LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

CHAIR OF THEORETICAL COMPUTER SCIENCE AND THEOREM PROVING

Benchmarking and Analysis of Automated Theorem Provers

Shivam Sambyal

Bachelor's Thesis
Computer Science plus Mathematics

Supervisor: Prof. Jasmin Blanchette

Advisor: Lydia Kondylidou

Submission Date: July 11, 2025

Disclaimer

I confirm that this thesis type is my own work and I have documented all sources and material used.

Munich, July 11, 2025 Shivam Sambyal

Acknowledgments

I wish to extend my sincere gratitude to my advisor, Lydia Kondylidou, for her unwavering guidance, insightful feedback, and encouragement throughout the course of this thesis. Additionally, I am profoundly grateful to my supervisor, Professor Jasmin Blanchette, for affording me the opportunity to engage with this challenging and intellectually stimulating subject in the field of automated proving and reasoning.

Abstract

This thesis provides a comparative analysis of the automated reasoning capabilities of two state-of-the-art theorem provers: the SMT solver cvc5 and the saturation-based prover Vampire. The primary focus is on assessing their performance on selected higher-order problems sourced from the TPTP library. Particular emphasis is placed on instances in which cvc5 fails to produce a result, whereas Vampire succeeds by employing classical inference techniques such as skolemization, resolution, and superposition. The study comprises benchmark-driven experiments alongside theoretical proof reconstructions aimed at identifying fundamental differences in reasoning strategies. The findings offer valuable insights into the limitations of cvc5 within symbolic domains and suggest potential avenues for architectural enhancements.

Keywords:Automated Theorem Proving, SMT Solvers, cvc5, Vampire, TPTP, Benchmarking, First-Order Logic, Higher-Order Logic, Symbolic Reasoning, Skolemization, Resolution, Superposition, Conjunctive Normal Form (CNF), Proof Reconstruction, Saturation-Based Provers

Contents

1 Introduction

1.1 Motivation

Automated Theorem Proving (ATP) occupies a vital position in contemporary formal verification, logic, and artificial intelligence. These systems are engineered to establish the validity or invalidity of logical formulas autonomously, utilizing a combination of deductive methodologies such as resolution, superposition, and satisfiability modulo theories (SMT) (7). Over the past several decades, ATPs have undergone significant advancements, enabling essential applications in software verification, the discovery of mathematical theorems, and knowledge reasoning systems. Nonetheless, despite these progressions, certain limitations persist, particularly in the context of quantifier reasoning and the management of intricate structural complexities (5).

One of the principal motivations for this thesis arises from a recurring challenge encountered during practical engagements with higher-order problems derived from the TPTP (Thousands of Problems for Theorem Provers) library (10). Although the solver cvc5 is a potent and highly modular SMT solver (1), it consistently failed to resolve a significant subset of these problems. Such failures were characterized by extended timeouts or non-terminating behaviors, which resulted in an outcome marked as "unknown."

This inquiry prompted a critical examination: what specific limitations of cvc5 contribute to its failure on certain problems that other provers, such as Vampire, are capable of successfully resolving? The thesis originated from this question, with the objective of benchmarking both systems against each other through the use of carefully selected higher-order logic problems. Unlike cvc5, Vampire is a saturation-based automated theorem prover (ATP) that specializes in clause-level manipulation and resolution calculus (12). It has demonstrated consistent top-tier performance in international ATP competitions such as the CASC (CADE ATP System Competition), particularly in the categories of first-order and higher oder logics (10).

Consequently, the focus of this thesis is to analyze the performance disparity between cvc5 and Vampire, particularly on problems involving deeply nested quantifiers, Skolemization, superposition, unification or clausification. While both provers are formally sound, they employ markedly different proof strategies: cvc5 utilizes a hybrid SMT architecture that is most effective for quantifier-free or theory-rich problems, whereas Vampire is designed to optimize saturation strategies across extensive clause sets.

To investigate these differences, we conducted an extensive benchmarking and theoretical analysis involving hundreds of problems sourced from twenty-three folders within the TPTP library. Each problem was initially evaluated using cvc5. If the result was a 'timeout' or 'unknown,' the same problem was subsequently re-evaluated with Vampire. This filtering process resulted in a curated subset of problems solvable by Vampire but not by cvc5, thereby providing valuable insights into the architectural and logical distinctions between the two systems.

This work not only furnishes a diagnostic benchmark dataset for future research endeavors but also establishes a foundation for enhancing existing solvers or developing hybrid architectures. It is our aspiration that this thesis serves as both a reference and a strategic roadmap for researchers seeking to narrow the performance gap in advanced higher-order automated reasoning systems.

1.2 Overview

This thesis is organized to guide the reader from the fundamental principles of automated reasoning to a comprehensive comparative analysis of two prominent theorem provers—cvc5 and Vampire—focusing specifically on their performance with higher-order logic problems sourced from the TPTP library.

