_1:M_1, \ldots, x_n:M_n$ when the $x_i$’s are pairwise distinct variables and $\Gamma(x_i) = M_i$ for all $1 \leq i \leq n$, and $\Gamma(y) = []$ for $y \notin \{x_1, \ldots, x_n\}$.

A typing is a pair $(\Gamma; \sigma)$, where $\Gamma$ is an environment and $\sigma$ is a type. A (typing) judgment is a tuple of the form $\Gamma \vdash t : \sigma$, where $(\Gamma; \sigma)$ is a typing and $t$ is a term (the subject of the judgment). The typing system $B$ for $\text{dBang}$ is defined by the rules in Figure 1. The axiom rule (var) is relevant, i.e. there is no weakening. Rules (abs), (app) and (es) are standard. Rule (bg) has as many premises as elements in the finite (possibly empty) index set $I$, and its conclusion types $t$ with a multitype gathering all the (possibly different) types in the premises typing $t$. In particular, when $I = \emptyset$, the rule has no premises, and it types any term $t$ with $[]$, leaving the subterm $t$ untyped. Rule (der) forces the argument of a dereliction to be typed by a multitype of cardinality 1.

A (type) derivation in system $B$ is a tree obtained by applying the rules in Figure 1. The judgment at the root of the type derivation $\Pi$ is the conclusion of $\Pi$. We write $\Pi \vdash_B \Gamma \vdash t : \sigma$ when $\Pi$ is a derivation in system $B$ with conclusion $\Gamma \vdash t : \sigma$, and $\vdash_B \Gamma \vdash t : \sigma$ if there exists some derivation $\Pi \vdash_B \Gamma \vdash t : \sigma$. A term $t$ is $B$-typable if $\vdash_B \Gamma \vdash t : \sigma$ for some typing $(\Gamma; \sigma)$.

System $B$ enjoys subject reduction and expansion with respect to $\rightarrow_F$, and characterizes surface-normalizing clash-free terms.

\[
\begin{align*}
\frac{}{x : [\sigma] \vdash x : \sigma} \quad & (\text{var}) \\
\frac{\Gamma \vdash t : M \Rightarrow \sigma \quad \Delta \vdash u : M}{\Gamma + \Delta \vdash t u : \sigma} \quad & (\text{app}) \\
\frac{\Gamma, x : M \vdash t : \sigma}{\Gamma \vdash \lambda x.t : M \Rightarrow \sigma} \quad & (\text{abs}) \\
\frac{\Gamma \vdash \lambda x.t : M \Rightarrow \sigma}{\Gamma \vdash \lambda x.t : M \Rightarrow \sigma} \quad & (\text{der})
\end{align*}
\]

\begin{figure}
\caption{Type System $B$ for the $\text{dBang}$-calculus.}
\end{figure}

\textit{Theorem 3} ([23, 13]). Let $t, u \in \Lambda_1$.
\begin{enumerate}
\item If $t \rightarrow_F u$, then for any typing $(\Gamma; \sigma)$, one has $\vdash_B \Gamma \vdash t : \sigma$ if and only if $\vdash_B \Gamma \vdash u : \sigma$.
\item $t$ is $B$-typable if and only if $t$ surface-reduces to a surface clash-free normal form.
\end{enumerate}

3 Meaningfulness = Typability + Inhabitation

In this section, we introduce the notion of meaningfulness for $\text{dBang}$ and we establish a logical characterization of meaningfulness via system $B$. Intuitively, a term $t$ is meaningful if it can be supplied by some arguments (possibly binding some free variables of $t$) so that it reduces to some observable term. In $\text{dBang}$, the observables are the bang terms since they are the only terms enabling substitution to be fired.

\textit{Definition 4}. A term $t$ is $\text{dBang}$-meaningful if there are a testing context $T$ and $u \in \Lambda_1$ such that $T(t) \rightarrow_{S} u$, where testing contexts are defined by the grammar $T := \circ | Ts | (\lambda x.T)s$.\footnote{Thanks to a factorization theorem for $\text{dBang}$ [14], in our definition of $\text{dBang}$-meaningfulness $\rightarrow_{S}$ can equivalently be replaced by $\rightarrow_{F}$. For the same reason, the same remark also applies to Definition 14.} A term $t$ is $\text{dBang}$-meaningless if it is not $\text{dBang}$-meaningful.
Meaningfulness and Genericity in a Subsuming Framework

Figure 2 A type derivation of $xx$ in system $B$.

Figure 3 Inhabitation of $[\alpha] \Rightarrow [\alpha]$ in system $B$.

For example, $I_1$ is $\text{dBang}$-meaningful, take the testing context $T = \varnothing \cup u$. Both $\Omega$ and $x\Omega$ are $\text{dBang}$-meaningless: every testing context they are plugged in cannot erase $\Omega$, which is not normalizing and does not reduce to a bang term. Note that all testing contexts are surface, and that the hole in a testing context is always in the functional position of an application, in particular if the hole is in the scope of some $\lambda$, then this $\lambda$ must be applied.

Readers familiar with the advanced theory of $\lambda$-calculus may wonder about the relevance of our notion of $\text{dBang}$-meaningfulness. In particular, we could have just naively extended the well-known notion of call-by-name solvability: a term $t$ is $\text{dBang}$-solvable if there are a testing context $T$ such that $T(t) \Rightarrow^*_S I_1$. We found at least two reasons to not use $\text{dBang}$-solvability: the first one is that we would lose consistency of the smallest $\lambda_{\text{dBang}}$-theory generated by equating all $\text{dBang}$-unsolvable terms (see discussion after Proposition 8), while the second one is that we would lose genericity (see discussion after Corollary 11).

In an adequate calculus, meaningfulness is usually characterized both operationally (normalizability) and logically (typability): a term is meaningful iff it is normalizing for a well-known notion of call-by-name reducibility: a term is meaningful iff it is typable in a suitable type system. Indeed, we establish a multitype characterization of meaningfulness based on typability and inhabitation in system $B$, similarly to what happens in the $\lambda$-calculus with pairs $[12, 29, 26]$. Intuitively, suppose that a term $t$ is $\text{dBang}$-meaningful, so there is a testing context $T$ such that $T(t)$
reduces to an observable, i.e. a bang, which can be (trivially) typed with the typing \( \langle \emptyset ; \tau \rangle \) in system \( B \). By Theorem 3.1, \( T(t) \) must also be typable by the same typing \( \langle \emptyset ; \tau \rangle \), meaning that \( t \) is typable by some environment \( x_1 : \mathcal{M}_1, \ldots, x_n : \mathcal{M}_n \) and some type \( \mathcal{N}_i \Rightarrow \cdots \Rightarrow \mathcal{N}_n \Rightarrow \tau \), where each of the \( \mathcal{M}_i \)'s and \( \mathcal{N}_i \)'s is inhabited, i.e. there is a term with such a type.

