IsaAbduct: A Multi-Strategy Abduction Pipeline for Isabelle/HOL

Tiago Campos[1], Sophie Tourret[2,3], Jasmin Blanchette[4], Haniel Barbosa[1]

[1]Universidade Federal de Minas Gerais, Belo Horizonte, Brazil ✉{tiago.ferreira, hbarbosa}@dcc.ufmg.br
[2]Inria, Loria, CNRS, Lorraine Univ., Nancy, France ✉sophie.tourret@loria.fr
[3]Max-Planck-Institut für Informatik, Saarbrücken, Germany
[4]Ludwig-Maximilians-Universität München, Munich, Germany ✉jasmin.blanchette@ifi.lmu.de

***Abstract*—Isabelle/HOL is a widely used proof assistant for higher-order logic. However, constructing an Isabelle/HOL proof is not always straightforward. In particular, the user may not know which intermediate lemmas or assumptions are required. A common situation is that a user attempts a proof, the automated reasoning tools fail to find one, and the system returns only "No proof found." This can be frustrating. A more helpful outcome would provide insight into (1) what is missing or (2) which reasoning steps the automated tools attempted. The first aspect corresponds to abduction: finding additional premises that would make the goal provable. The second aspect can be supported by reporting the facts that were relevant during the (ultimately unsuccessful) proof search.**

In this work, we present IsaAbduct, a new command for Isabelle/HOL that integrates an abductive reasoning engine to suggest plausible missing assumptions and to provide additional feedback about the proof search process. The command is based on the syntax-guided synthesis (SyGuS) solver cvc5 and the saturation-based prover E, and operates by exchanging grammar-based hypotheses from cvc5's enumeration and relevant facts extracted from E's saturation search, so that cvc5 and E synergistically guide each other's progress.

Our evaluation shows that IsaAbduct produces a variety of useful hypotheses for the user and indicates that the different backend configurations are complementary: the integrated approach leads to higher-quality suggestions in comparison with using either the cvc5 or the E pipeline in isolation.

***Index Terms*—abduction; proof assistants; Isabelle/HOL; SMT; saturation.**

I. Introduction

Isabelle/HOL is a widely used proof assistant for higher-order logic, and it has been applied to formalizing both hardware and software systems as well as mathematics. It enjoys a mature ecosystem for automating reasoning in higher-order logic, featuring several state-of-the-art tools integrated directly into its architecture for discharging proof obligations. The Sledgehammer [22] subsystem of Isabelle/HOL translates proof obligations into formats suitable for various first- and higher-order automatic theorem provers (ATPs) including satisfiability-modulo-theories (SMT) solvers, invokes these external provers to search for proofs, and reconstructs the discovered proofs within Isabelle/HOL.

In the usual scenario, there are three possible outcomes when a user tries to discharge a proof obligation. The first outcome is the preferable one: the obligation is valid and Sledgehammer is able to find a proof and reconstruct it within Isabelle/HOL. The second outcome is when the obligation is valid but Sledgehammer is unable to find a proof. The third outcome is when the obligation is not valid, and Sledgehammer is also unable to find a proof. The last two outcomes are not very helpful to the user, because they do not indicate what is missing to complete the proof.

One way to mitigate this issue is to try to make the attempted goal valid via *abduction*. The abduction problem asks whether there exists a formula that is consistent with a given set of facts that, when added to these facts, suffices to entail the given goal. Therefore, abductive reasoning can provide additional premises to a goal that allows automatic provers to prove the goal. The challenges related to this approach are to solve the abduction problem effectively and for its solution to be *useful* for users, since many solutions, such as the goal itself, may be undesirable.

For example, suppose we want to prove that inserting an element into a list yields a sorted list. A missing assumption to prove the goal could be "the input list has length at most one." This would make the proof easy, but it is too specific and not very useful. A better suggestion would be that the list was already sorted with respect to the order used by the insertion.

In this paper, we introduce the IsaAbduct command that supplements Sledgehammer when it fails to discharge a proof obligation. To increase the chances that we find useful candidates for closing goals, instead of just using an off-the-shelf abduction solver [23], as was recently done in a command with a similar objective in Rocq [3], we devise new abduction strategies and employ them in combination. We use a syntax-guided synthesis (SyGuS) abduction solver to generate missing assumptions and extend it to restrict the generated abducts based on symbols relevant to the goal. We instrument a saturation-based theorem prover to generate potential missing assumptions by reporting the clauses that it derives during an attempted proof of the goal. We use the cvc5 SMT solver and the E superposition prover [27] as the respective abduction engines. The results from each engine are used to refine the abduction problem sent to the other: abducts found by one engine become goals for the other, yielding new abducts. Ultimately, the results to be suggested to the user can also be filtered by determining whether they are provable in the current context

https://doi.org/

by Sledgehammer. Suggested premises that cannot be proved automatically might also be useful, but they require the user to investigate.

This raises two empirical questions: whether these candidate-generation and filtering mechanisms can recover missing premises in realistic Isabelle/HOL developments, and what criteria are useful for judging suggestions that cannot be proved automatically. To assess the value of IsaAbduct, we have evaluated it on a random selection of proof goals from Isabelle/HOL's *Archive of Formal Proofs* that could be discharged via Sledgehammer, from which existing premises were deleted and IsaAbduct was requested to make the goals provable again. Our results show that in almost all the cases IsaAbduct could successfully make the goal provable again with the found abducts, and that more often than not the abducts themselves could be automatically proved as well via Sledgehammer.

In summary, our contributions are:

- a saturation-based abduction engine leveraging the E prover;
- extensions to cvc5's SyGuS solver to better focus abduction search on relevant facts;
- the IsaAbduct command, which leverages grammar-based and saturation-based abduction engines to effectively generate useful abducts in Isabelle/HOL;
- an evaluation across numerous Isabelle/HOL theories attesting the effectiveness and limitations of the proposed approach.

Related Work: Abduction has generally received much less attention than deduction (the process of deriving a valid conclusion from given premises), but there is some related work. In the context of the superposition calculus, Echenim and Peltier formalized a calculus to generate missing hypotheses in first-order logic [13]. Wernhard proposed describing the problem of abduction in logic programming as a second-order quantifier elimination problem [28]. This is similar to how we will state the problem formally in the context of SMT solving, although we do not rely on second-order quantifiers to solve it. Echenim and Peltier proposed two generic methods to generate implicates of a formula. The implicates of a formula modulo a theory T are the ground clauses that are logical consequences of the formula under T—that is, every model of T that satisfies the formula also satisfies these clauses. Generating such implicates is useful in practice, since each implicate represents a constraint whose negation, when added to the formula, leads to inconsistency with respect to T. Therefore, implicates can be used to generate abducts, which they achieve either by using saturation [15] or by leveraging the SMT solver CVC4 [14]. The former approach is limited in that it applies only to ground formulas. The latter is limited in that the user must provide the candidates to search for implicates. Some other relevant tools include Explain [10], which performs abductive reasoning in Presburger arithmetic combined with propositional logic, and also a specialized tool for automated error diagnosis in software verification using abduction [11].

In the context of Isabelle/HOL, Nagashima and Goc [19] implemented an expansion-node algorithm for exploring abduction. Their approach uses several strategies and tools, including Quickcheck [7], LiFtEr [18], and PSL [20], to condense an abduction graph whose nodes correspond to proof goals. The overall objective, however, contrasts with our objective, since its aim is still to prove the goal directly. The framework combines explicit conjecturing with tactic-based subgoal generation, prioritizing auxiliary lemmas that contribute to the proof of the goal.

