Prepublication November 2026

# THORN: Architecture and Soundness Results for a Near-Prolog Theorem Prover

Mark Tarver
Shen Group
dr.mtarver@gmail.com

## Abstract

THORN is a compact theorem prover that adapts the architecture of the Prolog Technology Theorem Prover (PTTP) to single-conclusion sequents. It retains occurs-check unification, compilation into contrapositives and incrementally bounded depth-first search, while replacing ancestor framing with the insertion of complementary local assumptions. This architecture supports quantified queries beyond conventional Prolog goal lists and incorporates conditional equality reasoning. We examine THORN's relationship to PTTP and develop a route to soundness through simulation in an augmented PTTP theory. We establish soundness, completeness and termination for its propositional preprover, explain the limitations of first-order reasoning without compiled axioms, and exhibit incompleteness in its equality procedure. Implementation and performance results illustrate the benefits and limitations of the approach. We conclude by mentioning how guidance from large language models can mitigate its principal weakness: the combinatorial growth of unguided proof search.

## Introduction

The Prolog Technology Theorem Prover (PTTP) was introduced by Mark Stickel [9] in the mid-80s to extend Prolog inferencing to first-order logic. At that time logic programming had made great strides in the speed of the execution of Prolog programs. The progress from the early efforts of a decade before with Marseille Prolog in 1973 to Quintus Prolog based on the Warren Abstract Machine (WAM) [16] on the desktop Sun 3/50 represented an improvement of more than two orders of magnitude wrt LIPS (logical inferences per second).

Stickel believed that a theorem prover based on Prolog technology could occupy the same place in theorem proving as the brute force chess engines that, by that time, were playing increasingly effective chess. Out of this idea came the implementation of the PTTP [10] which is described in this paper. In the event, though Stickel achieved some of the raw performance he expected, the PTTP running faster than its rivals in terms of LIPS, the problem-solving capability of the PTTP was less than he hoped for [12].

In this paper we will examine THORN (Theorem prover based on HORN clause logic) which makes many of the same architectural moves as Stickel's PTTP, but is adapted towards reasoning with sequents rather than Prolog goals. We will first reprise the design of the PTTP and then compare the approach of THORN. We will then extend our attention to the problems that the PTTP was not designed to compute and how THORN's sequent based architecture allows THORN to address them. We will later examine the soundness, completeness and termination results wrt THORN.

Lastly we will look at Stickel's critique of the PTTP and to what degree that applies to THORN. We will argue that THORN has specific strengths play to the collaboration of LLMs and that this line of thought has been empirically confirmed in the CoST (Codex/Shen/THORN) program for solving first-order problems.

## The Architecture of the PTTP

The PTTP copies conventional resolution theorem provers by first reducing the list of assumptions to a list of clauses. The next move is more distinctive. For every clause of *n* literals, the PTTP generates *n contrapositives*. A contrapositive is generated from a clause $[l_1,\ldots,l_n]$ of n literals by selecting one literal $l_i$ as the head and the rest as candidates for the body. The literals destined for the body have their polarities switched (that is a negative literal of the form ~*p* becomes *p* and a positive literal of the form *p* becomes ~*p*) before they are assembled into the body.

Example: [*p*, ~ *q*, *r*] becomes (1) *p* :- *q*, ~ *r*. (2) ~*q* :- ~*p*, ~*r*. (3) *r* :- ~*p*, *q*.

The literals in the contrapositives are *signed*; that is, they acquire an extra argument, indicating their original polarity. Thus *p* :- *q*, ~*r*. might become (*p true*) :- (*q true*), (*r false*). At this point the contrapositives have collapsed syntactically into Horn clauses and can be compiled into a logic program.

However ordinary Prolog execution is insufficient to solve first-order problems expressed in this idiom, and so Stickel instituted the following changes for the PTTP.

1. The restoration of occurs-check unification. By default standard Prolog does not use occurs-check unification which allows illegal unifications and non-terminating dereference chains leading to memory failure.

