



Experiments on Automating Boolos’ Curious Inference

Alexander Steen¹ and Melanie Taprogge^{1,2}

¹ Institute of Mathematics and Computer Science, University of Greifswald, Greifswald, Germany
{alexander.steen,s-metapr}@uni-greifswald.de

² Université Paris-Saclay, ENS Paris-Saclay, LMF, CNRS, **INRIA**, France

Abstract

Boolos’ Curious Inference (BCI) is a well-known example of a class of first-order reasoning problems that require infeasibly many proof steps in a first-order proof calculus but exhibit short and elegant proofs in second-order (or higher-order) logic. Consequently, BCI remains intractable for contemporary first-order automated theorem provers. Although the original problem formulation is also beyond the capabilities of current higher-order provers, previous work by Benzmüller et al. demonstrates that augmenting the problem statement with suitable higher-order formulas allows these provers to succeed. Building on this work, we introduce a procedure for identifying and generating such candidate augmentations. This procedure is implemented in the new *ciao* tool. Using *ciao* as a preprocessing tool makes BCI fully automatically solvable for the very first time.

1 Introduction

In this paper, we report on empirical experiments for automating Boolos’ Curious Inference (BCI) [9]. BCI is a conceptually simple, yet combinatorially difficult reasoning problem. Intuitively, it asserts that a predicate holds for the number 1, and, if it holds for some number n , it also holds for the successor of n . The task is then to show that the predicate holds for an astronomically large number $a(n, m)$, where a is a fast-growing Ackermann-style function. This reasoning problem was studied by Boolos due to its curious property: It requires an infeasibly high number of modus ponens steps in cut-free first-order calculi, with a proof so large that it cannot realistically be constructed during our lifetime. When moving to second-order logic, however, Boolos shows that a concise proof can be constructed making use of second-order comprehension principles.

In the context of automated reasoning, the correctness of Boolos’ solution was verified by formalizations in various proof assistants [5, 14]. The automation of (reasonably hard instances of) BCI by first- and higher-order ATP systems, however, remains an open challenge. Previous work suggests that, in the long run, only higher-order ATP systems will be able to tackle this challenge, as they, in principle, have the ability to synthesize auxiliary axioms that serve as controlled cuts for introducing proof search shortcuts [7].

In the following, we report on experiments with a fully automatic preprocessing procedure that scans reasoning problems for potential inductive structures and augments them with suitable auxiliary axioms. We show that this procedure recovers the higher-order structure needed

for automatically solving BCI instances. Also, the solving time required to prove simpler BCI instances (already solvable by ATP systems before) is drastically reduced by this approach. The procedure is implemented as a stand-alone tool called **ciao** (comprehension instance **augmentor**), available as open-source software (under a BSD-3 license) via Zenodo [18] and GitHub¹.

In Sect. 2 some important preliminaries are briefly presented. BCI is introduced in Sect. 3. The procedure for generating auxiliary axioms is presented in Sect. 4 and evaluated in Sect. 5. Finally, Sect. 6 concludes and sketches further work.

2 Preliminaries

First-Order Logic. BCI is formulated in classical first-order logic (FOL) with equality. It is assumed the reader is familiar with its syntax and semantics; an introduction is left out. We write $\text{fv}(s)$ for the set of free variables of a term or formula s , and $\text{sym}(s)$ for the set of non-logical symbols occurring in s . Vector notation such as $\vec{x}$ and $\vec{t}$ abbreviates finite sequences of variables or terms.

Higher-Order Logic. The notion of higher-order logic (HOL) employed in this work is a variant of Church's Simple Type Theory [10] as usually employed by contemporary higher-order ATP systems. HOL is based on a simply typed λ -calculus and hence provides λ -notation as an expressive binding mechanism to denote unnamed functions, predicates and sets (by their characteristic functions), and it comes with built-in principles of Boolean and functional extensionality as well as type-restricted comprehension.