A close approach to ours was recently implemented in Rocq [3], but it only directly uses an off-the-shelf SyGuS solver, cvc5, to generate abducts, without any further orchestration. It also does not provide extra functionalities such as trying to prove the validity of the suggested premises. The approach was also evaluated only on a handful of problems from a specific Rocq theory.

II. Preliminaries

Our approach integrates abduction as an Isabelle/HOL command and relies on Sledgehammer. In this section, we introduce both abduction and Sledgehammer.

A. Abduction

We work in a higher-order language with variables of arbitrary finite type, lambda abstraction, and application, as well as arithmetic theories with the usual interpreted symbols and functions. Therefore, we use higher-order notations for function applications, i.e., $f\,x\,y$ denotes the function f applied to arguments x and y. The following notations and conventions are also used: x, y denote variables; σ denotes a type; $A : \sigma$ denotes a term of type σ; $\bar{x}$ denotes a tuple of variables of an arbitrary finite size; $E[\bar{x}]$ indicates that the variables in $\bar{x}$ occur in the expression E; $F[X, \bar{x}]$ denotes a *formula context* where $\bar{x}$ occurs and $X : \sigma$ is a fresh variable standing for an unknown term, so that substituting a term $A : \sigma$ for X produces a formula, denoted by $F[A/X, \bar{x}]$; $\models$ denotes logical entailment.

Definition 1. *Let Γ be a set of formulas, and let $P[\bar{x}]$ be an observation whose free variables are $\bar{x}$. A higher-order abduction problem asks for a term $M : \sigma$ such that, given a formula context $F[X, \bar{x}]$ with a fresh variable $X : \sigma$,*

$$\Gamma, F[M/X, \bar{x}] \models P[\bar{x}] \tag{1}$$

To avoid trivial solutions, we also require that

$$\Gamma, F[M/X, \bar{x}] \not\models \bot \tag{2}$$

The term M is called an abduct of P with respect to Γ.

Example 1. *Consider $F[X, x, y] \equiv X\ x\ y$. Let $\bar{x} = (x, y)$, where $x, y : \mathbb{N}$. Let $\Gamma = \{x = y^2\}$, and let $P[x, y]$ be the proposition $x > y$. The context Γ alone does not imply $P[x, y]$. For example, if $y = 0$ or $y = 1$, then $x = y^2$, but $x > y$ fails.*

We may therefore abduct the missing predicate

$$M = \lambda x\, y.\ non\_unit\ y$$

where non_unit is an interpreted predicate on natural numbers, defined by

$$non\_unit\ n \equiv n \neq 0 \wedge n \neq 1$$

The relevant background theorem is

$$\forall n : \mathbb{N}.\ non\_unit\ n \rightarrow n^2 > n$$

Thus, from the abduced fact $M\ x\ y$, we obtain $non\_unit\ y$. By the background theorem, $y^2 > y$. Since $x = y^2 \in \Gamma$, we conclude $x > y$. Therefore, $\Gamma \cup \{M\ x\ y\} \models P[x, y]$, so the abduct is $M = \lambda x\, y.\ non\_unit\ y$.

B. Sledgehammer

We rely on the Sledgehammer architecture to implement our two abductive reasoning engines. Sledgehammer is Isabelle/HOL's interface to external ATPs, including cvc5 [2], E [25], Vampire [5], and Z3 [17]. When it is invoked, it typically proceeds as follows. First, its relevance filter selects the most promising definitions and lemmas—generically called *facts*—from the current proof context. Next, the external provers are invoked to search for a proof using these facts. If a proof is found, Sledgehammer attempts to reconstruct it inside Isabelle/HOL. Reconstruction can follow two main approaches. In one mode, the ATP proof is discarded, and Isabelle/HOL's *metis* or some other automated tactic is invoked to re-establish the result using the subset of facts identified by the ATP. Alternatively, the ATP proof can be translated into a structured proof script that can be replayed directly within Isabelle/HOL [6], [16], [26].

III. The Abduction Engines

We built two abductive engines on top of Sledgehammer, which are based on syntax-guided synthesis and on saturation.

A. Grammar-Guided Abduction Engine

Definition 1 gives a semantic view of higher-order abduction. For automated reasoning, it is more convenient to work with a syntax-restricted formulation over a background theory T: we fix a finite conjunction of premises Φ, a goal G, and, optionally, a grammar R describing the allowed shape of explanations. The task is then to find a first-order formula A such that $A \wedge \Phi \models_T G$ and $A \wedge \Phi$ is satisfiable in T. In this formulation, the abduction problem reduces to syntax-guided synthesis (SyGuS) [1], [21], and we can use an off-the-shelf SyGuS solver [23] to search for abducts.

The entailment condition can be checked equivalently by asking that $A \wedge \Phi \wedge \neg G$ be unsatisfiable in T. When a grammar R is present, it restricts only the space of candidate formulas explored during search; the notion of abduction itself does not depend on it. We use the terminology of Reynolds et al. [23]: given a grammar R, a set of premises Φ, and a goal G in some theory T, an abduct is a formula A that is R-generated and such that $A \wedge \Phi$ is satisfiable and $A \wedge \Phi \wedge \neg G$ is unsatisfiable in T. We call A an abduct of G with respect to Φ in T. Example 2 illustrate the role of the grammar.

Example 2. *Consider the theory of linear integer arithmetic and the following S-expression grammar for candidate expressions, where x and y are integer variables:*

$$\begin{array}{rcl} E & ::= & x \mid y \mid 0 \mid 1 \mid true \mid false \\ & \mid & (-\ E) \mid (-\ E\ E) \mid (+\ E\ E) \mid (*\ E\ E) \\ & \mid & (div\ E\ E) \mid (mod\ E\ E) \mid (abs\ E) \\ & \mid & (not\ E) \mid (and\ E\ E) \mid (distinct\ E\ E) \\ & \mid & (\leq E\ E) \mid (<\ E\ E) \mid (\geq E\ E) \mid (>\ E\ E) \end{array}$$

Now assume the premise set Φ contains $(>\ y\ (*\ (-\ 2)\ x))$ and $(\leq\ 0\ x)$, and the goal G is $(>\ y\ 0)$. We may then ask which abduct A, built from this grammar, can be added to Φ so that the goal follows. One possible solution is $(and\ (distinct\ x\ 0)\ (\geq\ y\ x))$. Note that the grammar bounds the space of candidate explanations explored by the search procedure: for example, it cannot generate disjunctive candidates, since it has no production for or.

SMT problems can also involve uninterpreted functions and quantified facts, so grammars are often richer than the fragment above. For example, we can extend the grammar with first-order quantifiers by adding productions for $forall$ and $exists$:

$$\begin{array}{rcl} E & ::= & \cdots \mid (forall\ ((Var\ Sort))\ E) \\ & \mid & (exists\ ((Var\ Sort))\ E) \end{array}$$

Example 3 shows why grammars for abductive search may need to accommodate quantifiers and defined symbols, not only quantifier-free arithmetic expressions.

Example 3. *Let the premise set be $\Phi = \{(forall\ ((x\ Bool)\ (y\ Bool))\ (=\ (P\ x\ y)\ (P\ y\ x)))\}$, stating that P is symmetric, and let the goal be $G = (=\ (P\ x'\ x')\ false)$. Here the abduction problem searches for the operator P itself: in the terminology of Definition 1, P plays the role of the fresh variable X, and the abduct A is a defining equation for P. For example, abduction may define P as xor, using only the connectives and and not:*

$$\begin{array}{rcl} (P\ u\ v) & = & (and\ (not\ (and\ (not\ u)\ (not\ v)))\\ & & \quad (not\ (and\ u\ v))) \end{array}$$

This equation, which is universally quantified over u and v, is the abduct A: it makes $(P\ x'\ x')$ evaluate to $false$, so $A \wedge \Phi \models G$, whereas Φ alone does not entail G. Moreover, A is consistent with Φ, since xor is symmetric.