2. The abandonment of Prolog's unrestricted depth-first search as the *modus operandi* of search. Instead Stickel replaced this by *incrementally bounded depth first search* which prevented the PTTP from going down an infinite branch of the search tree.

3. The introduction of *reduction* and *contraction* as additional inference mechanisms.

Reduction was an addition to Prolog's inference mechanism that allowed the PTTP to emulate ancestor resolution and tackle a wider range of problems than Prolog. When a goal G rose to the top of the goal stack and became the current goal, it was not simply popped off the goal stack but popped off and the *framed* version pushed on the stack in its place. That is G became an object of inference just as in Prolog, but the framed goal |G| remained on the stack, recording its existence in the derivation. If a framed goal came to the front of the stack, it was discarded in *contraction* (since it recorded the existence of a goal that had been solved).

However a framed goal could contribute to a solution by resolving with a current goal. If $(f\ b\ x_1 \ldots x_n)$ was the current goal and $|\ (f\ b^*\ y_1\ \ldots\ y_n)\ |$ was a framed goal where *b* and *b** were polar opposites (in our scheme different Booleans) and $x_1 \ldots x_n$ and $y_1\ \ldots\ y_n$ unified then $(f\ b\ x_1 \ldots x_n)$ was treated as solved and the MGU was used to dereference the unsolved goals. This was Stickel's version of ancestor resolution - called *reduction* in the PTTP and it gave the PTTP the capability to solve first-order problems that could not be addressed by Prolog reasoning [10].

Example: we are given $\forall x\ (((f\ x) \vee (g\ x)) \rightarrow (h\ x)), ((f\ a) \vee (g\ a)) \vdash (h\ a)$ to prove. The assumptions are not all Horn clauses and there is no unit clause. The problem cannot be solved by Prolog reasoning. We generate these contrapositives (for clarity we will omit signing and take it as read).

1. $(h\ X)$ :- $(f\ X)$
2. $(h\ X)$ :- $(g\ X)$
3. $(\sim (f\ X))$ :- $(\sim (h\ X))$
4. $(\sim (g\ X))$ :- $(\sim (h\ X))$
5. $(f\ a)$ :- $(\sim (g\ a))$
6. $(g\ a)$ :- $(\sim (f\ a))$

The goal is $\vdash (h\ a)$. By 1. the goal is

$\vdash (f\ a), |(h\ a)|$

by 5. $\vdash (\sim (g\ a)), |(f\ a)|, |(h\ a)|$.

by 4. $\vdash (\sim (\text{h a})), |(\sim (g\ a))|, |(f\ a)|, |(h\ a)|$.

solved by reduction $((\sim (h\ a))$ with $|(h\ a)|$.

The remaining framed goals are removed by contraction. The PTTP is sound and complete wrt the first-order problems that can be submitted to it.

## THORN and the PTTP

The PTTP can process any query expressible as a conjunction of literals (what we shall call here a *PTTP query*). However not all first-order queries are PTTP queries. The query, $p \rightarrow q$ for example is not a PTTP query. The obvious answer is to assume $p$ in the course of proving $q$. However the idea of carrying local assumptions requires a richer data structure than the PTTP accommodates. The new object of proof is not a goal, but rather a goal with a list of local assumptions for proving it. That is, the object of proof is a (single conclusion) *sequent*.

THORN makes exactly this move; the objects of proof are sequents and the contrapositives of the PTTP are interpreted as derivation rules within a sequent calculus. $q$ :- $p$ is interpreted to mean that any sequent whose conclusion is $q$ can be replaced by one whose conclusion is $p$. The natural idea - that proving $p \rightarrow q$ involves assuming $p$ and proving for $q$ - falls out from a broader strategy for dealing with first order queries.

This strategy for non-quantified queries is as follows; convert the query into conjunctive normal form (effectively a list of clauses) and solve each disjunction $(p_1 \vee \ldots p_n)$ in turn as follows. Select one $p_i$ and make it the conclusion of the sequent and form the complements of the rest and make them assumptions; test the sequent to see if the conclusion is in the assumptions.