A similar argument holds for other type systems and calculi [29, 26] with their own notions of meaningfulness and observable. The point is to identify the set of types \( \mathcal{T}_{B}^{\text{obs}} \) associated with the observables. In any type system \( S \) whose types are those of Section 2.2, given a set of types \( \mathcal{T}_{B}^{\text{obs}} \) for observable terms, the set of arguments \( \text{args}_S(\sigma) \) of a type \( \sigma \) is the set of multitypes appearing to the left of arrows, until reaching the type of an observable. Formally, if \( \sigma \in \mathcal{T}_{B}^{\text{obs}} \) then \( \text{args}_S(\sigma) = \emptyset \), otherwise \( \text{args}_S(\sigma) := \{ \mathcal{M} \Rightarrow \sigma \} \) and \( \text{args}_S(\mathcal{M}) = \emptyset \). In system \( B \), we set \( \mathcal{T}_{B}^{\text{obs}} := \{ \mathcal{M} | \mathcal{M} \text{ multitype} \} \), because bang terms – the observables in \( \text{dBang} \) – can be only typed by multisets. For example, \( \text{args}_B(\{ \tau \Rightarrow (\mathcal{M} \Rightarrow [\alpha]) \}) = ([\tau], \mathcal{M}) \). The cases of \( \text{dCBN} \) and \( \text{dCBV} \) type systems are discussed in Section 5, this is why our definitions deal with a generic type system \( S \).

\[ \textbf{Definition 5.} \] Let \( S \) be a type system and \( \text{inh}\{\cdot\} \) be a predicate on the types of \( S \). A set \( S \) of types is inhabited, noted \( \text{inh}\{\cdot\}(S) \), if \( \text{inh}\{\cdot\}(\sigma) \) for all \( \sigma \in S \). We write \( \text{inh}\{\cdot\}(\Gamma) \) if \( \text{inh}\{\cdot\}(\text{im}(\Gamma)) \). A typing \( (\Gamma;\sigma) \) or a judgment \( \Gamma \vdash t : \sigma \) is \( S\text{-testable} \) if \( \text{inh}\{\cdot\}(\Gamma) \) and \( \text{inh}\{\cdot\}(\text{args}_S(\sigma)) \). A term \( t \) is \( S\text{-testable} \) if \( \vdash_B (\Gamma \vdash t : \sigma) \) for some \( S\text{-testable} \) typing \( (\Gamma;\sigma) \).

A type \( \sigma \) is inhabited in system \( B \), noted \( \text{inh}_B(\sigma) \), if \( \Pi \vdash_B \emptyset \vdash t : \sigma \) for some \( \Pi \) and \( t \). For instance, in system \( B \), the type \( \emptyset \) is inhabited by any bang, use rule (bg) with no premises; the environment \( \emptyset \) is trivially inhabited; the type \( \{ \alpha \} \Rightarrow [\alpha] \) is inhabited, see Figure 3. The term \( \lambda x . ! x \) is \( B\text{-testable} \) because \( \vdash_B (\emptyset \vdash \lambda x . ! x : [\emptyset] \Rightarrow [\emptyset] \Rightarrow (\emptyset, [\emptyset], [])) \) is \( B\text{-testable} \).

\[ \textbf{Lemma 6.} \] Let \( t \in \mathcal{L}_t \) and \( \mathcal{T} \) be a testing context. If \( \vdash_B (\emptyset \vdash T(t) : [\emptyset]) \), then \( \vdash_B (\Gamma \vdash t : \sigma) \) with \( \text{inh}_B(\sigma) \).

Inhabitation serves as a crucial tool to produce an observable from a typable term. As said before, any multitype assigned to a variable \( x \) by the environment \( \Gamma \) in the derivation of a meaningful term \( t \) should be inhabited. Hence, the environment \( \Gamma \) has to be inhabited. However, relying solely on the inhabitation of \( \Gamma \) is not sufficient, as illustrated by the typable term \( \vdash_B (\emptyset \vdash \lambda x . x x : [\emptyset] \Rightarrow [\emptyset] \Rightarrow [\emptyset] \Rightarrow (\emptyset, [\emptyset], [])) \), which, despite having a trivially inhabited environment, is \( \text{dBang} \)-meaningless. We thus also test the inhabitation of type arguments of the type \( \sigma \) of \( t \). This therefore means that \( B\text{-testability} \) is sufficient to ensure \( \text{dBang} \)-meaningfulness. Surprisingly, this actually provides a characterization of \( \text{dBang} \)-meaningfulness.

\[ \textbf{Theorem 7 (Logical Characterization).} \] Let \( t \in \mathcal{L}_t \) : \( t \) is \( \text{dBang} \)-meaningful iff \( t \) is \( B\text{-testable} \).

Now that we have a logical characterization of \( \text{dBang} \)-meaningfulness, we can reason about the consequences of equating all \( \text{dBang} \)-meaningless terms in a \( \lambda_{\text{dBang}} \)-theory, that is, in a quotient of \( \mathcal{L}_t \) that roughly equates all terms with the same semantics. Formally, a \( \lambda_{\text{dBang}} \)-theory is an equivalence \( \equiv_{\text{dBang}} \) on \( \mathcal{L}_t \) containing \( \rightarrow_{\text{F}} \) and closed under full contexts. Let \( \mathcal{H}_{\text{dBang}} \) (also noted \( \equiv_{\text{H}_{\text{dBang}}} \)) be the smallest \( \lambda_{\text{dBang}} \)-theory equating all \( \text{dBang} \)-meaningless terms. Theorem 7 entails that \( \mathcal{H}_{\text{dBang}} \) is consistent, that is, it does not equate all terms.

\[ \textbf{Proposition 8 (Consistency of } \mathcal{H}_{\text{dBang}} \text{).} \] There exist \( t, u \in \mathcal{L}_t \) such that \( t \not\equiv_{\text{H}_{\text{dBang}}} u \).

Replacing \( \text{dBang} \)-meaningfulness by \( \text{dBang} \)-solvability would result in the loss of consistency. Indeed, take an arbitrary term \( t \in \mathcal{L}_t \) and the two \( \text{dBang} \)-unsolvable terms \( !\Omega \) and \( \Omega \) that the resulting (alternative) theory, written \( \mathcal{H}_{\text{dBang}}^{\text{solv}} \), would equate. By contextuality, we would have \( \langle \lambda x . t \rangle !\Omega \equiv_{\mathcal{H}_{\text{dBang}}^{\text{solv}}} \langle \lambda x . t \rangle \Omega \), and by reduction \( t \equiv_{\mathcal{H}_{\text{dBang}}^{\text{solv}}} \langle \lambda x . t \rangle !\Omega \) (suppose
Meaningfulness and Genericity in a Subsuming Framework

$x \notin \text{fv}(t)$ . Notice that $(\lambda x.t) \Omega$ is also dBang-unsolvable since the term $\Omega$ cannot be erased, thus $(\lambda x.t) \Omega \equiv_{H_{\text{bang}}} \Omega$. By transitivity $t \equiv_{H_{\text{bang}}} (\lambda x.t) \Omega \equiv_{H_{\text{bang}}} (\lambda x.t) \Omega \equiv_{H_{\text{bang}}} \Omega$. Since $t$ is arbitrary, we easily conclude that all terms are equated in $H_{\text{bang}}^0$, making it inconsistent.

We also corroborate our definition of meaningfulness by proving that it fulfills a pair of genericity properties, and show that $H_{\text{bang}}$ admits a unique maximal consistent extension $H_{\text{bang}}^*$ (Section 4). Finally, we also show that dBang-meaningfulness, $H_{\text{bang}}$ and $H_{\text{bang}}^*$ subsume the well-established corresponding notions for dCBN and dCBV (Section 5).

