Brief Announcement: On Strong Observational Refinement and Forward Simulation

John Derrick
University of Sheffield, UK

Simon Doherty
University of Sheffield, UK

Brijesh Dongol
University of Surrey, UK

Gerhard Schellhorn
Universität Augsburg, Germany

Heike Wehrheim
Universität Oldenburg, Germany

Abstract

Hyperproperties are correctness conditions for labelled transition systems that are more expressive than traditional trace properties, with particular relevance to security. Recently, Attiya and Enea studied a notion of strong observational refinement that preserves all hyperproperties. They analyse the correspondence between forward simulation and strong observational refinement in a setting with finite traces only. We study this correspondence in a setting with both finite and infinite traces. In particular, we show that forward simulation does not preserve hyperliveness properties in this setting. We extend the forward simulation proof obligation with a progress condition, and prove that this *progressive forward simulation* does imply strong observational refinement.

2012 ACM Subject Classification Security and privacy → Formal methods and theory of security; Computing methodologies → Concurrent computing methodologies

Keywords and phrases Strong Observational Refinement, Hyperproperties, Forward Simulation

Digital Object Identifier 10.4230/LIPIcs.DISC.2021.55

Related Version *Full Version*: `https://arxiv.org/abs/2107.14509`

Funding *John Derrick*: EPSRC Grant EP/R032351/1.
Brijesh Dongol: EPSRC grants EP/V038915/1, EP/R032556/1, EP/R025134/2 and VeTSS.
Heike Wehrheim: DFG Grant SFB 901.

1 Introduction

Hyperproperties [2] form a large class of properties over sets of sets of traces, characterising, in particular, security properties such as generalised non-interference that are not expressible over a single trace. Like with trace properties, every hyperproperty can be characterised as the conjunction of a hypersafety and hyperliveness property.

Recently, Attiya and Enea proposed *strong observational refinement*, a correctness condition that preserves all hyperproperties, even in the presence of an adversarial scheduler. An object O_1 strongly observationally refines an object O_2 if the executions of any program P using O_1 as scheduled by some admissible deterministic scheduler cannot be observationally distinguished from those of P using O_2 under another deterministic scheduler. They showed that strong observational refinement preserves all hyperproperties. Furthermore, they prove that forward simulation implies strong observational refinement. Forward simulation alone is sound but not complete for ordinary refinement, and in general both backward and forward simulation are required. Forward simulation is furthermore known to not preserve liveness properties, which motivates our study of forward simulation and observational refinement in the context of infinite traces and hyperliveness.

As a result we show – by example – that forward simulation does not preserve hyperliveness. Furthermore, forward simulation alone cannot guarantee strong observational refinement when requiring admissiblity of schedulers, i.e., when schedulers are required to continually schedule enabled actions. To address these limitations, we employ a version of forward simulation extended with a progress condition, thereby guaranteeing strong observational refinement and preservation of hyperliveness.

2 Motivating Example

```
int* current_val initially 0
int fetch_and_add(int k):
F1. do n = LL(&current_val)
F2. while (!SC(&current_val, n + k))
F3. return n
```

Figure 1 A fetch-and-add with a nonterminating schedule when `LL` and `SC` are implemented using the algorithm of [4].

We give an example of an abstract atomic object O_2 and a non-atomic implementation O_1 such that there *is* a forward simulation from O_1 to O_2, but hyperliveness properties are not preserved for all schedules. As the atomic abstract object O_2 we choose a *fetch-and-add* object with just one operation, `fetch_and_add(int k)`, which adds the value integer `k` to a shared integer variable and returns the value of that variable before the addition. Let P be a program with two threads t_1 and t_2, each of which executes one `fetch_and_add` operation and assigns the return value to a local variable of the thread. For any scheduler S, the variable assignment of both threads will eventually occur. This "eventually" property can be expressed as a hyperproperty.

Now, consider the *fetch-and-add* implementation presented in Figure 1. This implementation uses the `load-linked/store-conditional` (LL/SC) instruction pair. The `LL(ptr)` operation loads the value at the location pointed to by the pointer `ptr`. The `SC(ptr,v)` conditionally stores the value `v` at the location pointed to by `ptr` if the location has not been modified by another `SC` since the executing thread's most recent `LL(ptr)` operation. If the update actually occurs, `SC` returns `true`, otherwise the location is not modified and `SC` returns `false`. In the first case, we say that the `SC` *succeeds*. Otherwise, we say that it *fails*.

Critically, we stipulate that the LL and SC operations are implemented using the algorithm of [4]. This algorithm has the following property. If thread t_1 executes an `LL` operation, and then thread t_2 executes an `LL` operation *before* t_1 has executed its subsequent `SC` operation, then that `SC` is guaranteed to fail. This happens even though there is no intervening modification of the location.

Now, let O_1 be a *labelled transition system* (LTS) representing a multithreaded version of this `fetch_and_add` implementation, using the specified `LL/SC` algorithm. Consider furthermore the program P (above) running against the object O_1. A scheduler can continually alternate the `LL` at line `F2` of t_1 and that of t_2, such that neither `fetch_and_add` operation ever completes. Therefore, unlike when using the O_2 object, the variable assignments of P will never occur, so the O_1 system does not satisfy the hyperproperty for all schedulers.