Quantifiers in a quantified query must be eliminated before the above technique can be used. Quantifier elimination follows the procedure of conversion to prenex form and the removal of the quantifiers with the proviso that the usual Skolemisation procedure is inverted. Universally bound

variables are replaced by arbitrary terms subject to the *eigenvariable* condition and existentially bound variables generate free variables. This corresponds to the convention that a universally quantified proposition is proved for an arbitrary case but an existentially bound one by finding a witness.

This treatment of quantifiers in queries is *Herbrandisation*, [2] which eliminates the universal quantifiers, but preserves derivability (that is, the Herbrandised conclusion is derivable just when the original is). THORN follows this by treating the existentially bound variables as free variables, discarding the existential quantifiers. After quantifier elimination the treatment follows the propositional pattern *modulo* unification.

This leaves only one remaining point of architectural difference. THORN follows the practice of using occurs-check unification and incrementally bounded search, copying the PTTP faithfully. However framing is replaced by a process called *indirect chaining*, whereby the usual Prolog backward chaining mechanism is supplemented. The idea is that when a sequent is subject to an inference rule, the complement of the conclusion is inserted into the list of local assumptions. This corresponds to the principle that it is always valid to assume the negation of what is to be proved at any stage of a proof – a principle that is integral to indirect proof – hence the phrase *indirect chaining* to describe this process.

Indirect chaining achieves the same effect as Stickel's framing, since it memoises a past goal and places the complement into the list of local assumptions. It is a fairly straightforward inductive proof (see later) to show that THORN proofs exactly mirror those of the PTTP in this regard *modulo* the architectural differences in the data structures. Moreover loop detection in the PTTP (detected by seeing if a current goal is identical to a framed goal) is again, with architectural changes, demonstrable in THORN (check to see if the conclusion is the complement of a local assumption introduced in the course of proof).

Note if the axioms supplied to the THORN compiler generate a Horn clause set, that that in the event THORN assumes that Prolog inferencing is sufficient to solve any problem submitted wrt that axiom set and hence backward chaining, and not indirect chaining, is used. In summary THORN runs under six basic derivation rules.

1. *Herbrandisation*; prefaced by conversion to prenex form, used to eliminate universal quantifiers as an initial step.
2. $\exists$*right*: given a goal $\exists x\ p$, generate $p[Y/x]$ where $Y$ is a fresh variable that replaces all free instances of $x$. This is effectively a basis of Prolog's query approach.
3. &*right*; given $(p \,\&\, q)$ to prove, prove $p$ and then $q$.
4. *Hypdisj*; which states that given a disjunction $(p_1 \vee \ldots p_n)$ to prove, select any $p_i$ and form a context $\Delta$ composed of the complements of every $p_j$ not equal to $p_i$ and solve for $\Delta$ |- $p_i$.
5. *Indirect chaining*; given a sequent $\Delta$ |- $p$ and a program clause $p^*$ :- $q_1 \ldots q_n$ (standardised apart), if $p$ and $p^*$ unify with MGU $\sigma$ replace $\Delta$ |- $p$ on the stack by by $\sim p$, $\Delta$ |- $q_1 \ldots \sim p$, $\Delta$ |- $q_n$ and dereference the variables on the stack by $\sigma$. In the case where the clause is $p^*$ (a unit clause), remove $\Delta$ |- $p$ from the stack and dereference the stack variables by $\sigma$.
6. *Proof by hypothesis*; given a sequent $\Delta$ |- $p$ where $p$ is unifiable with some member of $\Delta$, treat the sequent as solved, using the MGU to dereference the remaining stack.

