Multiple Definitions from a Single Resolution Proof

Friedrich Slivovsky

School of Computer Science & Informatics, University of Liverpool, UK
f.slivovsky@liverpool.ac.uk

1 Motivation

Several problems in formal methods—Boolean functional synthesis [1], QBF, DQBF—ask for Boolean functions satisfying a propositional specification. Identifying implicitly defined variables and computing their explicit definitions can simplify these problems, and sometimes solve them outright.

By Beth's theorem [3], the explicit definition of a defined variable y in φ in terms of X is a Craig interpolant of $\varphi \wedge y \wedge \varphi' \wedge \neg y'$, where φ' renames every variable of φ outside X. A SAT solver produces a resolution refutation of this formula, from which an interpolation system [4] reads off the definition. This pipeline has been used effectively in preprocessing [5,2], but it requires a separate SAT call per defined variable, and on instances with many definitions these calls dominate the running time.

We propose an alternative that obtains all n definitions from a *single* SAT call: the solver refutes a propositional formula expressing joint implicit definability of the n targets, and a partial assignment recovers the formula for any individual target. We modify an interpolation system to compute an interpolant under such a partial assignment, and use this generalisation to extract all n definitions from the refutation in a single pass, in time and circuit size $O(nm)$. Experiments on Boolean functional synthesis benchmarks indicate that the bottleneck shifts from SAT solving to circuit size.

2 Joint Definability and Interpolation

Let $\varphi = \varphi(X, Y, Z)$ with $Y = \{y_1, \ldots, y_n\}$. Each y_i is defined by X in φ iff

$$\underbrace{\varphi(X, Y, Z)}_{A} \wedge \underbrace{\varphi(X, Y', Z')}_{B} \wedge \underbrace{\textstyle\bigwedge_{i=1}^{n} \big((y_i \vee y_i' \vee e_i) \wedge (\neg y_i \vee \neg y_i' \vee e_i)\big) \wedge \bigvee_{i=1}^{n} \neg e_i}_{G}$$

is unsatisfiable. The encoding of definability for an individual y_i is recovered under the partial assignment $\sigma_i = \{y_i, \neg y_i', \neg e_i\} \cup \{e_j : j \neq i\}$: group A simplifies to $\varphi \wedge y_i$, B to $\varphi' \wedge \neg y_i'$, and all clauses in G are satisfied.

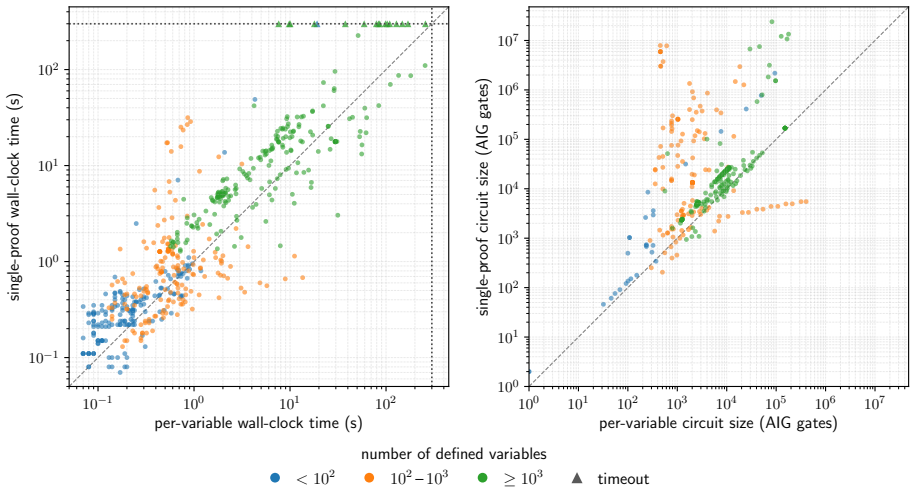

Fig. 1. SINGLE-PROOF vs. PER-VARIABLE on 594 Boolean functional synthesis instances, 300 s timeout. **Left**: wall-clock time. **Right**: AIG sizes of the multi-output definition circuit on the 366 instances solved by both that have ≥ 1 defined variable. Both axes logarithmic; dotted lines mark the timeout.

