# Pushdown Normal-Form Bisimulation: A Nominal Context-Free Approach to Program Equivalence (Extended Abstract)

Vasileios Koutavas[1], Yu-Yang Lin[1], and Nikos Tzevelekos[2]

[1] Trinity College Dublin
[2] Queen Mary University of London

**Abstract.** We present Pushdown Normal Form (PDNF) Bisimulation for verifying contextual equivalence in higher-order functional languages with local state. PDNF matches the expressiveness of Normal-Form bisimulation while remaining decidable for programs with unbounded call stacks. It relies on the fact that nested reentrant traces form a pushdown system, enabling reachability to be simulated via finite-state automata. We define PDNF over a stackless LTS with on-the-fly saturation, and implement it in a tool that verifies examples previously out of reach. Published at LICS 2024 [5].

*Background.* Contextual equivalence asks whether two programs exhibit the same operational behaviour in all contexts. Normal-Form (NF) bisimulation addresses this by symbolically handling unknown higher-order arguments. However, NF bisimulation can lead to unbounded exploration, making verification intractable in practice and undecidable in general. In particular, nested reentrant calls between a term and its context create unbounded stacks during verification. This pattern commonly arises in callback-driven code, e.g., event listeners, reactive systems, GUI frameworks, and many other uses of the Observer pattern.

*Method.* We present Pushdown Normal Form (PDNF) Bisimulation, an alternative to NF bisimulation that finitises verification for programs with unbounded nested or reentrant calls. PDNF is defined over a stackless LTS with an on-the-fly saturation procedure[1,2], abstracting away the call stack without loss of precision. This relies on the fact that such traces form a context-free language, which can be simulated by finite-state automata recognizing initial/final stack content. Our approach parallels exact-stack analyses in control-flow analysis[3], and yields a decidable equivalence for terms reaching bounded stackless configurations, while retaining support for up-to techniques, such as garbage collection.

*Example 1.* Consider the following equivalent terms, the NF bisimulation game of which is depicted in Fig. 1 (left).

$M =$
```
let x = ref 0 in
fun f -> f(); !x
```

$N =$
```
fun f -> f(); 0
```

$V =$
```
fun f -> f(); !l
(for location l)
```

$V' =$
```
fun f -> f(); 0
```

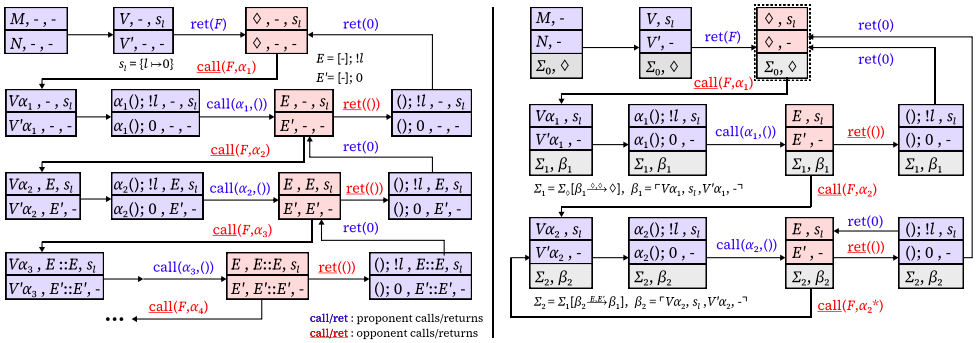
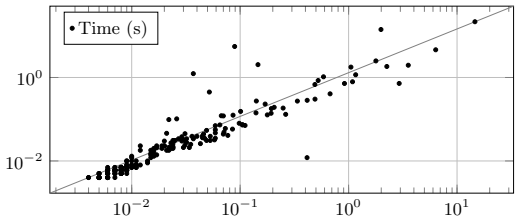



**Fig. 1.** NF bisimulations for terms $M$ and $N$ (Example 1); standard/stacked (left) and pushdown/stackless (right). We use "$\diamond$" to denote the top-level continuation and entry point, and set $\Sigma_0$ to be the empty continuation graph. We write "-" for the empty store and continuation stack.

| | HOBBIT tests: 129 eq's and 78 ineq's | | | HOBBIT-PDNF tests: 12 eq's | |
|---|---|---|---|---|---|
| | PDNF | HOBBIT | H+reentry | PDNF | HOBBIT |
| Eq. Proven | 72 | 62 | 67 | 11 | 0 |
| Ineq. Proven | 77 | 78 | 78 | N/A | N/A |




(X) HOBBIT vs. (Y) HOBBIT-PDNF over HOBBIT's test suite

**Table 1.** Summary of experiments comparing HOBBIT-PDNF to HOBBIT

*Implementation.* We implemented PDNF bisimulation in a prototype tool, HOBBIT-PDNF [6], by modifying the HOBBIT tool [4] for ML-like programs. HOBBIT-PDNF replaces HOBBIT's LTS and bisimulation with our Stackless LTS and PDNF bisimulation, while reusing its front-end, reduction semantics, most enhancement techniques, and Z3-based constraint solving. The tool is in principle sound (no false results) and bounded-complete (explores all paths up to a bound).

We evaluated HOBBIT-PDNF against HOBBIT, a reference implementation of standard (stacked) bisimulation, on a combined test suite of 6680 lines of code. Experiments ran with a 150-second timeout on an Ubuntu 23.04 machine equipped with an Intel Core i7 (1.90GHz) CPU and 32GB RAM, running OCaml 4.10.0 and Z3 4.8.10. Results are shown in Table 1. HOBBIT-PDNF performed strictly better on equivalences, timing out on one inequivalence. HOBBIT-PDNF correlates linearly against HOBBIT, suggesting comparable performance.

## References


1. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In A. W. Mazurkiewicz and J. Winkowski, editors, *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.
2. A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. In F. Moller, editor, *Second International Workshop on Verification of Infinite State Systems, Infinity 1997, Bologna, Italy, July 11-12, 1997*, volume 9 of *Electronic Notes in Theoretical Computer Science*, pages 27–37. Elsevier, 1997.
3. T. Gilray, S. Lyde, M. D. Adams, M. Might, and D. V. Horn. Pushdown control-flow analysis for free. In R. Bodík and R. Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 691–704. ACM, 2016.
4. V. Koutavas, Y. Lin, and N. Tzevelekos. From bounded checking to verification of equivalence via symbolic up-to techniques. In D. Fisman and G. Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Proceedings, Part II*, volume 13244 of *Lecture Notes in Computer Science*, pages 178–195. Springer, 2022.
5. V. Koutavas, Y.-Y. Lin, and N. Tzevelekos. Pushdown normal-form bisimulation: A nominal context-free approach to program equivalence. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '24, New York, NY, USA, 2024. Association for Computing Machinery.
6. Y.-Y. Lin. Laifsv1/hobbit-pdnf: Initial release, May 2024.