

Building Blocks for Reasoning on the BEAM

Johannes Schuster¹ and Christoph Benzmüller^{1,2}

¹ University of Bamberg, Kapuzinerstraße 16, 96047 Bamberg, Germany
`johannes-christian.schuster@stud.uni-bamberg.de`

² Freie Universität Berlin, Kaiserswerther Str. 16-18, 14195 Berlin, Germany
`christoph.benzmueller@uni-bamberg.de`

Abstract

We report on automated reasoning infrastructure for the Elixir ecosystem and the BEAM virtual machine. Five packages supply the components a higher-order reasoning tool on this platform would otherwise write from scratch: data structures for higher-order terms and proof problems, parsing of the TPTP THF dialects, higher-order unification, a decidable term ordering, and uniform access to external provers. On top of them sits SHOT, a higher-order free-variable tableau prover organised around the actor model, and an interactive environment for teaching automated reasoning. The platform choice also carried the hypothesis that concurrency would repay the constant factor a managed runtime costs on the symbolic inner loop. We report that our current procedure for computing simultaneous closure bottlenecks concurrency. On the TPTP TH0 and TH1 problems a worker pool multiplies inference by a factor of 3.27 over a single worker, the rate of global closure attempts does not follow, and coverage is unchanged. The cause lies in the component that reconciles closure across branches. The libraries are early and under revision, and we present them as such.

Keywords: Automated Theorem Proving, Higher-Order Logic, Tableaux, Concurrency, Elixir

1 Introduction

Automated theorem provers are implemented in a small set of languages. First-order saturation provers such as E and Vampire [30, 23] are written in C and C++, and interactive systems such as Isabelle/HOL [26] in the ML family. The BEAM, the virtual machine underlying Erlang and Elixir [2],¹ has seen little use for theorem proving, and a reasoning tool on this platform starts by writing its term and proof data structures, its TPTP parser, its unification procedure and its bridge to external provers. This paper addresses that gap.

Choosing a managed runtime over an ahead-of-time compiled language is a trade. The BEAM offers preemptive scheduling of lightweight processes, per-process heaps without a globally blocking garbage collector, and failure isolation with supervision. It costs a constant factor on the term-rewriting and unification loop that dominates a prover’s running time. We took the trade on the hypothesis that a proof search decomposed into independent concurrent work would recover the factor. Section 4 reports that our component for simultaneous closure is too weak to support that.

The hypothesis constrains the calculus. A saturation calculus shares its clause set by construction, and mutual simplification makes each inference depend on the others, so its parallelisation is a question about shared state [10]. A tableau calculus decomposes differently. Expanding a branch uses only the formulae on that branch, while closure over rigid free variables is a single constraint on all open branches at once. We take this as a statement about the

¹<https://elixir-lang.org>

shape of the search space. On the current record saturation provers dominate tableau provers at higher order, and nothing below disputes that.

One of the five packages, `AtpClient`, and its front-ends are the subject of a companion paper [32]. We use it here as the prover-access layer of a larger stack and do not re-derive that material.

Section 2 describes the packages and their measured limits, Section 3 the calculus and architecture of SHOT, and Section 4 its evaluation. Section 5 separates what the measurements support from what they do not, and Sections 6 to 8 cover the orchestration and teaching uses, related work, and conclusions.

2 Building Blocks

The five packages are open source under the MIT license and published on Hex, the Elixir package registry, with sources on GitHub² and documentation on HexDocs.³ Each addresses one requirement common to reasoning tools and is usable without the others. Figure 1 shows the composition.

Terms and proofs. `ShotDs` represents the terms of Church’s simple type theory in $\beta\eta$ -long normal form [16] with de Bruijn indices for bound variables, so α -equivalence is syntactic identity and substitution is capture-avoiding without a renaming pass. Terms live in a shared table and are addressed by integer identifiers. Equality of terms is an integer comparison, and a term occurring many times in a derivation is stored once. The rules of Section 3 act on identifiers throughout. `ShotTx` is the main component of the prover and provides the API.

THF parsing. The same package parses a fragment of the TPTP THF dialects [7, 20] and gives access to the TPTP problem library [36]. The parser is hand-written recursive descent over a generated lexer, and it elaborates while it parses. Types are inferred by unification in the style of algorithm W with rank-1 polymorphism, and a polymorphic problem is monomorphised against the types it uses. The output is typed terms in the representation above.

That design has a measurable cost. In the sweep of Section 4, 628 of 5109 problems exceeded the time budget inside the parser and 221 more were rejected by it, so about one problem in six never reached the calculus. The corpus coverage reported below is bounded by the parser.

Separating parsing from elaboration is the response, and it is under way in a separate package, `tptp`⁴, currently at version 0.1.0. Its grammar is generated mechanically from the TPTP syntax BNF. Parsing yields a concrete syntax tree in which every node carries its source span, and no typing, normalisation or logical interpretation takes place; a linter and a pretty-printer sit on the same tree. The reasoning layers are to be rebased onto it. That work is unfinished and nothing here depends on it.