4 Typed and Surface Genericity in dBang

In Section 3, we proved that dBang-meaningfulness is captured by typability in system $B$ with some $B$-testable typing. While this concise characterization formulated as “meaningfulness = typability + inhabitation” [26] provides a high level understanding, its practical manipulation might pose some challenges. Suppose we study some properties of a dBang-meanful term $t$ through the logical characterization (Theorem 7), thus having a type derivation $\Pi \vdash_B \Gamma \vdash t : \sigma$ with $(\Gamma; \sigma)$ $B$-testable. If we proceed by induction on $\Pi$, then there is no guarantee that all the judgments appearing in $\Pi$ have $B$-testable typings as well, which would make the reasoning awkward and the logical characterization of Theorem 7 difficult to exploit. But this is not the case. Upcoming Lemma 9 states that $B$-testability propagates bottom-up: if the conclusion of a derivation $\Pi$ has a $B$-testable typing, then so does every other judgment in $\Pi$.

We write $\Pi \vdash_B \Gamma \vdash t : \sigma$ if $\Pi \vdash_B \Gamma \vdash t : \sigma$ and each judgment in $\Pi$ is $B$-testable, and $\Pi \vdash_B t$ if $\Pi \vdash_B \Gamma \vdash t : \sigma$ holds for some typing $(\Gamma; \sigma)$.

Lemma 9. Let $t \in \Lambda_t$. Then $\Pi \vdash_B \Gamma \vdash t : \sigma$ with $(\Gamma; \sigma)$ $B$-testable iff $\Pi \vdash_B \Gamma \vdash t : \sigma$.

Proof. ($\Rightarrow$): By an induction on $\Pi$. ($\Leftarrow$): Trivial.

We can therefore easily use the logical characterization of dBang-meaningfulness to prove the following first genericity result for dBang: in a dBang-meanful term $s$, a dBang-meanless subterm can be replaced by any term, without impacting the typing of $s$.

Theorem 10 (Typed Genericity). Let $t \in \Lambda_t$ be dBang-meanless and $F$ be a full context. If $\vdash_B \Gamma \vdash F(t) : \sigma$, then $\vdash_B \Gamma \vdash F(u) : \sigma$ for all $u \in \Lambda_t$.

Proof. By induction on $F$, using both Theorem 7 and Lemma 9.

This proof relies on the fact that the dBang-meanless subterm $t$ cannot be explicitly typed in any of the judgments of $\Pi$, as typing $t$ in $B_\alpha$ is equivalent to being dBang-meanful (by Theorem 7 and Lemma 9). Thus, typed genericity fails when weakening the hypothesis from $B$-typability to $B$-typability. For example, given the dBang-meanless term $t = xx$ and the context $F = y \circ F(t)$ is $B$-typable as witnessed by $\vdash_B y : [N \Rightarrow \alpha], x : [M \Rightarrow N, M] \vdash F(t) : \alpha$ – note that the type of $x$ is not inhabited – while $F(\Omega) = y \Omega$ is not $B$-typable.

As a consequence of typed genericity, we can now prove a qualitative surface genericity result, stating that dBang-meanless subterms have no bearing on the significance of dBang-meanful terms: in a dBang-meanful term $s$, a dBang-meanless subterm can be replaced by any term, still keeping $s$ dBang-meanful. We call this genericity result surface, despite it universally quantifies over full contexts, as dBang-meanful is defined in terms of surface reduction. The corresponding results for dCBN and dCBV are also called surface in [15] and light in [10], they are both later generalized to a stratified notion in [15].

Corollary 11 (Qualitative Surface Genericity). Let $F$ be a full context. If $F(t)$ is dBang-meanful for some dBang-meanless $t \in \Lambda_t$, then $F(u)$ is dBang-meanful for all $u \in \Lambda_t$.
Proof. Let $u \in \Lambda$. As $F(t)$ is $dBang$-meaningful, then $\Pi \vdash_{dBang} F(t)$ holds for some $\Pi$ by Theorem 7 and Lemma 9. As $t$ is $dBang$-meaningless, then $\Pi \vdash_{dBang} F(u)$ holds for some $\Pi'$ by Theorem 10, and hence $F(u)$ is $dBang$-meaningful by Theorem 7 and Lemma 9.

As for consistency, surface genericity fails when replacing $dBang$-meaningfulness with $dBang$-solvability. Indeed, consider the full context $F := (\lambda y.x) \circ$ and the two $dBang$-unsolvable terms $t = \Omega$ and $u = \Omega$. One then has that $F(t) = (\lambda y.x) \Omega \to^* x$ is trivially $dBang$-solvable, while $F(u) = (\lambda y.x) \Omega$ is not, as the term $\Omega$ cannot be erased.

Genericity is a sanity check on meaningfulness: it holds only if all $dBang$-meaningless terms are truly meaningless. Still, some truly meaningless terms might be misinterpreted as $dBang$-meaningful. Indeed, when crafting a notion of $dBang$-meaningless that would satisfy genericity, one might not take all $dBang$-meaningless terms as $dBang$-meaningless. This can be formally proved using a property stating that neutral normal forms can create echoes via a single substitution, technical details can be found in [50].

We expect $H_{dBang}^*$ to extend the theory $H_{dBang}$. Moreover, to check that all truly meaningless terms are actually $dBang$-meaningless, we also want this theory to be maximal, meaning that no more terms can additionally be equated without compromising consistency.

\begin{remark}
In $H_{dBang}^*$, a term reducing to a bang will only be equated to terms which also reduce to bangs. This can be formally proved using a property stating that neutral normal forms can create echoes via a single substitution, technical details can be found in [50].
\end{remark}

We expect $H_{dBang}^*$ to extend the theory $H_{dBang}$. Moreover, to check that all truly meaningless terms are actually $dBang$-meaningless, we also want this theory to be maximal, meaning that no more terms can additionally be equated without compromising consistency.

\begin{definition}[Observational Equivalence]
Let $t, u \in \Lambda$, then $t$ and $u$ are open-observational equivalent (resp. observational equivalent), noted $t \equiv^o u$ (resp. $t \equiv_{dBang} u$) if for every full context $F$ (resp. full context $F$ such that $F(t)$ and $F(u)$ are closed), $F(t) \to^* !t'$ for some $t' \in \Lambda$ iff $F(u) \to^* !u'$ for some $u' \in \Lambda$.

Note that, differently from $\equiv$, $\equiv^o$ quantifies over all full contexts and not only on closing full contexts, hence $\equiv^o \subseteq \equiv$. Finally, we now prove that the $dBang$-theory $H_{dBang}^*$ coincides with the observational equivalences $\equiv$ and $\equiv^o$.
\end{definition}

\begin{theorem}
Let $t, u \in \Lambda$, then (1) $t \equiv u$ iff (2) $t \equiv^o u$ iff (3) $t \equiv_{dBang} u$.
\end{theorem}

Proof. Let $t, u \in \Lambda$. Let us show that (3) $\Rightarrow$ (2) $\Rightarrow$ (1) $\Rightarrow$ (3).

$= (3) \Rightarrow (2):$ Let $t \equiv_{dBang} u$. Suppose $F$ is an arbitrary full context such that $F(t) \to^* !t'$ for some $t' \in \Lambda$. Since $H_{dBang}^*$ is a $dBang$-theory (Theorem 13) then it is contextual and hence $F(t) \equiv_{dBang} F(u)$. By Remark 12, $F(u) \to^* !u'$ for some $u' \in \Lambda$. Therefore, $t \equiv u$.

$= (2) \Rightarrow (1):$ Immediate.

