Interaction Equivalence

Beniamino Accattoli, Adrienne Lancelot, Giulio Manzonetto, and Gabriele Vanoni

A cornerstone of the theory of programming languages is the acceptance of contextual equivalence as a meaningful notion—if not as *the* notion—of program equivalence. Introduced by Morris to study the untyped λ-calculus [13], contextual equivalence has the advantage that its definition is almost language-agnostic. Given a language $\mathcal{L}$, one simply needs the notion of contexts C of $\mathcal{L}$, usually defined as terms with a single occurrence of a special additional constant $\langle\cdot\rangle$ (the *hole* of the context), and a chosen predicate $\Downarrow$ of *observation* for $\mathcal{L}$, typically some notion of termination, which is the only language-specific ingredient. Then, contextual equivalence is defined as:

$$t \equiv^{\mathrm{ctx}} u \quad \text{if for all contexts } C \,.\, [\; C\langle t\rangle \Downarrow \quad \Leftrightarrow \quad C\langle u\rangle \Downarrow \;]$$

Contextual equivalence embodies a *black box* behavioral principle with respect to termination. Nothing is said about the structure of t and u, it is only prescribed that any use of t in a larger program can be replaced by u (and vice-versa) without affecting the observables. One of the most studied contextual equivalences is *head contextual equivalence* $\equiv_{\mathtt{h}}^{\mathrm{ctx}}$ for the untyped λ-calculus [4], where one observes termination of head reduction $\to_{\mathtt{h}}$. Another contextual equivalence that was studied at length is the one of Plotkin's PCF [16], where the observation is termination on the same value of ground type, typically a natural number. Both played crucial roles in the historical development of denotational semantics of programming languages.

Contextual equivalence is also relevant in more applied settings. The typical example being program transformations at work in compilers, for which contextual equivalence guarantees a strong form of soundness [15].

Equational Contextual Equivalence. In order for contextual equivalence $\equiv^{\mathrm{ctx}}$ to be a proper semantical notion, one actually has to show that it is an *equational theory* for $\mathcal{L}$. This means that the following two properties must be satisfied:

1. *Compatibility*: $\equiv^{\mathrm{ctx}}$ is stable under context closure, that is, $t \equiv^{\mathrm{ctx}} u$ implies $C\langle t\rangle \equiv^{\mathrm{ctx}} C\langle u\rangle$ for all contexts C, and;
2. *Invariance*: $\equiv^{\mathrm{ctx}}$ includes β-conversion $=_\beta$, that is, if $t =_\beta u$ then $t \equiv^{\mathrm{ctx}} u$.

While compatibility holds by definition, invariance is not immediate, because of the universal quantification on contexts in $\equiv^{\mathrm{ctx}}$, which is notoriously hard to manage. For head contextual equivalence $\equiv_{\mathtt{h}}^{\mathrm{ctx}}$, for instance, a rewriting-based proof of the invariance property requires to prove the confluence of β-reduction and various non-trivial properties of head reduction.

Termination vs Time Cost, Equationally. It is natural to wonder whether contextual equivalence can be refined by replacing the termination predicate $\Downarrow$ with a finer one $\Downarrow^k$ indicating termination in k evaluation steps. Intuitively, the number of evaluation steps is taken as a time cost model. With the compiler analogy in mind, it is indeed natural to ask that a program transformation preserves the time behavior. This refinement is exactly Sands' notion of *cost equivalence*, the symmetric form of his more famous *improvement preorder* [19,18,17]:

$$t \equiv^{\text{cost}} u \text{ if for all contexts } C, \exists k \geq 0 \,.\, [\; C\langle t\rangle \Downarrow^k \quad \Leftrightarrow \quad C\langle u\rangle \Downarrow^k \;]$$

Cost equivalence falls short, however, of being a refinement of contextual equivalence. The price to pay for the added quantitative information is that it can no longer be seen as a semantics as it is *not* an equational theory. The reason is both simple and deep. It is simple because (in, say, the λ-calculus) if $t \to_\beta u$, then $C\langle u\rangle$ might take one step less to terminate than $C\langle t\rangle$, so β-conversion $=_\beta$ cannot be included in cost equivalence $\equiv^{\text{cost}}$, breaking the invariance of equational theories.

Internal and External. The issue is deep as it reflects a more general tension between quantitative intensional properties such as time cost, which by their nature cannot be invariant under evaluation, and semantical notions such as equational theories, that are expected to hide the computational process. More generally, one might identify two perspectives on programs. The *internal view* studies programs *in isolation*, and it is concerned with qualitative properties such as, *e.g.*, termination, confluence, productivity, or quantitative properties such as time or space cost. The *external view*, instead, studies how programs *interact* with other programs, as it is the case for contextual equivalences, labelled transition systems (LTSs), or process calculi. Usually, the external view hides the internal dynamics of programs, for instance via the invariance requirement for contextual equivalence, or via the silent τ transitions of LTSs.