Given a grammar R, we use cvc5 as a SyGuS solver to search for the abduct. Its SyGuS strategies optimize the search space by caching models that do not satisfy the goal. Suppose that we have a set of premises $\Phi = \{(=\ x\ true), (=\ (f\ x)\ false)\}$ and a goal $G = (=\ false\ (f\ (f\ x)))$. Let $C = \{(=\ x\ (f\ (f\ x)))\}$ be a candidate abduct. C is consistent with Φ, but it is not an abduct: checking the satisfiability of $\Phi \wedge C \wedge \neg G$ yields a model $M = \{x \mapsto true,\ (f\ (f\ x)) \mapsto true\}$, showing that C together with Φ does not entail G. The model M is more than a witness for discarding C: cvc5 caches it, and any future candidate C' that evaluates to true under M can be discarded immediately, since M would again witness the satisfiability of $\Phi \wedge C' \wedge \neg G$. This replaces a satisfiability check with a cheap model evaluation, which matters because satisfiability checking is usually the bottleneck of the algorithm, especially when the premises contain quantifiers.

By default, cvc5 automatically generates grammars when

the user does not provide one. These grammars are constructed from the symbols that occur in the goal and the premises, and are tailored to leverage the fast and smart enumeration techniques employed by the solver [24], as the choice of grammar has a significant impact on the search. In the search for abducts, we have observed that not all symbols appearing in the premises are necessarily relevant for enumerating candidate abducts, even though these premises may still be useful for ruling out invalid candidates. However, restricting the grammar to symbols from a subset of the premises requires providing a custom grammar, which can be tedious and error-prone, and may inadvertently undermine the solver's enumeration techniques.

To address the issue of default grammar construction, we introduce an extension to cvc5's abduction solver: the *grammar-assertions* option. This option allows grammar construction to be restricted to a subset of the premises provided to the solver, namely those that the user deems relevant to the search for abducts. In particular, it restricts the generated grammar to the symbols occurring in this selected subset of premises. This feature is especially relevant for IsaAbduct because the premises sent to the SyGuS solver are extracted from a proving context. The local context consists of the premises available at the current proof state, such as the assumptions of the current goal, while the global context consists of the theorems and definitions available outside the proof state. In general, the local context is more relevant for enumerating candidate abducts, since symbols occurring locally are more likely to appear in the missing hypotheses. The global context, on the other hand, is useful for ruling out inconsistent candidates, thereby increasing the chances of finding a useful abduct. Therefore, when using the *grammar-assertions* option, we include only the local context in the grammar construction, while the abduction problems themselves still contain both the local and global contexts.

Example 4. *Consider a variant of the linear integer arithmetic problem following Example 2. Let the premise set be*

$$P = (> y\ (* (-2)\ x)), \qquad H = (> (inc\ x)\ 0),$$

$$F = (forall\ ((z\ Int))\ (= (inc\ z)\ (+ z\ 1))).$$

and let the goal G be $(> y\ 0)$. Note that H and F together imply the original premise $(\leq\ 0\ x)$, therefore a possible solution is still $A = (and\ (distinct\ x\ 0)\ (\geq y\ x))$. However, while H and F are necessary to enforce that x is non-negative, they would normally lead to the inclusion in the grammar of a production that involves the function inc:

$$\begin{array}{rcl} E & ::= & \cdots \\ & & \mid (inc\ E) \end{array}$$

The highlighted production however is unnecessary to find the solution A above, and it would be avoided if we use only P from the premises for the purpose of generating productions for the grammar, since P does not contain inc. This could be done for cvc5 with the new `:grammar-assertions` option listing only P. The grammar would be E as in Example 2, and H and F would still be used to filter out candidates.

B. Saturation-Based Abduction Engine

The second abduction engine is built into Sledgehammer and relies on the E prover. Like other superposition provers, E works by saturation: it negates the goal, clausifies the axioms and the negated goal, and derives, from the initial clauses, new clauses using inference rules. Clauses deemed to be redundant (e.g., if they are subsumed by other clauses) are removed. If E derives the empty false clause $\bot$, it concludes that the problem clauses including the negated goal are unsatisfiable, meaning that the goal is provable.

When E times out before deriving $\bot$, it typically has derived thousands of other clauses, forming a partial saturation. Each of these clauses represents a hole in a proof. If we managed to prove the negation of one of these clauses, we could directly derive $\bot$. Thus, all of the clauses appearing in a saturation are, once negated, candidate abducts.

For example, suppose the Isabelle/HOL goal is $set\_mset\ (mset\_set\ A) = A$, where $mset\_set$ converts a set into a multiset and set_mset performs the opposite conversion. The goal is not provable because sets may be infinite, whereas Isabelle/HOL multisets are always finite; thus, if A is an infinite set, $mset\_set\ A$ denotes some unspecified, arbitrary multiset, which is then converted to a set that may differ from A. When we invoke Sledgehammer, the underlying ATPs all time out, but from E's failure, Sledgehammer retrieves a list of candidate abducts. Among these is the particularly short and plausible-looking formula $finite\ A$, which is the negation of the clause $\neg\ finite\ A$ (up to double-negation elimination).

For each of the possibly thousands of abduct candidates, Sledgehammer gives a score computed from the formula. The lower the score, the more relevant the candidate is considered. Most Isabelle/HOL constructs are given a score of 2, but free variables are given a score of 1, because we want to prioritize general assumptions such as $x \leq y$ over specific assumptions such as $x \leq c$, where c is a constant. Moreover, Sledgehammer filters out candidates matching certain syntactic formats. For example, $x = t$, where x is a free variable and t is any term, is very specific and rarely useful in practice, and $t < 0$ on natural numbers is false.

IV. The IsaAbduct Command

We implemented IsaAbduct within the Sledgehammer architecture so we can exploit its infrastructure to translate goals into external systems and parse back the results. IsaAbduct is used interactively: the user invokes it on a proof state to find missing assumptions that would make the goal provable. The user then inspects the candidates and can either add one to the premise set or prove it separately as a lemma. Either option enables the user to prove the original goal. Figure 1 illustrates a typical interaction with IsaAbduct: the output panel lists the candidates found by the tool, and the user can decide which one to add to the premise set or prove separately.

A high-level overview is shown in Figure 2: IsaAbduct receives an Isabelle/HOL proof state together with a timeout

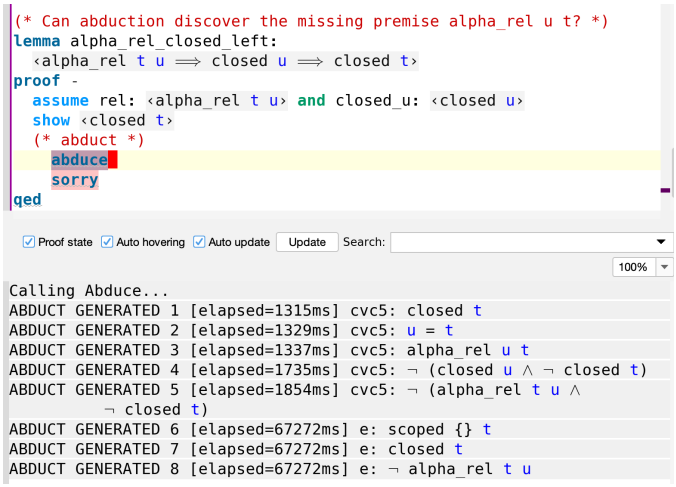

Figure 1. Invoking `abduce` on a lemma `alpha_rel_closed_left` in Isabelle/jEdit. The output panel shows the candidate abducts generated by cvc5 and E.

and a maximum search depth (the number of rounds between the two abduction engines). First, selected relevant facts are filtered through the Sledgehammer relevance filter, then they are dispatched to the abduction engines. The abducts are iteratively refined by exchanging abducts between them. We illustrate how IsaAbduct works via the following example.