## Soundness, Completeness, Termination in Propositional Calculus

Theorem: *THORN is sound, complete and terminating for propositional calculus*

Assume the input is ground and quantifier free. THORN converts the query into CNF. Let $P$ be any propositional calculus theorem, then $P$ is a tautology and the CNF $PC = (P_a \ldots \& P_m)$ of $P$ is a tautology. If $PC$ is a tautology then every $P_i$ in $(P_a \ldots \& P_m)$ is a tautology. If $P_i = (Q_1 \ldots Q_n)$ is a tautology iff $Qi$ and $\sim Qi$ occur in $P_i$. Therefore making either $Qi$ and $\sim Qi$ the conclusion and making the other a complement in the assumptions will lead to a proof by hypothesis. By searching all $n$ such arrangements in the manner of THORN, a proof of this form will be found if it exists. We refer to this process as the *preprover* process and the above soundness, completeness and termination properties wrt PC are guaranteed in the absence of any compiled axiom set.

## Working from a Zero Axiom Set

THORN solves tautologies very quickly, and the process is extended to incorporate unification. Thus a conclusion can be proved from a hypothesis where the two unify. The fact that THORN uses unification and is guaranteed to terminate makes THORN is the absence of compiled axioms *a weak first-order theorem-prover in its own right*. It is sound, complete and terminating for propositional calculus, but sound, terminating and therefore incomplete for first-order logic.

THORN generates a sequent where all the assumptions are literals and the conclusion is a literal and this can be subject to THORN's sequent based interpretation of contrapositives. Hence at the point where THORN cannot prove the sequent, *it calls upon the stored axioms*. The zero axioms case is where no such axioms exist (the background theory is empty), in which case THORN collapses into this weak ATP. As said THORN is a weak prover in the absence of compiled axioms and fails several of the quantified Pelletier problems, though it solves the propositional ones very quickly.

Why does the preprover sometimes fail at quantified problems? This is due to the rigid nature of the variables generated. Any universally quantified assumption dynamically assumed by THORN is not compiled to code, but instead represented by locally bound variables each of which can take one and only one value. A proof where a universally quantified statement needs to be instantiated in more than way is not computable by the preprover. An example of such a problem is

$(\forall x\ ((jewish\ (mother\ x)) \rightarrow (jewish\ x))) \ \&\ (jewish\ (mother\ (mother\ bob)))) \rightarrow (jewish\ bob)$

which is first-order provable but which requires using the quantified statement twice with different instantiations. This is precisely the problem afflicting tableau systems for which the standard response is *amplification* – the universally quantified propositions are repeated (amplified) $n$ times with the renaming of variables when $n$ different instantiations are needed. However though we know any solvable first-order problem in tableau is solvable for some finite value of $n$ [1] [3], there is no decision procedure for finding that value. A complete procedure involves iterating amplification until a solution is found. THORN does not use iterate amplification, accepting $n = 1$, so obviously zero axiom THORN is incomplete.

# The Inference Relations Between THORN and the PTTP

First we concentrate on those queries which are (signed) Prolog queries and therefore capable of being processed by the PTTP.

Reflection Theorem: *THORN is inferentially equivalent to the PTTP wrt PTTP queries and so is sound and complete for all such queries.*

The proof is by induction on the length *n* of the prefixes of any branch of the searrch.

**Base case: *n*=1.** The initial goal *G* unifies with a unit clause *H* common to both systems. Both close the branch using the same MGU. Reduction cannot apply because no ancestor has yet been framed; correspondingly, neither THORN nor the PTTP has a local assumption against which to close.

**Inductive Case: assume the correspondence holds through the first n inferences:** the current goals coincide, and THORN's local assumptions are the complements of PTTP's active framed goals, under the same accumulated substitution. For a branch closing at inference *n*+1, there are two cases:

1. **Closure by reduction.** PTTP unifies the current goal *G* with the complement of an active framed goal H. By the inductive hypothesis, THORN has precisely ~*H* among its local assumptions. It therefore closes by hypothesis with the same MGU. Conversely, each such THORN assumption corresponds to an active PTTP frame.
2. **Expansion.** Both systems unify the current goal *G* with a fresh variant of the head of the same contrapositive, using the same MGU. PTTP retains *G* as a framed ancestor of the resulting subgoals; THORN inserts ~G into their local assumptions. Applying the MGU throughout preserves the correspondence for each resulting subgoal, including shared variables. The frame and corresponding local assumption remain available over exactly the same subproof.