HOL is a typed logic. The set of simple types is freely generated from a set of base types and the function type constructor $\rightarrow$. It is assumed that the set of base types includes the type o (for truth values) and ι (for individuals). The syntax is given by the following abstract grammar:

$$s, t ::= c_\tau \in \Sigma \mid X_\tau \in \mathcal{V} \mid (\lambda X_\tau. s_\nu)_{\tau \rightarrow \nu} \mid (s_{\tau \rightarrow \nu} t_\tau)_\nu$$

where Σ is a (typed) signature, and $\mathcal{V}$ is a set of variables with (countably) infinitely many variables for each type. We use upper-case letters for variables of function type, and lower-case letters otherwise. Types (written in subscripts) may be dropped if irrelevant or clear from the context. It is assumed that Σ contains the usual logical connectives; and binary connectives are written infix by convention. Formulas are HOL terms of type o .

HOL automation by ATP systems is generally studied with respect to general semantics [11, 2], for which sound and complete proof calculi can be achieved. Every theorem of HOL with general semantics is also a theorem of HOL with standard semantics (but not vice versa). For a thorough introduction of HOL semantics we refer to relevant literature [8, 3, 6].

HOL validates the following comprehension principle:

$$\exists R_{\vec{\tau} \rightarrow \tau}. \forall x_{\tau_1}^1 \dots x_{\tau_n}^n. (R x^1 \dots x^n) = s_\tau[x^1, \dots, x^n] \quad (\text{COM})$$

where $s[x^1, \dots, x^n]$ denotes a term with $\text{fv}(s) = \{x^1, \dots, x^n\}$, and $R \notin \text{fv}(s)$. The comprehension principle (COM) asserts the existence of functions (hence also predicates, relations, sets) that can be expressed within the language of HOL. Its validity is easy to see in HOL, simply take $R := \lambda x_{\tau_1}^1 \dots x_{\tau_n}^n. s_\tau$. In HOL ATP systems such λ -terms are usually constructed during unification procedures. However, more complicated instances need to be step-wise approximated by primitive substitution, see e.g. inference rule (PS) as implemented in Leo-III [17]. This process is usually a difficult obstacle for proof search if complex terms need to be instantiated.

¹See github.com/leoprover/ciao

3 Boolos' Curious Inference (BCI)

BCI, adapted from Boolos' original article [9], is stated by the following consequence:

$$\forall x. a(x, e) = s(e) \quad (\text{A1})$$

$$\forall x. a(e, s(x)) = s(s(a(e, x))) \quad (\text{A2})$$

$$\forall x y. a(s(x), s(y)) = a(x, a(s(x), y)) \quad (\text{A3})$$

$$d(e) \quad (\text{D1})$$

$$\forall x. d(x) \Rightarrow d(s(x)) \quad (\text{D2})$$

$$\therefore d(a(s(s(s(s(e))))), s(s(s(s(e)))))) \quad (\text{C})$$

The intended interpretation of the problem is as follows: e represents the number 1, and s represents the successor function.² Formulas (A1)–(A3) give an axiomatization of the Ackermann-style function a (see [1]). The predicate d is specified by axioms (D1) and (D2) to contain e and to be closed under s (intuitively intending to capture all natural numbers generated from e). Formula (C), the conjecture to be proved, asserts that d holds for $a(5, 5)$.

A first-order proof is conceptually simple: An exhaustive unfolding of the conjecture term using axioms (A1)–(A3) yields a deeply nested successor term. Proving that d holds for this term can, in principle, be achieved by repeatedly applying axiom (D2) via modus ponens, once for each successor occurring in the fully unfolded term, starting from the base case (D1).

However, in a standard first-order proof system, any such derivation of the conjecture from axioms (A1)–(D2) must contain a number of symbols that is so vast that it is physically impossible to ever write down such a derivation in full [9]. Boolos demonstrates that this phenomenon is inherent to the limited expressive power of FOL [9], a limitation that in practice is also reflected in the behavior of first-order ATPs: The TPTP library contains five instances of this problem, all based on equations (A1)–(D2), but asserting d of a on different arguments, with Boolos' original problem representing the largest instance. In their original form, all but the two smallest remain unproved by ATP systems. Fig. 1 shows BCI in the TPTP cnf format.