There is, however, a forward simulation from O_1 to O_2. The underlying `LL`/`SC` implementation can be proven correct by means of forward simulation, as can the `fetch_and_add` implementation. Therefore, standard forward simulation is insufficient to show that all hyperproperties are preserved, contradicting Lemma 5.2 of [1].

3 Progressive Forward Simulation implies Strong Observ. Refinement

We will use the notation of Attiya and Enea [1], in particular that of an LTS $A = (Q, \Sigma, s_0, \delta)$ and of a (deterministic, admissible) scheduler $S : \Sigma^* \to 2^\Sigma$ (full definitions can also be found in [3]). The main change we make is that the traces in trace sets $T(A)$ and $T(A, S)$ (S-scheduled traces) now include finite and infinite sequences[1]. A scheduler is *admitted* by an LTS A if for all finite traces σ of A consistent with S, the scheduler satisfies (i) $S(\sigma)$ is non-empty and (ii) all actions in $S(\sigma)$ are enabled in $\mathit{state}(\sigma)$. Besides being admissible, schedulers for programs P and objects O (LTSs of the form $P \times O$) also have to be *deterministic*: they must resolve the nondeterminism on the actions of the object. An object O_1 *strongly observationally refines* the object O_2, written $O_1 \leq_S O_2$, iff for every deterministic scheduler S_1 admitted by $P \times O_1$ there exists a deterministic scheduler S_2 admitted by $P \times O_2$ such that $T(P \times O_1, S_1)|\Sigma_P = T(P \times O_2, S_2)|\Sigma_P$ for all programs P.

Contrary to the claim in [1], standard forward simulation does not imply strong observational refinement (details in [3]). In the example given in Section 2, a deterministic admissible scheduler S_1 for P and O_1 could drive $P \times O_1$'s execution along the infinite trace of `LL` and `SC` operations, so that calls to `fetch_and_add` never return. On the other hand, any scheduler for the O_2 system must eventually execute call and return actions for both `fetch_and_add` operations, and subsequently execute the writes to the program variables. Thus, $T(P \times O_1, S_1)|\Sigma_P \neq T(P \times O_2, S_2)|\Sigma_P$. To guarantee strong observational refinement, forward simulation additionally has to guarantee some sort of progress, so that the scheduler S_2 is always able to schedule some action without producing a trace not present in $P \times O_1$ under S_1. This guarantee can be made if we disallow infinite stuttering.

▶ **Definition 1** (Progressive Forward Simulation). *Let $A_i = (Q_i, \Sigma_i, s_0^i, \delta_i)$, $i = 1, 2$, be two LTSs and Γ an alphabet. A relation $F \subseteq Q_1 \times Q_2$ together with a well-founded order $\ll \subseteq Q_1 \times Q_1$ is called a* progressive Γ-forward simulation *from A_1 to A_2 iff*

- $(s_0^1, s_0^2) \in F$, *and*
- *for all $(s_1, s_2) \in F$, if $(s_1, a, s_1') \in \delta_1$ and $a \in \Sigma_1$, then there exist $\alpha \in \Sigma_2^*$ and $s_2' \in Q_2$ such that $a \mid \Gamma = \alpha \mid \Gamma$, $(s_2, \alpha, s_2') \in \delta_2$ and $(s_1', s_2') \in F$. Whenever $\alpha = \varepsilon$ then $s_1' \ll s_1$.*

The definition requires that the concrete state decreases in the well-founded order when the abstract sequence α in the forward simulation is empty and $s_2 = s_2'$ (stuttering). Progressiveness prohibits an infinite sequence of concrete internal steps that map to the empty abstract sequence. For object O_1 above with the `fetch_and_add` implementation no such well-founded ordering can be given. We have (full proof in [3]):

▶ **Theorem 2.** *If there exists a progressive $(C \cup R)$-forward simulation from O_1 to O_2, then $O_1 \leq_S O_2$.*

[1] The work of [1] just considers finite traces. However, they still assume schedulers to always be able to schedule a next action which seems to contradict the fact that all traces are finite.

4 Conclusion

In this paper, we have reported on our findings that forward simulation does not imply strong observational refinement in a setting with infinite traces. We have proposed a notion of progressive forward simulation implying strong observational refinement. In future work, we will investigate whether the reverse direction also holds.

References

1. H. Attiya and C. Enea. Putting strong linearizability in context: Preserving hyperproperties in programs that use concurrent objects. In J. Suomela, editor, *DISC*, volume 146 of *LIPIcs*, pages 2:1–2:17. Schloss Dagstuhl, 2019. doi:10.4230/LIPIcs.DISC.2019.2.
2. M. R. Clarkson and F. B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.
3. J. Derrick, S. Doherty, B. Dongol, G. Schellhorn, and H. Wehrheim. On strong observational refinement and forward simulation, 2021. arXiv:2107.14509.
4. V. Luchangco, M. Moir, and N. Shavit. Nonblocking k-compare-single-swap. In *SPAA*, pages 314–323. ACM, 2003. doi:10.1145/777412.777468.