HIPrec : Verifying Recursive Programs with a Satisfiability Solver

Quang Loc Le, Muoi Tran, and Wei-Ngan Chin

Department of Computer Science, National University of Singapore

Abstract. HIPrec is based on an automated verification framework to reason with recursive integer programs. It uses a novel satisfiability solver that works via inductive predicates. The key idea of HIPrec is its reduction approach to the verification problem by utilizing an unfolding-based satisfiability that can precisely capture program semantics. This paper describes the verification approach taken by HIPrec and provides instructions on how to install and use the tool.

1 Verification Approach

HIPrec verifies recursive programs by transforming each integer program into a set of inductive predicates with precise program semantics; and reduces the error reachability problem to satisfiability solving problem. There are a number of existing verification tools targeted on verification of recursive programs [1–3]. This paper gives an informal overview of our approach and key components that were required to handle the recursive benchmarks from the verification competition. Interested readers can find more details on the theory behind HIPrec in [4].

HIPrec builds upon the HIP system [5] and S2SAT satisfiability solver [4]. HIP system was originally developed as a platform for automated verification of separation logic-based specification for heap-manipulating programs. It was also one of the early pioneers for an expressive specification system where users are allowed to write and reason with non-trivial inductive predicates. Such predicates can describe recursive data structures, such as AVL-trees that are hard to verify. Over the years, this automated verification platform has been gradually extended with a variety of new efficient and expressive features, including *specialization calculus* [6] to prune infeasible disjuncts while unfolding, *error calculus* [7] to find bugs, and *second-order bi-abduction* to support specification inference for pure property [8], and shape analysis [9].

```
1 int fibo(int n){                          6  int main(){
2  if(n<1) return 0;                        7   int(x=5);
3  else if(n==1) return 1;                  8   int result=fibo(x);
4  else return fibo(n − 1)+fibo(n − 2);     9   if(result! =5) __VERIFIER_error();
5 }                                         10  return 0; }
```

Fig. 1. fibo_5_true-unreach-call.c of Recursive category.

S2SAT [4] is a semi-decision procedure for an expressive fragment of separation logic over user-defined predicates with Presburger arithmetic properties. Our solver

combines model satisfaction and abstract interpretation. It iteratively refines disjunctive separation logic formulas via a *context-sensitive* unfolding mechanism. In each iteration, it searches for either one model (for satisfiable queries) or a proof of unsatisfiability on all disjuncts (for unsatisfiable queries). The unsatisfiability solving is realized by sound invariants from inductive predicates. These predicate invariants are inferred automatically and are particularly helpful for ensuring that algorithm terminates.

For illustration, we show how to employ HIPrec to verify those programs in Recursive category of the SV-COMP competition. In this category, a program is safe if error locations are unreachable. To verify it, each method m is first transformed into its equivalent recursive predicate m_v. The list of arguments of this predicate includes also a special `res` for output (if applicable) and another special parameter e to denote status of a program path ($e{=}0$ for safety and $e{=}1$ for error). Each program path in m corresponds to a disjunct in m_v. Let us illustrate further how we use the proposed satisfiability solver for verifying a fibonacci-like recursive program in Fig. 1. The methods `fibo` and `main` are transformed into predicates `fibo_v` and `main_v`, as follows.

$$\begin{array}{l}\texttt{pred}\ \texttt{fibo\_v}(n,\texttt{res},e)\ \equiv\ \texttt{emp}\wedge n{<}1\wedge\texttt{res}{=}0\wedge e{=}0\ \vee\ \texttt{emp}\wedge n{<=}1\wedge\texttt{res}{=}1\wedge e{=}0\\ \ \vee\ \texttt{fibo\_v}(n{-}1,r_1,e_1){*}\texttt{fibo\_v}(n{-}2,r_2,e_2)\wedge e_1{=}0\wedge e_2{=}0\wedge\texttt{res}{=}r_1{+}r_2\wedge e{=}0;\\ \texttt{pred}\ \texttt{main\_v}(\texttt{res},e)\ \equiv\ \texttt{fibo\_v}(x,\texttt{result},e_1)\wedge x{=}5\wedge e_1{=}0\wedge\texttt{result}{\neq}5\wedge e{=}1\\ \ \vee\ \texttt{fibo\_v}(x,\texttt{result},e_1)\wedge x{=}5\wedge e_1{=}0\wedge\texttt{result}{=}5\wedge\texttt{res}{=}0\wedge e{=}0\end{array}$$

We note that the first disjunct of `main_v` predicate encodes the errors that are denoted by the presence of $e{=}1$. Safety of the program is reduced to solving the following query: $\texttt{main\_v}(\_,e)\wedge e{=}1$. If S2SAT decides this query as *unsat*, then the error is unreachable. If S2SAT decides this query as *sat*, then the program is unsafe since the error is reachable. In the latter case, S2SAT returns counter-example from the satisfiable disjunct. From this counter-example, HIPrec symbolically traverses the input program to generate a witness. Interested readers may visit our HIPrec website to learn more about our approach for verifying safety of recursive programs.