Example 5. *Consider the datatype* `lam`*, representing lambda terms; the predicate* `scoped`*, which checks whether a term is well scoped in a variable context; the definition* `closed`*, where* `closed t` *abbreviates* `scoped {} t` *and therefore means that* `t` *has no free variables; and the predicate* `alpha_rel`*, which establishes alpha-equivalence between two lambda terms. Assume we have already proved a lemma* `alpha_rel_closed_right`*, which establishes that if a closed term* `t` *is alpha-equivalent to a term* `u`*, then* `u` *is also closed. We want to prove the variant* `alpha_rel_closed_left`*, which transfers closedness in the other direction: assuming that* `u` *is closed and requiring to show that* `t` *is closed.*

```
datatype lam =
  Var nat
| App lam lam
| Lam nat lam

fun scoped :: ⟨nat set ⇒ lam ⇒ bool⟩ where
  ⟨scoped Γ (Var x) ⟷ x ∈ Γ⟩
| ⟨scoped Γ (App t u) ⟷ scoped Γ t ∧ scoped Γ u⟩
| ⟨scoped Γ (Lam x t) ⟷ scoped (insert x Γ) t⟩

definition closed :: ⟨lam ⇒ bool⟩ where
  ⟨closed t ⟷ scoped {} t⟩

inductive alpha_rel :: ⟨lam ⇒ lam ⇒ bool⟩
    where
  alpha_Var: ⟨alpha_rel (Var x) (Var x)⟩
| alpha_App:
    ⟨alpha_rel t t' ⟹ alpha_rel u u' ⟹
     alpha_rel (App t u) (App t' u')⟩
| alpha_Lam:
    ⟨alpha_rel t u ⟹ alpha_rel (Lam x t) (
        Lam x u)⟩
| ...
| alpha_trans:
    ⟨alpha_rel t u ⟹ alpha_rel u v ⟹
        alpha_rel t v⟩

lemma alpha_rel_closed_right:
  ⟨alpha_rel t u ⟹ closed t ⟹ closed u⟩
...
qed

lemma alpha_rel_closed_left:
  ⟨alpha_rel t u ⟹ closed u ⟹ closed t⟩
proof - ???
qed
```

In Example 5, to prove lemma `alpha_rel_closed_left`, we may need to assume the reverse alpha-equivalence relation for `t` and `u`, i.e., `alpha_rel u t`. Together with `closed u`, this missing fact is sufficient to conclude `closed t`. The user may not be aware of it, however. In this case, the `abduce` command can help the user find this missing assumption. `abduce` is the user-facing entry point of IsaAbduct, illustrated in Figure 1. In Example 5, it yields the abducts shown in Figures 3 and 4 that are being generated by each engine in isolation.

The abduction we are looking for is candidate c_3, namely *alpha_rel u t*, generated by cvc5. This is the missing reverse relation needed to reuse `alpha_rel_closed_right`: from *alpha_rel u t* and *closed u*, we obtain *closed t*. Thus, by using the abductive reasoning engine, we were able to find a missing assumption that helps prove the goal. We can still examine other abduction suggestions; candidate c_1, namely *closed t*, and candidate *e2* restate the goal itself, while *e1* is its unfolding through `closed_def`, which makes both of them spurious. The remaining cvc5 candidates, c_2, c_4, and c_5, close the current goal but are less useful to guide the proof search. This means that the user may still not know exactly what they are looking for, but these suggestions can still help point them in the right direction.

These observations motivate the quality metrics used in Section V-A: candidate abducts should not be evaluated only by whether they are sufficient to close the current goal. Consider for instance a goal could be proved once an abduct is found that states that a given set is finite, as in the saturation example of Section III-B. The finiteness assumption however may not be provable from the surrounding context of that goal, or may be an assumption a user may not want to add.

A. Implementation

We implemented IsaAbduct as a new command, `abduce`, for Isabelle/HOL. The `abduce` command relies on two pipelines: one for cvc5 and another for E. The tool also includes a depth parameter; this parameter controls how many iterations of the

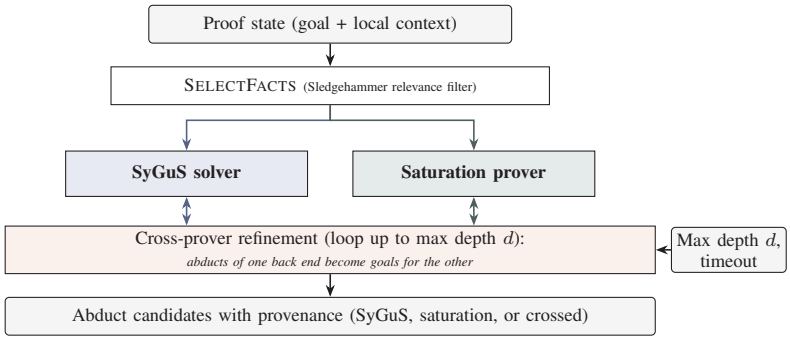

Figure 2. Architecture of IsaAbduct. The `abduce` command takes an Isabelle/HOL proof state together with a depth and timeout, selects relevant facts via Sledgehammer's filter, and passes them to two abductive backends: a SyGuS solver and a saturation-based prover. A cross-prover refinement loop reuses abducts of one backend as goals for the other, up to the configured depth, producing a tagged set of candidates.

Abducts from cvc5

c_1 $(closed\ t)$
c_2 $u = t$
c_3 $(alpha\_rel\ u\ t)$
c_4 $\neg((closed\ u) \wedge \neg(closed\ t))$
c_5 $\neg((alpha\_rel\ t\ u) \wedge \neg(closed\ t))$

Figure 3. Candidate abduct premises generated by cvc5 for the lemma `alpha_rel_closed_left`.

Abducts from E

e1 $scoped\ \{\}\ t$
e2 $closed\ t$

Figure 4. Candidate abduct premises generated by E for the lemma `alpha_rel_closed_left`.

abduction process the tool should perform. The motivation is to allow the user to decide how deep the interaction between the two backends should be, so that they can balance the time spent on abduction with the amount of candidates generated. For example, with depth 0, the tool only generates abducts from the original goal; with depth 1, it also generates abducts from the abducts generated in the previous step; and so on.

