HOMER: A Higher-Order Observational Equivalence Model checkER*

David Hopkins and C.-H. Luke Ong

Oxford University Computing Laboratory

Abstract. We present HOMER, an observational-equivalence model checker for the 3rd-order fragment of Idealized Algol (IA) augmented with iteration. It works by first translating terms of the fragment into a precise representation of their game semantics as visibly pushdown automata (VPA). The VPA-translates are then passed to a VPA toolkit (which we have implemented) to test for equivalence. Thanks to the fully abstract game semantics, observational equivalence of these IA-terms reduces to the VPA Equivalence Problem. Our checker is thus sound and complete; because it model checks *open* terms, our approach is also compositional. Further, if the terms are inequivalent, HOMER will produce both a game-semantic and an operational-semantic counter-example, in the form of a play and a separating context respectively. We showcase these features on a number of examples and (where appropriate) compare its performance with similar tools. To the best of our knowledge, HOMER is the first implementation of a model checker of 3rd-order programs.

1 Theory and Implementation

Motivation. Higher-order functions are commonly used in functional programming. The functions map and foldr are standard examples of 2nd-order programs. 3rd and higher-order functions arise naturally in language processors [10]. Higher-order programs also crop up in imperative / object-oriented languages. E.g. any algorithm or data structure parameterised by, say, a comparison function is 2nd-order. A program that relies on such a 2nd-order function (being defined in an external library, say) is 3rd-order. Perhaps the most significant higher-order program is Google's MapReduce system [11]. Here we present the first model checker for 3rd-order programs.

Reynold's ***Idealized Algol*** (IA) [9] is a higher-order procedural language that combines imperative constructs (such as block-allocated assignable variables, sequencing and iteration) with higher-order functional features. It is essentially a call-by-name variant of (core) ML. E.g. the imperative term $\mathsf{while}\ !X > 0\ \mathsf{do}\ \{Y := !Y * !X\ ;\ X := !X - 1;\}$ and the lambda-term $\lambda f^{A\to B\to C}.\lambda g^{A\to B}.\lambda x^{A}.f\,x\,(g\,x)$ are both valid in IA.

Here we consider the fragment of IA containing up to 3rd-order terms over finite base types. I.e. we allow functions of types $((b_1 \to b_2) \to b_3) \to b_4$, say, where each b_i is one of the base types: com (commands), exp and var (expressions and variables respectively, with values taken from a finite prefix of the natural numbers). In addition, we allow while-loops but not full recursion. We denote this fragment IA^*_3.

* We thank A. Murawski for useful discussions and Microsoft Research PhD Scholarship Programme for funding this work. HOMER builds on and extends Hopkins' dissertation [7].

A. Bouajjani and O. Maler (Eds.): CAV 2009, LNCS 5643, pp. 654–660, 2009.

Two terms $\Gamma \vdash M_1, M_2 : A$ are ***observationally*** (or *contextually*) ***equivalent***, written $\Gamma \vdash M_1 \cong M_2$, just if for every program context $C[-]$ such that both $C[M_1]$ and $C[M_2]$ are closed terms of type com, $C[M_1]$ converges if and only if $C[M_2]$ converges. I.e. two terms are observationally equivalent whenever no program context can possibly distinguish them. An intuitively compelling notion of program equivalence, observational equivalence has a rich theory. For example, $\lambda x^{exp}.\mathsf{new}\, X \,\mathsf{in}\, \{X := x;\, !X\} \;\cong\; \lambda x^{exp}.x$, because the internal workings of the function are not detectable from the outside. However, these terms are not equivalent to $\lambda x^{exp}.\mathsf{if}\, x \,\mathsf{then}\, x \,\mathsf{else}\, x$ — because expressions can have side-effects, the outcome of evaluating x twice may be different from evaluating x only once. A much less obvious equivalence is

$$p : com \to com \;\vdash\; \mathsf{new}\, x := 0 \,\mathsf{in}\, \{p\,(x := 1);\, \mathsf{if}\, !x = 1 \,\mathsf{then}\, \Omega \,\mathsf{else}\, \mathsf{skip}\} \quad\cong\quad p\;\Omega$$

where Ω is the term that immediately diverges. This example shows that "snapback" (i.e. a term that first evaluates its com-type arguments and then immediately undoes their side-effects) is not definable in IA. The above equivalence holds because in either case, if p ever evaluates its argument, the computation will diverge.