Unification. `ShotUn` implements depth-bounded Huet pre-unification: imitation and projection over flex-rigid pairs, with flex-flex pairs retained as constraints [19]. Problems inside the decidable pattern fragment [24] take a dedicated path. The depth bound makes the procedure sound and complete only in the limit.

²github.com/jcschuster/{ShotDs,ShotUn,ShotTo,ShotTx,AtpClient}

³<https://hexdocs.pm/{shot-ds,shot-un,shot-to,shot-tx,atp-client}>

⁴<https://tptp.hexdocs.pm>

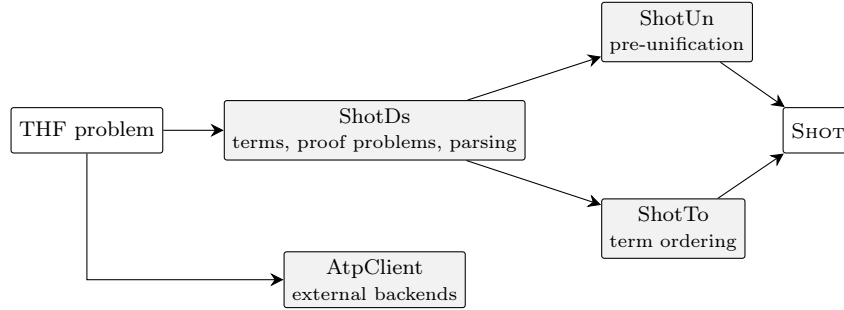


Figure 1: Composition of the packages. The same problem text is either parsed by ShotDs for the native search or dispatched unchanged to an external backend.

Term ordering. ShotTo implements the computability path order for $\beta\eta$ -normal higher-order rewriting [25]. The order’s parameters are supplied as inputs and are not searched for, so comparing one pair of terms is a decidable predicate callable from inside the proof search.

External provers. AtpClient is a uniform Elixir interface to heterogeneous backends. The uniformity is over transports and result formats and not over the input language: local executables, SystemOnTPTP [37] over HTTP, StarExec [34] and Isabelle are reached through one API and return one result type based on the SZS ontology [35]. Isabelle’s `sledgehammer` [9] normalises external-prover output in order to reconstruct an Isabelle proof; AtpClient targets no host proof assistant and returns the normalised verdict to an arbitrary BEAM application. The companion paper [32] describes it together with its front-ends KinoAtpClient and AtpMcp.

Maturity. These packages are a first cut, and the evaluation below assesses them as much as the prover. The interfaces are still moving: at the time of writing the versions are `shot_ds` 1.3.1, `shot_un` 0.2.1, `shot_to` 0.2.0, `shot_tx` 0.1.0 and `atp_client` 0.6.2, and the term representation is being revised along with the parser. They are a working foundation with known limits and not a stable API.

3 SHOT

3.1 Rigidity

SHOT is a refutation tableau prover for classical higher-order logic in the free-variable tradition of leanTAP [5] and leanCoP [28]. Among higher-order systems its closest relatives are TPS [1] and Satallax [12]; other lines are the extensional higher-order paramodulation of Leo-III [33] and λ -superposition as in Zipperposition [38].

Closure in such a tableau is not local to a branch. A variable introduced by a γ -rule is rigid and denotes the same term on every branch, so the tableau closes only under a substitution that closes all open branches simultaneously. A branch that commits to a value for such a variable is either unsound, when a sibling closes under an inconsistent substitution and the search reports success, or incomplete, when the commitment is imposed on the siblings and a global solution binding the variable elsewhere falls outside the searched space.

The design follows from that observation. Expanding a branch and collecting the literal pairs that could close it uses only the formulae of that branch and is independent of all other

branches. Reconciling those possibilities into one substitution is a single constraint over all of them. The rules are arranged so that the first part is as large as possible and the second is confined to one component.

3.2 Calculus

No branch-level rule applies a substitution. A rule that finds two literals which could be made complementary records the disagreement pair as evidence and continues; the pre-unifiers computed to test the pair are discarded. The consequences reach the term level. Demodulation is restricted to rewriting with ground equations under an empty matcher, decided by identity of term identifiers, because rewriting with an equation that carries a rigid variable can destroy the pair a sibling branch needs. Only comparisons stable under every substitution are used. Flex-flex pairs are deferred. The price is that an open branch is an under-approximation. It has not seen the structure the eventual substitution will introduce.

The calculus is in the lineage of extensional higher-order tableaux [21], and discussed in detail in [31]⁵. The rule set is α , β , γ , δ , equality expansion, primitive substitution, ordered demodulation (ordered by ShotTo) and two decomposition rules for atoms. Rules are drawn from a priority queue ordered by static cost, so decomposition precedes branching and the two unbounded generators are tried last. Closure costs 0; tautology and atomic decomposition 1; α , δ and equality expansion at type o cost 2; instantiation over c instances $2 + c$; γ over a type with c instantiations so far $3 + 2c$, or $5 + 2c$ when the instances come from the instance-based generator; the finite γ -rule 3; β 4; extensional equality expansion 10; Leibniz equality expansion 15; primitive substitution at depth d with offset c costs $20 + 5d + 2c$. Leibniz expansion is priced above extensional expansion because it introduces a quantifier at a predicate type, which feeds the most expensive rule in the table.