We generate additional candidate abducts by using abducts from one backend as goals for the other, and vice versa. For example, we use the abducts of E as goals for cvc5 (Strategy E $\rightarrow$ cvc5) and abducts from cvc5 as goals for E (Strategy cvc5 $\rightarrow$ E). This exchange exploits the complementarity of the two engines: saturation search is guided by E's internal heuristics and by the facts selected by the relevance filter, whereas the SyGuS solver enumerates candidates directly from a grammar, yielding a more diverse but less guided set of hypotheses. This preserves soundness in the following sense: if we have an abduct A and a set of clauses P that help prove the goal G, then $A \wedge P \models G$. The abduct A' of A given P is also a valid abduct of G for the same P since $A' \wedge P \models A$. Moreover, any abduction obtained from another candidate is at least as strong as, or equivalent to, the candidate it abduces. Indeed, if A' is an abduct of A, then $(P \wedge A' \rightarrow A) \ \wedge \ (P \wedge A \rightarrow G) \ \models (P \wedge A' \rightarrow G)$. This cross-prover strategy demonstrates better results when using E first and then cvc5, because cvc5 can struggle to find abducts when the goal has many quantifiers or the grammar is too complex, while E handles quantifiers better but is more limited by the context. In Figure 5, e denotes a candidate generated by E and c denotes a candidate generated by cvc5; $e{\rightarrow}c$ denotes a candidate first generated by E and then consumed by cvc5 in the interchange step, and $c{\rightarrow}e$ denotes the reverse direction. We can see the favored abduction suggestion is generated later in the cross-prover search, namely candidate $e{\rightarrow}c_4 = 2 \leq size\ t + size\ x$. It captures the constraint that the size of nodes in the tree must be at least 2, and it is more general than the other candidates. Candidate $e{\rightarrow}c_4$ illustrates that interchanging candidates between pipelines can enrich the set of generated suggestions.

Operationally, the process begins by selecting facts from the current subgoal via the usual Sledgehammer interface. The code gathers local hypotheses, local named facts, chained facts, and a relevance-filtered set of outer facts. Algorithm 1 describes the whole procedure. In lines 1 and 2, we initialize the sets of facts used by E and cvc5. Notice that these sets may differ, since each prover can use its own custom configuration. For example, when generating abducts with cvc5, IsaAbduct uses a default configuration, that may rule out external facts containing quantifiers, e.g, those quantified facts that are outside the current proof context. Thus, no configuration is required for normal interactive use, although users may override the default when needed.

Concretely, the problem is translated to SMT-LIB [4] using the facts selected by Sledgehammer, as shown in Figure 6. The excerpt is taken from the query generated for the proof obligation in Figure 1. Notice that some facts are omitted.

First, when sending an abduction query to cvc5, we rely on Sledgehammer to handle the translation from Isabelle/HOL to SMT-LIB. We generate a SyGuS problem whose grammar is built from the local proof context, extended with additional facts selected by Sledgehammer relevance filter. The current goal is extracted from the proof state and used as the abduction goal for cvc5. The abducts produced by cvc5 are then parsed back into Isabelle/HOL syntax through the same pipeline and stored in a queue, where they can later be consumed by E. For E, we

Theory Editor (left pane)

```
datatype 'a tree =
  Node 'a
| Empty
| Branch "'a tree" "'a tree"

fun size :: "int tree ⇒ int" where
  ⟨size (Node n) = n⟩
| ⟨size Empty = 0⟩
| ⟨size (Branch u v) = size u + size
    v⟩

(* Can abduction find a useful size
   premise? *)
lemma branch_size_at_least_two: ⟨size
    (Branch t x) ≥ 2⟩
  abduce
```

Abducts from E

e1 $size\ (Branch\ t\ x) = dbl\ 1$

$(\ldots)$

Abducts from E → cvc5

$(\ldots)$

$e{\rightarrow}c_1$ $t = x \wedge 1 = size\ t$

$e{\rightarrow}c_2$ $t = x \wedge 1 \leq size\ t$

$e{\rightarrow}c_3$ $2 = size\ t + size\ x$

e→c4 $\boxed{2 \leq size\ t + size\ x}$

$e{\rightarrow}c_5$ $t = x \wedge 0 = size\ t$

$e{\rightarrow}c_6$ $t = x \wedge size\ t \leq 0$

$(\ldots)$

Figure 5. Isabelle/HOL proof state and selected abduction suggestions, with omitted candidates indicated by $(\ldots)$.

use an analogous pipeline. We again rely on Sledgehammer to perform the translation, provide E with the goal and the relevant facts, and extract abducts from the saturation process. These abducts are also stored in a queue, so that they can be consumed by cvc5. Lines 5 and 6 of Algorithm 1 initialize these queues, and lines 8–19 process them.

When jobs in the queue are at the same depth, meaning that they do not depend on one another, they are sent in parallel, and their results are displayed as soon as they arrive.

One issue when sending a problem to cvc5 is that the grammar generated from the proof context can become too large, especially when additional facts contain quantifiers. These facts are useful for refuting candidate models, but the abduct itself does not necessarily need to mention their symbols. A possible mitigation strategy is therefore to restrict the grammar to the symbols occurring in only a subset of the assertions. Therefore IsaAbduct employs the *grammar-assertions* feature we added to cvc5 to restrict the grammar to the goal and the local context hypotheses only. Global facts may still be provided to the solver, but since they excessively enlarge the grammar, and usually they do not contain symbols that are relevant to the goal, they are discarded in the grammar. This keeps the search space smaller while still allowing the solver to use additional facts to rule out invalid candidates.

For example, in Figure 6 we use the local assumptions `a0` and `a1`, corresponding to `closed u` and `alpha_rel t u`, as grammar assertions. This means that the generated candidates will be guide by the syntax of these assertions, and are therefore more likely to be relevant to the goal, although cvc5 can still rely on other facts to discard candidates that do not satisfy the current proof context.

V. Evaluation

There is no standard technique for evaluating abduction quantitatively. Thus, we begin this section by a motivated description of our metrics before presenting the benchmarks and the results.

A. Evaluation Metrics

Example 5 already shows why abduction cannot be evaluated only by whether a candidate closes the current goal. Candidate c_3, namely *alpha_rel* u t, is the abduct we are looking for, whereas candidate c_1, namely *closed* t, is merely the original goal and is therefore not useful as an explanation. Other candidates can be valid but too strong, too specific, or difficult to justify from the existing context.

One good metric to evaluate the quality of the abduction engine is to measure the relative weakness or strength of the candidates. Given two abduction candidates A_1 and A_2, we say that A_1 is *strictly weaker* than A_2 if $A_2 \models A_1$ and $A_1 \not\models A_2$. Similarly, we say that A_2 is *strictly stronger* than A_1. When A_1 and A_2 entail each other, we say that they are *equivalent*. Here entailment is understood in the proof context of the abduction problem, and in the evaluation it is checked by Sledgehammer within a timeout.

Another important metric is whether a candidate abduct can itself be discharged by Sledgehammer from the original proof context. In that case, the candidate is not merely a new assumption that would close the current goal; it is a lemma that Isabelle/HOL can recover automatically from the surrounding facts. We therefore keep and report such provable abducts separately, since they are especially actionable: the user can turn them into intermediate proof obligations rather than adding unsupported assumptions. Throughout the evaluation, *provable* accordingly means that Sledgehammer finds a proof from the stated context within the timeout; failure to do so does not imply unprovability in principle.