In HOL, much shorter proofs are possible, as systems can reason at a more abstract level rather than carrying out the individual unfoldings. This is already demonstrated by the concise proof relying on the comprehension principle of second-order logic presented by Boolos himself [9]. In automated reasoning, however, higher-order versions of the problem obtained by

```
%-----
cnf(ax1,axiom, f(N,one) = s(one) ).
cnf(ax2,axiom, f(one,s(X)) = s(s(f(one,X))) ).
cnf(ax3,axiom, f(s(N),s(X)) = f(N,f(s(N),X)) ).
cnf(ax4,axiom, d(one) ).
cnf(ax5,axiom, ( ~ d(X) | d(s(X)) ) ).

cnf(conj,negated_conjecture, ~ d(f(s(s(s(s(one))))),s(s(s(s(one)))))) ).
%-----
```

Figure 1: BCI problem statement in TPTP cnf format (problem SY0038-1.005.005.p in the TPTP library [19], without metadata header). The function symbol f in the TPTP representation corresponds to the symbol a from above.

²But note that s need not be injective, and e might have predecessors. This is, however, not of major importance for the considerations.

direct translation of the first-order formalization also remain beyond the capabilities of ATPs. Here, the provers face a different kind of difficulty, as current higher-order ATPs are generally unable to discover the necessary lemmas for such proofs fully automatically. A significant step toward overcoming these limitations has recently been made [7]: There, the problem specification is augmented with the following two definitions expressible in HOL:

$$ind = \lambda X_{\iota \rightarrow o}. Xe \wedge \forall y. Xy \Rightarrow X(sy) \quad (\text{IND})$$

$$p = \lambda y z. (\lambda w. \forall X_{\iota \rightarrow o}. ind X \Rightarrow Xw) (ayz) \quad (\text{P})$$

Intuitively, equation (IND) encodes a predicate for being an inductive set over e and s , while equation (P) encodes membership of an instance of a in the least closure generated by this definition. These equations merely introduce shorthand notations for λ -terms. It is important to note that adding these equations as additional assumptions does not change the provability of the problem: The existence of such functions is guaranteed by (COM), and the symbols on the left-hand side of the equations are fresh.

The correspondingly augmented higher-order TPTP problems are solvable by E [15], Leo-III [17], and Zipperposition [4]. As shown by Benzmüller et al., the notations affect the systems' proof search, allowing them to automatically prove lemmas that identify d as an inductive set over e and s , and show that the instance of a is necessarily in any such inductive set, from which the conjecture follows [7]. The proofs found by ATPs are discussed in detail in previous work [7].

4 Synthesis of Shorthand Notations

The augmentations enabling proofs of larger BCI instances were, up until now carried out manually [7]. We address this limitation by outlining a procedure, given in Algorithm 1, for identifying inductively defined structures in reasoning problems and generating the useful shorthand notations fully automatically.

Algorithm 1 is divided into three phases. First, axioms that structurally match common encodings of inductive base cases and steps are collected for each predicate symbol in the problem. Second, the candidate definitions are filtered: We only keep rules for symbols that are relevant to the conjecture and whose collected rules are sufficient to describe an inductive structure. In the third and final phase, the shorthand notations are generated. This includes (i) a predicate characterizing predicates or relations that are closed under the collected rules, (ii) a predicate expressing membership in the least such closure, and (iii) conjecture-guided instantiations of this membership notation.

The individual phases of Algorithm 1 are discussed next.

4.1 Phase 1: Collecting candidate definitions per symbol

During the first phase, the axioms Ax are scanned for formulas that explicitly define inductive structures. For each predicate symbol h , the map $Cand[h]$ collects normalized versions of all detected rules. Currently, three syntactic forms are considered:

$\forall \vec{X}. h(\vec{t})$	base cases,
$\forall \vec{X}. B_1 \wedge \dots \wedge B_k \Rightarrow h(\vec{t})$	implication-based rules,
$\forall \vec{X}. \neg B_1 \vee \dots \vee \neg B_k \vee h(\vec{t})$	Horn clauses.