In either case, the closing substitution is propagated through the shared variables in the same way.

Our next proof relies on a trick used to coax the PTTP into answering non-PTTP queries which we can call *indirection*. Given any first order problem *P* to solve in the PTTP and a set $\Delta$ of compiled axioms from which to prove *P*, assume and compile $\Delta \cup \{P \rightarrow K)$ where *K* is a fresh symbol. If *K* is fresh, then the only way that a proof can be obtained is through a proof of *P*. We write $\Delta \vdash_{\text{PTTP}} P$ to mean that *P* is derivable in the PTTP from $\Delta$ and $\Delta \vdash_{\text{THORN}} P$ to show that *P* is derivable in the THORN from $\Delta$ using the rules cited so far . We show that THORN as given so far is sound; that is;

Theorem: if $\Delta \vdash_{\text{THORN}} P$ then $\Delta \vdash P$.

Proof: Assume $\Delta \vdash_{\text{THORN}} P$. Assume *P* is in prenex form where the matrix is in CNF. Assume $\Delta \vdash_{\text{THORN}} P$, we will prove $\Delta \vdash P$. Our proof is a proof by induction over the *n* logical constants used in *P*.

Base Case: $n = 0$. Then $P$ is an atom and by the reflection theorem $\Delta \vdash_{\text{PTTP}} P$ and since the PTTP is sound $\Delta \vdash P$ .

Inductive case: the theorem holds for all $Q$ with $n$ constants, let $P$ be a proposition with $n$+1 constants. The proof is by cases.

Case 1: $P = (\sim Q)$. Since $P$ is in CNF, $Q$ is an atom and $P$ is a negative literal and hence by the reflection theorem $\Delta \vdash_{\text{PTTP}} P$ and since the PTTP is sound $\Delta \vdash P$ .

Case 2: $P = (Q \vee R)$; since $\Delta \vdash_{\text{THORN}} (Q \vee R)$ then by *hypdisj* $\Delta \cup \{\sim Q\} \vdash_{\text{THORN}} R$ or $\Delta \cup \{\sim R\} \vdash_{\text{THORN}} Q$. By the inductive hypothesis since $R$ and $Q$ are of rank $n$ then either $\Delta \cup \{\sim Q\} \cup \{R \rightarrow K\} \vdash_{\text{PTTP}} K$ or $\Delta \cup \{\sim R\} \cup \{Q \rightarrow K\} \vdash_{\text{PTTP}} K$. But if $K$ is fresh, given the PTTP is sound, then either $\Delta \cup \{\sim Q\} \vdash R$ or $\Delta \cup \{\sim R\} \vdash Q$ and hence $\Delta \vdash (Q \vee R)$ and so $\Delta \vdash P$.

Case 3: $P = (Q \ \& \ R)$; since $\Delta \vdash_{\text{THORN}} (Q \ \& \ R)$ then by *&r*, $\Delta \vdash_{\text{THORN}} Q$ and $\Delta \vdash_{\text{THORN}} R$ and by the inductive hypothesis $\Delta \cup \{Q \rightarrow K) \vdash_{\text{PTTP}} K$ and $\Delta \cup \{R \rightarrow K) \vdash_{\text{PTTP}} K$. . But if $K$ is fresh, given the PTTP is sound, then. $\Delta \vdash Q$ and $\Delta \vdash R$ and so $\Delta \vdash (Q \ \& \ R)$ and hence $\Delta \vdash P$.