More generally, evaluating abduction is inherently difficult, since the usefulness of a suggested hypothesis is partly subjective. An abductive reasoning problem may admit several premises that suffice to prove the same goal. These candidate

Algorithm 1: ABDUCE—cross-prover abductive reasoning

Input: Proof context, goal, depth limit, timeout
Output: Abductive hypotheses with provenance

```
facts ← SELECTFACTS(context, goal);
// Depth 0: run both backends on the
   original goal
cvc5_facts ← FILTERCVC5FACTS(facts);
abducts_cvc5 ←
     CVC5-ABDUCT(context, cvc5_facts, goal, timeout);
abducts_e ← E-ABDUCT(context, facts, goal, timeout);
// Depth 1..d: cross-prover
   refinement (breadth-first)
work_for_e[0] ← abducts_cvc5;
work_for_cvc5[0] ← abducts_e;
abducts_all ← abducts_cvc5 ∪ abducts_e;
for level ← 1 to depth do
    foreach hypothesis h in work_for_cvc5[level-1] do
        // Usually at depth higher than
           0 fewer facts are selected
        new ← CVC5-ABDUCT(context,
         FILTERCVC5FACTS(SELECTFACTS(context,
         h)), h, timeout);
        abducts_all ← abducts_all ∪ new;
        work_for_e[level] ← work_for_e[level] ∪ new;
    end
    foreach hypothesis h in work_for_e[level-1] do
        new ←
          E-ABDUCT(context,
               SELECTFACTS(context, h), h, timeout);
        abducts_all ← abducts_all ∪ new;
        work_for_cvc5[level] ←
             work_for_cvc5[level] ∪ new;
    end
end
return abducts_all;
```

premises may differ substantially in strength and explanatory value: some are too specific to the current goal, while others capture a more reusable missing assumption. In such cases, the challenge is not only to find a correct premise, but also to identify one that is sufficiently general to be useful. Example 6 illustrates this issue in a simple equational scenario.

Example 6. *Consider the goal* $inv\ (inv\ x) = id\ x$ *given* $id\ x = x$*. Candidate abducts include*

1: $inv\ x = x$	*(unappealing)*
2: $inv\ x = id\ x$	*(unappealing)*
3: $inv\ (inv\ x) = id\ x$	*(unappealing)*
4: $inv\ (inv\ x) = x$	*(appealing)*

In this scenario, multiple abducts can be used to prove the goal. Although abducts 1 and 2 can be used, they are likely not provable from the context. Abduct 3 is simply the goal itself, whereas abduct 4 is weaker and more general, making it the more useful candidate.

```
(set-option :produce-abducts true)
(set-option :sygus-stream true)
(set-logic AUFLIA)

(declare-sort Lam$ 0)
(declare-sort Nat_set$ 0)
(declare-fun t$ () Lam$)
(declare-fun u$ () Lam$)
(declare-fun bot$ () Nat_set$)
(declare-fun closed$ (Lam$) Bool)
(declare-fun scoped$ (Nat_set$ Lam$) Bool)
(declare-fun alpha_rel$ (Lam$ Lam$) Bool)

(assert (! (closed$ u$) :named a0))
(assert (! (alpha_rel$ t$ u$) :named a1))
(assert (!
  (forall ((?v0 Lam$))
    (= (closed$ ?v0) (scoped$ bot$ ?v0)))
  :named a2))
(assert (!
  (forall ((?v0 Lam$) (?v1 Lam$))
    (=> (and (alpha_rel$ ?v0 ?v1)
             (closed$ ?v0))
        (closed$ ?v1)))
  :named a3))

; ... (additional facts selected by Sledgehammer)

(get-abduct A
  (! (closed$ t$) :named conjecture54)
  :grammar-assertions (a0 a1))
```

Figure 6. Excerpt of the SMT-LIB query generated for the proof obligation shown in Figure 1. The local assumptions are used as grammar assertions for cvc5's abductive search.

Therefore, the algorithm should ideally produce high-quality abducts, not only any abducts. Similar to previous approaches, such as inductive invariant generation and error localization via abduction [10]–[12], we use logical strength, relative to other potential solutions, and simplicity, in the sense of leading to fewer proof obligations, as proxies for quality.

We report four quantitative metrics. First, *coverage* measures the number of problems for which the tool generates at least one abduct. Second, *provability* measures whether Sledgehammer can prove the abduct itself from the full original context. Third, we compare the *strength* of each abduct A relative to the removed premise p that serves as the reference point in our benchmark (Section V-B). Since p is known to be sufficient in the original proof, strictly weaker or equivalent abducts are especially interesting: they recover useful missing information while avoiding unnecessarily strong assumptions. Fourth, *timing* measures the elapsed time until the first generated abduct and the first proved abduct, capturing both responsiveness and the time needed to obtain an automatically recoverable suggestion.

B. Benchmark

We use a controlled benchmark in which one premise is removed from each proof obligation and IsaAbduct is asked to generate hypotheses for the resulting goal. This gives a

concrete reference point, namely the removed premise, against which generated abducts can be compared. We generated benchmark instances with Mirabelle [9] from 30 *Archive of Formal Proofs* theories, selected randomly from the eligible AFP theory list using hashes of the theory names to make the selection reproducible, with 10 proof obligations from each theory. More precisely, we implemented a custom Mirabelle action, `abduct`, that removes one arbitrary premise from the current proof obligation, invokes IsaAbduct on the resulting goal, and records the omitted premise, the generated abducts, and timing information. We used a timeout of three minutes to obtain at least one abduct.[1] Removing a premise of an existing theorem is a tentative way of approximating real-world scenarios where users are trying to prove something but may have a missing condition. While this methodology is limited, it does allow for a controlled set of experiments, and is on par with similar evaluations of previous approaches [23].

We evaluate four configurations: cvc5, E, E $\rightarrow$ cvc5, and cvc5 $\rightarrow$ E. The first means using cvc5 to produce abducts by itself; the second E by itself; the third is the configuration that adds E abducts into the premises considered by cvc5; the fourth is the one where cvc5's abducts are added to E's premises. All four configurations correspond to a refinement depth of 1; Algorithm 1 supports arbitrary depths, but a systematic exploration of deeper refinements is left for future work. We use a modified version of cvc5 1.3.3 and the E version 3.2.5-ho bundled with Isabelle/HOL. The experiments use a 2025 Isabelle/HOL development version and the officially available AFP 2025.

Overall, IsaAbduct generated at least one abduct for 240 of the 300 analyzed problems (80.0%), producing 5405 abducts in total. Table I shows how these abducts are distributed across backend configurations. The largest contribution comes from E $\rightarrow$ cvc5, followed by cvc5 $\rightarrow$ E, cvc5, and E. This shows that cross-prover refinement, and especially refining E-generated candidates in cvc5, substantially enlarges the set of available suggestions, although candidate volume alone does not determine usefulness.

Table I
BACKEND CONTRIBUTION TO ABDUCT GENERATION.

Backend	Abducts	%
cvc5	1083	20.0
E	413	7.6
E $\rightarrow$ cvc5	2424	44.8
cvc5 $\rightarrow$ E	1485	27.5

Table II reports problem-level coverage for each backend, showing the number of problems for which each backend produced at least one abduct. Here, "% w/ abducts" is relative to the 240 problems for which IsaAbduct produced at least one abduct overall, whereas "% total" is relative to all 300 benchmark problems. The "Unique" column reports the number of problems for which only that backend produced an abduct.

[1]The experimental data is available at [8].

Table II
PROBLEMS WITH AT LEAST ONE ABDUCT PRODUCED BY EACH BACKEND.

Backend	Problems	% w/ abducts	% total	Unique
cvc5	211	87.9	70.3	98
E	119	49.6	39.7	15
E $\rightarrow$ cvc5	98	40.8	32.7	0
cvc5 $\rightarrow$ E	84	35.0	28.0	0

Table III
PER-BACKEND PROVABILITY AND RELATIVE STRENGTH. PERCENTAGES ARE COMPUTED WITH RESPECT TO THE "TOTAL" COLUMN; THE OMITTED ABDUCTS NUMBER CORRESPONDS TO EQUIVALENT OR INCOMPARABLE CANDIDATES. "UNIQUE" GIVES THE NUMBER OF PROBLEMS FOR WHICH ONLY THAT BACKEND GENERATED A PROVABLE ABDUCT.

