Automatically Verifying Temporal Properties of Programs with Cyclic Proof

Gadi Tellez
Department of Computer Science
University College London
gadi.tellez.13@ucl.ac.uk

James Brotherston
Department of Computer Science
University College London
j.brotherston@ucl.ac.uk

We propose a deductive reasoning approach to the automatic verification of temporal properties of pointer programs, based on *cyclic proof*. The simplicity and expressive power of temporal logics allow us to verify complex *safety* ("something bad cannot happen") and *liveness* properties ("something good eventually happens") of heap programs [11].

As an example consider the following nonterminating program that nondeterministically alternates between emptying the heap and appending an arbitrary number of elements to the head of a list structure.

```
while(true){
    if(*) {
        while(x!=nil) {
            temp:=x.next; free(x); x:=temp;
    } else {
        while(*) {
            y:=new(); y.next:=x; x:=y;
} } }
```

Employing temporal logic allows us to not only verify common safety and nonterminating properties, but we could also express more elaborated ones, say for example, that it is always possible for the heap to be empty, expressed in CTL as $AGEF(\mathsf{emp})$,

Historically, perhaps the most popular approach to ensuring that a program has a given temporal logic property has been *model checking*: one first builds an abstract model that overapproximates all possible executions of the program, and then checks that the desired temporal property holds of this model (see e.g. [4, 5, 2]). However, this approach has been applied in the literature mainly to integer programs; the situation for memory-aware programs over heap data structures becomes significantly more challenging, mainly because of the difficulties in constructing suitable abstract models.

One possible approach is simply to translate such heap-aware programs into integer variables, in such a way that properties such as memory safety or termination of the original program follows from a corresponding property in its integer translation [12, 4, 5]. However, for more general temporal properties, this technique might produce unsound results. In general, it is not clear whether it is feasible to provide suitable translations from heap to integer programs for any temporal property we might wish to prove.

Here, we instead approach the above problem via the main (perhaps less fashionable) alternative to model checking, namely the *direct deductive verification* of pointer problems. We formulate a proof system manipulating temporal judgements about programs, and attempt to directly construct a proof that a program has a given temporal property by means of an automatic proof search.

Given some fixed program, the judgements of our system express a temporal property of the program when started from any state satisfying precondition written in the *symbolic heap* [6] fragment of *separation logic* [14] extended with user-defined inductive predicates [8], a well-known language for abstractly describing the heap memory.

We examine two concrete temporal languages, the first based on *computation tree logic* (CTL) [1], where the temporal operators quantify over execution paths from a given program state, and the second on *linear time logic* (LTL) [13] where the temporal operators quantify events along a given *path*. To achieve the notion of seeing the execution of a program as a collection of traces, a different handling of nondeterminism in our system is required, for which we employ the technique of *prophecy variables* (cf. [3]) that effectively predict the outcome of nondeterministic choices.

The core of our proof systems is a set of *symbolic execution* rules that simulate program execution steps, so that there is a direct and natural correspondence between traces of the program and paths of judgements in our proofs. To handle the fact that symbolic execution can in general be applied *ad infinitum*, we employ the increasingly popular technique of *cyclic proof* [15, 7, 8, 9], in which proofs are finite cyclic graphs subject to a global soundness condition. Using this approach, we are able to verify temporal properties in an automatic and sound way.

Our implementation is built on top of the CYCLIST theorem prover [9], a mechanised cyclic theorem proving framework. The core algorithm performs iterative depth-first proof search, aimed at closing open nodes in the derivation tree by either applying an inference rule or forming a back-link to an identical previous node previously discovered. The automated application of derivation rules is largely guided by the structure of the temporal property required to prove.

TAPAS 2016, September 7th 2016, Edinburgh, UK
ACM ISBN .
DOI:

We first attempt to form back-links as early as possible to reduce the size of our explorations. If a back-link cannot be soundly formed, we instead attempt to apply symbolic execution; if this is not possible, we try unfolding temporal operators and inductive predicates in the precondition to enable symbolic execution to proceed. When all open nodes have been closed, a global soundness check of the cyclic proof is performed automatically. The details of the implementation, source code and benchmarks are publicly available at [10].

We evaluate our tool on handcrafted nondeterministic and nonterminating programs that operate on heap data structures similar to the example previously introduced. Moreover, we also cover sample programs taken from the Windows Update system, the back-end infrastructure of the PostgreSQL database server and an implementation of the acquire-release algorithm taken from previous model checking papers [3, 4]. Our experiments demonstrate the viability of our approach since our runtimes are mostly in the range of milliseconds and show similar performance to existing tools for the model checking benchmarks. The main advantages of our approach are that, thanks to the use of separation logic and a deductive proof system, we do not need to apply approximation or transformations to the program before attempting to verify it, and, furthermore, we never obtain false positive results. However, we do not guarantee termination or completeness (and achieving both is of course impossible). Thus we believe that our technique offers an interesting and potentially useful complement to model checking approaches (although not a replacement for them).

1. REFERENCES

[1] E. Clarke and A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs, Workshop*, pages 52–71, 1981.

[2] E. Clarke and D. Kroening and F. Lerda. A Tool for Checking ANSI-C Programs In Proceedings of TACAS , pages 168–176, 2004.

[3] B. Cook and E. Koskinen. Making Prophecies with Decision Predicates In Proceedings of POPL-38, pages 399–410, 2011.

[4] B. Cook and E. Koskinen. Reasoning About Nondeterminism in Programs In Proceedings of PLDI-34, pages 219–230, 2013.

[5] B. Cook and H. Khlaaf and N. Piterman. On Automation of CTL* Verification for Infinite-State Systems In Proceedings of CAV-27, 2015.

[6] J. Berdine and C. Calcagno and P. W. O'Hearn. A Decidable Fragment of Separation Logic. In Proceedings of FSTTCS-24, pages 97–109. Springer-Verlag, 2004.

[7] J. Brotherston. Formalised inductive reasoning in the logic of bunched implications. In *Proc. SAS-14*, volume 4634 of *LNCS*, pages 87–103. Springer-Verlag, 2007.

[8] J. Brotherston, R. Bornat, and C. Calcagno. Cyclic proofs of program termination in separation logic. In *Proceedings of POPL-35*, pages 101–112. ACM, 2008.

[9] J. Brotherston, N. Gorogiannis, and R. L. Petersen. A generic cyclic theorem prover. In *Proceedings of APLAS-10*, LNCS, pages 350–367. Springer, 2012.

[10] Cyclist: software distribution. https://github.com/ngorogiannis/cyclist/.

[11] L. Lamport. Proving the Correctness of Multiprocess Programs. In IEEE Trans. Software Eng., pages 125–143, 1977.

[12] S. Magill and M. Tsai and P. Lee and Y. Tsay. Automatic numeric abstractions for heap-manipulating programs. In *Proceedings of POPL-37*, pages 211–222, 2010.

[13] A. Pnueli. The Temporal Logic of Programs. In *Proceedings of FOCS-18*, pages 46–57. IEEE Computer Society, 1977.

[14] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of LICS-17*, pages 55–74. IEEE Computer Society, 2002.

[15] C. Sprenger and M. Dam. On the structure of inductive reasoning: circular and tree-shaped proofs in the μ-calculus. In *Proceedings of FOSSACS-6*, volume 2620 of *LNCS*, pages 425–440. Springer-Verlag, 2003.