Case 4: $P = (\exists x \, Q)$; since $\Delta \vdash_{\text{THORN}} (\exists x \, Q)$, by $\exists$*right* there is some assignment $\sigma$ to the free variable $x$ such that $\Delta \vdash_{\text{THORN}} \sigma(Q)$ and hence by the inductive hypothesis $\Delta \cup \{\sigma(Q) \rightarrow K) \vdash_{\text{PTTP}} K$.. But if $K$ is fresh, given the PTTP is sound, then $\Delta \vdash \sigma(Q)$ and hence by $\exists$ introduction $\Delta \vdash (\exists x \, Q)$, and hence $\Delta \vdash P$.

Case 5: $P = (\forall x \, Q)$; since $\Delta \vdash_{\text{THORN}} (\forall x \, Q)$ then *Herbrandisation* for any fresh symbol assigned to $x$ under $\sigma$, $\Delta \vdash_{\text{THORN}} \sigma(Q)$ and hence by the inductive hypothesis $\Delta \cup \{\sigma(Q) \rightarrow K) \vdash_{\text{PTTP}} K$. But if $K$ is fresh, given the PTTP is sound, then $\Delta \vdash \sigma(Q)$ and hence by $\forall$ introduction $\Delta \vdash (\forall x \, Q)$, and hence $\Delta \vdash P$.

One obvious question here is why not use indirection as the standard theorem-proving technique? There are two reasons for this. The first is that indirection has a computational cost; it requires invoking the compiler for every problem submitted to the theorem prover. This is not a small cost, since any predicate implicated in the query, which is also found in the background theory, will incur the overhead of compiling the background theory as well. A second reason is that the proofs gained by indirection are obviously contrived to meet a limitation of software architecture, rather than dictated by the problem itself and, to that degree, the proofs are more contrived and less natural.

## Equality

Much of the author's interest in automated reasoning is directed at reasoning with conditional equalities in the representation of functional programs. The PTTP has no provision for dealing with equality and so therefore some mechanism had to be added to THORN to enable it to handle equality. Given a goal $(f \ x_1 \ .. \ x_n)$ to prove, with equality reasoning enabled, THORN attaches a special provision which we will describe. Equality reasoning is enabled if either an axiom contains an equality predicate or the problem does.

The special provision is that given ($f\ x_1$ .. $x_n$) to prove, THORN selects a non-variable subterm $y$ of $x_1$ .. $x_n$ and attempts to prove $y = z$ (rewriteA) or $z = y$ (rewriteB) for some $z$ on the condition that $y$ and $z$ are not syntactically identical and replaces $y$ by $z$ in a new goal. The role of the unknown $z$ is supplied by a fresh variable and unification is used. This procedure is applied until a proof is obtained or all possibilities are exhausted. This procedure, based on Leibnitz Law, is obviously sound in first-order logic. The reflexivity of = (*=refl*) is always assumed by the compilation of $\forall x\ x = x$. Thus with the addition these three rules to the 6 rules of inference explained previously, THORN runs under 9 rules of inference.

This obligation to prove $y = z$ or $z = y$ is not restricted to using only unconditional equalities but also extends to conditional equalities. An identity is a goal and can be at the head of a body with zero elements (an unconditional equality) or 1 or more (a conditional one). In either case THORN handles the process in the usual way.

THORN's treatment of equality fails to solve some simple equational problems. Thus from the compilation of ($f\ a$) and ~($f\ b$) one cannot prove in THORN that ~($a = b$). We conjecture that for compilation of purely equational theories, where = is the only predicate and $\forall$ and $\exists$ the only constants, that the THORN equality procedure is complete for any equational query in prenex form where $\forall$ and $\exists$ and = are at most the only logical constants.

## Independent Confirmation of THORN Using Mistral

THORN is an implementation of the proof strategies described in this paper and though the strategies may be sound, there is a question as to whether the encoding reflects those strategies. The LLM Mistral was given this paper and on that basis constructed an interpreter for THORN which was tested over 30,000 proof steps drawn from 179 first-order problems. The test was to see if the results from Mistral's interpreter matched the execution of THORN; they did [15].