An equation is expanded according to the type of its arguments. At type o it becomes an equivalence, at a function type it is expanded extensionally, and at any other base type by the Leibniz definition. Equality is therefore handled branch-locally and does not enter the global constraint (Section 3.4).

A quantifier over a type built only from o is enumerated over the finite domain of that type instead of being instantiated with a free variable. Where an opaque argument of type o blocks decomposition, one rule moves it out of argument position onto the branch as an equation and a second discharges the resulting parameter by case analysis. Disabling the finite rule costs 20 refutations in Table 1.

One rule is absent, an oversight that surfaced during the evaluation. Nothing turns a pair of literals with the same rigid head and opposite polarity into a disequality goal, so a branch separable only in that way saturates without producing a candidate. Section 4.4 measures the effect.

The γ counter, the primitive substitution depth and the unification depth are bounded, and iterative deepening raises them together. A rule blocked by a bound is parked and not discarded, so raising the bound resumes the rule where it stopped instead of re-deriving the branch. Removing iterative deepening costs 87 refutations.

Before the expensive rules are reached, a formula is simplified through a reduced ordered binary decision diagram over the identifiers of its atoms. The identifiers give a total and stable variable order for free, so the diagram is canonical and tautology and unsatisfiability on the propositional level are off the rule budget. The readback mentions only the atoms the function depends on.

⁵An interactive Livebook version of the thesis is available at <https://github.com/jcschuster/shot-thesis>.

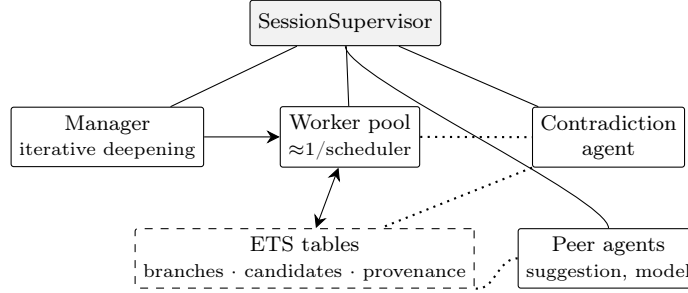


Figure 2: A SHOT session. Solid lines are supervision and coordination, the dashed store is shared state, and dotted lines are the publish–subscribe channel carrying branch evidence.

3.3 Architecture

A session is an OTP supervision tree (Fig. 2) under a `rest_for_one` strategy. The owner of the shared tables starts first, so the tables outlive a crash of the manager or of any agent, and the worker pool starts last.

A branch is immutable data in a shared table and not a process. We considered the alternative of one process per branch, as in Goéland [13], and rejected it. Branches are created faster than they are closed, each retains its full formula set, and all of them publish to the same single consumer, so the memory held would track the size of the tableau instead of the size of the pool.

A worker takes a branch, applies the branch core, and interprets the result. The core is a total function on immutable data. The two effects it may request but not perform are notifying the contradiction agent of candidate disagreements and recording provenance. Workers yield after a bounded number of steps, so branches are fairly redistributed through the work queue. No worker binds a rigid variable.

Workers broadcast branch lifecycle events to a per-session publish–subscribe topic. Ordering between events from different workers is not guaranteed and the subscribers tolerate that; in exchange an additional consumer is free. Two exist: a suggestion agent that proposes instantiations from evidence elsewhere in the tableau, and a model agent that probes a saturated branch for a countermodel through Isabelle.

Expansion records the origin of each fresh variable in the γ -rule or primitive substitution (which is at quantifier instantiation level) that introduced it, so a closed set of branches can be reassembled into a textbook derivation instead of a trace of the search. Figure 3 shows a rendered tableau proof automatically generated from the prover’s output.

3.4 Global closure

The contradiction agent holds, for every open branch, the list of disagreement pairs that would close it. Closure is a constraint satisfaction problem in which one option is chosen per branch such that the union of the choices is simultaneously unifiable. The agent sorts the option lists by length, enumerates the cartesian product lazily, hands the union of each choice to the pre-unification procedure, and stops at the first solution or at the deadline of the proof attempt. A branch with an empty option list aborts the dispatch, since no choice can then succeed. This is a batch variant of incremental closure [18]: it shares the idea of collecting closing options per branch and solving simultaneously, but re-solves the combined constraint at each search

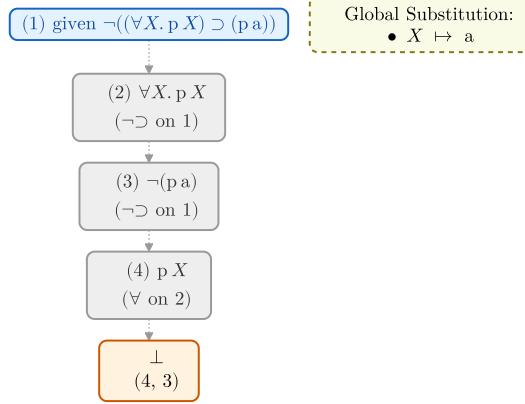


Figure 3: Tableau proof of $(\forall X. p X) \supset p a$, automatically generated from the prover’s output as Mermaid diagram.

rather than maintaining it as branches arrive (Section 5). This permits binding-free branch rules, since no branch backtracks over an instantiation when no branch makes one.

