Formal Verification Analysis of Tritype Programs

Arshad Beg

May 26, 2025

Analysis of Verification on `baseline_Example1-Tritype.c`

The verification of `baseline_Example1-Tritype.c` using Frama-C and four SMT solvers (Alt-Ergo, Z3, CVC4, and CVC5) revealed a consistent pattern. With minimal ACSL specification, only two implicit verification goals—termination and unreachability—were generated and successfully verified by all four provers. This indicates that the function is syntactically well-formed and does not exhibit trivial issues like infinite loops or unreachable code paths. However, due to the absence of user-defined postconditions or preconditions, the analysis provides limited insight into functional correctness. The warning about the missing `assigns` clause suggests that memory side-effects are not specified, potentially causing inaccurate assumptions for callers. Similarly, the absence of RTE (Run-Time Error) guards indicates that common runtime errors like overflows or division by zero are not being verified. While the results demonstrate soundness at a structural level, the lack of deep specification significantly limits the utility of the verification. All provers perform equally under these trivial conditions.

Comparison of Prover Results on `baseline_Example1-Tritype.c`

Table 1: Prover Results for `baseline_Example1-Tritype.c`

Prover	**Total Goals**	**Proved**	**Notes**
Alt-Ergo	2	2	All default goals proved
Z3	2	2	All default goals proved
CVC4	2	2	All default goals proved
CVC5	2	2	All default goals proved

Analysis of Verification on `pathcrawler_augmented_Example1-Trityp`

The `pathcrawler_augmented_Example1-Tritype.c` file, enriched with detailed ACSL annotations, exhibits significantly different verification behavior. A total of 20 goals were generated, including 18 based on user-specified preconditions and postconditions, plus the standard termination and unreachability checks. The analysis reveals a distinct divide in prover effectiveness. While all provers verified termination and basic logic, their ability to handle complex ensures clauses varied. Z3 and CVC5 each failed to prove seven goals—Z3 due to timeouts and CVC4 due to unknown statuses—highlighting their struggles with intricate logical paths and case distinctions. Alt-Ergo and CVC5 fared slightly better, with only five unverified goals each. Notably, the most complex properties, such as correct classification of triangle types (Scalene, Isosceles, Equilateral) and handling of inequality rules, were consistently problematic across all provers. This reflects the challenges solvers face when dealing with disjunction-heavy logic or subtle arithmetic constraints embedded in functional specifications.

Comparison of Prover Results on `pathcrawler_augmented_Example1`

Table 2: Prover Results for `pathcrawler_augmented_Example1-Tritype.c`

Prover	Total Goals	Proved	Failed Type	Failed Count
Z3	20	13	Timeout	7
Alt-Ergo	20	15	Timeout	5
CVC4	20	13	Unknown	7
CVC5	20	15	Timeout	5

Summary and Recommendations

From the analysis, it is evident that enhancing ACSL specifications significantly increases the value of formal verification, albeit at the cost of greater computational effort and prover resource usage. The baseline version passed trivially due to lack of functional goals, offering limited assurance. In contrast, the augmented version uncovered practical challenges in the implementation logic of triangle classification. Complex postconditions, especially those involving mutually exclusive branches and arithmetic inequalities, frequently led to timeout or unknown statuses. To address these, it is recommended to introduce modular proofs by breaking down composite postconditions, use auxiliary lemmas, and ensure complete specifications including `assigns` clauses. Additionally, enabling RTE guards will allow safety verification alongside correctness. Future work may explore combining solvers or using Why3's backend to orchestrate multiple strategies. This evaluation highlights both the strength and current limitations of SMT-based automatic verifiers in verifying real-world conditional logic encoded in C programs.