$= (1) \Rightarrow (3):$ We can easily prove that $\equiv$ is a consistent $dBang$-theory. As (3) $\Rightarrow$ (2) $\Rightarrow$ (1), we have $\equiv \supseteq H_{dBang}^* \supseteq H_{dBang}$ (the last inclusion holds by Theorem 13). By maximality of $H_{dBang}^*$ (Theorem 13), then necessarily $\equiv \subseteq H_{dBang}^*$. ▲
5 Subsuming CBN and CBV Meaningfulness

In this section we show that the notions of meaningfulness for dCBN and dCBV in the literature [15] are subsumed by the one proposed in Section 3 for dBang. We also deduce surface genericity for dCBN and dCBV as a consequence of surface genericity for dBang.

5.1 dCBN and dCBV Calculi

Both dCBN [7, 8, 1] and dCBV [11] are specified using ES and action at a distance, as explained in Section 2.1 for dBang. Both dCBN and dCBV share the same term syntax. The sets Λ of terms and Υ of values are inductively defined below.

(Terms) \( t, u ::= v \mid t \cdot u \mid t[x\backslash u] \) 
(Values) \( v ::= x \mid \lambda x.t \)

From now on, we set \( I := \lambda x.x \), \( \Delta := \lambda x.xx \), and \( \Omega := \Delta\Delta \). Note that the syntax contains neither der nor \(!\). The distinction between terms and values is irrelevant in dCBN but crucial in dCBV. The two calculi also share the same list contexts \( L_\|, L_q \) and full contexts \( F_R, F_Y \), but use specialized surface contexts \( S_\| \) and \( S_Y \) for dCBN and dCBV, respectively. Again, contexts can be seen as terms with exactly one hole \( ⋄ \) and are inductively defined below.

(List Contexts) \( L_\|, L_q ::= ⋄ \mid L_\| [x\backslash t] \)
(dCBN Surface Contexts) \( S_\| ::= ⋄ \mid S_\| t \mid \lambda x. S_\| \mid S_\| [x\backslash t] \)
(dCBV Surface Contexts) \( S_Y ::= ⋄ \mid S_Y t \mid I S_Y \mid S_Y [x\backslash t] \mid t[x\backslash S_Y] \)
(Full Contexts) \( F_R, F_Y ::= ⋄ \mid F_R t \mid I F_R \mid \lambda x. F_R \mid F_R [x\backslash t] \mid t[x\backslash F_R] \)

We now consider the following rewrite rules:

\( L_\|(\lambda x.t) :\ x \rightarrow_{dB} L_\| (t[x\backslash u]) \quad t[x\backslash u] :\ x \rightarrow_{dB} t\{x\backslash u\} \quad t[x\backslash L_q(v)] :\ x \rightarrow_{dB} L_q(t\{x\backslash v\}) \)

Rules dB and sV are both capture-free: no free variable of \( u \) (resp. \( t \)) is captured by the list context \( L_\| \) (resp. \( L_q \)). The differences between dCBN and dCBV are in the previous notions of surface contexts, and in the rewrite rules. The dCBN surface reduction \( \rightarrow_S \) is the union of the dCBN surface closure of rewrite rules dB and s, while the dCBV surface reduction \( \rightarrow_S^* \) is the union of the dCBV surface closure of the rewrite rules dB and sV. Finally, we use \( \rightarrow_S^* \) (resp. \( \rightarrow_S^* \)) to denote the reflexive-transitive closure of the relation \( \rightarrow_{\|} \) (resp. \( \rightarrow_{S_\|} \)).

Example 16. For example, \( t_0 := (\lambda x. yxx)(\Pi) \rightarrow_S y(\Pi) =: t_1 \) and \( t_0 = (\lambda x. yxx)(\Pi) \rightarrow_S^* (yxx)[x\backslash \Pi] \rightarrow_S (yxx)[x\backslash\{x\backslash x\}] \rightarrow_S (yxx)[x\{x\backslash\{x\}\}] \rightarrow_S^* (yxx)[x\{x\backslash\{x\}\}] \rightarrow_S^* (yxx)[x\{x\backslash\{x\}\}] =: t_2 \).

The dCBN surface reduction is a (non-deterministic diamond variant of) the well-known head reduction [20], and the dCBV surface reduction is the weak reduction not reducing under \( \lambda \)'s.

The quantitative type systems \( N \) for dCBN and \( V \) for dCBV are presented in Figures 4 and 5, respectively. Types and judgments are the same as for system B. A derivation \( \Pi \) in system \( N \) with conclusion \( \Gamma \vdash t : \sigma \) is noted \( \Pi \triangleright N \Gamma \vdash t : \sigma \); we write \( \triangleright N \Gamma \vdash t : \sigma \) if there is a derivation \( \Pi \triangleright N \Gamma \vdash t : \sigma \). We use similar notations for system V.

The salient property of type systems \( N \) and \( V \) is characterizing normalization in dCBN and dCBV, respectively.

Lemma 17 ([23, 24]). Let \( t \in \Lambda \), then:

- \( t \) is dCBN surface normalizing iff it is \( N \)-typable.
- \( t \) is dCBV surface normalizing iff it is \( V \)-typable.
Example 20. Let $t, u \in \Lambda$.
1. If $t \rightarrow_S^* u$ then $t^\circ \rightarrow_S^* u^\circ$.
2. If $t \rightarrow_S^* u$ then $t^\circ \rightarrow_S^* u^\circ$.

Example 20. In Example 16, we showed that $t_0 \rightarrow_S^* t_1$ and $t_0 \rightarrow_S^* t_2$. Recalling Example 18, one has $t_0 \rightarrow_S^* (y!x!x)[x[\![](I!\Pi)]\![]] \rightarrow_S t_1^\circ$ and $t_0 \rightarrow_S (\text{der}(y!x!x)[x[\![](I!\Pi)]\![]] \rightarrow_S (\text{der}(y!x!x)[x[\![](I!\Pi)]\![]] \rightarrow_S t_2^\circ$. 

Both $dCBN$ and $dCBV$ can be embedded into $dBang$ by decorating each term with the $!$ and $\text{der}$ modalities. The embedding for $dCBN$ is standard, while various embeddings for $dCBV$ have been proposed in the literature [44, 57, 58, 46, 23, 24, 14], each with its own strengths and weaknesses. In this work, we use the embeddings from [23, 24] defined below:

$$
x^\circ := x \quad x^y := !x
$$

$(\lambda x.t)^a := \lambda x.t^a \quad (\lambda x.t)^y := !\lambda x.t^y$

$(t u)^a := t^a !u^a \quad (t u)^y := \begin{cases} L(s)^y & \text{if } t^y = L(s) \\
\text{der}(t^y)^y & \text{otherwise}
\end{cases}$

$(t[x\backslash u])^a := t[x\backslash u]^a \quad (t[x\backslash u])^y := t[x\backslash u]^y$

These translations are extended to contexts as expected by setting $\circ^a := \circ$ and $\circ^y := \circ$. 

Example 18. Recalling Example 16, one has $t_0^a = (\lambda x. y!x!x)[x[\![](I!\Pi)]\![]]$, $t_0^y = y!(I!\Pi)(I!\Pi)$, $t_0^a = (\lambda x. (\text{der}(y!x!x))[x[\![](I!\Pi)]\![]])$, and $t_0^y = \text{der}(y!x!x)[x[\![](I!\Pi)]\![]]$. 

Let us give some intuition on these embeddings. In $dCBN$, any argument (right-hand side of application or substitution) can be erased/duplicated, just as bang terms in the $dBang$-calculus, so that arguments must be translated to bang terms. In $dCBV$, only values can be erased/duplicated so that values – and only values – must be translated to bang terms. However, this remark alone is not sufficient to achieve a $dCBV$ embedding enjoying good properties, and in particular to translate $dCBV$-normal forms to $dBang$-normal forms. The translation of applications is precisely designed in order to guarantee this property.