Two properties of the implementation matter for Section 4. At most one search runs at a time, and evidence arriving during a search does not start another. The agent does track whether a completed search answered from a snapshot that later evidence invalidated, but consults that information only when a caller is blocked on a verdict; on the ordinary path a search reporting no closure ends, and the next search waits for the next piece of evidence. Removing the agent costs 364 refutations, more than every other component combined.

The union handed to the unifier carries no equational context. The constraint is simultaneous rigid unification, and equality has already been discharged branch-locally by demodulation and the expansion rules. Treating equality inside the constraint would make this simultaneous rigid *E*-unification [17], which is undecidable [14] already at first order, so a principled version of this component would rest on an incomplete but terminating procedure [15] or on a bounded variant [3]. Section 5 returns to this.

4 Evaluation

4.1 Method

We report a one-at-a-time ablation over the TPTP TH0 and TH1 problems, a study of how the search scales with the worker pool, and runs over a structured set of higher-order problems [6].

Only refutations are scored. A refutation counts when the prover reports a theorem and the problem’s TPTP status agrees. Countermodel claims are counted separately and are not independently attested, since the model agent’s external backend was disabled for these runs. The ablation ran at 2000 ms per problem on an AMD Ryzen 7 4700U with 15 GiB of memory under Arch Linux, with eight schedulers, on 15–16 August 2026, at commit 2a05f05 of ShotTx (Elixir 1.20, OTP 28). The probe of Section 4.3 ran on an AMD Ryzen 7 3700X with sixteen schedulers.

Run-to-run variation is not negligible. Over three repeats of the structured set under identical parameters the number of problems solved moves by about ten, and the standard deviation

configuration	parameter	refut.	net	gained	lost	unique
eager definition unfolding	<code>unfold_defs: :eager</code>	730	+48	72	24	44
no β -variant	<code>beta_variant: false</code>	703	+21	41	20	16
prim. batch 16	<code>prim_subst_batch_size: 16</code>	685	+3	21	18	2
prim. batch 4	<code>prim_subst_batch_size: 4</code>	683	+1	24	23	0
default	—	682	—	—	—	3
no primitive substitution	<code>primitive_substitution: false</code>	681	−1	23	24	2
no suggestion agent	<code>suggestions_enabled: false</code>	681	−1	34	35	11
single worker	<code>worker_pool_size: 1</code>	680	−2	36	38	10
no instance-based γ	<code>instance_based_gamma: false</code>	679	−3	16	19	1
no atom decomposition	<code>atom_decomposition: false</code>	670	−12	26	38	3
no finite α -quantification	<code>finite_o_quantification: false</code>	662	−20	17	37	0
γ -limit 3	<code>initial_gamma_limit: 3</code>	661	−21	17	38	1
γ -limit 5	<code>initial_gamma_limit: 5</code>	657	−25	18	43	3
no demodulation	<code>demodulation: false</code>	641	−41	20	61	6
no iterative deepening	<code>iterative_deepening: false</code>	595	−87	15	102	0
no contradiction agent	<code>contradiction_agent: false</code>	318	−364	5	369	0

Table 1: One-at-a-time ablation, 16 configurations over 5109 problems at 2000 ms, from which 4183 reach the prover. *refut.* counts refutations agreeing with the problem status; *unique* counts problems solved by that configuration and no other.

of the difference between two runs is 5.3 problems. Single-run differences below that are not effects.

4.2 Corpus ablation

Of the 5109 problems, 628 exceed the time budget inside the parser, 221 are rejected by it and 77 carry no conjecture, which leaves 4183 that reach the prover. On these the default configuration reports 685 theorems, of which 682 agree with the problem’s TPTP status, and 62 countermodel claims (branch saturations), of which 61 agree; it exhausts the search budget on 3423 problems and reports 8 unknown and 5 errors.

Table 1 gives the ablation; each row changes one parameter from the default. The *gained* and *lost* columns count problems whose verdict changes in either direction. They are the reason the net column must be read against the variance of Section 4.1: the single-worker configuration differs from the default by two problems net, over 74 problems that change verdict.

The components that carry the search are the contradiction agent, iterative deepening and demodulation, in that order. The default configuration is untuned. Eager definition unfolding alone gains 48 refutations and disabling the β -variant gains 21, either of which exceeds the positive contribution of every component except the agent. Both swept numeric parameters are worse than the default in both directions, which says that the defaults are not obviously misplaced and not that they are optimal. The configurations are also complementary. Together they solve 905 problems against the default’s 743 (refutations + saturations), a fifth more, and several configurations that are worse overall solve problems no other configuration reaches. Every entry in the table is thus a lower bound on what the calculus reaches.