## Performance and Implementation

THORN is implemented in Shen [13], a modern logic-functional programming language that incorporates a Prolog compiler. The implementation takes ~600 lines of code and, since Shen runs under 14 platforms, a wide array of languages. The most performant platform is ShenJIT0.9 which compiles Shen (and Shen Prolog) into native code. All the timings discussed in [15] are collected under this platform on an Envy workstation with 16GB of main memory. All input is type checked to be conformant to FOL syntax, including functions that define axiom sets. Every query returns either *true* or *false* under a 5 second time out; and if *true*, a proof is output to a file *prf.txt*. Timeout can be controlled by the user.

A problem in evaluating THORN as an ATP is that many standard benchmarks are designed to be used in a zero-axiom environment which includes many of the Pelletier problems [7]. Aside from the problems 1-17, which are propositional, (all 17 being solved by THORN in 0.03s), the quantificational problems 18-46 showed a rather modest success rate of 18 out of 29. Of the quantificational problems 18-46, 12 are zero-axiom problems of which THORN solved 2.

But zero-axiom problems were never the intended domain of application for THORN. Indeed, such problems cannot be submitted directly to the original PTTP either without first introducing

indirection. In the context of using indirection, THORN is of course a sequent based simulation of the PTTP and it works well enough to solve 9 of the unsolved 11 problems in milliseconds. The remaining two 5-second failures were the result of a combinatorial explosion wrt conversion to CNF.

More interesting are non-zero-axiom problems where the system is asked to prove a result from a set of fixed axioms - which is more characteristic of proofs in mathematics and program verification.

Concentrating on 15 non-zero cases in Pelletier, with exceptions like Schubert's Steamroller (solved in 0.16 sec), the results are modest; the solution rate is 6/15 or only 40%. The later Pelletier problems often need intermediate results that are easy to recognise mathematically but expensive to discover by backward search. THORN repeatedly explores those possibilities without retaining them as named lemmas. Equational reasoning in THORN is very expensive and equational reasoning dominates the later Pelletier problems. Although THORN's algebra proofs are very clear, any algebra proof beyond a dozen steps generally times out under the 5 second limit.

The question arises whether this is all due to the familiar combinatorial explosion or to a failure in the proof process. The answer is - almost certainly the former, because heuristically guided by an LLM selecting proof plans, THORN can solve all 15 problems; (see [15] for a table of results on this compared to the unguided search). This feat is possible within the CoST [14] (Codex/Shen/THORN) planning environment which we describe briefly in the next section.

## Integrating Lightweight Goal Directed ATPs and LLMs

Stickel, initially enthusiastic about the PTTP [9], was later more sombre in his assessment [12] comparing the PTTP to systems like Otter [6].

*Caveat: PTTP's high inference rate is achieved at the cost of not allowing more flexible search strategies or elimination of redundancy in the search space by subsumption. Although PTTP is one of the fastest theorem provers available when evaluated by its inference rate and performance on easy problems and it has been used to solve reasoning problems in planning and natural-language-understanding systems effectively, its high inference rate can be overwhelmed by its exponential search space and it is unsuitable for many difficult theorems.*

This remains true of THORN. Its strengths are the compact encoding, the speed of inferencing which means that moderate (and sometimes difficult) problems succumb quickly, and the clarity of the resulting proofs. THORN proofs are clear because backward chaining is, by the common wisdom, easier to grasp than less directed methods of resolution found in, for example, Prover9 [5] or superposition found in Vampire [4]. Equational reasoning in THORN is simply an application of Leibnitz Law and the proofs are very clear indeed. But the weaknesses observed by Stickel are still true and place both THORN and the PTTP at a disadvantage compared to much bigger and more sophisticated systems like Prover9 and Vampire.

However recent work by the author [14] on combining THORN with heuristic guidance from LLMs in the CoST system (Codex/Shen/THORN) has shown that theorems can be solved in this system which would defy raw search. The key has been using LLM-generated plans to break up search space into small lemmas which THORN could solve *en route* to the final solution. This same technology solved all the non-zero axiom Pelletier problems. This research is fully described in [14].

