# Meta-F*: Proof Automation with SMT, Tactics, and Metaprograms: An Overview

Caleb Bowers

## I. Overview

What follows is an attempt to provide an overview and some key insights into [1], which introduces the "tactics and metaprogramming framework for the F* program verifier" Meta-F*. The development and use of Meta-F* enables the verification of assertions not solvable by traditional SMT methods and can be used to generate verified code automatically. The paper on the whole focuses on providing the motivation, the theoretical foundations, the design, and experimental evaluation of Meta-F*. I specifically will cover the sections that address the tactics for assertions and canonicalization on an individual level (section 2.1) and section 6, which speaks to Meta-F*'s experimental evaluation.

### A. *Background*

Historically, there have been two somewhat independent approaches to verifying programs: *i*) tactics and metaprogramming (often called interactive theorem provers, IPT) and *ii*) computing verification conditions from programs. The former relies on specifying properties of pure programs in expressive higher-order logics with its proofs conducted using a variety of imperative programming languages, such as the functional language ML [2]. The latter make use of program code annotations that are encoded into an automated theorem solver such as an SMT solver. These two fields maintain an arm's length relationship, but this paper contends that wedding the two approaches more closely together would provide several benefits:

- *Increased Expressiveness*: Program users wishing to verify will no longer be limited by the automated theorem prover of choice, but can be extended to undecidable and difficult to automate theories.
- *Reduction in Boilerplate*: Users no longer need to construct excessively detailed proofs for every portion of the program, since ITPs provide exceptional domain specific automation to complete those proofs.
- *Explicit Access to Proof Context*: Rather than depending on the control flow of the program to provide logical proof context, IPTs provide explicit access to the proof context and can structure and explore the proof space as a result.

To accomplish the combining of IPTs and verification conditions computations, the authors have developed Meta-F*, which enables programmers to build code that is more easily verified, eliminate excessive annotation and proof creation for code, and to inspect the proof state to manipulate it to address logical contextual issues by combining aspects of IPT *tactics* and SMT solvers.

### B. *Tactics*

The authors provide an example of how IPT tactics can be used to reduce the work involved in verifying a condition on a program. They use the cryptographic MAC, Poly1305 [3]. The main takeaway from this example is that using SMT solvers requires the user to heavily guide the SMT solver by frequently applying lemmas manually at the proper points in the condition for the code, which can be quite error prone as proofs and programs grow, especially for verifying low-level cryptographic primitives such as the case is for parts of Poly1305. By using IPT tactics in combination with SMT solvers, users can reduce the dependence their proof has on lemmas by making use of IPT tactics to construct the canonicalized form of the tactic goal[1] for areas where they would have originally used lemmas to guide the SMT solver.

### C. *Experimental Evaluation of Meta-F**

The author's experimental evaluation of Meta-F* compares the verification of the previously mentioned Poly1305 assertion for different methods: SMT solver only, canonicalized tactics only, and the entire suite of Meta-F* capability. Their results ultimately demonstrate that implementing a hybrid solution as found in Meta-F*, implementing domain specific tactics for a proof in concert with SMT solvers provides a much more tractable solver approach than simply relying on one method to solve a given assertion, at least for this

[1] Canonicalized here means to specify a unique representation for every relevant object

given experiment. For their example, even if Meta-F* did not exist, a user would definitely want to make use of IPT tactics for this problem as it provides an entire order of magnitude gain in compute time. While this does not demonstrate IPT tactic superiority (nor do the authors claim it does), it does demonstrate that there exist problems where a hybrid approach provides significant gains and is worth the effort to implement.

## II. Conclusion

This paper, at times, was difficult to follow. The authors definitely relied on readers already possessing a significant amount of knowledge in the field of verification and IPT practices, and they would have benefited readers by providing a terms and definitions page. They also occasionally gave into the temptation to use bigger, more intelligent sounding words when a simpler word would have sufficed and not obscured their meaning. That being said, this framework demonstrates promise and relative ease of use, so if an SMT solver were not getting the job done, Meta-F* could provide a viable hybrid solution.

## References

[1] G. Martínez, D. Ahman, V. Dumitrescu, N. Giannarakis, C. Hawblitzel, C. Hritcu, M. Narasimhamurthy, Z. Paraskevopoulou, C. Pit-Claudel, J. Protzenko, T. Ramananandro, A. Rastogi, and N. Swamy, “Meta-f*: Metaprogramming and tactics in an effectful program verifier,” *CoRR*, vol. abs/1803.06547, 2018. [Online]. Available: http://arxiv.org/abs/1803.06547

[2] R. Milner, M. Tofte, and D. Macqueen, *The Definition of Standard ML.* Cambridge, MA, USA: MIT Press, 1997.

[3] D. J. Bernstein, “The poly1305-aes message-authentication code,” in *Fast Software Encryption*, H. Gilbert and H. Handschuh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 32–49.