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 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 \(\lambda x.e \xrightarrow{\text{app}(v)} e[x \mapsto v]\) 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 \(E[x \mapsto v]\) in call-by-value, and issuing a label corresponding to the shape of this normal form. Here the label \(\text{app}(x)\) 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 be 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 \(s \rightarrow t\) 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):

\[
\begin{align*}
\text{label} & : \text{scope} \to \text{Type} \\
\text{label} \; \Gamma := \text{app}_{s,t} \; (s \to t \in \Gamma) \mid \text{ret}_s \; (\neg s \in \Gamma) \\
\text{next}_\Gamma & : \text{label} \; \Gamma \to \text{scope} \\
\text{next}_\Gamma \; (\text{app}_{s,t} \; i) & := s, \neg t, \Gamma \\
\text{next}_\Gamma \; (\text{ret}_s \; i) & := s, \Gamma
\end{align*}
\]

The astute reader will have recognized that these label and scope transition rules already form an LTS! It is total and deterministic: 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 \(D\) [Cap05]. We give the types of the configurations and active/passive transition functions below:

\[
\begin{align*}
\text{conf-act} & : \text{scope} \to \text{Type} \\
\text{trans-act} & : \text{conf-act} \; \Gamma \to D((m : \text{label} \; \Gamma) \times \text{conf-pas} \; (\text{next}_\Gamma \; m)) \\
\text{conf-pas} & : \text{scope} \to \text{Type} \\
\text{trans-pas} & : \text{conf-pas} \; \Gamma \to (m : \text{label} \; \Gamma) \to \text{conf-act} \; (\text{next}_\Gamma \; m)
\end{align*}
\]

**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.

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.

\[
\begin{align*}
\text{strat}^+(G : \text{game} \; I \; J) \; X \; i & := \text{ret} \; (X \; i) \mid \text{tau} \; (\text{strat}^+ \; G \; X \; i) \mid \text{vis} \; (\text{active} \; (G.\text{ply}) \; (\text{strat}^- \; G \; X) \; i) \\
\text{strat}^- (G : \text{game} \; I \; J) \; X \; j & := \text{passive} \; (G.\text{opp}) \; (\text{strat}^+ \; G \; X) \; j
\end{align*}
\]

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 interaction trees 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, of which we give a simple one:

\[
\begin{align*}
\text{sync} & : \Sigma_i(\text{active} \; H \; X \; i \times \text{passive} \; H \; Y \; i) \to \Sigma_j(Xj \times Yj) \\
\text{compo} & : \Sigma_i(\text{strat}^+ \; G \; X \; i \times \text{strat}^- \; G^\perp \; Y \; i) \to D(\Sigma_iXi + \Sigma_jYj)
\end{align*}
\]
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