Game Semantics. The *fully abstract*[1] game semantics [1] of IA has proved extremely powerful. In this model, a type A is interpreted as a game $[\![A]\!]$ between P and O, and a term $\Gamma \vdash M : A$ as a *strategy* $[\![\Gamma \vdash M]\!]$ for P to play in the game $[\![\Gamma \vdash A]\!]$. A strategy is just a set of plays (forming a playbook for how P should respond at each step he is to play), and a *play* is a sequence of moves, each is a question or an answer equipped with a *pointer*[2] to an earlier move. A play is *complete* if every question in it has been answered, so the game has run to completion.

The highly accurate game semantics characterises observational equivalence in terms of complete plays i.e. $\Gamma \vdash M_1 \cong M_2$ if and only if $comp[\![\Gamma \vdash M_1]\!] = comp[\![\Gamma \vdash M_2]\!]$, where $comp\,\sigma$ is the set of complete plays in strategy σ. Murawski and Walukiewicz [8] have shown that the complete plays in the strategy denotation of an IA^*_3-term are recognisable by a visibly pushdown automaton.

Visibly Pushdown Automata. The *visibly pushdown automata* (VPA) [3] are a subclass of pushdown automata in which the stack actions (push, pop or no-op) are completely determined by the input symbol. They are more expressive than finite automata, yet enjoy many nice closure properties of the regular languages. Because they are closed under complementation and intersection, and have a decidable emptiness problem, the *VPA Equivalence Problem* ("Given two VPA, do they accept the same language?") is decidable. So by representing the set of complete plays in a strategy denotation of an IA^*_3-term as a VPA, it is decidable (in EXPTIME [8]) if a given pair of β-normal IA^*_3-terms are observationally equivalent.

Implementation. Following the algorithm of Murawski and Walukiewicz [8], we have created a tool, called HOMER, for checking observational equivalence of IA^*_3 terms-in-context. Given two such terms, it first translates them to their VPA representations.

[1] Full abstraction is a strong goodness-of-fit measure. A denotational semantics is *fully abstract* if the theory of equality induced by the semantics coincides with observational equivalence.

[2] Which models the operand-to-operator, and variable-to-binder relation within a term.

These are then fed into a VPA toolkit, which we have created, to check for equivalence by complementing, intersecting and emptiness-checking to test for inclusion in both directions. The complementation and intersection operations are straightforward implementations from [3]. Since the VPA-translates are deterministic by construction, they are complemented just by complementing the final states. Intersection is by a product construction (which works for VPA – but not general PDA – because the two automata always perform the same stack action). More complex is the emptiness test. We experimented on a few algorithms before settling on Schwoon's pre* algorithm for general PDA, [12]. When the two terms are inequivalent, this will produce as a counter-example a play recognisable by exactly one of the two VPA-translates. The tool will use this play to generate a *separating context* - a context that converges when its hole is filled by one term, but diverges when filled by the other. HOMER is written in about 8 KLOC of $F\#$, including about 600 LOC for the VPA toolkit.

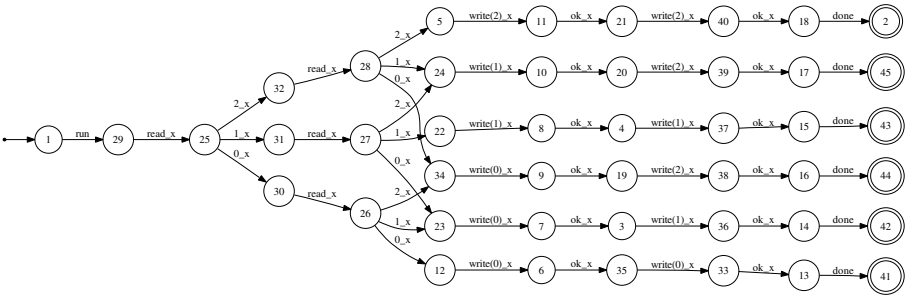
2 Evaluation and Tests

All tests in the following have been run on a laptop with a 2.53GHz Intel Core 2 Duo processor and 4GB RAM under Windows Vista. The base type `exp%N` *is* $\{0, 1, \cdots N-1\}$. *Unless specified otherwise,* `exp` *coincides with* `exp%N`*, which defaults to* `N` = *3.*

