A summary on categorical contextual reasoning

Stelios Tsampas[1], Andreas Nuyts[1], Dominique Devriese[2], and Frank Piessens[1]

[1]KU Leuven, Belgium

[2]Vrije Universiteit Brussel, Belgium

Contextual equivalence is the standard notion of program equivalence for operational semantics. Despite its prevalence and due to its complex nature, it is considered very hard to reason about. We summarize our recent work towards a general, categorical perspective on contextual equivalence using distributive laws [1].

In the early nineties, Turi and Plotkin observed [8] that general system specifications known as Structural Operational Semantics [6] (SOS) exhibited categorical *naturality*. More specifically, it was shown that SOS specifications are *distributive laws* of *syntax* over *behavior*, that is natural transformations $\lambda : \Sigma^* B^\infty \Longrightarrow B^\infty \Sigma^*$, where (Σ^*, η, μ) and $(B^\infty, \epsilon, \nu)$ are freely/cofreely generated by endofunctors Σ and B modelling the syntax and behavior for a given system[2].

Distributive laws generate bialgebraic semantics in that the induced interpretation function f, that maps programs (elements of $\Sigma^* 0$) to behaviors (elements of $B^\infty 1$) is both an algebra and a coalgebra homomorphism:

$$\begin{array}{ccccc}
\Sigma^*\Sigma^*0 & \xrightarrow{\eta_0} & \Sigma^*0 & \xrightarrow{h} & B^\infty\Sigma^*0 \\
\downarrow{\scriptstyle \Sigma^* f} & & \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle B^\infty f} \\
\Sigma^* B^\infty 1 & \xrightarrow{g} & B^\infty 1 & \xrightarrow{\epsilon_1} & B^\infty B^\infty 1
\end{array}$$

Where h and g are obtained via *lifting* of the initial algebra η_0 and final coalgebra ϵ_1 respectively [3]. In more concrete terms, *bisimilarity* of programs can be defined as equality under f, and is a *congruence*. This is a desirable *well-behavedness* property and it is the reason why distributive laws are an ideal abstract setting to study operational semantics [3].

[1] https://people.cs.kuleuven.be/~stylianos.tsampas/ctx.pdf, submitted to MFCS 2019 [7]

[2] This can be generalized further but it suffices for our purposes.

In our work we looked at the notion of Morris style *contextual equivalence* [2], widely used as a program equivalence in operational semantics, from a categorical perspective. Intuitively, contextual equivalence captures observational indistinguishability where observers, or contexts, are terms with a *hole*. For instance, given a set of programs A with $a \Downarrow$ denoting that program a successfully terminates and a set C of "terms with a hole", a typical definition of contextual equivalence looks like this:

Definition 0.1 $a_1 \sim_{\text{ctx}} a_2 \; \texttt{iff} \; \forall c.\, c\,[\![a_1]\!] \Downarrow \iff c\,[\![a_2]\!] \Downarrow$

It is the quantification over program contexts that makes reasoning about contextual equivalence inherently hard. Consequently, there is an absence of a unifying, general approach to it.

Contexts and distributive laws The starting point of our approach is the category theoretic notion of a program context. Assuming a distributive law $\lambda : \Sigma^* B^\infty \Longrightarrow B^\infty \Sigma^*$ with Σ and B endofunctors on a well-behaved category $\mathbb{C}$, we say that a functor $H : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ is a context functor (with application to (X, Y) denoted as $H_X Y$) if there exist natural transformations $\mathrm{hole} : \forall (X, Y).1 \to H_X Y$ and $\mathrm{con} : \forall X. X \times H_X X \to X \uplus \Sigma X$ making the following diagram commute for all X:

$$\begin{tikzcd}
X \times 1 \arrow[r, "\pi_1", "\sim"'] \arrow[d, "\mathrm{id}_X \times \mathrm{hole}_{(X,X)}"'] & X \arrow[d, "\iota_1"] \\
X \times H_X X \arrow[r, "\mathrm{con}_X"] & X \uplus \Sigma X
\end{tikzcd}$$

The idea behind con is that it takes a metavariable $x \in X$ and a context $c \in H_X X$, meaning a layer of syntax H_X in which deeper holes have already recursively been plugged and returns either x, if the context is the hole itself, or the corresponding syntax in Σ. We show that the above definitions are instantiated by both single-hole [4] and multi-hole contexts.

Contextual co-closures Looking back at Definition 0.1, the next step is to be able to (categorically) reason about *all* program contexts for a given "adequate" relation. If we represent relations categorically as spans, our definition of contexts allows us to do so:

Definition 0.2 *A span $A \xleftarrow{r_1} R \xrightarrow{r_2} A$ is called contextually closed if there is a function $[\![\,]\!] : C \times R \to R$ making the following diagram commute:*

$$\begin{tikzcd} C_A \times A \arrow[d, "\llbracket\,\rrbracket"] & C_A \times R \arrow[l, "\mathrm{id}\times r_1"'] \arrow[r, "\mathrm{id}\times r_2"] \arrow[d, "\llbracket\,\rrbracket"] & C_A \times A \arrow[d, "\llbracket\,\rrbracket"] \\ A & R \arrow[l, "r_1"'] \arrow[r, "r_2"] & A \end{tikzcd}$$

The contextual co-closure $A \xleftarrow{\bar{r}_1} \bar{R} \xrightarrow{\bar{r}_2} A$ of an arbitrary span $A \xleftarrow{r_1} R \xrightarrow{r_2} A$ is the final contextually closed span on A with a span morphism $\bar{R} \to R$.