These embeddings preserve reductions, which will allow us to show that meaningfulness if preserved through embedding (Theorems 25 and 30). 

Lemma 19 (Simulation [23, 24]). Let $t, u \in \Lambda$.
1. If $t \rightarrow_S^* u$ then $t^\circ \rightarrow_S^* u^\circ$.
2. If $t \rightarrow_S^* u$ then $t^\circ \rightarrow_S^* u^\circ$. 

Figure 4 Type System $\mathcal{N}$ for the $dCBN$-calculus.

$\begin{array}{ll}
\Gamma, x : M \vdash t : \sigma & \rightarrow \Gamma \vdash \lambda x.t : M \Rightarrow \sigma \\
\Gamma \vdash t : [\tau_i]_{i \in I} \Rightarrow \sigma & \rightarrow \Gamma +_{i \in I} \Delta \vdash t : \sigma \\
\Gamma, x : [\tau_i]_{i \in I} \vdash t : \sigma & \rightarrow \Gamma +_{i \in I} \Delta \vdash t[x\backslash u] : \sigma
\end{array}$

Figure 5 Type System $\mathcal{V}$ for the $dCBV$-calculus.

$\begin{array}{ll}
x : M \vdash x : M \Rightarrow \sigma & \rightarrow \Gamma \vdash [\Rightarrow] \Delta \vdash u : M \\
\Gamma + \Delta \vdash [\Rightarrow] \Delta \vdash u : M & \rightarrow \Gamma, x : [\Rightarrow] \Delta \vdash \sigma \\
\end{array}$
As the dCBV-embedding uses der, some d!-step might be needed in the simulation process.

These embeddings also preserve typing, which will make possible to project dBang meaningfulness and surface genericity onto dCBN and dCBV. More precisely, the two embeddings are proven to be sound and complete with respect to system B.

\[ \text{Proposition 21 ([23, 24])} \] Let \( t \in \Lambda \) and \( (\Gamma; \sigma) \) be a typing.
1. One has \( \triangleright_{\mathcal{N}} \Gamma \vdash t : \sigma \) if and only if \( \triangleright_{\mathcal{B}} \Gamma \vdash t^0 : \sigma \).
2. One has \( \triangleright_{\mathcal{V}} \Gamma \vdash t : \sigma \) if and only if \( \triangleright_{\mathcal{B}} \Gamma \vdash t^0 : \sigma \).

A straightforward corollary is that dCBN and dCBV inhabitation properties are well subsumed in dBang, as illustrated in [13]. In simpler words, any type inhabited in dCBN (resp. dCBV) is also inhabited in dBang. As expected, the converse is false.

In dCBV and dBang, typing an arbitrary term and typing an argument is similar, as it can be seen in the right premise \( \Delta \vdash u : \mathcal{M} \) of the typing rules (app) and (es) of systems \( \mathcal{N} \) and \( \mathcal{B} \). This is not the case in dCBN, as the right premise of the (app) and (es) rules of system \( \mathcal{N} \) requires, not a single derivation, but a finite set \( (\Delta_i \vdash u : \tau_i)_{i \in I} \) of typing derivation for the same term \( u \). In the logical characterization (Theorem 7), we check that arguments of a given type can be inhabited. We therefore need to reflect the typability of arguments – rather than typability of arbitrary terms – in the definition of dCBN inhabitation.

\[ \text{Definition 22.} \] In system \( \mathcal{N} \), a non-multitype \( \sigma \) is inhabited, noted \( \text{inh}_{\mathcal{N}}(\sigma) \), if \( \Pi \triangleright_{\mathcal{N}} \emptyset \vdash t : \sigma \) for some \( \Pi \) and \( t \). A multitype \( [\tau_i]_{i \in I} \) is inhabited in system \( \mathcal{N} \), noted \( \text{inh}_{\mathcal{N}}([\tau_i]_{i \in I}) \) if there exists \( u \in \Lambda \) such that for each \( i \in I \), \( \triangleright_{\mathcal{N}} \emptyset \vdash u : \tau_i \).

In system \( \mathcal{V} \), a type \( \sigma \) is inhabited, noted \( \text{inh}_{\mathcal{V}}(\sigma) \), if \( \Pi \triangleright_{\mathcal{V}} \emptyset \vdash t : \sigma \) for some \( \Pi \) and \( t \).

In particular, the type \([ \_ ]\) is inhabited in both dCBN and dCBV (i.e. \( \text{inh}_{\mathcal{N}}([\_]) \) and \( \text{inh}_{\mathcal{V}}([\_]) \)). Similarly, the environment \( \emptyset \) is also trivially inhabited in both (i.e. \( \text{inh}_{\mathcal{N}}(\emptyset) \) and \( \text{inh}_{\mathcal{V}}(\emptyset) \)).

### 5.2 dCBN Meaningfulness and Surface Genericity

In this subsection, our attention shifts towards the dCBN-calculus, where we show that its notion of meaningfulness is subsumed by that of dBang. This observation enables us to project the surface genericity theorem accordingly. We start by introducing dCBN-meaningfulness.

\[ \text{Definition 23.} \] A term \( t \in \Lambda \) is dCBN-meaningful if there is a testing context \( T_b \) such that \( T_b(t) \rightarrow^{*}_{\text{inh}} \emptyset \), where testing contexts are defined by \( T_b := \emptyset | T_b u | (\lambda x. T_b) u \).

For example \( t = x(\lambda y. \Omega) \) is dCBN-meaningful as \( T_b(t) \rightarrow^{*}_{\text{inh}} \emptyset \) for \( T_b = (\lambda x. c)(\lambda z. 1) \), while \( \Omega \) and \( \lambda x. \Omega \) are dCBN-meaningless as for whatever testing context \( \Omega \) and \( \lambda x. \Omega \) are plugged into, \( \emptyset \) will not be erased. According to the definition of dCBN-meaningfulness, it is natural to define the types of observable terms in dCBN as the identity types, i.e. \( T_b^{\text{obs}} := \{ [\sigma] \Rightarrow [\sigma] \mid \sigma \text{ type} \} \).

Unlike dBang, dCBN-meaningfulness can be characterized both operationally, through surface normalizability, and logically, through typability in system \( \mathcal{N} \). Moreover, this logical characterization turns out to be equivalent to \( \mathcal{N} \)-testability, meaning that dCBN-meaningfulness can also be characterized via typability and inhabitation, as already observed in [27].

\[ \text{2} \] Usually, dCBN-meaningfulness (aka solvability) is defined using contexts of the form \((\lambda x_1 ... x_m. c)Y_1 ... Y_n \) \((m, n \geq 0) \) [19, 20, 71], instead of testing contexts. It is easy to check that the two definitions are equivalent in dCBN. The benefit of our definition is that the same testing contexts are also used to define dCBV-meaningfulness (Section 5.3).
As for observational equivalence in $dCBN$, we can now project surface genericity from $dBang$. Let $H$ be $dCBN$ under full contexts. Let $dBang$ is $dCBN$.

Proof.

We present here an operational proof. Let $t$ be $dCBN$-meaningful, thus $T_B(t) \to^*_B I$ for some testing context $T_B$. By induction on $T_B$, one has that $(T_B(t))^p = T^*_B(t^p)$. By simulation (Lemma 19.1), one deduces that $T^*_B(t^p) \to^*_B \lambda x.x$ thus $T^*_B(t^p) !y \to^*_B (\lambda x.x)!y \to^*_B !y$. Notice that $T^*_B !y$ is a $dBang$-testing context. We thus conclude that $t^p$ is $dBang$-meaningful.