Algorithm 1 Generate shorthand predicates from explicit inductive definitions

```

1: Input: axioms  $Ax$ , conjecture  $C$ 
2:
3: /* Collect candidate definitions per symbol */
4:  $Cand \leftarrow$  empty map from predicate symbols to sets of rules (default  $\emptyset$ )
5: for each  $A \in Ax$  do
6:   if  $A$  can be normalized to  $\forall \vec{X}. B_1 \wedge \dots \wedge B_k \Rightarrow h(\vec{t})$  then
7:      $\mathcal{B} \leftarrow \langle B_1, \dots, B_k \rangle$ 
8:      $Cand[h] \leftarrow Cand[h] \cup \{(\mathcal{B}, h(\vec{t}))\}$ 
9:
10: /* Select relevant inductive definitions */
11:  $Defs \leftarrow$  empty map from predicate symbols to sets of rules
12: for each  $h$  with  $Cand[h] \neq \emptyset$  do
13:   if  $h \in \text{sym}(C)$  and  $\text{SUFFICIENT}(Cand[h])$  then
14:      $Defs[h] \leftarrow Cand[h]$ 
15:
16: /* Construct shorthand notations */
17:  $Aug \leftarrow \emptyset$ 
18: for each  $h$  with  $Defs[h] \neq \emptyset$  do
19:   /* Closure notation: */
20:    $Aug_1 \leftarrow \text{INDUCTIVERELATIONPREDICATE}(Defs[h])$ 
21:   /* Least-closure notation: */
22:    $Aug_2 \leftarrow \text{LEASTCLOSUREPREDICATE}(Aug_1)$ 
23:    $Aug \leftarrow Aug \cup \{(Aug_1, Aug_2)\}$ 
24:   /* Conjecture-guided shorthand notations: */
25:   for each occurrence  $h(t_1, \dots, t_n)$  in  $C$  do
26:     for each  $i$  such that  $t_i = g(\vec{s})$  do
27:        $Aug^{h,g,i} \leftarrow \text{SPECIALIZEDMEMBERSHIP}(Aug_2, g, i)$ 
28:        $Aug \leftarrow Aug \cup \{Aug^{h,g,i}\}$ 
29: return  $Aug$ 

```

Note that all three shapes are captured by Horn clauses [13]. They correspond to implication-based rules in CNF, and base cases are special instances of Horn clauses where the set of negative literals is empty. For uniformity, we therefore represent each collected instance as a pair

$$(\mathcal{B}, h(\vec{t})), \text{ where the list of premises is } \mathcal{B} = \langle B_1, \dots, B_k \rangle.$$

The predicate symbol h is treated as the symbol defined by the rule, and for each collected formula, the corresponding pair is added to $Cand[h]$.

In the case of BCI, the only predicate for which such rules are collected is d : Equation (D1) is represented as $(\langle \rangle, d(e))$, and Equation (D2) is represented as $(\langle d(x) \rangle, d(s(x)))$. Thus the first phase yields $Cand[d] = \{(\langle \rangle, d(e)), (\langle d(x) \rangle, d(s(x)))\}$.

4.2 Phase 2: Filtering

The second phase filters the candidate definitions collected in $Cand$. First, the algorithm only considers candidates for predicate symbols expected to be relevant for proving the conjecture. Here, we use direct occurrence in the conjecture C as a straightforward criterion; less restrictive heuristics are discussed in Section 6. Second, the algorithm checks whether the collected rules for a predicate symbol plausibly describe an inductive structure, which requires at least one rule describing a base case and one describing an inductive step. Recall that after Phase 1, each candidate rule is represented in the normal form

$$(\mathcal{B}, h(\vec{t})), \text{ with } \mathcal{B} = \langle B_1, \dots, B_k \rangle$$

which can be interpreted as $\forall \vec{X}. B_1 \wedge \dots \wedge B_k \Rightarrow h(\vec{t})$. Rules with $|\mathcal{B}| = 0$ are treated as possible base cases and ones with $|\mathcal{B}| > 0$ are treated as possible inductive steps. The predicate `SUFFICIENT` used in Algorithm 1 therefore keeps $Cand[h]$ only if it contains at least one possible base case and at least one possible inductive step:

$$\text{SUFFICIENT}(R) := \exists(\mathcal{B}_0, h(\vec{t})) \in R. |\mathcal{B}_0| = 0 \wedge \exists(\mathcal{B}_1, h(\vec{u})) \in R. |\mathcal{B}_1| > 0.$$

For BCI, the candidate set collected for d satisfies both filtering conditions: d occurs in the conjecture (c), and $Cand[d]$ contains both the empty-premise rule ($\langle \rangle, d(e)$) and the non-empty-premise rule ($\langle d(x) \rangle, d(s(x))$). Consequently, Phase 2 keeps this candidate definition unchanged, that is, $Defs[d] = Cand[d]$.