Here THORN worked very well, contributing to proofs in Region Connection Calculus and purely equational algebra built sometimes of dozens of lemmas (one proof occupying over 20,000 lines). In one group theory proof, the LLM tossed THORN an algebraic lemma that took ~15 million inferences to prove, executed < 1 second. This was one of many cases where raw execution speed proved essential.

Since the LLM tamed the worst aspects of search space, a redeeming quality of THORN, inferential speed and the clarity of the proofs, became *the* defining assets in the project. The need for a massive ATP with domain specific algorithms to tame combinatorial explosions was not a requirement. Instead the focus was on the clarity of the proofs. This might point a pattern for how work in automated reasoning might develop in the future – LLMs with light portable goal directed ATPs . The algebraic proofs were in particular, very understandable *modulo* the complexity of the material which ranged into postgraduate maths.

## Summary

We have described THORN as an extension to the work done on the PTTP to create a more flexible tool for research and application, suitable for a wider range of queries. We have also supplied a soundness proof for THORN and shown it is complete and terminating for propositional calculus. THORN is a lightweight implementation, not designed to be competitive against larger systems like Prover9 and Vampire on challenge problems and not optimised to work in a zero axiom environment. It excels when embedded in a larger system such as an interactive proof editor – where it can drastically cut the cruft from proofs. THORN proved to be excellent when used in tandem with a planning mechanism supplied by an LLM, the biggest plus being the legibility of the resulting proofs.

## References


[1] Bibel, W. (1987). *Automated Theorem Proving*. 2nd revised edition. Vieweg, Braunschweig/Wiesbaden.

[2] Buss, S. R. (1995). "On Herbrand's Theorem." In *Logic and Computational Complexity*, Lecture Notes in Computer Science, vol. 960, Springer, pp. 195–209.

[3] Fitting, M. (1996). *First-Order Logic and Automated Theorem Proving*. 2nd edition. Springer-Verlag.

[4] Kovács, L., Voronkov, A. First-Order Theorem Proving and Vampire. Computer Aided Verification (CAV 2013), LNCS 8044, 1–35, Springer, 2013.

[5] McCune, W. (2010). *Prover9 and Mace4*. Version 2009-11A. Argonne National Laboratory.

[6] McCune, W., and Wos, L. (1997). "Otter: The CADE-13 Competition Incarnations." *Journal of Automated Reasoning*, 18(2), 211–220.

[7]Pelletier, F. J. (1986). "Seventy-Five Problems for Testing Automatic Theorem Provers." *Journal of Automated Reasoning*, **2**, 191–216.

[8] Smullyan, R. M. (1968). *First-Order Logic*. Springer-Verlag, Berlin and Heidelberg. [9] Stickel, M. E. (1984). "A Prolog Technology Theorem Prover." *New Generation Computing*, **2**, 371–383.

[10] Stickel, M. E. A Prolog Technology Theorem Prover. Journal of Automated Reasoning, 4(4), 353–380, 1988.

[11] Stickel, M. E. A Prolog technology theorem prover: a new exposition and implementation in Prolog. *Theoretical Computer Science*, 104(1), 109–128, 1992
[12] Stickel, M. E. Program notes on a Prolog implementation of the PTTP https://courses.cs.umbc.edu/671/fall09/code/prolog/pttp/src/pttp.prolog?utm_source=chatgpt.com
[13] Tarver M. Book of Shen 5th edition, https://www.thegreatbritishbookshop.co.uk/products/the-book-of-shen-5th-edition?_pos=6&_sid=5dada33f0&_ss=r.
[14] Tarver M. CoST home page, shenlanguage.org/CoST.html
[15] THORN home page, shenlanguage.org/THORN.html
[16] Warren, D. H. D. (1983). *An Abstract Prolog Instruction Set*. Technical Note 309, SRI International, Menlo Park, California.