Sorting. Verifying sorting programs is a challenging test for model checkers because of the complex interplay between data flow and control flow [2]. An implementation of bubble sort is given below. Input to the program takes the form of an array of elements of the set $\{0, 1, 2\}$; we evaluate the model for different values of n, the length of the array. The program communicates with its environment only through the non-local variable `x`. In a call-by-name setting, because `x` can represent any var-typed procedure, it is legitimate to use it as an input/output stream. The program initially populates the array by reading from `x`, and after sorting writes the values back to `x` in order. These reads and writes are the only actions visible from the outside.

```
1  x : var%3 |-
2  new a[N]%3 in
3  {new i%N+1 in while !i < N do {a[!i] := !x; i := (!i + 1)}};
4  {
5    new flag%2 in
6    flag := 1;
7    while !flag do{
8      new i%N in
9      flag := 0;
10     while !i < N-1 do{
11       if !a[!i] > !a[!i + 1] then{
12         new temp%3 in
13         flag := 1;
14         temp := !a[!i] ;
15         a[!i] := !a[!i + 1];
16         a[!i + 1] := !temp
17       }
18       else skip;
19       i := !i + 1
20     }
21   }
22 };
23 {new i%N+1 in while !i < N do {x:= !a[!i];i := !i + 1}}
```

The automaton produced when $n = 2$ is shown in the following using Graphviz[3]. Since this is a 1st-order program, the VPA-translate degenerates to a deterministic finite automaton. It can be seen that there is a trace through the automaton for each of the 9 possible input combinations. [4]



Since the internals of the sorting program are hidden, bubble sort is observationally equivalent to any other sorting algorithm. Replacing the body of the bubble sort algorithm above (but keeping the same input and output format) with the code for insertion sort, we can feed them both into HOMER and check that (for a given value of n) they are in fact equivalent. (Timing data are provided under *Related Work* in the sequel.)

Kierstead Terms. An interesting 3rd-order example are the family of *Kierstead terms*, $K_{n,i} := \lambda f^{(o\to o)\to o}.f(\lambda x_1.f(\lambda x_2.\cdots f(\lambda x_{n-1}.f(\lambda x_n\,.\,x_i))))$, where $1 \leq i \leq n$, and o is a base type. $K_{2,1}$ and $K_{2,2}$ are the simplest pair of terms for which the location of pointers is critical. In the VPA-translate, each 3rd-order question has a tag that indicates the target of its pointer. Over base-type com, the table below gives the times to compile $K_{n,i}$, and to determine $K_{n,i} \ncong K_{n,j}$ whenever $i \neq j$, as n varies.

n	Time (Compile)	Time (Compare)	State Space (Final)	State Space (Max)
10	1.5s	2.6s	64	1172
20	3.8s	6.5s	97	3340
80	35s	70s	250	8848
160	220s	7.5 mins	730	26128

Counter-Example: Separating Context. It is not at all obvious how to construct by hand a context that witnesses inequivalent Kierstead terms. Below is a context that separates $K_{2,1}$ and $K_{2,2}$, as generated by HOMER (suitably adjusted for readability). By following the (somewhat complex) execution process, we can see how the assignments and tests on the local variable `X` force the program to follow the single path from which the context was derived: it terminates on $K_{2,2}$, but diverges on $K_{2,1}$ when it tries to run its arguments' argument from the first time it is called (thus deviating from the path).

[3] `http://www.graphviz.org/`

[4] HOMER uses a somewhat naïve forward reachability checker to remove unreachable states at each intermediate stage. This is very fast, and reduces the state space considerably, but it does not make any attempt to merge bisimilar states or delete states that can never reach a final state.

```
(   fun G:(((com -> com) -> com) -> com).new X in
  X:=1;
  G ( fun z:(com -> com).
    if !X = 1 then
    (
      X:=2;
      z omega;
      if !X = 5 then X:=6 else omega
    )
    else if !X = 2 then
    (
      X:=3;
      z (if !X = 3 then X:=4 else omega);
      if !X = 4 then X:=5 else omega
    )
    else
      omega
  );
  if !X = 6 then X:=7 else omega
)([-])
```

Shortest Prefix. Let us represent a sequence of numbers by a function of type $exp \to exp$ that maps i to the ith value in the sequence. We assume that if i exceeds the length of the sequence then evaluation diverges. A predicate over such sequences has type $(exp \to exp) \to exp \to bool$ where the second argument is the length of the sequence being tested. Given such a predicate and a sequence, we can find the length of the shortest prefix of the sequence that satisfies the predicate using the following 3rd-order program.

```
|- fun p : (exp%N -> exp%M) -> exp%N -> exp%2 . fun xs : exp%N -> exp%M .
  new i%N in
  new done%2 in
  new found%2 in
  while not !done do{
    if (p (fun index:exp%N . if index > !i then omega else (xs index)) !i) then {
      found := 1;
      done := 1
    }
    else if !i = N-1 then done := 1 else i := !i + 1
  };
  if !found then !i else omega
```