4.3 Phase 3: Constructing the shorthand notations

The implementation produces concrete formulas in the `thf` dialect of the TPTP language. Here, we present the generated formulas in standard HOL notation and exemplify the corresponding TPTP augmentations for BCI. The augmented file also contains explicit `thf` type declarations for each generated notation.

Generating *ind*. For every predicate symbol h collected in $Defs$, the algorithm generates a fresh higher-order predicate ind^h , which holds exactly for those predicates or relations that satisfy all defining rules collected for h .

Assume that h has type $\tau = \tau_1 \rightarrow \dots \rightarrow \tau_n \rightarrow o$. Then P is a fresh variable of type τ , and ind^h is a fresh symbol of type $\tau \rightarrow o$. The construction then proceeds in two steps. First, every rule in $Defs[h]$ is translated into a formula over the fresh variable P . For a rule $r = (\mathcal{B}, h(\vec{t}))$, with $\mathcal{B} = \langle B_1, \dots, B_k \rangle$, let $\varphi[P/h]$ denote the result of replacing every occurrence of h in φ by P . The rule r is then represented by

$$\hat{r}^P := \forall \text{fv}(r). \left(\bigwedge_{B \in \mathcal{B}} B[P/h] \right) \Rightarrow (h(\vec{t}))[P/h],$$

where $\text{fv}(r) = \text{fv}(B_1) \cup \dots \cup \text{fv}(B_k) \cup \text{fv}(h(\vec{t}))$. As usual, the empty conjunction is interpreted as $\top$, so base rules with $|\mathcal{B}| = 0$ reduce to universally quantified instances of the renamed rule head. For typed problems, the variables in $\text{fv}(r)$ are quantified with the types recovered from the quantifier context of the original formula.

For BCI, h is the predicate d of type $\iota \rightarrow o$, the fresh variable is therefore a predicate variable $P_{\iota \rightarrow o}$, and ind^d has type $(\iota \rightarrow o) \rightarrow o$. Writing the two rules in $Defs[d]$ as r_0 and r_1 , their translations are $\hat{r}_0^P = P e$ and $\hat{r}_1^P = \forall x_{\iota}. P x \Rightarrow P(s x)$.

Second, the conjunction of all formulas in $Defs[h]$ is formed, and the fresh variable P is abstracted over:

$$ind^h = \lambda P_{\tau}. \bigwedge_{r \in Defs[h]} \hat{r}^P.$$

Thus ind^h characterizes the predicates or relations of the same type as h that contain all base facts and are closed under all inductive steps collected for h . For BCI, this gives $ind^d = \lambda P_{\iota \rightarrow o}. P e \wedge \forall x_{\iota}. P x \Rightarrow P(s x)$. This is precisely the notation shown in equation (IND), originally introduced as part of the manual augmentation of BCI by Benzmüller et al. [7].

```

%-----
thf(ind0_decl,          type, ind0: (( $\$i > \$o$ ) >  $\$o$ )).
thf(intersect_ind0_decl, type, intersect_ind0: ( $\$i > \$o$ )).
thf(intersect_ind_f0_decl, type, intersect_ind_f0: ( $\$i > (\$i > \$o)$ )).
thf(ind0_axiom,
  ( ind0 = ( ^ [RO:  $\$i > \$o$ ] :
    ( ! [X:  $\$i$ ] :
      ( ( RO @ X )
        => ( RO @ ( s @ X ) ) )
      & ( RO @ one ) ) ) ) ).
thf(intersect_ind0_axiom,
  ( intersect_ind0 = ( ^ [AO:  $\$i$ ] :
    ! [RO:  $\$i > \$o$ ] :
      ( ( ind0 @ RO )
        => ( RO @ AO ) ) ) ) ).
thf(intersect_ind_f0_axiom,
  ( intersect_ind_f0
    = ( ^ [A1:  $\$i$ , AO:  $\$i$ ] : ( intersect_ind0 @ ( f @ A1 @ AO ) ) ) ) ).
%-----

