CPArec: Verifying Recursive Programs via Source-to-Sourcce Program Transformation

Yu-Fang Chen[1], Chiao Hsieh[1,2*], Ming-Hsien Tsai[1], Bow-Yaw Wang[1], and Farn Wang[2]

[1] Institute of Information Science, Academia Sinica, Taiwan
[2] Graduate Institute of Electrical Engineering, National Taiwan University, Taiwan

Abstract. CPArec is a tool for verifying recursive C programs via source-to-source program transformation. It uses a recursion-free program analyzer CPAChecker as a black box and computes function summaries from the inductive invariants generated by CPAChecker. Such function summaries enable CPArec to check recursive programs.

1 Verification Approach

The CPArec tool handles recursive programs by an iterative source-to-source transformation technique proposed in [2]. In each iteration, it transforms the original recursive program P into a non-recursive program P' that *under-approximates* the behaviors of P. The program P' will be sent to a black box program verifier V that does not support recursion. If an assertion violation in the program P' is found by the verifier V, it also indicates an assertion violation in the program P. Otherwise, the verifier should generate an *inductive invariant* as a proof for the unreachability of the assertion violation, from which CPArec extracts candidates of *function summaries.*

Based on Tarski's fix-point theorem, CPArec reduces the problem of checking the correctness of function summary candidates again to assertion checking. More specifically, it first replaces all function calls in P with the corresponding function summary candidates and obtain a new non-recursive program P''. Then it checks if all behaviors of P'' are included in the behaviors encoded in the function summary candidate of P. This step is again handled by a source-to-source program transformation with some additional assertions added. If the verifier V reports that all assertions are not violated, then CPArec found correct function summaries and thus proved the correctness of P. Otherwise, it produces a more refined version of P by unwinding the function calls and proceeds to the next iteration of the verification procedure. The execution flow of CPArec can be found in Figure 1.

2 Software Architecture

Currently, CPArec using CPAChecker (over 140 thousands lines of Java codes) as the underlying program analyzer [1]. Itself contains 1256 lines of

[*] Corresponding Author: bridge@iis.sinica.edu.tw

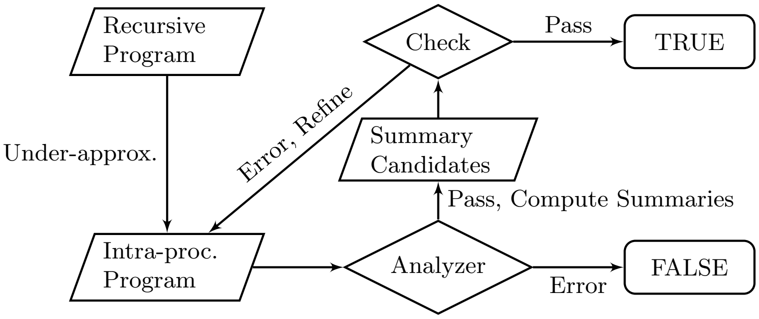

Fig. 1: The Execution Flow of CPArec

OCaml codes for syntactic source-to-source transformation using the CIL [3] library. The rest of the algorithm are implemented in 705 lines of Python codes. Among them only 270 lines are for extracting function summaries. Since syntactic transformation is independent of the underlying program analyzer, only about 14% of codes need to be rewritten should another analyzer be employed. When extracting summaries from inductive invariants, we sometimes need to quantify out additional variables that are neither formal parameters nor return variables. CPArec uses the tool RedLog [4] for quantifier elimination.

3 Strengths and Weaknesses

Compared with other analysis algorithms for recursive programs, the one implemented in CPArec is very *lightweight*. It only performs syntactic transformation and requires standard functionalities from underlying intraprocedure program analyzers. Moreover, our technique is very *modular*. Any intraprocedural analyzer providing proofs of inductive invariants can be employed by our tool. With the interface between CPArec and the program analyzers described in the previous section, incorporating recursive analysis with existing program analyzers thus only requires minimal implementation efforts. Recursive analysis hence benefits from future advanced intraprocedural analysis with little cost through our lightweight and modular technique.

On the other hand, we suffer the same limitation as the black-box analyzer. For instance, using CPAChecker, we can only produce *linear* summaries. However, in the recursive category of the competition, several examples require nonlinear summaries for proving correctness. Moreover, we get the modularity for the price of losing some flexibility. For example, we cannot optimize the way how the underlying program analyzer constructs the trace formula and sends to SMT solver. This step potentially can reduce the amount of variables that we need to quantify out and may benefit performance.

4 Setup and Configuration

CPArec is available at

```
https://github.com/fmlab-iis/cparec
```

The submitted version is v0.1-alpha. The simplest way to execute CPArec is to first download the binary from the web-site. To setup the environment in Ubuntu 12.04 64-bit, JAVA Runtime, Python 2.7, the Python Networkx package, and the Python PyGraphviz package are required. Run following command to install above packages in Ubuntu 12.04 64-bit:

```
sudo apt-get install openjdk-7-jre python python-networkx python-pygraphviz
```

To process a benchmark example `program.c`, one should use the following script:

```
python <path_to_cparec>cparec/main.py program.c
```

No further parameters are needed. CPArec will print the verification result to the console. We will only participate in the recursive category of the competition.

5 Software Project and Contributors

CPArec is an open-source project from the programming language and formal method (PLFM) group at the Institute of Information Science, Academia Sinica, Taiwan. The main contributors are the authors of this paper. The programs are written by Chiao Hsieh and Ming-Hsien Tsai.

References

1. Dirk Beyer and M. Erkan Keremoglu. CPAchecker: A tool for configurable software verification. In *CAV*, pages 184–190, 2011.
2. Yu-Fang Chen, Chiao Hsieh, Ming-Hsien Tsai, Bow-Yaw Wang, and Farn Wang. Verifying recursive programs using intraprocedural analyzers. In *SAS*, pages 118–133, 2014.
3. George C. Necula, Scott McPeak, Shree Prakash Rahul, and Westley Weimer. CIL: intermediate language and tools for analysis and transformation of C programs. In *CC*, pages 213–228, 2002.
4. Redlog. `http://www.redlog.eu/`.