Backend	Total	Prov.	%	Weak.	%	Strong.	%	Unique
cvc5	1083	186	17.2	106	10.0	366	34.4	71
E	413	134	32.4	81	20.1	141	35.1	25
E $\rightarrow$ cvc5	2424	125	5.2	71	3.0	820	34.5	3
cvc5 $\rightarrow$ E	1485	135	9.1	16	1.3	736	57.6	0

Across all backends, at least one generated abduct was proved for 146 problems (48.7% of all problems, or 60.8% of the problems with abducts).

Among the base backends, cvc5 has the wider coverage, producing at least one abduct for 211 problems (70.3% of the full benchmark), whereas E does so for 119 problems (39.7%). Neither base backend subsumes the other: cvc5 produces abducts for 121 problems where E produces none, and E for 29 problems where cvc5 produces none, with 90 problems covered by both. The multi-strategy configurations apply to fewer problems, 98 for E $\rightarrow$ cvc5 and 84 for cvc5 $\rightarrow$ E, because they depend on candidates produced earlier in the pipeline. Nevertheless, once applicable, E $\rightarrow$ cvc5 contributes many additional hypotheses, which explains its dominant share in Table I.

Table III evaluates the generated abducts more closely. E has the highest proportion of provable abducts (32.4%) and also the highest proportion of weaker abducts (20.1%), suggesting that its candidates are often tightly connected to the removed premise, perhaps due to the E abduction engine's preference for free variables occurring in the goal. cvc5 has the next best weaker-abduct rate (10.0%), while cvc5 $\rightarrow$ E has the next best provability rate (9.1%) among the remaining configurations. The multi-strategy configurations produce additional candidates, but these candidates are less often weak relative to the removed premise: E $\rightarrow$ cvc5 contributes the most abducts overall, with 5.2% provable and 3.0% weaker than the removed premise, whereas cvc5 $\rightarrow$ E produces the highest proportion of stronger abducts (57.6%). Overall, these results suggest that E provides the strongest direct candidate quality, cvc5 provides the broadest problem coverage, and cross-prover refinement is useful for expanding the search space and exposing additional hypotheses.

Timing results show that useful suggestions often appear early, but with a long tail. Table IV summarizes the average and median estimated time to each backend's first generated abduct and to the first proved abduct. For generated abducts, the estimate divides the earliest backend timestamp by the number

Table IV
ESTIMATED TIME (IN SECONDS) TO EACH BACKEND'S FIRST GENERATED ABDUCT AND FIRST PROVED ABDUCT.

Backend	Gen.	Avg.	Med.	Prov.	Avg.	Med.
cvc5	174	20	5	80	14	5
E	98	218	138	52	175	121
E → cvc5	77	177	62	33	169	98
cvc5 → E	69	346	198	9	134	93

of timed abducts from that backend in the problem. For proved abducts, it divides the earliest proved `generated_at_ms` timestamp for that backend by the number of timed abducts from that backend in the problem. Among the 116 problems for which the first proved abduct was directly timed, the median time was 23 s and the average was 166 s, with a minimum of 1 s and a maximum of 1485 s. The distribution of these times is as follows: in 49 cases (42.2%) the first proved abduct appeared within 1–10 s, in 15 cases (12.9%) within 10–30 s, and in 7 cases (6.0%) within 30–60 s; the remaining cases fall in the 1–5 minute bucket (14 cases, 12.1%), the 5–15 minute bucket (29 cases, 25.0%), and the 15–30 minute bucket (two cases, 1.7%). In particular, a lower time limit of 30 s would still yield a proved abduct for 64 of these problems (55.2%). Note that these figures can exceed the three-minute timeout of the benchmark: that timeout is the budget for *generating* abducts, whereas the times reported here measure how long it took to find the first *provable* abduct among the generated candidates. We emphasize that provability checking is performed only as part of the evaluation. In interactive mode, abducts are presented to the user as soon as they are generated.

VI. CONCLUSION AND FUTURE WORK

We presented IsaAbduct, a new tool for Isabelle/HOL that provides abductive reasoning. Our strategy generates diverse candidates with a useful range of strengths, which can help the user to find the abduct that best suits their needs. We evaluated our tool on a set of problems extracted from the *Archive of Formal Proofs* and showed that it can generate abducts for most benchmark problems. We observed that combining cvc5 and E expands the candidate space, while the direct cvc5 and E pipelines often provide the strongest provability and relative-weakness rates. We also observed that many of the generated abducts make it possible to complete the proof, further indicating their usefulness.

As future work, we would like to explore whether the tool could be used to discharge proof obligations entirely by generating abducts that Sledgehammer can then use to prove the goal. This direction is motivated by two observations. First, some generated abducts are provable from the original context and can therefore be turned into intermediate lemmas that help complete the proof. Second, as discussed above, in the majority of timed cases, the first proved abduct was found within the first minute. We believe this could be feasible if the tool parallelizes abduct generation and the task of proving each abduct.

Acknowledgment

Campos' and Barbosa's research are partially supported by a gift from Amazon Web Services and by the Defense Advanced Research Projects Agency (DARPA) under contract FA8750-24-2-1001. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of DARPA. Campos is also supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Blanchette's research was cofunded by the European Union (ERC, Nekoka, 101083038). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

[1] R. Alur, R. Bodík, G. Juniwal, *et al.*, "Syntax-guided synthesis," in *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20–23, 2013*, IEEE, 2013, pp. 1–8.

[2] H. Barbosa, C. W. Barrett, M. Brain, *et al.*, "cvc5: A versatile and industrial-strength SMT solver," in *Tools and Algorithms for Construction and Analysis of Systems (TACAS), Part I*, D. Fisman and G. Rosu, Eds., ser. Lecture Notes in Computer Science, vol. 13243, Springer, 2022, pp. 415–442. DOI: 10.1007/978-3-030-99524-9_24.

[3] H. Barbosa, C. Keller, A. Reynolds, A. Viswanathan, C. Tinelli, and C. Barrett, "An interactive SMT tactic in Coq using abductive reasoning," in *Proceedings of the 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR '23)*, ser. EPiC Series in Computing, Also available as HAL Id hal-04287029v1, vol. 94, EasyChair, 2023, pp. 11–22. DOI: 10.29007/432m.

[4] C. Barrett, P. Fontaine, and C. Tinelli, *The Satisfiability Modulo Theories Library (SMT-LIB)*, https://www.smt-lib.org/, 2016.

[5] F. Bártek, A. Bhayat, R. Coutelier, *et al.*, "The Vampire diary," in *Computer Aided Verification—37th International Conference, CAV 2025, Zagreb, Croatia, July 23–25, 2025, Proceedings, Part III*, R. Piskac and Z. Rakamaric, Eds., ser. Lecture Notes in Computer Science, Springer, 2025, pp. 57–71. DOI: 10.1007/978-3-031-98682-6_4.

[6] J. C. Blanchette, S. Böhme, M. Fleury, S. J. Smolka, and A. Steckermeier, "Semi-intelligible Isar proofs from machine-generated proofs," *Journal of Automated Reasoning*, vol. 56, no. 2, pp. 155–200, 2016. DOI: 10.1007/S10817-015-9335-3.