2 Software Architecture

The system firstly transforms the C programs into a core language with the help of CIL [10], before executing it symbolically via Hoare-style rules. Our symbolic execution engine generates a set of inductive predicates and verification conditions. Finally, these conditions are then passed to S2SAT. To decide on pure logic formulas S2SAT utilizes off-the-shelf provers, such as Z3 [11] and Omega [12]. For finding closed-form approximation to recursive predicates, HIPrec employs an in-house developed disjunctive fix-point analyser, called FIXCALC.

3 Strengths and Weaknesses of the Approach

The main strength of our tool is an expressive satisfiability solver, named S2SAT. This can decide formulas over separation logic with (both heap and non-heap) inductive predicates. It allows us to gradually evolve our verification system into larger fragment of recursive programs. Our weakness is that HIPrec is presently limited to recursive

programs without heap. Although our satisfiability solver supports separation logic formulas, the translation engine presently works for only non-heap programs. This shortcoming is a source of inspiration for future works on our toolset.

4 Tool Setup and Configuration

HIPrec website: `http://loris-7.ddns.comp.nus.edu.sg/~project/hiprec`.
Download and Setup.
1. Download `hiprec.tar.gz` file from the tool website above.
2. Uncompress `hiprec.tar.gz`. The uncompressed folder, called `hiprec`, contains HIPrec and all auxiliary tools (provers, fixed point computation) to run HIPrec.
Run. Change current working directory to `hiprec` and run HIPrec as follows
`./hiprec /path/to/c_program_to_analyse`
HIPrec outputs a triple: Verification result:(ANSWER, witness:WITNESS, TIME(seconds)). ANSWER is one of TRUE, FALSE, UNKNOWN. When the ANSWER is FALSE, WITNESS captures a link for witness that shows how error is reachable.
Participation Statement. We participate in the Recusive sub-category.

References

1. P. Suter, A. Kksal, and V. Kuncak. Satisfiability modulo recursive programs. *Static Analysis*, vol. 6887, pp. 298–315, Lecture Notes in Computer Science, 2011.
2. Y. Chen, C. Hsieh, M. Tsai, B. Wang, and F. Wang. Verifying recursive programs using intraprocedural analyzers. *SAS*, pp. 118–133, 2014.
3. M. Brain, S. Joshi, D. Kroening, and P. Schrammel. Safety verification and refutation by k-invariants and k-induction. *SAS*, pp. 145–161, 2015.
4. Q. L. Le, S. Jun, and W.-N. Chin. Satisfiability modula heap-based programs. *CAV*, 2016.
5. W. N. Chin, C. David, H. Nguyen, and S. Qin. Automated verification of shape, size and bag properties via user-defined predicates in separation logic. *Sci. Comput. Program.* 77(9):1006–1036, 2012.
6. W.-N. Chin, C. Gherghina, R. Voicu, Q.-L. Le, F. Craciun, and S. Qin. A specialization calculus for pruning disjunctive predicates to support verification. *CAV*, 2011.
7. Q. L. Le, A. Sharma, F. Craciun, and W. N. Chin. Towards complete specifications with an error calculus. *NASA*, pp. 291–306, 2013.
8. M. Trinh, Q. L. Le, C. David, and W. N. Chin. Bi-Abduction with Pure Properties for Specification Inference. *APLAS*, 2013.
9. Q. L. Le, C. Gherghina, S. Qin, and W. N. Chin. Shape Analysis via Second-Order Bi-Abduction. *CAV*, 2014.
10. G. Necula, S. McPeak, S. Rahul, and W. Weimer. CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs. *CC'02*, pp. 213–228, 2002.
11. L. M. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. *TACAS*, 2008.
12. P. Kelly, V. Maslov, W. Pugh, and et al et al. *The Omega Library Version 1.1.0 Interface Guide*, Nov. 1996.