Across all 81 744 rows of the sweep, 22 answers contradict the problem’s TPTP status. Each is a countermodel claimed on a problem whose status is *Theorem*, pointing at an incomplete calculus rendering countermodel claims unsound. No refutation contradicts a status.

	pool	single worker	ratio
<i>sweep, 3383 problems where both exhaust the budget</i>			
tableau steps	5 414 451	1 654 858	3.27
branches closed locally	121 798	38 853	3.13
peak open branches (median)	66	21	3.14
global closure searches	2469	2115	1.17
per 1000 steps	0.46	1.28	0.36
wall clock on the 705 solved by both (ms)	61 003	77 785	1.28
refutations over the full sweep	682	680	—
<i>instrumented probe, sixteen schedulers</i>			
share of wall clock inside a search	46.5 %	51.6 %	0.90
checks suppressed by a search in flight	5 559 732	677 507	8.21
per search dispatched	595	81	7.30
branches per search (mean)	21.41	10.37	2.06
candidates per search (mean)	996.84	161.05	6.19
searches finding a closure	12.4 %	13.6 %	0.91
searches abandoned at the deadline	17.8 %	10.8 %	1.65

Table 2: Worker pool against a single worker. The upper block derives from the sweep of Section 4.2; the lower block is a separately instrumented run.

4.3 Pool against single worker

Table 2 compares the default pool of one worker per scheduler against a single worker.

At the level of branch expansion the pool behaves as designed. It performs more than three times the inference and more than three times the local closure work, holds a frontier three times as wide, and reaches the problems solved by both configurations in about three quarters of the wall clock. None of that reaches coverage. Global closure is attempted 1.17 times as often, which against 3.27 times the inference is a rate per step of 0.36, and the refutation count moves by two problems (sweep variance not measured).

The probe locates the cause. A single search occupies the agent for 46.5 % of wall clock under the pool and for 51.6 % under the single worker, so the saturation of the agent is present without concurrency and is not an artefact of the wider frontier. What the wider frontier changes is the cost of a search. It draws on twice as many branches and six times as many candidate pairs, which lengthens the product the agent enumerates and raises the share of searches abandoned at the deadline from 10.8 % to 17.8 %. Evidence arriving during a search is meanwhile suppressed at 595 triggers per search against 81. More inference yields more evidence, more evidence yields longer searches, and longer searches discard more evidence.

4.4 The structured set

The structured set [6] contains 130 problems targeting known challenge areas of higher-order reasoning: choice, Boolean and functional extensionality, Leibniz against primitive equality, congruence, and two formulations of Cantor’s theorem. Over three repeats SHOT closes 82 of them at least once and 76 in every repeat; 48 are never reached. Table 3 sets the coverage of one such run against reference systems run through SystemOnTPTP via AtpClient at a ten-second limit (SHOT at default timeout of five seconds). No answer of SHOT contradicts an answer of a reference system.

The set is classified by the principles a problem requires, and the classification separates the results. Of the 70 problems needing no functional principle beyond β and η , 57 are closed in every repeat. Of the 35 requiring ξ or functional extensionality, 5 are. The injective Cantor

system	decided	budget exhausted	gave up	unknown
Vampire	121	9	0	0
Leo-III	120	4	6	0
Satallax	118	0	0	12
E	116	2	12	0
Zipperposition	112	6	0	12
SHOT	79	48	0	3

Table 3: Coverage over the 130 problems of the structured set, recorded on 9 August 2026. The SHOT row is one run of three; across the three, 82 problems are closed at least once and 76 in every run. External provers were queried through SystemOnTPTP via AtpClient with a ten second timeout. SHOT was run mostly with default parameters, with a five second timeout.

formulation is closed in none. This is consistent with the missing rule of Section 3: the branches in question separate only by turning a pair of literals with the same rigid head into a disequality goal.

The median solve time over the problems it closes is 0.035 s for SHOT, against 0.012 s for E, 0.013 s for Zipperposition, 0.024 s for Vampire and 1.374 s for Leo-III, with maxima of 4.905, 4.197, 1.411, 0.116 and 2.486 s. These are self-reported times from different machines, and the reference systems were reached through the network, so only the order of magnitude is meaningful.

Four threats to validity remain. Timeouts are cooperative; the prover checks its deadline at rule-application boundaries, and a long unification or parser call can overrun it, which is visible in the parse timeouts above. Countermodel claims are unchecked. Each configuration of the sweep ran once, so only the larger effects in Table 1 are safe to read, and the denominator itself moves slightly between configurations because parse failure is timing-dependent. The figures in Table 3 use externally hosted provers under a different budget on different hardware, and are a coverage comparison and not a performance comparison.

5 The BEAM as a Platform

The measurements separate into three layers with different verdicts. Running external provers is where the runtime pays without qualification. Each invocation is a supervised process with its own heap, so a backend that crashes, hangs or floods its output is a contained and anticipated failure, and cancellation, timeouts and portfolio invocation follow from the process model instead of per-backend signal handling. Nothing in Section 4 tests this and nothing needs to; it is a property of the runtime.