$(\Leftarrow)$ Let $t^p$ be $dBang$-meaningful, then using Theorem 7, it is $B$-testable and thus $B$-typable.

By Proposition 21.1, $t$ is $\Lambda$-typable and hence $t$ is $dCBN$-meaningful by Theorem 24.

Observe for example that $1$ and $\Gamma^0 = I_1$ are both $dCBN/dBang$-meaningful while $\Omega$ and $\Omega^0 = \Omega$ are both $dCBN/dBang$-meaningless.

Theorem 25 states that $dCBN$-meaningfulness precisely aligns with $dBang$-meaningfulness on its image via $\pi$, strengthening the idea that these two notions are adequately chosen. Thanks to Theorem 25, we can now project surface genericity from $dBang$ to $dCBN$.

Theorem 24 (Characterizations of $dCBN$-Meaningfulness [29, 27, 23]). Let $t \in \Lambda$.

1. (Operational) $t$ is $dCBN$-meaningful iff $t$ is $dCBN$ surface-normalizing.
2. (Logical) $(1) t$ is $dCBN$-meaningful iff $N$-typable and hence $t$ is $\Lambda$-typable. $(2)$ $t$ is $dCBN$-meaningful iff $\Lambda$-typable, and thus $dBang$-meaningful.

We now discuss some crucial consequences of our previous results, captured by the use of $\Lambda_{dCBN}$-theories. A $\Lambda_{dCBN}$-theory is an equivalence $\equiv$ on $\Lambda$ containing $\to_B$, and closed under full contexts. Let $H_{\Lambda_{dCBN}}$ (also noted $\equiv_{\Lambda_{dCBN}}$) be the smallest $\Lambda_{dCBN}$-theory equating all $dCBN$-meaningless terms, and let $H^*_dCBN$ be defined as follows:

$$H^*_dCBN := \{(t, u) \mid \forall F_B \text{ full context, } F_B(t) \text{ dCBN-meaningful } \equiv F(u) \text{ dBang-meaningful}\}$$

As for $dBang$, $H^*_dBang$ is the maximal consistent $\Lambda_{dCBN}$-theory containing $H_{\Lambda_{dCBN}}$ and it coincides with observational equivalence in $dCBN$ (see [15]). Thanks to the preservation of meaningfulness via the $dCBN$-embedding, $^p$ (Theorem 25), we can actually relate the theories $H^*_dBang$ and $H^*_dCBN$ (in $dBang$) to the corresponding ones in $dCBN$, that is, $H_{\Lambda_{dCBN}}$ and $H^*_\Lambda_{dCBN}$ respectively.

Theorem 27. Let $t, u \in \Lambda$.

1. If $t \equiv_{\Lambda_{dCBN}} u$ then $t^p \equiv_{\Lambda_{dBang}} u^p$.
2. If $t^p \equiv_{\Lambda_{dBang}} u^p$ then $t \equiv_{\Lambda_{dCBN}} u$.

Proof. Let $t \in \Lambda$ be $dCBN$-meaningless and $F_B$ be a full context. Suppose that $F_B(t^p)$ is $dCBN$-meaningful: by Theorem 25 and since $(F_B(t))^p = F_B^p(t^p)$ (simple induction on $F_B$), $F_B^p(t^p) \equiv dBang$-meaningful, and $t^p$ is $dBang$-meaningless. By Corollary 11, for any $u \in \Lambda$, $F_B^p(u^p) \equiv dBang$-meaningful, and hence $F_B(u)$ is $dCBN$-meaningful using Theorem 25.

$H^*_dBang$ := $\{t \mid \exists u \in \Lambda, t^p \equiv dBang u \Rightarrow H^*_dBang \}$

$H^*_dCBN$ := $\{t \mid \exists u \in \Lambda, t \equiv dCBN u \Rightarrow H^*_dCBN \}$
Meaningfulness and Genericity in a Subsuming Framework

Proof.
1. Immediate consequence of Theorem 25 and Lemma 19.1.
2. Let \( t, u \in \Lambda \) such that \( t^\eta \equiv_{\mathrm{dCBang}} u^\eta \). Let \( F_\eta \) be a full context and suppose that \( F_\eta (t) \) is \( \mathrm{dCBN} \)-meaningful. Using Theorem 25, one deduces that \( (F_\eta (t))^n = F_\eta^n (t^\eta) \) is \( \mathrm{dBang} \)-meaningful.

Since \( t^n \equiv_{\mathrm{dCBang}} u^n \), one has that \( F_\eta^n (u^n) = (F_\eta (u))^n \) is \( \mathrm{dBang} \)-meaningful. Using Theorem 25, one concludes that \( F_\eta (u) \) is \( \mathrm{dCBN} \)-meaningful and therefore \( t \equiv_{\mathrm{H}^*_{\mathrm{dCBas}}} u \).

We strongly conjecture that the converse of Theorem 27.1 also holds. Perhaps unexpectedly, the converse of Theorem 27.2 is actually false. Indeed, \( \eta \)-expansion is included in \( H^*_{\mathrm{dCBas}} \) (see [20]) but not in \( H^*_{\mathrm{dCBang}} \). Thus \( x \equiv_{\mathrm{H}^*_{\mathrm{dCBas}}} \lambda y.x y \) but \( x^n \neq_{\mathrm{H}^*_{\mathrm{dCBas}}} \lambda y.x y = (\lambda y.x y)^n \). The context \( F = \circ [x]\!\!/w \) separates \( x \) and \( \lambda y.x y \). However, through Theorem 15, this phenomenon is not so surprising as it tells us that the \( \mathrm{dCBN} \) observational equivalence does not coincide with \( \mathrm{dBang} \) observational equivalence on the image of \( n \), since \( \mathrm{dBang} \) is a finer language than \( \mathrm{dCBN} \), with more contexts to separate terms operationally.

5.3 \( \mathrm{dCBV} \)-Meaningfulness and Surface Genericity

We now move to the \( \mathrm{dCBV} \)-calculus, where we show that its notion of meaningfulness is subsumed by that of the \( \mathrm{dBang} \)-calculus, and then project surface genericity theorem accordingly.

Adapting meaningfulness from \( \mathrm{dCBN} \) to \( \mathrm{dCBV} \) by replacing \( \mathrm{dCBN} \)-reduction with \( \mathrm{dCBV} \)-reduction may seem initially promising. This notion, known as \( \mathrm{dCBV} \)-solvability, has appealing properties [64, 71, 11, 30, 47, 6]. Unfortunately, Accattoli and Guerrieri showed that generivity fails in such setting [6], and that equating unsolvable terms yields an inconsistent theory (see e.g. [6]). Consequently, \( \mathrm{dCBV} \)-meaningfulness cannot be identified with \( \mathrm{dCBV} \)-solvability. Identifying appropriate notions to capture \( \mathrm{dCBV} \) meaningful \( \lambda \)-terms and formally validating these notions has been a longstanding and challenging open question.

Paolini and Ronchi Della Rocca [64, 71] introduced the notion of \( \text{potentially valuability for CBV} \), also studied in [63, 11, 30, 42] and renamed \( (\mathrm{dCBV} \text{ scrutiny}) \) in [6]. This notion, which we introduce below, proves to be suitable \( \mathrm{dCBV} \)-meaningfulness. Notably, it aligns seamlessly with \( \mathrm{dBang} \)-meaningfulness through the \( \mathrm{dCBV} \)-embedding and thus enjoys a genericity theorem.