We construct sequences that agree with `xs` up to index `i` and then diverge and check whether `p` holds for such a sequence. The boolean flags are required because of a restriction on comparing values of different types (e.g. `exp%N` and `exp%M` with unequal `N` and `M`). Fixing the type of the data to `exp%3` and varying the size of the input sequence, the timing and state space data are below.

n	Time (Compile)	State Space (Final)	State Space (Max)
5	2s	197	4706
10	4s	592	18486
15	9s	1187	47966
20	18s	1982	99146

Related Work. To our knowledge, HOMER is the first implementation of a model checker of 3rd-order programs. The tools most similar to ours are MAGE [4], and

GAMECHECKER [5]. Though these are model checkers based on game semantics, they only check up to 2nd-order programs, which are much simpler as they can be represented by finite automata. Further they check for reachability rather than equivalence, so a fair comparison is thus hard to make even for 2nd-order programs. However, if we augment bubble sort so that as it outputs each of the values, it also asserts that each element is less than the next one, we can use MAGE to check that none of the assertions fail.

In the table below, compile time is the time to just compile the model, paramaterised by n, the size of the array. Compare time is the total time to compile both bubble and insert sort and check that they are observationally equivalent. The times for MAGE are how long it took to assert that bubble sort always results in a sorted array.

n	Compile	Compare	State Space (Final)	State Space (Max)	Time (MAGE)
5	1.3s	2.3s	495	1983	0.6s
10	55s	7.5 mins	60501	353613	154s

For $n = 15$, both MAGE[5] and HOMER gave up complaining of insufficient memory.

Conclusions. Since HOMER is (we believe) the first third-order model checker, there are no benchmarks against which we can properly measure its performance. When compared to MAGE, HOMER's performance is of the order of magnitude. Given the complexity of model-checking observational equivalence of third-order problems (EXPTIME-complete), this is encouraging. Further, as this is a prototype implementation, we used a simple explicit representation of the state-space. We expect a significant performance gain when techniques that have proved successful in other model-checkers (including MAGE), such as symbolic representation, lazy model checking or CEGAR, are exploited. Another encouraging point is that the final models produced are compact. Though the intermediate VPA have larger state-spaces, they are still much smaller than that examined by a naive state space exploration.

References

1. Abramsky, S., McCusker, G.: Linearity, sharing and state. In: Algol-Like Langs. (1997)
2. Abramsky, S., et al.: Applying Game Semantics to Compositional Software Modelling and Verification. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 421–435. Springer, Heidelberg (2004)
3. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Proc. STOC (2004)
4. Bakewell, A., Ghica, D.R.: On-the-fly techniques for game-based software model checking. In: Proc. TACAS (2009)
5. Dimovski, A.S., Ghica, D.R., Lazic, R.: Data-abstraction refinement. In: Hankin, C., Siveroni, I. (eds.) SAS 2005. LNCS, vol. 3672, pp. 102–117. Springer, Heidelberg (2005)
6. Ghica, D.R., McCusker, G.: Reasoning about Idealized Algol Using Regular Languages. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, p. 103. Springer, Heidelberg (2000)

[5] It may have been possible to increase the memory available to MAGE's Haskell compiler.

7. Hopkins, D.: A model checker for a higher-order procedural language. MCompSc dissertation, University of Oxford (2009), http://users.comlab.ox.ac.uk/luke.ong/publications/HopkinsReport.pdf
8. Murawski, A., Walukiewicz, I.: Third-order idealized algol with iteration is decidable. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 202–218. Springer, Heidelberg (2005)
9. Reynolds, J.C.: The essence of Algol. In: Algorithmic Languages. North-Holland, Amsterdam (1981)
10. Okasaki, C.: Even higher-order functions for parsing. J. Funct. Program. (1998)
11. Cataldo, A.: The Power of Higher-Order Composition Languages in System Design. Ph.D thesis, UC Bekerley (2006)
12. Schwoon, S.: Model-Checking Pushdown Systems. Ph.D thesis, Tech. Univ. of Munich (2002)