```

Figure 2: Augmentation for BCI instance from Fig. 1. The generated symbols `ind0`, `intersect_ind0` and `intersect_ind_f0` correspond to ind^d , $least^d$, and $least^{d,a,1}$, respectively.

Generating the least-closure notation. The second generated notation defines a fresh symbol $least^h$ of type τ with the following shape:

$$least^h = \lambda y_1 \tau_1 \dots y_n \tau_n. \forall P_\tau. ind^h P \Rightarrow P y_1 \dots y_n.$$

In the body of the definition, $\forall P. ind^h P \Rightarrow P y_1 \dots y_n$ holds for any tuple $(y_1, \dots, y_n)$ that is contained in every predicate or relation satisfying ind^h . The predicate $least^h$ therefore describes membership in the least closure determined by ind^h . For BCI, this gives $least^d = \lambda y_\iota. \forall P_{\iota \rightarrow o}. ind^d P \Rightarrow P y$.

Generating specialized least-closure notations. The two notations defined above make the inductive structure and membership in its least closure explicit. For BCI, however, the general least-closure notation is not enough to render the TPTP representations of the problem provable. The second notation introduced manually in previous work, shown above as equation (P), is more specific: it names the membership of applications of a in this least closure.

To generate such notations, the algorithm inspects occurrences of h in the conjecture by scanning it for atoms of the following shape:

$$h(t_1, \dots, t_n) \text{ with a } t_i \text{ of the shape } t_i = g(\vec{s}) \text{ with } g : \vec{\sigma} \rightarrow \tau_i$$

The algorithm then generates a specialized least-closure notation by abstracting over the arguments of g and, if h is not unary, over the remaining arguments of $least^h$. This results in a fresh symbol $least^{h,g,i}$ of type $\vec{\sigma} \rightarrow \vec{\tau}_{\neq i} \rightarrow o$.

$$least^{h,g,i} = \lambda \vec{x}_{\vec{\sigma}} \vec{y}_{\vec{\tau}_{\neq i}}. least^h \vec{y}_{<i} (g \vec{x}) \vec{y}_{>i}.$$

Here, $\vec{y}_{\vec{\tau}_{\neq i}}$ abbreviates the variables for all argument positions of $least^h$ except the i -th one, and $\vec{y}_{<i}$ and $\vec{y}_{>i}$ denote the corresponding arguments before and after that position. If h is unary, these additional $\vec{y}$ -arguments disappear. For BCI, the relevant atom is the conjecture itself, which is headed by the unary predicate d , and its argument is headed by the binary

function a . The corresponding specialized symbol therefore has type $\iota \rightarrow \iota \rightarrow o$ and is given by $least^{d,a,1} = \lambda y_\iota z_\iota. least^d(ayz)$. This is the counterpart of equation (P): it abstracts over the arguments of a and states that the resulting term is in the least closure determined by d .

The augmentation produced by Algorithm 1 on BCI from Fig. 1 is shown in Fig. 2.

5 Evaluation

The procedure detailed above is implemented in `ciao`. The tool accepts problems in the first-order TPTP dialects `cnf` (clausal normal form), `fof` (first-order form), and `tff` (typed first-order form), and augments them with additional `thf` (typed higher-order form) formulas.