For proof search the hypothesis of Section 1 is testable, and the criterion is narrower than whether a calculus is parallelisable. The tableau does admit the split of Section 3, and the pool exploits the local half as intended. The outcome turns on whether the global half scales with it, and in SHOT it does not. The agent is one process holding one search at a time, and each additional worker enlarges the problem that process must solve and shortens the time it has. In Bonacina’s terms [10] this is coarse-grained parallelism at the level of the search with a single arbitration point, and the arbitration point is the bottleneck.

The deficiency lies in that component and not in rigidity. Nothing about a rigid variable requires the simultaneous constraint to be solved by one process enumerating combinations. Three changes follow from Table 2. A search should have a deadline proportional to the remaining budget instead of the deadline of the whole proof, since only 12.4 % of searches find

a closure and an abandoned search costs its whole slot. The staleness check the agent already computes should extend to the ordinary path, so a search invalidated by evidence that arrived during it re-dispatches. Several searches over disjoint sets of branches are admissible under the constraint structure, and a subset with no solutions kills the whole dispatch. Beyond these, incremental closure in the sense of Giese [18] maintains the constraint as branches arrive instead of re-solving it, and the pre-unification inside a search is itself a search that could run concurrently. Moving equality into the constraint, as in Section 3.4, would replace demodulation’s role in closure by simultaneous rigid E -unification; the practical forms are an incomplete but terminating procedure [15] or a bounded encoding solvable by SAT [3]. At higher order this compounds with the undecidability of unification itself, and we note the direction without having assessed it.

The constant factor on the inner loop is real and should be properly quantified by future work. BeamAsm, the BEAM’s just-in-time compiler, removes interpreter dispatch but performs little optimisation across instructions,⁶ so the loop-invariant and register-allocation gains available to an ahead-of-time compiler are largely absent. Sharing terms by identifier and caching derived facts in the shared tables narrows the gap at the level of the data structure.

Native implemented functions are the obvious escape and are more expensive than they look. A native function runs as an extension of the emulator, outside the environment the runtime otherwise guarantees: a crash in it takes down the whole virtual machine, an error in it can leave the emulator’s internal state inconsistent so that a failure surfaces later and elsewhere, and one that runs long before returning degrades the scheduler it occupies.⁷ The documented remedies are to chunk the work and reschedule, to use a dirty scheduler at the cost of a context switch per call, or to hand the work to a separately managed thread. The kernels worth moving in SHOT are pre-unification enumeration, order comparison and term traversal, which are the operations that run long and fail, and so the ones for which preemption and failure isolation were the reason to be on this runtime.

The criterion this evidence suggests is that the step deciding the verdict must scale with the work feeding it. Instance-based methods, which replace the global substitution by propositionally decidable tests over ground instances, satisfy it by construction. This invites future work to lean more heavily into that paradigm, in the legacy of Satallax [12].

6 Integration and Teaching

The prover-access layer makes the same runtime a place to run other provers. Each invocation is supervised, so a portfolio over several backends with cancellation once one answers is a few lines of code, and the same mechanism supports cross-running a benchmark through several systems for soundness checking, which is how the agreement in Section 4.4 was established. A further consumer is a Model Context Protocol⁸ server, AtpMcp, which exposes the backends as tools to language-model agents.

Teaching motivated much of this work. The packages support an interactive environment built on Livebook,⁹ supported by KinoAtpClient. Branch state is immutable data in a shared ETS table with copy-on-lookup semantics, and is inspectable without instrumentation, so a notebook can drive and observe proof search step by step. A problem is entered in TPTP syntax,

⁶<https://www.erlang.org/doc/apps/erts/beamasm.html>

⁷https://www.erlang.org/doc/apps/erts/erl_nif.html

⁸<https://modelcontextprotocol.io/specification/2025-11-25>

⁹<https://livebook.dev>

dispatched either to SHOT or to an external backend, and examined in the same document. The teaching material and the research code are essentially the same code.

7 Related Work

Earlier free-variable tableau provers established the sequential lineage SHOT extends into a concurrent one: HARP [27] and the many-valued 3TAP [4] are first-order systems from that tradition, and Zenon [11] produces tableau-derived proofs checkable by a host proof assistant; none targets higher-order logic or runs its search concurrently.

Goéland [13] is the closest first-order relative. It runs a concurrent search in which processes correspond to individual branches and exchange substitutions over channels, since branch closure may instantiate variables shared across branches. SHOT makes the opposite choice at the same design point. Branches stay values and arbitration belongs to one component. Section 4.3 measures what that costs. HOT [22] is the direct predecessor of SHOT as a concurrent prover for higher-order tableaux, and the closest system to it in scope. Its concurrency, however, serves a blackboard of cooperating expert processes rather than branch expansion itself, so the comparison in Section 4.3 between a worker pool and a single worker has no counterpart there. OANTS [8] organised proof search around a blackboard of suggestion agents, and Leo-III [33] carries an agent framework whose fine-grained mode was not found competitive. Arbitration cost is the common thread; SHOT arbitrates only closure. Bonacina’s taxonomy [10] classifies such strategies by granularity and by whether parallelism acts on inference, search or both; the configuration measured here is parallel search with a shared frontier and one arbitration point.