**Definition 28.** A term \( t \in \Lambda \) is \( \mathrm{dCBV} \)-meaningful if there exists a testing context \( T_V \) and a value \( v \) such that \( T_V (t) \rightarrow^*_V v \), where testing contexts are defined by \( T_V := \circ | T_V u | (\lambda x. T_V) u \).

For example \( t = x (\lambda y.z) \) is \( \mathrm{dCBV} \)-meaningful as \( T_V (t) \rightarrow^*_V \lambda y.z \) for \( T_V = (\lambda x. \circ) (\lambda z. z) \), while \( \Omega \) and \( x \Omega \) are \( \mathrm{dCBV} \)-meaningless as for whatever testing context \( \Omega \) and \( x \Omega \) are plugged into, \( \Omega \) will not be erased. Note that the set of testing contexts is the same as those of \( \mathrm{dCBN} \).

Notice that this definition closely mirrors that of \( \mathrm{dBang} \)-meaningfulness, with the primary difference being the replacement of \( \mathrm{dBang} \) values for those of \( \mathrm{dCBV} \). Since values are typed with multitypes, it is natural to take them as types of the observable terms in \( \mathrm{dCBV} \) (i.e. \( T^\text{obs}_V := \{ M | M \text{ multitype} \} \)). Consequently, and thanks to the preservation of typing (Proposition 21.2), one easily shows that testability is preserved through the \( \mathrm{dCBV} \) translation: if a term \( t \) is \( V \)-testable, then its image \( t^\eta \) is \( B \)-testable.

As in \( \mathrm{dCBN} \) and unlike \( \mathrm{dBang} \), \( \mathrm{dCBV} \)-meaningfulness can actually be characterized both operationally, through surface normalizability, and logically, through typability in system \( V \). Moreover, the logical characterization turns out to be equivalent to \( V \)-testability, meaning that \( \mathrm{dCBV} \)-meaningfulness is also characterized by means of typability and inhabitation.

**Theorem 29** (Characterizations of \( \mathrm{dCBV} \)-Meaningfulness [11, 6, 23]). Let \( t \in \Lambda \).

1. **(Operational)** \( t \) is \( \mathrm{dCBV} \)-meaningful iff \( t \) is \( \mathrm{dCBV} \) surface-normalizing.

2. **(Logical)** (1) \( t \) is \( \mathrm{dCBV} \)-meaningful iff (2) \( t \) is \( V \)-typable iff (3) \( t \) is \( V \)-testable.
The notion of observable aligns in \texttt{dCBV} and \texttt{dBang}, at least from the type perspective. This yields a simple fully semantical proof of the preservation of \texttt{dCBV}-meaningfulness.

\begin{lemma}
Let $t \in \Lambda$, then $t$ is \texttt{dCBV}-meaningful iff $t^\nu$ is \texttt{dBang}-meaningful.
\end{lemma}

\begin{proof}
$(\Rightarrow)$ We present here a semantical proof. Let $t$ be \texttt{dCBV}-meaningful, then using Theorem 29, one has that $t$ is $\nu$-testable thus, by preservation of testability, $t^\nu$ is $\mathcal{B}$-testable and one concludes that $t^\nu$ is \texttt{dBang}-meaningful according to Theorem 7.

$(\Leftarrow)$ Let $t^\nu$ be \texttt{dBang}-meaningful, then using Theorem 7, it is $\mathcal{B}$-testable thus $\mathcal{B}$-typable. By Proposition 21.2, $t$ is $\nu$-typable and thus $t$ is \texttt{dCBV}-meaningful by Theorem 29.
\end{proof}

Observe for example that $\Omega$ and $\Omega^\nu = !\Omega^\nu$ are both \texttt{dCBV/dBang}-meaningful while $\Omega$ and $\Omega^\nu = \Omega^\nu$ are both \texttt{dCBV/dBang}-meaningless.

Theorem 30 states that \texttt{dCBV}-meaningfulness precisely aligns with \texttt{dBang}-meaningfulness on its image, strengthening the idea that these two notions are adequately chosen. Thanks to Theorem 30, we can now project surface genericity from \texttt{dBang} to \texttt{dCBV}.

\begin{lemma}[\texttt{dCBV} Qualitative Surface Genericity]
Let $F_\nu$ be a full context. If $F_\nu(t)$ is \texttt{dCBV}-meaningful for some \texttt{dCBV}-meaningless $t \in \Lambda$, then $F_\nu(u)$ is \texttt{dCBV}-meaningful for every $u \in \Lambda$.
\end{lemma}

\begin{proof}
Let $t \in \Lambda$ be \texttt{dCBV}-meaningless and $F_\nu$ be a full context. Suppose that $F_\nu(t)$ is \texttt{dCBV}-meaningful, then using Theorem 30, $(F_\nu(t))^\nu$ is \texttt{dBang}-meaningful, and $t^\nu$ is \texttt{dBang}-meaningless.

By induction on $F_\nu$, $(F_\nu(t))^\nu = F_\nu^\nu(t^\nu)$ thus $F_\nu^\nu(t^\nu)$ is \texttt{dBang}-meaningful. By Corollary 11, for any $u \in \Lambda$, $F_\nu^\nu(u^\nu)$ is \texttt{dBang}-meaningful. So, by typing preservation (Proposition 21.2), $(F_\nu(u))^\nu$ is \texttt{dBang}-meaningful, and hence $F_\nu(u)$ is \texttt{dCBV}-meaningful using Theorem 30.
\end{proof}

We now discuss some crucial consequences of our previous results, captured by the use of $\lambda_{\texttt{dCBV}}$-theories. A $\lambda_{\texttt{dCBV}}$-theory is an equivalence $\equiv$ on $\Lambda$ containing $\to_F$ and closed under full contexts. Let $H_{\texttt{dCBV}}$ (also noted $\equiv_{\texttt{dCBV}}$) be the smallest $\lambda_{\texttt{dCBV}}$-theory equating all \texttt{dCBV}-meaningless terms, and let $H^*_\texttt{dCBV}$ be defined as follows:

$$H^*_\texttt{dCBV} := \{ (t, u) \mid \forall F_\nu \text{ full context, } F_\nu(t) \texttt{dCBV}-meaningful \Leftrightarrow F_\nu(u) \texttt{dBang}-meaningful \}$$

As for \texttt{dBang} and \texttt{dCBN}, $H^*_\texttt{dCBV}$ is the maximal consistent $\lambda_{\texttt{dCBV}}$-theory containing $H_{\texttt{dCBV}}$ and coincides with observational equivalence in \texttt{dCBV} (see [15]). Again, thanks to the preservation of meaningfulness via the \texttt{dCBV}-embedding $\cdot^\nu$ (Theorem 30), we can relate the theories $H_{\texttt{dBang}}$ and $H^\nu_{\texttt{dBang}}$ (in \texttt{dBang}) to the corresponding ones in \texttt{dCBV}, that is, $H_{\texttt{dCBV}}$ and $H^*_\texttt{dCBV}$.