[7] L. Bulwahn, "The new Quickcheck for Isabelle—random, exhaustive and symbolic testing under one roof," in *Certified Programs and Proofs—Second International Conference, CPP 2012, Kyoto, Japan, December 13–15, 2012. Proceedings*, C. Hawblitzel and D. Miller, Eds.,

ser. Lecture Notes in Computer Science, Springer, 2012, pp. 92–108. DOI: 10.1007/978-3-642-35308-6_10.

[8] T. Campos, S. Tourret, J. Blanchette, and H. Barbosa, *Experimental data for IsaAbduct: A multi-strategy abduction pipeline for Isabelle/HOL*, Zenodo, version 1.0, 2026. DOI: 10.5281/zenodo.21715742.

[9] M. Desharnais, P. Vukmirović, J. Blanchette, and M. Wenzel, "Seventeen provers under the hammer," in *13th International Conference on Interactive Theorem Proving (ITP 2022)*, J. Andronick and L. de Moura, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 237, Dagstuhl, Germany: Schloss Dagstuhl—Leibniz-Zentrum fur Informatik, 2022, 8:1–8:18, ISBN: 978-3-95977-252-5. DOI: 10.4230/LIPIcs.ITP.2022.8.

[10] I. Dillig and T. Dillig, "Explain: A tool for performing abductive inference," in *Computer Aided Verification—25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13–19, 2013. Proceedings*, N. Sharygina and H. Veith, Eds., ser. Lecture Notes in Computer Science, Springer, 2013, pp. 684–689. DOI: 10.1007/978-3-642-39799-8_46.

[11] I. Dillig, T. Dillig, and A. Aiken, "Automated error diagnosis using abductive inference," in *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, ACM, 2012, pp. 181–192. DOI: 10.1145/2254064.2254087.

[12] I. Dillig, T. Dillig, B. Li, and K. L. McMillan, "Inductive invariant generation via abductive inference," in *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013*, A. L. Hosking, P. T. Eugster, and C. V. Lopes, Eds., ACM, 2013, pp. 443–456. DOI: 10.1145/2509136.2509511.

[13] M. Echenim and N. Peltier, "A superposition calculus for abductive reasoning," *Journal of Automated Reasoning*, vol. 57, no. 2, pp. 97–134, Aug. 2016. DOI: 10.1007/s10817-015-9344-2.

[14] M. Echenim, N. Peltier, and Y. Sellami, "A generic framework for implicate generation modulo theories," in *Automated Reasoning—9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14–17, 2018, Proceedings*, D. Galmiche, S. Schulz, and R. Sebastiani, Eds., ser. Lecture Notes in Computer Science, Springer, 2018, pp. 279–294. DOI: 10.1007/978-3-319-94205-6_19.

[15] M. Echenim, N. Peltier, and S. Tourret, "Prime implicate generation in equational logic," *Journal of Artificial Intelligence Research*, vol. 60, pp. 827–880, 2017. DOI: 10.1613/JAIR.5481.

[16] H. Lachnitt, M. Fleury, H. Barbosa, *et al.*, "Improving the SMT proof reconstruction pipeline in Isabelle/HOL," in *16th International Conference on Interactive Theorem Proving (ITP 2025)*, Y. Forster and C. Keller, Eds., ser. LIPIcs, vol. 352, Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2025, 26:1–26:22. DOI: 10.4230/LIPIcs.ITP.2025.26.

[17] L. M. de Moura and N. S. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, C. R. Ramakrishnan and J. Rehof, Eds., ser. Lecture Notes in Computer Science, Springer, 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24.

[18] Y. Nagashima, "LiFtEr: Language to encode induction heuristics for Isabelle/HOL," in *Programming Languages and Systems—17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1–4, 2019, Proceedings*, A. W. Lin, Ed., ser. Lecture Notes in Computer Science, Springer, 2019, pp. 266–287. DOI: 10.1007/978-3-030-34175-6_14.

[19] Y. Nagashima and D. S. Goc, *Proof by abduction in Isabelle/HOL*, Extended abstract presented at the 9th Conference on Artificial Intelligence and Theorem Proving (AITP 2024), 2024.

[20] Y. Nagashima and R. Kumar, "A proof strategy language and proof script generation for Isabelle/HOL," in *Automated Deduction—CADE 26—26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6–11, 2017, Proceedings*, L. de Moura, Ed., ser. Lecture Notes in Computer Science, Springer, 2017, pp. 528–545. DOI: 10.1007/978-3-319-63046-5_32.

[21] S. Padhi, E. Polgreen, M. Raghothaman, A. Reynolds, and A. Udupa, "The sygus language standard version 2.1," *CoRR*, vol. abs/2312.06001, 2023. DOI: 10.48550/ARXIV.2312.06001. arXiv: 2312.06001.

[22] L. C. Paulson and J. C. Blanchette, "Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers," in *The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011*, G. Sutcliffe, S. Schulz, and E. Ternovska, Eds., ser. EPiC Series in Computing, EasyChair, 2010, pp. 1–11. DOI: 10.29007/36DT.

[23] A. Reynolds, H. Barbosa, D. Larraz, and C. Tinelli, "Scalable algorithms for abduction via enumerative syntax-guided synthesis," in *Automated Reasoning: 10th International Joint Conference, IJCAR 2020, Paris, France, July 1–4, 2020, Proceedings, Part I*, Paris, France: Springer-Verlag, 2020, pp. 141–160. DOI: 10.1007/978-3-030-51074-9_9.

[24] A. Reynolds, H. Barbosa, A. Nötzli, C. W. Barrett, and C. Tinelli, "cvc4sy: Smart and fast term enumeration for syntax-guided synthesis," in *Computer Aided Verification—31st International Conference, CAV 2019, New York City, NY, USA, July 15–18, 2019, Proceedings,*

Part II, I. Dillig and S. Tasiran, Eds., ser. Lecture Notes in Computer Science, Springer, 2019, pp. 74–83. DOI: 10.1007/978-3-030-25543-5_5.

[25] S. Schulz, S. Cruanes, and P. Vukmirovic, "Faster, higher, stronger: E 2.3," in *Automated Deduction—CADE 27—27th International Conference on Automated Deduction, Natal, Brazil, August 27–30, 2019, Proceedings*, P. Fontaine, Ed., ser. Lecture Notes in Computer Science, Springer, 2019, pp. 495–507. DOI: 10.1007/978-3-030-29436-6_29.

[26] H. Schurr, M. Fleury, and M. Desharnais, "Reliable reconstruction of fine-grained proofs in a proof assistant," in *Proc. Conference on Automated Deduction (CADE)*, A. Platzer and G. Sutcliffe, Eds., ser. Lecture Notes in Computer Science, vol. 12699, Springer, 2021, pp. 450–467. DOI: 10.1007/978-3-030-79876-5_26.

[27] P. Vukmirovic, J. Blanchette, and S. Schulz, "Extending a high-performance prover to higher-order logic," in *Tools and Algorithms for the Construction and Analysis of Systems—29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22–27, 2023, Proceedings, Part II*, S. Sankaranarayanan and N. Sharygina, Eds., ser. Lecture Notes in Computer Science, Springer, 2023, pp. 111–129. DOI: 10.1007/978-3-031-30820-8_10.

[28] C. Wernhard, "Abduction in logic programming as second-order quantifier elimination," in *Frontiers of Combining Systems—9th International Symposium, FroCoS 2013, Nancy, France, September 18–20, 2013. Proceedings*, P. Fontaine, C. Ringeissen, and R. A. Schmidt, Eds., ser. Lecture Notes in Computer Science, Springer, 2013, pp. 103–119. DOI: 10.1007/978-3-642-40885-4_8.