Simultaneous rigid E -unification was introduced for extension procedures of this kind [17] and shown undecidable [14]. The relevant responses are complete proof procedures built on incomplete but terminating rigid unification [15] and bounded variants that are NP-complete and SAT-encodable [3], as used in a branch of Princess [29]. Incremental closure [18] is the technique Section 3.4 implements at higher order.

8 Conclusion

What exists is a layer of BEAM packages covering terms and proofs, THF parsing, higher-order unification, term ordering and external-prover access; a tableau prover on top of them whose rules are binding-free and whose closure is one constraint; and a teaching environment sharing the same code. The runtime pays for orchestrating provers and for interactive work. In this implementation it does not pay for proof search. The pool multiplies inference by a factor of three and coverage does not follow, because the component deciding the verdict does not scale with the work feeding it. That component is where the next work goes, together with separating parsing from elaboration so the corpus figures measure the prover.

Use of AI assistants

Large-language-model assistants were used during development of the software described here, for code review, refactoring and test generation, and during preparation of this text. All design decisions, measurements and claims are the authors’ and the authors take full responsibility for them.

References

- [1] P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A theorem-proving system for classical type theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.
- [2] J. Armstrong. *Making reliable distributed systems in the presence of software errors*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 2003.
- [3] P. Backeman and P. Rümmer. Theorem proving with bounded rigid E-unification. In A. P. Felty and A. Middeldorp, editors, *Automated Deduction – CADE-25*, volume 9195 of *Lecture Notes in Computer Science*, pages 572–587. Springer, 2015.
- [4] B. Beckert, R. Hähnle, P. Oel, and M. Sulzmann. The tableau-based theorem prover 3TAP version 4.0. In M. A. McRobbie and J. K. Slaney, editors, *Automated Deduction – CADE-13*, volume 1104 of *Lecture Notes in Computer Science*, pages 303–307. Springer, 1996.
- [5] B. Beckert and J. Posegga. leanTAP: Lean tableau-based deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.
- [6] C. Benzmüller and C. E. Brown. A structured set of higher-order problems. In J. Hurd and T. F. Melham, editors, *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005, Proceedings*, volume 3603 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 2005.
- [7] C. Benzmüller, F. Rabe, and G. Sutcliffe. THF0 - the core of the TPTP language for higher-order logic. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 491–506. Springer, 2008.
- [8] C. Benzmüller and V. Sorge. OANTS – combining interactive and automated theorem proving. In M. Kerber and M. Kohlhase, editors, *Symbolic Computation and Automated Reasoning*, pages 81–97. A.K.Peters, 2001.
- [9] J. C. Blanchette, C. Kaliszyk, L. C. Paulson, and J. Urban. Hammering towards QED. *Journal of Formalized Reasoning*, 9(1):101–148, 2016.
- [10] M. P. Bonacina. A taxonomy of parallel strategies for deduction. *Annals of Mathematics and Artificial Intelligence*, 29(1–4):223–257, 2000.
- [11] R. Bonichon, D. Delahaye, and D. Doligez. Zenon: An extensible automated theorem prover producing checkable proofs. In N. Dershowitz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning – LPAR 2007*, volume 4790 of *Lecture Notes in Computer Science*, pages 151–165. Springer, 2007.
- [12] C. E. Brown. Satallax: An automatic higher-order prover. In B. Gramlich, D. Miller, and U. Sattler, editors, *Automated Reasoning – IJCAR 2012*, volume 7364 of *Lecture Notes in Computer Science*, pages 111–117. Springer, 2012.
- [13] J. Cailler, J. Rosain, D. Delahaye, S. Robillard, and H. L. Bouziane. Goéland: A concurrent tableau-based theorem prover (system description). In J. Blanchette, L. Kovács, and D. Pattinson, editors, *Automated Reasoning – IJCAR 2022*, volume 13385 of *Lecture Notes in Computer Science*, pages 359–368. Springer, 2022.
- [14] A. Degtyarev and A. Voronkov. The undecidability of simultaneous rigid E-unification. *Theoretical Computer Science*, 166(1–2):291–300, 1996.
- [15] A. Degtyarev and A. Voronkov. What you always wanted to know about rigid E-unification. *Journal of Automated Reasoning*, 20(1):47–80, 1998.
- [16] G. Dowek. Higher-order unification and matching. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 16, pages 1009–1062. Elsevier and MIT Press, 2001.
- [17] J. H. Gallier, S. Raatz, and W. Snyder. Theorem proving using rigid E-unification: Equational matings. In *Proceedings of the Second Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 338–346. IEEE Computer Society, 1987.

