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 $p \parallel r$ 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

$$p \parallel r \rightarrow p_1 \parallel r_1 \rightarrow p_2 \parallel r_2 \rightarrow \ldots$$

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 $\text{must} (p, r)$, if in every maximal execution of $p \parallel r$, there exists a state $p_i \parallel r_i$ such that $r_i$ good, where $\text{good}$ is a decidable predicate indicating that the client has reached a successful state. Observe that $\text{must} (p, r)$ 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 $[\_] \parallel r$, as argued also by Thati [2003]. Then Morris preorder, when restricted to computations leading to successful states, boils down to the $\text{must}$-preorder of De Nicola and Hennessy [1984]:

$$p \text{ must } q \text{ if } \forall r. \text{ must} (p, r) \text{ implies } \text{must} (q, r).$$

The $\text{must}$-preorder is by definition an archetype of a liveness preserving preorder, moreover its definition is syntax-agnostic: to define $\text{must}$-preorder it is sufficient to have a reduction semantics for the parallel composition of programs, and some predicate $\text{good}$. 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 $\text{must}$-preorder, like Morris one, is contextual: to prove that $p \text{ must } q$, 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 $\text{must}$, i.e. a preorder $\leq_{\text{alt}}$ endowed with a practical proof method and such that the equality $\leq_{\text{alt}} = \text{must}$ 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 $\text{must}$-preorder has received comparatively less attention. Paul Laforgue, under the supervision of Giovanni Bernardi, though, has recently mechanised a characterisation of the $\text{must}$-preorder for the output-buffered agents with...
feedback proposed by Selinger [1997]. The code is available online [Bernardi et al. 2023], and a publication in under preparation. These agents are a very general setting to reason on asynchronous behaviours. In particular, asynchronous CCS and asynchronous \( \pi \)-calculus are instances of this family of agents.

2 \hspace{1em} \text{RESEARCH LINES FOR SUMMER INTERNS}

Much research remains to be done, and we are looking for outstanding interns willing to work ideally with pen-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 \textit{may}-preorder. The simplest preorder in testing theory is the \textit{may}-preorder, which is obtained stating that a server \( p \) satisfies a client \( r \) if there exists a maximal computation of \( p \upmodels r \) in which the client reaches a good state.

We would like to characterise the \( \sqsubseteq_{\text{may}} \) using the same technique we used to reason on \( \sqsubseteq_{\text{must}} \), 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 \textit{must}-preorder does not treat infinite branching LTSs. However it is easy to define them in Coq.

Following Bernardi and Hennessy [2015], what \textit{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 \( p \sqsubseteq_{\text{must}} q \) 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 \textsc{Erlang}. Both Tanti and Francalanza [2015] and Caruana [2019] define LTSs for \textsc{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.

\footnote{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,

\[ \forall \text{choreography } C. C \subseteq_{\text{must}} \Pi_{a \in \text{names}(C)} EPP(C, a) \]

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

REFERENCES