Interpolation under a partial assignment. Let $A \wedge B \wedge G$ be unsatisfiable, and let σ be a partial assignment with $dom(\sigma) \cap var(A) \cap var(B) = \emptyset$ and $\sigma \models G$. Then $A \wedge \sigma_A \wedge B \wedge \sigma_B$ is unsatisfiable, where $\sigma_S = \sigma|_{var(S)}$, and there is a Craig interpolant of $(A \wedge \sigma_A, B \wedge \sigma_B)$ over $var(A) \cap var(B)$. We compute one from a resolution refutation of $A \wedge B \wedge G$ by annotating each clause C with a partial interpolant $itp(C)$ that may be *undefined* ($\bullet$). For A- and B-leaves we follow McMillan's system [4]: $itp(C) = \bigvee\{\ell \in C : var(\ell) \in var(A) \cap var(B)\}$ for $C \in A$, and $itp(C) = \top$ for $C \in B$. For G-leaves we introduce a new rule, $itp(C) = \bullet$. For a resolvent C on pivot p with premises $C_1 \ni p$, $C_2 \ni \neg p$:

1. if $itp(C_1) = \bullet$, inherit $itp(C_2)$ (symmetrically);
2. else if $p \in dom(\sigma)$, take $itp(C_2)$ when $\sigma(p) = \top$ and $itp(C_1)$ when $\sigma(p) = \bot$;
3. otherwise apply McMillan's rule: $itp(C_1) \vee itp(C_2)$ if $p \in var(A) \setminus var(B)$, else $itp(C_1) \wedge itp(C_2)$.

Since G is satisfiable, every refutation uses at least one $A \cup B$ clause, and inheritance propagates definedness, so $itp(\bot) \neq \bullet$. An induction on the proof DAG shows that $itp(\bot)$ is a Craig interpolant of $(A \wedge \sigma_A, B \wedge \sigma_B)$.

Extracting all n definitions. Each clause carries a vector $(itp_1, \ldots, itp_n)$, with itp_i updated by the rules above instantiated at σ_i. By the above, $itp_i(\bot)$ is an explicit definition of y_i in terms of X. The traversal is single-pass with running time and output size $O(nm)$.

3 Implementation and Experiments

We implemented the algorithm in C++ using CADICAL for proof generation and ABC for And-Inverter Graphs (AIGs). A naive realisation maintaining n independent labels per clause is not viable. Two observations make this efficient. First, at almost every resolution step the rule for itp_i does not depend on i: the only pivots assigned by some σ_i are y_j, y'_j, e_j; at any other pivot the whole vector evolves identically, and at an assigned pivot at most one target deviates. We therefore store each clause as a *shared* label plus a small set of per-target *overrides*. Second, a backward pass marks and skips labels whose value is never read at the empty clause. We call this implementation SINGLE-PROOF.

We compare against a PER-VARIABLE baseline using incremental SAT that adds each defined variable to the shared-variable set of the next call. On a standard Boolean functional synthesis benchmark of 594 instances [1], with a 300 s timeout and 8 GB memory, PER-VARIABLE solved 568 instances and SINGLE-PROOF solved 551 (a strict subset); medians on the 551 common solves are 1.7× in wall-clock time and 2.2× in circuit size for SINGLE-PROOF (Fig. 1). To see what the best possible definition extraction from the joint proof could achieve, we re-ran both tools in proof-only mode (no circuit construction). SINGLE-PROOF then solved 579 instances and PER-VARIABLE 568, with a median time ratio of 0.74× on the 568 common solves. The joint refutation is therefore produced at least as quickly as the per-variable sequence and the gap in Fig. 1 is attributable to interpolation, not SAT solving.

4 Discussion

The bottleneck is in the circuit construction, not the SAT calls. Improving on the $O(nm)$ bound, to $O(m \log m + n)$ or even $O(m + n)$, is the main open challenge. A caveat is that PER-VARIABLE interleaves detection with extraction in a single SAT call, while SINGLE-PROOF has a separate detection stage. Accounting for it would shift the comparison slightly in favour of PER-VARIABLE. Identifying the implicitly defined subset of outputs is itself an interesting question.

References

1. Akshay, S., Chakraborty, S., Goel, S., Kulal, S., Shah, S.: Boolean functional synthesis: hardness and practical algorithms. Formal Methods Syst. Des. **57**(1), 53–86 (2021)
2. Golia, P., Slivovsky, F., Roy, S., Meel, K.S.: Engineering an efficient boolean functional synthesis engine. In: ICCAD (2021)
3. Koopmann, P., Wernhard, C., Wolter, F.: Interpolation in classical propositional logic. In: ten Cate, B., Jung, J.C., Koopmann, P., Wernhard, C., Wolter, F. (eds.) Theory and Applications of Craig Interpolation. Ubiquity Press (2026), to appear
4. McMillan, K.L.: Interpolation and SAT-based model checking. In: CAV (2003)
5. Slivovsky, F.: Interpolation-based semantic gate extraction and its applications to QBF preprocessing. In: CAV (2020)