Testing preorders for asynchronous processes

GIOVANNI BERNARDI, IRIF, France

1 INTRODUCTION

Code refactoring is a routine task, necessary to either update or develop software. Correct refactoring in turn necessitates ensuring that a (new) program q can be used in place of a program p. Usually this is tackled via refinement relations. In the setting of programming languages, the most well-known is the *extensional* preorder defined by Morris [1969, pag. 50], by letting $p \leq q$ if for all contexts C, whenever $C[p]$ reduces to a normal form N, then $C[q]$ also reduces to N.

In the setting of nondeterministic asynchronous client-server systems it is natural to reformulate the preorder by replacing reduction to normal forms (*i.e.* termination) with a suitable *liveness* property. Let $p \mathrel{\overline{\parallel}} r$ denote an asymmetric parallel composition in which the identities of the server p and the client r are distinguished, and whose computations have the form

$$p \mathrel{\overline{\parallel}} r \longrightarrow p_1 \mathrel{\overline{\parallel}} r_1 \longrightarrow p_2 \mathrel{\overline{\parallel}} r_2 \longrightarrow \dots$$

where each step represents either an internal computation of one of the two components, or an interaction between them via message-passing. We express liveness by saying that p *must pass* r, denoted MUST (p, r), if in every maximal execution of $p \mathrel{\overline{\parallel}} r$, there exists a state $p_i \mathrel{\overline{\parallel}} r_i$ such that r_i good, where good is a decidable predicate indicating that the client has reached a successful state. Observe that MUST (p, r) literally means that "in every execution something good must happen (on the client side)".

Servers are then compared according to their capacity to satisfy clients, namely to lead them to a successful state. In other words, servers are compared only via contexts of the form $[-] \mathrel{\overline{\parallel}} r$, as argued also by Thati [2003]. Then Morris preorder, when restricted to computations leading to successful states, boils down to the MUST-preorder of De Nicola and Hennessy [1984]:

$$p \mathrel{\underset{\sim}{\sqsubset}}_{\text{MUST}} q \text{ if } \forall r.\ \text{MUST}\ (p, r) \text{ implies } \text{MUST}\ (q, r).$$

The MUST-preorder is by definition an archetype of a liveness preserving preorder, moreover its definition is syntax-agnostic: to define MUST-preorder it is sufficient to have a reduction semantics for the parallel composition of programs, and some predicate good. For instance, the servers written in ERLANG could be compared according to clients written in ELIXIR, because we know how to model their parallel executions. The work of Hirschkoff et al. [2023] provides an analogous example for the Morris preorder itself.

The MUST-preorder, like Morris one, is *contextual*: to prove that $p \mathrel{\underset{\sim}{\sqsubset}}_{\text{MUST}} q$, a quantification over an *infinite* number of clients is required, and so the definition of the preorder does not entail an effective proof method. The solution to this problem is to devise an *alternative (semantic) characterisation* of the preorder $\mathrel{\underset{\sim}{\sqsubset}}_{\text{MUST}}$, *i.e.* a preorder $\preccurlyeq_{alt}$ endowed with a practical proof method and such that the equality $\preccurlyeq_{alt} = \mathrel{\underset{\sim}{\sqsubset}}_{\text{MUST}}$ is true.

In *synchronous* settings, that is when both inputs and output actions are blocking, such characterisations have been thoroughly investigated, and typical techniques to define them are either behavioural or logical. In the *asynchronous* setting, i.e. when send actions are not blocking, and communication takes places via a shared unordered buffer, the MUST-preorder has received comparatively less attention. Paul Laforgue, under the supervision of Giovanni Bernardi, though, has recently mechanised a characterisation of the MUST-preorder for the output-buffered agents with

Author's address: Giovanni Bernardi, IRIF, Paris, France, gio[AT]irif.fr.

feedback proposed by Selinger [1997]. The code is available online [Bernardi et al. 2023], and a publication in under preparation.[1] These agents are a very general setting to reason on asynchronous behaviours. In particular, asynchronous CCS and asynchronous π-calculus are instances of this family of agents.

2 RESEARCH LINES FOR SUMMER INTERNS

Much research remains to be done, and we are looking for outstanding interns willing to work ideally with pan-and-paper and in Coq. We present here a series of topics that we would like to investigate together with intern, possibly leading to a PhD thesis.

Characterisation of the MAY*-preorder.* The simplest preorder in testing theory is the MAY-preorder, which is obtained stating that a server p satisfies a client r if there exists a maximal computation of $p \mathbin{\overline{\parallel}} r$ in which the client reaches a good state.

We would like to characterise the $\underset{\sim}{\sqsubset}_{\textsc{may}}$ using the same technique we used to reason on $\underset{\sim}{\sqsubset}_{\textsc{must}}$, namely treating programs as forwarders. We claim that this is possible, and easy to be done. The result would be stronger than (i.e. implies) the ones that exists in the literature by [Castellani and Hennessy 1998; Boreale et al. 2002].

Characterisations for infinite branching state transition systems. The current characterisation of the MUST-preorder does not treat infinite branching LTSs. However it is easy to define them in Coq.

Following Bernardi and Hennessy [2015], what *seems* sufficient and necessary is to add to the characterisation a condition on the inclusion of infinite traces. In the synchronous settings, this amount to proving that if $p \underset{\sim}{\sqsubset}_{\textsc{must}} q$ and q performs an infinite trace w, so does p.

At present, it is not clear how to state this condition neither with pen-and-paper, neither in Coq. The difficulties with pen-and-paper are due to the asymmetry between output and input actions, while the difficulties in Coq are due to the finitary treatment of infinite traces. To make things worse, in the asynchronous setting not all traces can be tested.

