HipTNT+: A Termination and Non-termination Analyzer by Second-Order Abduction
(Competition Contribution)

Ton Chanh Le(✉), Quang-Trung Ta, and Wei-Ngan Chin

School of Computing, National University of Singapore, Singapore, Singapore
{chanhle,taqt,chinwn}@comp.nus.edu.sg

Abstract. HipTNT+ is a modular termination and non-termination analyzer for imperative programs. For each given method, the analyzer first annotates it with an initial specification with second-order unknown predicates and then incrementally derives richer known specifications with case analysis. Subsequently, the final inference result indicates either (conditional) termination, non-termination, or unknown. During the proving process, new conditions for the case analysis are abductively inferred from the failure of both termination and non-termination proof, which aim to separate the terminating and non-terminating behaviors for each method. This paper introduces the verification approach and the structure of HipTNT+, and instructs how to set up and use the system.

1 Overview

HipTNT+ is an automated verification and inference system for the termination and non-termination properties of imperative programs [2,3]. The system is built upon the HIP/SLEEK toolset [1], a separation logic-based platform for automatically proving and inferring functional correctness of heap-manipulating programs. The development of HipTNT+ follows an incremental process, in which a verifier with an appropriate specification logic for reasoning about both program termination and non-termination is first developed, prior to augmenting it with specification inference capability. In our approach, the outcomes of inference mechanism are represented by an enriched specification logic, that can be optionally re-verified by the verifier constructed in the earlier phase. This development methodology is helpful for debugging a new inference mechanism that is being implemented. In contrast, the other analyzers simply represent their outcomes in some internal forms, without automated re-scrutiny.

2 Verification Approach

HipTNT+ has been developed based on two technical innovations proposed by Le et al., that are *(i)* a unified resource-based specification logic [2],

T.C. Le—Jury member.

DOI: 10.1007/978-3-662-54580-5_25

and *(ii)* an abductive specification inference mechanism [3] for reasoning about both program termination and non-termination *at the same time*. These approaches analyze the program terminating and non-terminating behaviors on a per-method basis, thus providing a modular, reusable and scalable proving technique for these program properties.

2.1 Termination Verification via Resource Reasoning

To specify and verify the termination and non-termination of a program, Le et al. [2] propose a unified specification logic with three temporal predicates $\mathtt{Term}\ M$, $\mathtt{Loop}$, and $\mathtt{MayLoop}$, denoting definite termination (with a lexicographic ing function M), definite non-termination and possible (unknown) non-termination, respectively. The formal semantics of these predicates can be uniformly defined using a *resource capacity* predicate $\mathtt{LC}(\mathtt{L}, \mathtt{U})$ with a lower bound $\mathtt{L}$ and an upper bound $\mathtt{U}$ on the execution length, as follows:

$$\mathtt{Term}\ M \triangleq \mathtt{LC}(0,\ f(M))\ \mathtt{Loop} \triangleq \mathtt{LC}(\infty,\ \infty)\ \mathtt{MayLoop} \triangleq \mathtt{LC}(0,\ \infty)$$

where f is an order-embedding from a finite list of non-negative expressions into naturals.

Intuitively, a program terminates if its execution length has a finite upper bound. On the other hand, a non-terminating program has an infinite lower bound on the execution length. Verification conditions involving these temporal predicates can be discharged in terms of resource reasoning via a resource consumption entailment $\vdash_t$. Given the current program state ρ with an execution resource θ_a and a code fragment that requires a resource θ_c to execute, the entailment $\rho \wedge \theta_a \vdash_t \theta_c \blacktriangleright \theta_r$ checks if the required resource θ_c can be met (or subsumed) by the current resource θ_a. If succeeded, the entailment returns the (largest) remaining execution resource, denoted by the residue θ_r, after θ_c is consumed in θ_a.

2.2 From Verification to Inference

To infer the termination specification of each method in a program, Le et al. [3] first enhance the proposed specification logic by a pair of *second-order* temporal pre-predicate, for precondition, and post-predicate, for postcondition, to capture the unknown status of (non-)termination properties. They then extend the resource entailment procedure to handle entailments with these unknown temporal predicates, and employs a Hoare-style forward verification to collect a set of relational assumptions on them.

From these relational assumptions, a comprehensive summary of both termination and non-termination behaviors of each program's method is incrementally constructed. Specifically, the pre-assumptions collected when proving preconditions at method calls guide the overall inference process and can be used to infer ing functions when proving termination. The post-assumptions collected when proving postconditions contain information about the reachability or unreachability of the method's exits. Therefore, they can be used *(i)* to determine base-case scenarios with obvious termination property, *(ii)* to prove inductive unreachability for a definite non-termination, and *(iii)* to derive new conditions for further case-split via an abductive inference from the failure proofs of definite termination and non-termination. Note that the ranking functions and the abductive conditions can be inferred by the constraint-based synthesis technique via Farkas' lemma.