The mentioned issue with cost equivalence is then an instance of a more general question: is there a way of analyzing quantitative properties externally, without interference from the hidden internal dynamics? Concretely, is there a way of measuring the amount of *interactions* between a term and its context *modulo* the internal dynamics of the term and of the context?

Interaction Equivalence and the Checkers Calculus. In this talk, we provide a positive answer, published in [2] and studied further in Lancelot's PhD thesis [10]. The literature on improvements has focused on application-oriented λ-calculi based on call-by-need evaluation. Our focus is more theoretical, therefore, we place ourselves in the minimalistic setting of the (call-by-name) untyped λ-calculus, the semantics of which has been studied in depth—see the classic book [4] and its recent extension [5].

We introduce a new notion of equivalence, called *(head) interaction equivalence*, whose key property is being—as we prove—an equational theory, in contrast to what happens for cost equivalence. This is obtained via a reconciliation of the internal and the external perspectives, loosely inspired by *game semantics* [8,1]. In a following paper [11], we also study a notion of interaction improvement.

Interaction equivalence is based on the new *checkers calculus* $\Lambda_{\circ\bullet}$, a λ-calculus enriched with two players, black $\bullet$ and white $\circ$. We duplicate the abstraction and application constructors of the λ-calculus as to have white ($\lambda_\circ x.t$ and $t \circ u$) and black variants ($\lambda_\bullet x.t$ and $t \bullet u$) of each. Next, β-reduction comes in two flavors, a silent one (internal to a player) and an interaction (external) one, depending on whether the constructors involved in the redex belong to the same player or not:

Silent β	Interaction β
$(\lambda_\bullet x.t) \bullet u \to_{\beta_\tau} t\{x := u\}$	$(\lambda_\circ x.t) \bullet u \to_{\beta_{\bullet\circ}} t\{x := u\}$
$(\lambda_\circ x.t) \circ u \to_{\beta_\tau} t\{x := u\}$	$(\lambda_\bullet x.t) \circ u \to_{\beta_{\bullet\circ}} t\{x := u\}$

Interaction equivalence of t and u is then defined in the checkers calculus exactly as cost equivalence *except* that one uses the predicate $\Downarrow^{\bullet\circ k}_{\mathtt{h}_{\circ\bullet}}$ holding when checkers head reduction terminates using k interaction steps $\to_{\beta_{\bullet\circ}}$ and—crucially—*arbitrarily many* (possibly zero) silent steps $\to_{\beta_\tau}$.

Checkers interaction equivalence is then transferred to the ordinary λ-calculus via a *paint-and-wash* construction: two ordinary λ-terms t and u are interaction equivalent when their uniformly, say, black-painting $\overline{t}^\bullet$ and $\overline{u}^\bullet$ are interaction equivalent in the checkers calculus, *i.e.*:

$$t \equiv^{\mathrm{int}} u \text{ if for all checkers contexts } C, \exists k \geq 0 \,.\, [\; C\langle \overline{t}^\bullet \rangle \Downarrow^{\bullet\circ k}_{\mathtt{h}_{\circ\bullet}} \quad \Leftrightarrow \quad C\langle \overline{u}^\bullet \rangle \Downarrow^{\bullet\circ k}_{\mathtt{h}_{\circ\bullet}} \;]$$

Our first result is that interaction equivalence is an equational theory of the ordinary untyped λ-calculus. As for contextual equivalence, proving that $\equiv^{\mathrm{int}}$ is an equational theory is a non-trivial rewriting theorem, and the counting of interaction steps adds a further difficulty.

Inspecting Black Boxes. It often is hard to establish the contextual equivalence of two given terms, because of the universal quantification over all the contexts. It is then common to look for explicit reformulations via semantic trees or *normal form bisimulations* [12]. Head contextual equivalence $\equiv^{\mathrm{ctx}}_{\mathtt{h}}$ is one of the few (uneffectful) cases for which an explicit description is available, as they are not easy to obtain. Namely, $\equiv^{\mathrm{ctx}}_{\mathtt{h}}$ was characterized as the equational theory $\mathcal{B}\eta^\infty$ induced by the equality of Böhm trees [3] up to possibly infinite η-equivalence [7,20]. A natural question then arises: is there an explicit description of interaction equivalence?

Interaction Equivalence and Böhm Trees. The second contribution of our work is such a characterization of interaction equivalence by the well-known equational theory $\mathcal{B}$ induced by Böhm tree equality. In particular, interaction equivalence is *not* extensional, that is, it does not validate any form of η-equivalence, given that x and $\lambda y.xy$ clearly have a different number of interactions with any context providing an argument. Therefore, interaction equivalence $\equiv^{\mathrm{int}}$ has a simpler explicit description than head contextual equivalence $\equiv^{\mathrm{ctx}}_{\mathtt{h}}$, and provides an operational insight about η-equivalence. The failure of η-equivalence also reveals that our framework may not be that close to game semantics, which typically induces extensional equivalences (unless additional constraints are imposed [14,9]).

