An abstract, certified account of Operational Game Semantics

Peio Borthelle[1], Tom Hirschowitz[1], Guilhem Jaber[2], and Yannick Zakowski[3]

[1] Univ. Savoie Mont Blanc, CNRS, LAMA, 73000 Chambéry, France
[2] Nantes Université, LS2N, Nantes, France
[3] Inria, LIP, Lyon, France

Introduction Operational Game Semantics (OGS [Lai07; LL07]) is a method for building trace models for higher-order languages based on ideas from pointer games [HO00]—most importantly keeping traces first-order. These models are typically proven sound (and sometimes complete) w.r.t. contextual equivalence. We present a formal treatment of this technique in the Coq proof assistant. While there is no obvious obstruction to doing this, OGS is generally developed in set theory, which forces us to adapt some notions to better fit the type theory of Coq. In particular, (1) we choose the more computational coalgebraic point of view and (2) we adopt the intrinsically well-scoped approach to binders including for traces. Concretely this consists of an indexed adaptation of Interaction Trees [Xia+20] closely related to the *transition system* approach to games [LS14].

In doing so we also abstract away the specifics of the modelled language and develop an abstract account of OGS, expliciting the hypotheses needed for the soundness proof to go through. Building on Fiore et al.'s work on substitution [FPT99; Fio08; FS22], we treat a large class of simply-typed languages with deterministic but possibly non-terminating evaluation, presented as abstract machines. Notably, we have instantiated our framework with simply-typed λ-calculus and System D [DA20], a variant of polarized $\mu\tilde{\mu}$-calculus.

In this talk, we propose (1) a brief rundown on operational game semantics, (2) a high-level view of our contributions, and finally (3) a more technical elaboration of a specific aspect of the correctness proof, namely well-definedness of the composition of Proponent- and Opponent-strategies. Indeed, this crucial point involves the so-called *infinite chit-chat problem*, to which we provide an abstract solution we believe may interest the GALOP audience.

Operational Game Semantics Following interactive semantics, the behavior of a program can be represented as the (infinite) tree of its interactions with any execution environment. In OGS, the LTS associated to a term is computed by evaluating it to a normal form, say $E[x\ v]$ and issuing a label corresponding to the normal form. As embarking higher-order values in labels leads to challenging notions of bisimilarity, only an *abstracted*, or *inert* version of the normal form is sent, with fresh *channel names* bound and inserted in place of subterms we wish to hide. In the case of $E[x\ v]$, the label is $\mathrm{app}(x)$ and binds two channels, one to hide the argument v and one to hide the continuation context E. Such a transition leads the LTS to a passive state where the environment may resume computation by choosing an available channel to send a message to.

In this setup, labels are semantically simpler, but they bind and reference channel names. To tackle this formally, we resort to a static scoping discipline and use dependently typed data structures. Labels do not live on their own but are indexed by some channel scope. In addition, to each scoped label, we associate the freshly bound channels it introduces. The OGS LTS configurations are then indexed by *positions* consisting of two channel scopes: one for channels owned by the player, the other for channels owned by its opponent, each side sending messages on their opponent's owned channels and receiving on their own channels.

An LTS over such a label structure consists of sets of active and passive *configurations* indexed by active and passive *positions*, with two transition functions appropriately choosing or receiving the next label and producing the next configuration. Given an evaluator **eval** defined in the delay monad [Cap05] we construct such an LTS, together with interpretations $[\![-]\!]_{\mathrm{act}}$ and $[\![-]\!]_{\mathrm{pas}}$ respectively from terms to active configurations and from contexts-and-substitutions to passive configurations.

Productive Composition We recall that two terms u, v are CIU-equivalent iff:

$$\forall E, \sigma,\ \mathbf{eval}(E[u[\sigma]]) \approx_{\Downarrow} \mathbf{eval}(E[v[\sigma]])$$

where $\approx_{\Downarrow}$ means cotermination in the delay monad.

To prove the soundness of our model, the central technique of OGS is to provide a composition operation $\|$ between active and passive configurations, and to prove the following[1] for all terms u, evaluation contexts E and substitutions σ:

$$\mathbf{eval}(E[u[\sigma]]) \approx_{\Downarrow} [\![u]\!]_{\mathrm{act}} \,\|\, [\![E, \sigma]\!]_{\mathrm{pas}} \tag{1}$$

Intuitively this composition operator first extracts a message from the currently active configuration, then sends this message to the passive configuration and finally continues by swapping the roles and starting over. At each point, the exchanged message could may also be sent to a *final* channel, in which case the composition is finished. More formally, this composition operator is given by the fixpoint of an equation system, justified by the fact that the delay monad is completely iterative [Acz+03], *i.e.*, has an operator $\mathbf{iter} : (X \to \mathcal{D}(X + Y)) \to (X \to \mathcal{D}(Y))$.