This is certainly a path worth investigating for theoretical reasons.

Treating input-buffered agents. The axioms for output-buffered agents by Selinger [1997] have a symmetric set of axioms for input-buffered agents. It would be interesting to know if the proofs we devised so far can be adapted "out-of-the-box" to the LTS of input-buffered agents.

At present, this is interesting for theoretical reasons, and in particular to understand how general are our proofs.

Semantic models of subtyping for session types. Testing preorders provide semantic models of subtyping for binary session types, both in synchronous and asynchronous settings [Bernardi and Hennessy 2016a,b; Bravetti et al. 2021]. We would like to mechanise in our framework these results, in particular the ones about asynchronous semantics, and contrast and compare the various testing preorders used in the literature.

This venue is worth attention for practical purposes, and in particular to devise provably sound algorithms to prove that two types in a testing preorder. The problem is not trivial because in general it is undecidable.

Preorders for ERLANG*.* Both Tanti and Francalanza [2015] and Caruana [2019] define LTSs for ERLANG. We wish to study whether at least one of these LTS is an instance of output-buffered agents with feedback. If this is not the case we would like define an LTS that satisfies Selinger's axioms.

[1] A preliminary one received three week accepts at POPL 2024.

This study is definitely geared towards practical applications, and in particular devising sound techniques for code refactoring in Erlang and Elixir.

Liveness preserving choreographies. We would like to give to the compositional choreographies introduced by Montesi and Yoshida [2013] an asynchronous semantics, and then use the must-preorder to prove the correctness of the projection function EPP, *i.e.* prove the following fact,

$$\forall \text{ CHOREOGRAPHY } C.C \underset{\sim}{\sqsubset}_{\text{MUST}} \Pi_{a \in names(C)} EPP(C, a)$$

We expect this notion of correctness to be less restrictive than the one based on bisimulation.

REFERENCES

Giovanni Bernardi and Matthew Hennessy. 2015. Mutually Testing Processes. *Log. Methods Comput. Sci.* 11, 2 (2015). https://doi.org/10.2168/LMCS-11(2:1)2015

Giovanni Bernardi and Matthew Hennessy. 2016a. Using higher-order contracts to model session types. *Log. Methods Comput. Sci.* 12, 2 (2016). https://doi.org/10.2168/LMCS-12(2:10)2016

G. Bernardi, P. Laforgue, and L. Stefanesco. 2023. Machine checked characterisation of the MUST-preorder. https://shouldnothappen.com/must/.

Giovanni Tito Bernardi and Matthew Hennessy. 2016b. Modelling session types using contracts. *Math. Struct. Comput. Sci.* 26, 3 (2016), 510–560. https://doi.org/10.1017/S0960129514000243

Michele Boreale, Rocco De Nicola, and Rosario Pugliese. 2002. Trace and Testing Equivalence on Asynchronous Processes. *Inf. Comput.* 172, 2 (2002), 139–164. https://doi.org/10.1006/inco.2001.3080

Mario Bravetti, Marco Carbone, Julien Lange, Nobuko Yoshida, and Gianluigi Zavattaro. 2021. A Sound Algorithm for Asynchronous Session Subtyping and its Implementation. *Log. Methods Comput. Sci.* 17, 1 (2021). https://lmcs.episciences.org/7238

Caroline Caruana. 2019. Compositional Reasoning about Actor Based Systems.

Ilaria Castellani and Matthew Hennessy. 1998. Testing Theories for Asynchronous Languages. In *Foundations of Software Technology and Theoretical Computer Science, 18th Conference, Chennai, India, December 17-19, 1998, Proceedings (Lecture Notes in Computer Science, Vol. 1530)*, Vikraman Arvind and Ramaswamy Ramanujam (Eds.). Springer, 90–101. https://doi.org/10.1007/978-3-540-49382-2_9

Rocco De Nicola and Matthew Hennessy. 1984. Testing Equivalences for Processes. *Theor. Comput. Sci.* 34 (1984), 83–133. https://doi.org/10.1016/0304-3975(84)90113-0

Daniel Hirschkoff, Guilhem Jaber, and Enguerrand Prebet. 2023. Deciding Contextual Equivalence of ν-Calculus with Effectful Contexts. In *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings (Lecture Notes in Computer Science, Vol. 13992)*, Orna Kupferman and Pawel Sobocinski (Eds.). Springer, 24–45. https://doi.org/10.1007/978-3-031-30829-1_2

Fabrizio Montesi and Nobuko Yoshida. 2013. Compositional Choreographies. In *CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8052)*, Pedro R. D'Argenio and Hernán C. Melgratti (Eds.). Springer, 425–439. https://doi.org/10.1007/978-3-642-40184-8_30

James H. Morris. 1969. *Lambda-calculus models of programming languages.* Ph. D. Dissertation. https://dspace.mit.edu/handle/1721.1/64850

Peter Selinger. 1997. First-Order Axioms for Asynchrony. In *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings (Lecture Notes in Computer Science, Vol. 1243)*, Antoni W. Mazurkiewicz and Józef Winkowski (Eds.). Springer, 376–390. https://doi.org/10.1007/3-540-63141-0_26

Erica Tanti and Adrian Francalanza. 2015. Towards sound refactoring in erlang. https://api.semanticscholar.org/CorpusID:63046364

Prasannaa Thati. 2003. *A Theory of Testing for Asynchronous Concurrent Systems.* Ph. D. Dissertation. University of Illinois Urbana-Champaign, USA. https://hdl.handle.net/2142/81630