The characterization is proved via new interaction-based refinements of standard proof methods in the literature, namely the Böhm-out technique [4] and non-idempotent intersection types [6].

References

1. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. Inf. Comput. **163**(2), 409–470 (2000). https://doi.org/10.1006/INCO.2000.2930
2. Accattoli, B., Lancelot, A., Manzonetto, G., Vanoni, G.: Interaction equivalence. Proc. ACM Program. Lang. **9**(POPL) (Jan 2025). https://doi.org/10.1145/3704891, https://doi.org/10.1145/3704891
3. Barendregt, H.: The type free lambda calculus. In: Barwise, J. (ed.) Handbook of Mathematical Logic, Studies in Logic and the Foundations of Mathematics, vol. 90, pp. 1091 – 1132. Elsevier (1977)
4. Barendregt, H.: The Lambda Calculus – Its Syntax and Semantics, Studies in logic and the foundations of mathematics, vol. 103. North-Holland (1984)
5. Barendregt, H., Manzonetto, G.: A Lambda Calculus Satellite. College Publications (2022), https://www.collegepublications.co.uk/logic/mlf/?00035
6. Bucciarelli, A., Kesner, D., Ventura, D.: Non-idempotent intersection types for the lambda-calculus. Log. J. IGPL **25**(4), 431–464 (2017). https://doi.org/10.1093/JIGPAL/JZX018
7. Hyland, M.: A syntactic characterization of the equality in some models for the lambda calculus. Journal of the London Mathematical Society **s2-12**(3), 361–370 (1976). https://doi.org/https://doi.org/10.1112/jlms/s2-12.3.361
8. Hyland, M., Ong, C.L.: On full abstraction for PCF: I, II, and III. Inf. Comput. **163**(2), 285–408 (2000). https://doi.org/10.1006/INCO.2000.2917
9. Ker, A.D., Nickau, H., Ong, C.L.: Adapting innocent game models for the böhm tree λ-theory. Theor. Comput. Sci. **308**(1-3), 333–366 (2003). https://doi.org/10.1016/S0304-3975(02)00849-6
10. Lancelot, A.: Comparing functional programs, or how to put λ-terms back in order. Theses, Institut Polytechnique de Paris (Nov 2025), https://theses.hal.science/tel-05407883
11. Lancelot, A., Manzonetto, G., McCusker, G., Vanoni, G.: Interaction improvement. In: Bertrand, N., Milius, S. (eds.) Foundations of Software Science and Computation Structures. pp. 372–395. Springer Nature Switzerland, Cham (2026)
12. Lassen, S.B.: Bisimulation in untyped lambda calculus: Böhm trees and bisimulation up to context **20**, 346–374 (1999). https://doi.org/10.1016/S1571-0661(04)80083-5
13. Morris, J.H.: Lambda-calculus Models of Programming Languages. Ph.D. thesis, Massachusetts Institute of Technology (1968), https://books.google.is/books?id=DklAAQAAIAAJ
14. Ong, C.L., Di Gianantonio, P.: Games characterizing lévy-longo trees. In: Widmayer, P., Ruiz, F.T., Bueno, R.M., Hennessy, M., Eidenbenz, S.J., Conejo, R. (eds.) Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2380, pp. 476–487. Springer (2002). https://doi.org/10.1007/3-540-45465-9_41
15. Patrignani, M., Ahmed, A., Clarke, D.: Formal approaches to secure compilation: A survey of fully abstract compilation and related work. ACM Comput. Surv. **51**(6), 125:1–125:36 (2019). https://doi.org/10.1145/3280984
16. Plotkin, G.D.: LCF considered as a programming language. Theor. Comput. Sci. **5**(3), 223–255 (1977). https://doi.org/10.1016/0304-3975(77)90044-5
17. Sands, D.: Proving the correctness of recursion-based automatic program transformations. Theor. Comput. Sci. **167**(1&2), 193–233 (1996). https://doi.org/10.1016/0304-3975(96)00074-6

18. Sands, D.: Total correctness by local improvement in the transformation of functional programs. ACM Trans. Program. Lang. Syst. **18**(2), 175–234 (mar 1996). https://doi.org/10.1145/227699.227716
19. Sands, D.: Improvement theory and its applications, p. 275–306. Cambridge University Press, USA (1999)
20. Wadsworth, C.P.: The Relation Between Computational and Denotational Properties for Scott's $\mathcal{D}_\infty$-Models of the Lambda-Calculus. SIAM J. Comput. **5**(3), 488–521 (1976). https://doi.org/10.1137/0205036