The derived summary of the method's termination and non-termination characteristics is represented in the high-level specification logic, so that it can be reused in the inference of the remaining methods higher-up in the calling hierarchy. This enables better modularity and reuse for the proving process.

3 Software Architecture

As illustrated in Fig. 1, HipTNT+ has been built on top of the HIP/SLEEK platform, so that it can exploit the infrastructure of HIP/SLEEK, such as the front-end components, the Hoare-style verification, and the SMT solver's interface. Note that the annotated specifications are optional; when they are not given, the system automatically inserts a second-order specification for each method of the input program to trigger the inference process.

For reasoning about termination and non-termination, the core of HipTNT+ is made up of two main components:

– A prover for the resource-based termination logic. This prover implements the resource consumption entailment $\vdash_t$ to discharge verification conditions involving the temporal constraints. Moreover, it also generates a set of relational assumptions on the unknown temporal predicates as the input of the termination inference system.
– An abductive inference system for termination and non-termination analysis. This component implements the search procedure to simultaneously analyze the termination and non-termination behaviors of a program from a given set of relational assumptions, via case analysis with abductive inference.

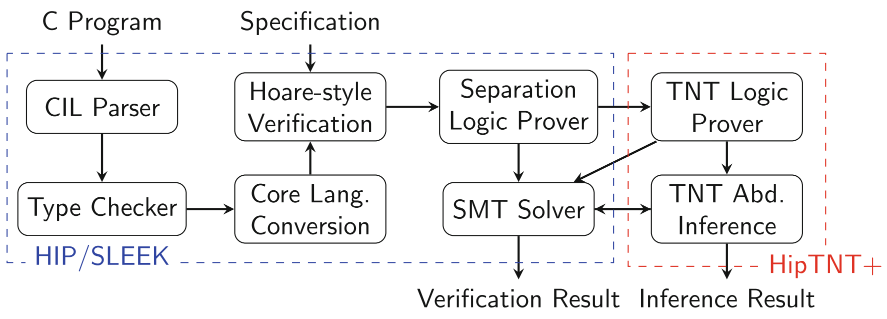

Fig. 1. Structure of HipTNT+

4 Strengths and Weaknesses

The incorporation of HipTNT+ in a verification toolset like HIP/SLEEK allows us to gradually evolve our termination analyzer with new capabilities. For example, HipTNT+ can analyze heap-based programs with ease because they are natively supported by HIP/SLEEK via separation logic. However, it is also our main weakness as we have to wait for the support from HIP/SLEEK to handle string-manipulating programs or programs with function pointers in the SV-COMP benchmarks.

5 Tool Setup and Configuration

Download and Installation. The competition submission of the HipTNT+ system is at version 2.0. A zip bundle containing a wrapper script and self-contained binaries of HipTNT+ v2.0 can be freely downloaded from http://loris-5.d2.comp.nus.edu.sg/hiptnt/plus/hiptnt_svcomp17.zip. The bundle also provides for your convenience executables of all needed third party provers, i.e., Omega Calculator[1] and Z3[2] provers.

To run the system, the wrapper script `hiptnt.sh` can be invoked via a command-line interface as follows: `./hiptnt.sh file.c`. Note that the current working directory must be the one that contains this wrapper. The system outputs the verification results, i.e., TRUE, FALSE + *Witness*, or UNKNOWN, to the console. A witness represents a counterexample to termination, indicating a feasible path of method call locations to a definite non-termination condition with Loop.

Participation Statement. HipTNT+ participates in the Termination category.

6 Software Project and Contributors

HipTNT+ is maintained by Ton Chanh Le, a member of the Software Verification research group, led by Wei-Ngan Chin, at the National University of Singapore. HipTNT+ is freely available for academic and non-commercial use at http://loris-5.d2.comp.nus.edu.sg/hiptnt/plus/. For third party provers, i.e., Omega Calculator and Z3, their original licensing requirements apply. Our thanks go to all contributors of the core verification system HIP/SLEEK. The full description of the system can be found at http://loris-5.d2.comp.nus.edu.sg/hip/, it is also where all of its contributors are listed.

Acknowledgement. Ton Chanh and Wei-Ngan are partially supported by the MoE Tier-2 grant MOE2013-T2-2-146.

[1] http://www.cs.umd.edu/projects/omega/.
[2] https://github.com/Z3Prover/z3.

References

1. Chin, W., David, C., Nguyen, H.H., Qin, S.: Automated verification of shape, size and bag properties via user-defined predicates in separation logic. Sci. Comput. Program. **77**(9), 1006–1036 (2012)
2. Le, T.C., Gherghina, C., Hobor, A., Chin, W.-N.: A resource-based logic for termination and non-termination proofs. In: Merz, S., Pang, J. (eds.) ICFEM 2014. LNCS, vol. 8829, pp. 267–283. Springer, Cham (2014). doi:10.1007/978-3-319-11737-9_18
3. Le, T.C., Qin, S., Chin, W.: Termination and non-termination specification inference. In: PLDI, pp. 489–498 (2015)