Games and Strategies using Coinductive Types

Peio Borthelle[1], Tom Hirschowitz[2], Guilhem Jaber[3], and Yannick Zakowski[4]

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

Introduction Bisimulations and game techniques for higher-order languages have proved to be powerful tools for reasoning about program equivalence and building models that scale to advanced features such as side effects or existential types. Yet, their usage in mechanized proofs is rare. In this work in progress, we argue that this observation is, in part, the consequence of several mismatches between the traditional presentation of games in set theory and idiomatic constructions from type theory. We hence present a formulation of games and strategies more amenable to manipulation in proof assistants.

The framework we propose is structured around a coinductive representation of labelled transition systems (LTS), inspired by *interaction structures* [**?**], by the Coq library of *interaction trees* [XZH+20], and building upon the work of Levy and Staton [LS14]. Our main contribution is to provide a unified account of *operational game semantics* (OGS [Lai07, LL07]), an LTS-based game model for which we prove the correctness of the generated bisimulation with respect to contextual equivalence[1]. The construction, and the proof, are parametrized by a rather loose notion of evaluator assumed to satisfy a succinct axiomatization. In this talk, we will focus on (1) introducing the standard approach of operational game semantics succinctly, before (2) giving a more detailed account of the peculiarities and advantages of our representation of games and strategies.

Operational Game Semantics The behavior of a program can be represented as the set of its interactions with any execution environment. These sets of interactions can be generated intensionally by an LTS, where the labels encode information exchanged. For higher-order languages, this interaction may typically be the application of the term at hand to an arbitrary value v. One might be tempted to describe it as the transition $\lambda x.e \xrightarrow{\mathbf{app}(v)} e[x \mapsto v]$ but embarking higher order values in labels leads to challenging notions of bisimilarity. Following a technique used in pointer-games [HO00], operational game semantics provides a way to keep the traces first-order: instead of full-blown terms, only an *abstracted* or inert version is exchanged, with fresh *channel names* in place of subterms we wish to hide. The LTS of a term is constructed by evaluating it to a normal form, say $E[x\,v]$ in call-by-value, and issuing a label corresponding to the shape of this normal form. Here the label $\mathbf{app}(x)$ is issued and will bind two fresh channels, one for the abstracted argument v provided to x and one for the abstracted continuation E. This transition leads the LTS to a passive state where the environment (Opponent) is able to resume computation by choosing an available channel.

Labels are now semantically simpler, but they bind and reference channel names. To tackle this, we resort to a static scoping discipline and use dependent-types. Labels are indexed by channel scoping information, containing types such as $s \to t$ for functions and $\neg s$ for continua-

[1] At the time of writing the mechanized version of the proof is not complete.

tions. The function **next** gives the new scope after a label has been issued (slightly simplified):

$$\begin{array}{l}\textbf{label} : \text{scope} \to \text{Type} \\ \textbf{label}\ \Gamma \coloneqq \textbf{app}_{s,t}\ (s \to t \in \Gamma) \mid \textbf{ret}_s\ (\neg s \in \Gamma)\end{array} \qquad \begin{array}{l}\textbf{next}_\Gamma : \text{label}\ \Gamma \to \text{scope} \\ \textbf{next}_\Gamma\ (\text{app}_{s,t}\ i) \coloneqq s, \neg t, \Gamma \\ \textbf{next}_\Gamma\ (\text{ret}_s\ i) \coloneqq s, \Gamma\end{array}$$

The astute reader will have recognized that these label and scope transition rules already form an LTS! We dub it the *game specification*. The OGS LTS proper is indexed over this specification LTS. As our languages of interest have general recursion, we allow usage of the delay monad $\mathcal{D}$ [Cap05]. We give the types of the configurations and active/passive transition functions:

$$\begin{array}{ll}\textbf{conf-act} : \text{scope} \to \text{Type} & \textbf{trans-act} : \text{conf-act}\ \Gamma \to \mathcal{D}((m : \text{label}\ \Gamma) \times \text{conf-pas}\ (\text{next}_\Gamma\ m)) \\ \textbf{conf-pas} : \text{scope} \to \text{Type} & \textbf{trans-pas} : \text{conf-pas}\ \Gamma \to (m : \text{label}\ \Gamma) \to \text{conf-act}\ (\text{next}_\Gamma\ m)\end{array}$$

From Polynomial Functors to Two-Player Games OGS is a symmetric game but in general, Proponent-chosen and Opponent-chosen labels might be different. Thus our two-player game specifications consist of two matching *half-game* descriptions. Descriptions are parametrized by a set of *states* for each player, each side giving for each state the set of allowed moves, and for each move the next state. Each half-game gives rise to two functors on families which we call the *active* and *passive* interpretation.