Here, we evaluate `ciao` on the BCI instances contained in the TPTP library [19]. For each problem, we compare three inputs: the original first-order problem (given in the library in the `cnf` format), its direct `thf` translation (i.e., merely a syntactic variation) produced with the `tptp-utils` tool [16], and the `thf` translation of the `ciao`-augmented problem³. All three inputs are then passed to `eprover-ho` from E 3.2.5-ho [15, 20], using the following command:

```
eprover-ho --delete-bad-limit=2000000000 --definitional-cnf=24 -s -R \
--proof-object --auto-schedule=8 --cpu-limit=360 <input>
```

The experiments were run on an 8-core Intel Core i9 @ 2.3 GHz system with 32 GB DDR4 memory at 2667 MHz, and the results are summarized in Table 1. The average time needed by `ciao` to generate the augmentation was 0.526 seconds.

Conjecture	TPTP file	Original FOL	Direct THF	THF + <code>ciao</code>
$d(a(3, 3))$	SY0038-1.003.003.p	0.049	0.045	0.045
$d(a(3, 4))$	SY0038-1.003.004.p	34.702	65.162	0.100
$d(a(2, a(3, 4)))$	SY0038-1.002.304.p	TIMEOUT	TIMEOUT	0.092
$d(a(4, 4))$	SY0038-1.004.004.p	TIMEOUT	TIMEOUT	0.192
$d(a(5, 5))$	SY0038-1.005.005.p	TIMEOUT	TIMEOUT	0.833

Table 1: TPTP instances of BCI used in the evaluation. Runtime cells record E wall-clock runtime in seconds for successful runs, and TIMEOUT for timeouts.

The smallest BCI instance, SY0038-1.003.003.p, is solved within milliseconds in all three settings, and the proof found for the augmented version of this instance does not use the generated notations. This reflects that the corresponding first-order unfolding is still small enough for E to handle directly without much difficulty.

The next instance, SY0038-1.003.004.p, can likewise be solved without augmentation: E finds a proof of the original first-order problem in 34.702 seconds, and the direct THF translation in 65.162 seconds. With the `ciao`-generated formulas, the runtime is reduced to 0.100 seconds, and the resulting proof uses the generated symbols corresponding to ind^d , $least^d$, and $least^{d,a,1}$.

The most significant effect of the augmentation can be observed for the three larger instances: The original first-order problems SY0038-1.002.304.p, SY0038-1.004.004.p, and SY0038-1.005.005.p all time out, as do their direct THF translations. These three problems all have a TPTP library difficulty rating of 1.00, indicating that they had not been solved by any fully automatic system in the rating set. This is consistent with the discussion in Section 3: in first-order form, the proof search has to proceed through an enormous number of concrete unfolding and successor steps, while a direct higher-order translation does not by itself provide

³This is necessary because E does not accept TPTP problems that mix `cnf` and `thf` formulas.

E with the comprehension-style lemmas needed for the short proof. After augmentation with *ciao*, all three instances become provable in less than one second of E wall-clock time, and all found proofs use the generated shorthand notations (ind^d , $least^d$, and $least^{d,a,1}$).

6 Conclusion and Future Work

In this paper, we presented a syntax-guided procedure for detecting explicitly stated inductive structures in first-order reasoning problems, and generating higher-order shorthand notations describing them. We implemented the procedure in the fully automated, stand-alone pre-processing tool *ciao*. This tool was evaluated on the BCI instances of the TPTP library, showing that it discovers suitable shorthand notations that render the problems provable for higher-order ATP systems. As a result, all BCI instances in the TPTP library become fully automatically provable, including – up to the author's knowledge for the very first time – the original $d(a(5, 5))$ instance.

Although *ciao* was motivated by BCI, the developed procedure addresses inductive structures more broadly. Within the TPTP library, similar encodings occur across domains, including list and data-structure problems in *DAT* and *SWC*, successor arithmetic in *NUM*, rewrite sequences in *SYO*, and algebraic iteration in *KLE*. Preliminary experiments suggest that *ciao*-style augmentations accurately describe many of these structures and can improve proof search.

In future work, we will extend the tool to address more general applicability: First, *ciao* currently only considers symbols occurring directly in the conjecture as candidates for inductive predicates or specialized least-closure functions. This criterion can be weakened, for instance by following strategies established in relevance filtering [12]. Second, the current implementation recognizes only a limited range of syntactic presentations of inductive structures. Future versions will also consider other common encodings, such as equations or equivalences.