- [18] M. Giese. Incremental closure of free variable tableaux. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning – IJCAR 2001*, volume 2083 of *Lecture Notes in Computer Science*, pages 545–560. Springer, 2001.
- [19] G. P. Huet. A unification algorithm for typed λ -calculus. *Theoretical Computer Science*, 1(1):27–57, 1975.
- [20] C. Kaliszyk, G. Sutcliffe, and F. Rabe. TH1: the TPTP typed higher-order form with rank-1 polymorphism. In P. Fontaine, S. Schulz, and J. Urban, editors, *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning co-located with International Joint Conference on Automated Reasoning (IJCAR 2016), Coimbra, Portugal, July 2nd, 2016*, volume 1635 of *CEUR Workshop Proceedings*, pages 41–55. CEUR-WS.org, 2016.
- [21] M. Kohlhase. Higher-order tableaux. In P. Baumgartner, R. Hähnle, and J. Posegga, editors, *Theorem Proving with Analytic Tableaux and Related Methods, 4th International Workshop, TABLEAUX ’95, Schloß Rheinfels, St. Goar, Germany, May 7-10, 1995, Proceedings*, volume 918 of *Lecture Notes in Computer Science*, pages 294–309. Springer, 1995.
- [22] K. Konrad. HOT: A concurrent automated theorem prover based on higher-order tableaux. In J. Grundy and M. Newey, editors, *Theorem Proving in Higher Order Logics – TPHOLs 1998*, volume 1479 of *Lecture Notes in Computer Science*, pages 245–261. Springer, 1998.
- [23] L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In N. Sharygina and H. Veith, editors, *Computer Aided Verification – CAV 2013*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013.
- [24] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.
- [25] J. Niederhauser and A. Middeldorp. The computability path order for beta-eta-normal higher-order rewriting. In C. W. Barrett and U. Waldmann, editors, *Automated Deduction - CADE 30 - 30th International Conference on Automated Deduction, Stuttgart, Germany, July 28-31, 2025, Proceedings*, volume 15943 of *Lecture Notes in Computer Science*, pages 207–225. Springer, 2025.
- [26] T. Nipkow, L. C. Paulson, and M. Wenzel. Isabelle/HOL – A Proof Assistant for Higher-Order Logic, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.
- [27] F. Oppacher and E. Suen. HARP: A tableau-based theorem prover. *Journal of Automated Reasoning*, 4:69–100, 1988.
- [28] J. Otten and W. Bibel. leanCoP: Lean connection-based theorem proving. *Journal of Symbolic Computation*, 36(1–2):139–161, 2003.
- [29] P. Rümmer. A constraint sequent calculus for first-order logic with linear integer arithmetic. In I. Cervesato, H. Veith, and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning – LPAR 2008*, volume 5330 of *Lecture Notes in Computer Science*, pages 274–289. Springer, 2008.
- [30] S. Schulz, S. Cruanes, and P. Vukmirović. Faster, higher, stronger: E 2.3. In P. Fontaine, editor, *Automated Deduction – CADE-27*, volume 11716 of *Lecture Notes in Computer Science*, pages 495–507. Springer, 2019.
- [31] J. Schuster. Extensionality and instance-based methods in tableau-based higher-order automated theorem proving. Master’s thesis, University of Bamberg, Aug. 2026.
- [32] J. Schuster, D. Fuenmayor, and C. Benzmüller. Elixir meets TPTP: Bringing automated reasoning to the BEAM ecosystem. In M. Hajdu, C. Nalon, and M. Suda, editors, *Proceedings of the 10th Workshop on Practical Aspects of Automated Reasoning (PAAR 2026)*, volume 4241 of *CEUR Workshop Proceedings*, Lisbon, Portugal, July 2026.
- [33] A. Steen and C. Benzmüller. The higher-order prover Leo-III. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *Automated Reasoning – IJCAR 2018*, volume 10900 of *Lecture Notes in Computer Science*, pages 108–116. Springer, 2018.
- [34] A. Stump, G. Sutcliffe, and C. Tinelli. StarExec: A cross-community infrastructure for logic solving. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Automated Reasoning – IJCAR*

- 2014, volume 8562 of *Lecture Notes in Computer Science*, pages 367–373. Springer, 2014.
- [35] G. Sutcliffe. The SZS Ontologies for Automated Reasoning Software. In G. Sutcliffe, P. Rudnicki, R. Schmidt, B. Konev, and S. Schulz, editors, *Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics*, number 418 in CEUR Workshop Proceedings, pages 38–49, 2008.
 - [36] G. Sutcliffe. The TPTP problem library and associated infrastructure – from CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.
 - [37] G. Sutcliffe. Stepping stones in the TPTP world. In C. Benzmüller, M. J. H. Heule, and R. A. Schmidt, editors, *Automated Reasoning – IJCAR 2024*, volume 14739 of *Lecture Notes in Computer Science*, pages 30–50. Springer, 2024.
 - [38] P. Vukmirović, A. Bentkamp, J. Blanchette, S. Cruanes, V. Nummelin, and S. Tourret. Making higher-order superposition work. In A. Platzer and G. Sutcliffe, editors, *Automated Deduction – CADE-28*, volume 12699 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2021.