$$\begin{array}{l}\text{record}\ \textbf{half-game}\ (I\ J : \text{Type}) \coloneqq \{\, \textbf{move} : I \to \text{Type}\,;\, \textbf{trans} : \forall i, \text{move}\ i \to J \,\} \\ \text{record}\ \textbf{game}\ (I\ J : \text{Type}) \coloneqq \{\, \textbf{ply} : \text{half-game}\ I\ J\,;\, \textbf{opp} : \text{half-game}\ J\ I \,\} \\ \textbf{active}\ (H : \text{half-game}\ I\ J)\ (X : J \to \text{Type})\ i \coloneqq (m : H.\text{move}\ i) \times X(H.\text{trans}\ m) \\ \textbf{passive}\ (H : \text{half-game}\ I\ J)\ (X : J \to \text{Type})\ i \coloneqq (m : H.\text{move}\ i) \to X(H.\text{trans}\ m)\end{array}$$

An indexed polynomial endofunctor [AGH+15] can be constructed by composing the active resp. passive interpretation of Proponent resp. Opponent half-games. We can then build strategies by taking an infinite tree construction on this endofunctor. As we wish to handle looping in strategies, following the lead of interaction trees, our construction of choice is a free complete Elgot monad [GMR16] which we give here in two mutually coinductive definitions. The three cases in active strategies correspond respectively to leaves, silent steps similar to the "later" node of the delay monad, and playing a move. Passive strategies correspond to waiting for an Opponent move.

$$\textbf{strat}^+(G : \text{game}\ I\ J)\ X\ i \coloneqq \textbf{ret}\ (X\ i) \mid \textbf{tau}\ (\text{strat}^+\ G\ X\ i) \mid \textbf{vis}\ (\text{active}\ (G.\text{ply})\ (\text{strat}^-\ G\ X)\ i)$$
$$\textbf{strat}^-\ (G : \text{game}\ I\ J)\ X\ j \coloneqq \text{passive}\ (G.\text{opp})\ (\text{strat}^+\ G\ X)\ j$$

Several dualities are at play. First, we can dualize a game by swapping the two components, hence reversing the players' roles. This dualization is definitionally involutive, an improvement over [XZH+20, **?**] where question-answer swapping is hard to make sense of. Second, on half-games there is a functor-functor interaction law [KRU20] between the passive and active interpretations which we dub *synchronization*. Intuitively it makes a sender and a receiver interact and progress. These two constructions together give rise to several more or less general composition operators between strategies and counter-strategies, of which we give a simple one:

$$\begin{array}{l}\textbf{sync} : \Sigma_i(\text{active}\ H\ X\ i \times \text{passive}\ H\ Y\ i) \to \Sigma_j(Xj \times Yj) \\ \textbf{compo} : \Sigma_i(\text{strat}^+\ G\ X\ i \times \text{strat}^-\ G^\perp\ Y\ i) \to \mathcal{D}(\Sigma_i Xi + \Sigma_j Yj)\end{array}$$

Moreover, like polynomial functors, half-games and games are closed under a number of combinators, some studied in [LS14] and strikingly similar to linear logic connectives which we have to investigate further.

References

[AGH+15] Thorsten Altenkirch, Neil Ghani, Peter G. Hancock, Conor McBride, and Peter Morris. Indexed containers. *J. Funct. Program.*, 25, 2015.

[Cap05] Venanzio Capretta. General recursion via coinductive types. *Log. Methods Comput. Sci.*, 1(2), 2005.

[GMR16] Sergey Goncharov, Stefan Milius, and Christoph Rauch. Complete elgot monads and coalgebraic resumptions. *Electronic Notes in Theoretical Computer Science*, 325:147–168, 2016. The Thirty-second Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXII).

[HO00] J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: i, ii, and III. *Inf. Comput.*, 163(2):285–408, 2000.

[KRU20] Shin-ya Katsumata, Exequiel Rivas, and Tarmo Uustalu. Interaction laws of monads and comonads. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 604–618. ACM, 2020.

[Lai07] James Laird. A fully abstract trace semantics for general references. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 667–679. Springer, 2007.

[LL07] Søren B. Lassen and Paul Blain Levy. Typed normal form bisimulation. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 2007.

[LS14] Paul Blain Levy and Sam Staton. Transition systems over games. In *Proceeding of the 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)*, pages 64:1–64:10. ACM, 2014.

[XZH+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. *Proc. ACM Program. Lang.*, 4(POPL):51:1–51:32, 2020.