Currently, a principal limitation is the inability to recognize inductive structures that are not stated explicitly. This occurs, for example, in the Isabelle-derived algebra problems from the *ALG* and *SWW* domains. For such cases, and for explicitly stated structures that do not follow any of the common variants we cover, a promising direction could be to use generative AI to suggest candidate augmentations, followed by formal checks ensuring that any added formulas are tautological and represent conservative shorthand notations rather than adding semantically new assumptions.

Overall, *ciao* renders three TPTP instances of BCI fully automatically provable for the first time, while providing an extensible foundation for the more efficient handling of first-order problems with inductive structure.

References

- [1] Wilhelm Ackermann. Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematische Annalen*, 99(1):118–133, 1928.
- [2] Peter B. Andrews. General models, descriptions, and choice in type theory. *J. Symb. Log.*, 37(2):385–394, 1972.
- [3] Peter B. Andrews. *An introduction to mathematical logic and type theory - to truth through proof*. Computer science and applied mathematics. Academic Press, 1986.
- [4] Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, Petar Vukmirović, and Uwe Waldmann. Superposition with lambdas. *J. Autom. Reason.*, 65(7):893–940, 2021.
- [5] Christoph Benzmüller and Chad Brown. The curious inference of Boolos in MIZAR and OMEGA. In Roman Matuszewski and Anna Zalewska, editors, *From Insight to Proof – Festschrift in Honour*

- of Andrzej Trybulec, volume 10(23) of *Studies in Logic, Grammar, and Rhetoric*, pages 299–388. The University of Białystok, Poland, 2007.
- [6] Christoph Benzmüller, Chad E. Brown, and Michael Kohlhase. Higher-order semantics and extensionality. *J. Symb. Log.*, 69(4):1027–1088, 2004.
 - [7] Christoph Benzmüller, David Fuenmayor, Alexander Steen, and Geoff Sutcliffe. Who finds the short proof? *Log. J. IGPL*, 32(3):442–464, 2024.
 - [8] Christoph Benzmüller and Peter Andrews. Church's Type Theory. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2025 edition, 2025.
 - [9] George Boolos. A curious inference. *J. Philos. Log.*, 16(1):1–12, 1987.
 - [10] Alonzo Church. A formulation of the simple theory of types. *J. Symb. Log.*, 5(2):56–68, 1940.
 - [11] Leon Henkin. Completeness in the theory of types. *J. Symb. Log.*, 15(2):81–91, 1950.
 - [12] Kryštof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction – CADE-23*, volume 6803 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2011.
 - [13] Alfred Horn. On sentences which are true of direct unions of algebras. *J. Symb. Log.*, 16(1):14–21, 1951.
 - [14] Jeffrey Ketland. Boolos's curious inference in Isabelle/HOL. *Archive of Formal Proofs*, pages 1–19, June 2022. Available via https://isa-afp.org/entries/Boolos_Curious_Inference.html.
 - [15] Stephan Schulz. E - a brainiac theorem prover. *AI Commun.*, 15(2-3):111–126, 2002.
 - [16] Alexander Steen. tptp-utils 1.4.2. Zenodo, September 2025. Available via [10.5281/zenodo.17205289](https://zenodo.org/record/17205289).
 - [17] Alexander Steen and Christoph Benzmüller. Extensional Higher-Order Paramodulation in Leo-III. *J. Autom. Reason.*, 65(6):775–807, 2021.
 - [18] Alexander Steen and Melanie Taprogge. ciao 1.0. Zenodo, June 2026. Available via <https://doi.org/10.5281/zenodo.20743945>.
 - [19] G. Sutcliffe. Stepping Stones in the TPTP World. In C. Benzmüller, M. Heule, and R. Schmidt, editors, *Proceedings of the 12th International Joint Conference on Automated Reasoning*, number 14739 in *Lecture Notes in Artificial Intelligence*, pages 30–50, 2024.
 - [20] Petar Vukmirović, Jasmin Blanchette, and Stephan Schulz. Extending a High-Performance Prover to Higher-Order Logic. In *TACAS (2)*, volume 13994 of *Lecture Notes in Computer Science*, pages 111–129. Springer, 2023.