We can now define a bisimilarity relation on Σ^*0 by relating $a_0 \sim_{\text{bis}} a_1$ if and only if $f(a_0) = f(a_1) \in B^\infty 1$. As part of our contributions, we show that if f is generated via a distributive law, then $\sim_{\text{bis}}$ is contextually closed:

Theorem 0.1 $a_1 \sim_{\text{bis}} a_2$ $\texttt{iff}$ $\forall c.\ c\llbracket a_1 \rrbracket \sim_{\text{bis}} c\llbracket a_2 \rrbracket$

Contextual equivalence is typically defined as the contextual co-closure of equitermination, which is strictly coarser than the contextual co-closure of bisimilarity. However, we move on to demonstrate that for certain behavior functors, we can alter a given distributive law so that two programs are bisimilar in the derived system if and only if they evaluate to the same value in the original system. By applying our proof method to the new system we effectively show that contextual equivalence coincides with bisimilarity.

Fully abstract compilation In the field of secure compilation [5], a compiler is fully abstract if it preserves and reflects contextual equivalence. Formally proving that a compiler is fully abstract is especially hard precisely due to the awkward definition of contextual equivalence. As such, there is great incentive for more effective formal methods [1].

Definition 0.3 ([9]) *Given distributive laws $\lambda_1 : \Sigma_1^* B_1^\infty \Longrightarrow B_1^\infty \Sigma_1^*$ and $\lambda_2 : \Sigma_2^* B_2^\infty \Longrightarrow B_2^\infty \Sigma_2^*$ a map of distributive laws is a pair of natural transformations $s : \Sigma_1^* \Longrightarrow \Sigma_2^*$ and $b : B_1^\infty \Longrightarrow B_2^\infty$, which respect the (co)monad laws and for which the following diagram commutes:*

$$\begin{tikzcd} \Sigma_1^* B_1^\infty \arrow[r, Rightarrow, "\lambda_1"] \arrow[d, Rightarrow, "s \circ \Sigma_1^* b"] & B_1^\infty \Sigma_1^* \arrow[d, Rightarrow, "b \circ B_1^\infty s"] \\ \Sigma_2^* B_2^\infty \arrow[r, Rightarrow, "\lambda_2"] & B_2^\infty \Sigma_2^* \end{tikzcd}$$

For our other contribution, we prove that maps of distributive laws are effectively compilers that preserve and (under a reasonable condition) reflect bisimilarity. Insofar as bisimilarity can be tuned to coincide with contextual equivalence, we argue that the coherence criterion given by the diagram in Definition 0.3 could be an effective formal method for testing full abstraction.

References

[1] Amal Ahmed, Deepak Garg, Catalin Hritcu, and Frank Piessens. Secure Compilation (Dagstuhl Seminar 18201). *Dagstuhl Reports*, 8(5):1–30, 2018. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9891, doi:10.4230/DagRep.8.5.1.

[2] Morris James Hiram, Jr. Lambda-calculus models of programming languages. page 135, 12 1968.

[3] Bartek Klin. Bialgebras for structural operational semantics: An introduction. *Theor. Comput. Sci.*, 412(38):5043–5069, 2011. URL: https://doi.org/10.1016/j.tcs.2011.03.023, doi:10.1016/j.tcs.2011.03.023.

[4] Conor Mcbride. The derivative of a regular type is its type of one-hole contexts (extended abstract), 2001.

[5] Marco Patrignani, Amal Ahmed, and Dave Clarke. Formal Approaches to Secure Compilation: A Survey of Fully Abstract Compilation and Related Work. *ACM Comput. Surv.*, 51(6):125:1–125:36, February 2019. doi:10.1145/3280984.

[6] Gordon D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.

[7] Stelios Tsampas, Andreas Nuyts, Dominique Devriese, and Frank Piessens. Towards categorical contextual reasoning. Submitted to MFCS 2019, May 2019. URL: https://tcs.rwth-aachen.de/mfcs2019/.

[8] Daniele Turi and Gordon D. Plotkin. Towards a mathematical operational semantics. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pages 280–291, 1997. URL: https://doi.org/10.1109/LICS.1997.614955, doi:10.1109/LICS.1997.614955.

[9] Hiroshi Watanabe. Well-behaved translations between structural operational semantics. *Electr. Notes Theor. Comput. Sci.*, 65(1):337–357, 2002. URL: https://doi.org/10.1016/S1571-0661(04)80372-4, doi:10.1016/S1571-0661(04)80372-4.