Now to prove equation (1) it is only natural to use an argument of uniqueness of fixpoints. Indeed, the hypotheses we ask on the language buy us the core of the proof that "substituting then evaluating" is a fixpoint of "one step of composition" (with respect to strong bisimulation in the delay monad). Alas, fixpoints in the delay monad are not unique in general. Milius *et al.* have proven that they are when the equation system is *guarded*, that is, it never directly loops back ([Acz+03; ML17], example 2.12), but our equations are not guarded! As it happens, when substituting the head variable of a normal form with a value, one may sometimes get again a normal form, in particular when the value is just another variable. This situation is what is commonly called a *chit-chat* between Proponent and Opponent.

Our way out of this hurdle is two-fold. First, we introduce a notion of *eventually guardedness* for equation systems (where the guard does not need to happen at every loop step but just some finite amount of steps apart) and prove the uniqueness of fixpoints[2] in this case for the delay monad. Second, we prove the eventual guardedness of the composition's equation system. This first requires to prove an acyclicity property on the suspended value environment of Proponent and Opponent. We solve this by using a refined static typing of the game structure. Instead of two unrelated scopes for Proponent and Opponent channels, we switch to a joint *interlaced* representation. This enables remembering the alternated sequence of scope-extensions by each player and statically forbids any loop from appearing.

At this point the equation system of composition is still not eventually guarded. Indeed we can show that after some finite amount of composition steps, either composition has ended, or we are guarded, or we have instanciated the head variable of a normal form with a non-value variable. But this last case does not necessarily produce a redex. The missing technical condition on the language which enables to conclude states that if one repeatedly replaces the head variable of a normal form with a non-variable value, then one eventually gets a non-normal form (*i.e.*, a redex). While easy to check and true in all instances we have looked at, this condition is non-trivial and was to our knowledge never explicitely stated.

References

[Acz+03] Peter Aczel, Jirí Adámek, Stefan Milius, and Jirí Velebil. "Infinite trees and completely iterative theories: a coalgebraic view". In: *Theor. Comput. Sci.* 300.1-3 (2003), pp. 1–45. DOI: 10.1016/S0304-3975(02)00728-4.

[1] The soundness follows by congruence of this composition operation.

[2] Still with respect to strong bisimulation.

[Cap05] Venanzio Capretta. "General recursion via coinductive types". In: *Log. Methods Comput. Sci.* 1.2 (2005). DOI: 10.2168/LMCS-1(2:1)2005.

[DA20] Paul Downen and Zena M. Ariola. "Compiling With Classical Connectives". In: *Log. Methods Comput. Sci.* 16.3 (2020). URL: https://lmcs.episciences.org/6740.

[Fio08] Marcelo P. Fiore. "Second-Order and Dependently-Sorted Abstract Syntax". In: *Proc. 23rd Symposium on Logic in Computer Science*. IEEE, 2008, pp. 57–68. ISBN: 978-0-7695-3183-0. DOI: 10.1109/LICS.2008.38.

[FPT99] Marcelo Fiore, Gordon Plotkin, and Daniele Turi. "Abstract Syntax and Variable Binding". In: *Proc. 14th Symposium on Logic in Computer Science*. IEEE, 1999. DOI: 10.1109/LICS.1999.782615.

[FS22] Marcelo Fiore and Dmitrij Szamozvancev. "Formal metatheory of second-order abstract syntax". In: *Proc. ACM Program. Lang.* 6.POPL (2022), pp. 1–29. DOI: 10.1145/3498715.

[HO00] J. M. E. Hyland and C.-H. Luke Ong. "On Full Abstraction for PCF: I, II, and III". In: *Inf. Comput.* 163.2 (2000), pp. 285–408. DOI: 10.1006/inco.2000.2917.

[Lai07] James Laird. "A Fully Abstract Trace Semantics for General References". In: *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*. Ed. by Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki. Vol. 4596. Lecture Notes in Computer Science. Springer, 2007, pp. 667–679. DOI: 10.1007/978-3-540-73420-8_58.

[LL07] Søren B. Lassen and Paul Blain Levy. "Typed Normal Form Bisimulation". In: *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*. Ed. by Jacques Duparc and Thomas A. Henzinger. Vol. 4646. Lecture Notes in Computer Science. Springer, 2007, pp. 283–297. DOI: 10.1007/978-3-540-74915-8_23.

[LS14] Paul Blain Levy and Sam Staton. "Transition systems over games". In: *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*. Ed. by Thomas A. Henzinger and Dale Miller. ACM, 2014, 64:1–64:10. DOI: 10.1145/2603088.2603150. URL: https://doi.org/10.1145/2603088.2603150.

[ML17] Stefan Milius and Tadeusz Litak. "Guard Your Daggers and Traces: Properties of Guarded (Co-)recursion". In: *Fundam. Informaticae* 150.3-4 (2017), pp. 407–449. DOI: 10.3233/FI-2017-1475.

[Xia+20] Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. "Interaction trees: representing recursive and impure programs in Coq". In: *Proc. ACM Program. Lang.* 4.POPL (2020), 51:1–51:32. DOI: 10.1145/3371119.