\begin{lemma}
Let $t$, $u \in \Lambda$.
1. If $t \equiv_{H_{\texttt{dCBV}}} u$ then $t^\nu \equiv_{H^\nu_{\texttt{dCBV}}} u^\nu$.
2. If $t^\nu \equiv_{H^\nu_{\texttt{dCBV}}} u^\nu$ then $t \equiv_{H_{\texttt{dCBV}}} u$.
\end{lemma}

\begin{proof}
1. Immediate consequence of Theorem 30 and Lemma 19.2

2. Let $t, u \in \Lambda$ such that $t^\nu \equiv_{H^\nu_{\texttt{dCBV}}} u^\nu$. Let $F_\nu$ be a full context and suppose that $F_\nu(t)$ is \texttt{dCBV}-meaningful. By Theorem 30, $(F_\nu(t))^\nu$ is \texttt{dBang}-meaningful, and by Theorem 7 $(F_\nu(t))^\nu$ is $\mathcal{B}$-testable. As $(F_\nu(t))^\nu \to_F F_\nu^\nu(t^\nu)$ (see [14]) and typing is preserved by reduction (Theorem 3.1), we deduce that $F_\nu^\nu(t^\nu)$ is also \texttt{dBang}-meaningful. Since $t^\nu \equiv_{H^\nu_{\texttt{dCBV}}} u^\nu$, one has that $F_\nu^\nu(t^\nu)$ is \texttt{dBang}-meaningful and so is $(F_\nu(u))^\nu$, thanks to Theorem 3.1 and since $(F_\nu(u))^\nu \to_F F_\nu^\nu(u^\nu)$. By Theorem 30, $F_\nu(u)$ is \texttt{dCBV}-meaningful and hence $t \equiv_{H_{\texttt{dCBV}}} u$.
\end{proof}
As in the dCBN-case, we strongly conjecture that the converse of Theorem 32.1 also holds. As in the dCBN-case again, the converse of Theorem 32.2 is actually false. Indeed, the \( \eta \)-expansion is included in \( H_{\text{dCBV}} \) (see [66, 53, 22, 3]) but not in \( H_{\text{dBang}} \), i.e. \( x \equiv_{H_{\text{dCBV}}} \lambda_2.x.y \) but \( x'^\prime = !_x \not\equiv_{H_{\text{dBang}}} \lambda_2.(x'.y)y = (\lambda_2.x.y)y' \): the context \( F = \text{der}(\phi)[x'\parallel !_w] \) separates the two.

Again, from the viewpoint of Theorem 15, this phenomenon is not so surprising as it tells us that the dBang and dBang observational equivalence does not coincide on the image of \( \cdot' \), since dBang is a finer language than dCBV, with more contexts to separate terms operationally.

6 Conclusion and Future Work

We defined a notion of meaningful term, in a unifying well-established framework dBang that is able to capture both dCBN and dCBV calculi. We validated this notion of meaningfulness by providing a (high-level) characterization based on both typability and inhabitation, and showing a (surface) genericity result. All these results in dBang are perfectly analogous to well-known results for dCBN and dCBV [15]. Furthermore, both meaningfulness and genericity in dBang are shown to capture their respective notions in dCBN and dCBV. This suggests that there is a sort of canonicity in our definition of dBang-meaningfulness.

It is natural to wonder why this work is not conducted on the usual CBV and CBN calculi but rather their distant version dCBN and dCBV, which make use of explicit substitutions. The main reason is the non-adequacy of Plotkin’s CBV calculus [66], meaning that some observational equivalent terms have different operational behaviors. Indeed, take the term \( t := (\lambda x.\Delta)(yy)\Delta \) which is observationally equivalent to the prototypical diverging term \( \Omega \). Since \( \lambda x.\Delta \) is applied to \( yy \) – which is not a value and cannot reduce to a value – it makes \( t \) a normal form in Plotkin’s CBV. This mismatch complicates the study of dCBV-meaningfulness. Notice that this issue is solved in dCBV as the term \( t \) now diverges: \( t \rightarrow \text{S}_n \Delta[x\parallel yy]\Delta \rightarrow \text{S}_n (zz) [\epsilon' \Delta] [x\parallel yy] \rightarrow \text{S}_n \Omega[x\parallel yy] \rightarrow \text{S}_n \ldots \), as expected. Furthermore, the observational equivalences generated by Plotkin’s CBV and dBang coincide, making the calculus switch harmless. Since adequacy for CBV is recovered thanks to ES and action at a distance, it is then natural to adopt a similar specification for CBN, knowing that standard CBN \( \lambda \)-calculus and dCBN are operationally and semantically equivalent.

While the logical characterization of meaningfulness for dBang (Theorem 7) requires additional hypotheses (typability and inhabitation) compared to those for dCBN (Theorem 24) and dCBV (Theorem 29), which only require typability, this dissimilarity should not be mistakenly interpreted as a weakness of our approach.

Firstly, the inhabitation condition becomes trivial in the case of dCBN and dCBV, as testability and typability coincide in both cases. Consequently, our approach to meaningfulness for dBang clearly provides a conservative extension of those for dCBN and dCBV.

Secondly, the use of distinct term constructors to specify data that cannot be intermingled seems unavoidable to embed both call-by-name and call-by-value calling paradigms within a single unifying framework. In the case of dBang, a clear distinction must be made between functions (represented by abstractions) and duplicable terms (represented by bang). This syntactic distinction, absent in both call-by-name and call-by-value, results in a unifying framework containing (at least) two built-in primitives that capture incompatible data. Then, the use of intersection types enable, in principle, such mismatch to exists even though a term cannot actually be a bang and a function at the same time. To address this issue, it may seem tempting to explore some syntactical restriction of intersection type systems such as uniformity [65] or compatibility [29], but both these cases result in a loss of completeness.
Finally, the characterization of meaningfulness through typability and inhabitation in a language equipped with incompatible data structures was initially studied in [29, 26], in the context of a λ-calculus with pair patterns. Clearly, functions cannot be pattern-matched by pair patterns, and pairs cannot be applied to arguments.

Besides that, several questions remain to be explored. First of all, we aim to show that our notion of meaningfulness for dBang allows us to prove a full genericity result in dBang in Barendregt’s sense as mentioned in Section 1 (meaningless subterms are computationally irrelevant in the evaluation of full normalizing terms). A notion of stratified reduction, a finer operational semantics generalizing surface reduction to different levels, has been recently defined for dCBN and dCBV [15]. Stratified reduction is a key tool to show a full genericity result for both dCBN and dCBV. We plan to transfer these techniques to the more general framework of dBang, so that full genericity for dCBN and dCBV can be simply obtained by projecting the more general notion of full genericity for dBang via CBN/CBV translations.

It has been observed [69] that dBang can be embedded in this pattern language. Nevertheless, these two languages are not semantically equivalent, as dBang allows only duplication of values (bang terms), whereas the pattern language allows duplication of arbitrary terms.

We also plan to further study the properties of the smallest theory HDbang generated by equating all the meaningful terms in dBang. We strongly conjecture that HDbang restricted to the image of the embedding $\cdot$ (resp. $\cdot_\nu$) is equivalent to HDcbn in dCBN (resp. HDcbv in dCBV).

We would like to extend our study to other natural objects in the theory of programming, such as Böhm trees for dBang and their related theorems (e.g. approximation and separability). Böhm trees for dBang are expected to encompass both dCBN [20] and dCBV [49] ones.

Unifying frameworks such as dBang should also provide other general results for dCBN and dCBV, such as standardization, separability, etc. All this is left to future work. Finally, a more ambitious goal would be to generalize these results to models of computations with effects, such as global memory, non-determinism, exceptions, etc. This would approach our study on dBang to a more general unifying framework such as call-by-push-value [54, 55].

References


Meaningfulness and Genericity in a Subsuming Framework


Meaningfulness and Genericity in a Subsuming Framework


69 Miguel Ramos. Embedding the distant bang calculus with pattern-matching primitives, 2024. Personal Communication.