Following the presentation of the motivation, scope, and objectives in the introduction, the second chapter establishes the necessary theoretical foundation. It introduces essential concepts in logic, the historical development of automated theorem proving, and the core mechanisms employed by both SMT solvers and general automated theorem provers. Additionally, this chapter provides information about the problem library utilized throughout the thesis and an introduction to benchmarking in automated theorem proving.

The third chapter details the main theoretical contributions of the thesis. It begins with a broad conceptual overview of how formal proofs are transformed within automated reasoning systems, including techniques such as negation, conversion to negation normal form (NNF), Skolemization, transformation into conjunctive normal form (CNF), and resolution via unification. This is followed by a comparative theoretical analysis of cvc5 and Vampire, emphasizing their respective proof strategies, capabilities, and limitations. The discussion establishes a logical foundation for interpreting the behaviors of the solvers observed in the subsequent experimental sections.

Empirical benchmarking is addressed in the fourth chapter, which describes the testing methodology employed to evaluate both solvers across a curated subset of TPTP problems. It details the experimental design, classification criteria, timeout management, and result collection procedures. The chapter concludes with a presentation of the benchmarking outcomes, illustrating clear patterns in the instances where cvc5 encounters difficulties and Vampire successfully resolves the problems.

Chapter five provides a detailed analysis of these results. It examines recurring failure patterns observed in cvc5, contrasting them with Vampire's saturation-based reasoning and symbolic rewriting capabilities, which frequently enable it to resolve many of the same problems successfully. The analysis also identifies particular problem patterns, offering an in-depth examination of how Vampire constructs proofs in scenarios where cvc5 is unable to proceed.

The final chapter concludes the thesis by summarizing the principal insights derived from both the theoretical and experimental investigations. It reflects on the architectural and strategic disparities between SMT and ATP paradigms and suggests potential avenues for enhancement and future research aimed at mitigating the limitations observed in cvc5.

2 Background

Automated reasoning constitutes a fundamental discipline within theoretical computer science and artificial intelligence, dedicated to the mechanization of logical inference processes. Its primary aim is to develop algorithmic systems capable of autonomously establishing the validity or invalidity of logical statements through formal deduction techniques. The significance of automated reasoning has increased in response to the growing complexity and scope of contemporary mathematical frameworks, hardware and software systems, and knowledge-based applications. In many of these fields, ensuring correctness is not merely advantageous but essential, particularly within safety-critical domains such as aerospace, cryptography, and medical systems (7). Manual verification in

these contexts has become infeasible, thereby driving the advancement and widespread adoption of automated reasoning tools (9).

Among these computational tools, two principal categories have emerged as particularly influential: Automated Theorem Provers (ATPs) and Satisfiability Modulo Theories (SMT) solvers. Both address issues related to logical validity and satisfiability; however, they employ distinctly different strategies, architectures, and underlying assumptions regarding their applications. Automated Theorem Provers predominantly rely on symbolic inference methods and are especially effective for problems characterized by substantial logical complexity and intricate structural reasoning (12). Conversely, SMT solvers integrate propositional logic with specialized theory solvers designed for particular domains, such as arithmetic, arrays, or data types (2), thereby rendering them particularly suitable for problems involving concrete constraints and quantifier-free logic.

2.1 Automated Reasoning and Theorem Proving

Automated reasoning fundamentally pertains to the development of systems capable of deriving conclusions from a specified set of premises. Within this expansive domain, Automated Theorem Proving (ATP) specifically denotes the class of algorithms and systems engineered to validate or disprove mathematical theorems and logical assertions through formal methodologies. ATP systems function by employing well-defined inference rules applied to axioms and hypotheses to infer new truths.

A vital distinction within ATP concerns the type of logic utilized. Propositional logic, while decidable and relatively efficient to resolve, possesses limited expressive capacity. First-order logic (FOL) introduces quantifiers and predicates, rendering it semi-decidable—solutions can be obtained if a proof exists, although the process may not terminate otherwise. Higher-order logic (HOL) further enhances expressiveness by permitting quantification over predicates and functions, yet this often entails undecidability. These differences significantly influence the capabilities and design considerations of ATP systems (9).

Notable ATP tools such as Vampire, E, and Prover9 employ saturation-based techniques—including resolution, paramodulation, and unification—to derive contradictions and thereby establish the validity of theorems [14]. Their primary advantage lies in addressing logic-intensive, theory-light problems characterized by complex symbolic structures. These systems have become indispensable in fields such as formal mathematics, software verification, and symbolic artificial intelligence. As discussed by Voronkov (12), Vampire's clause saturation and selection strategies are optimized for scalability across thousands of clauses, facilitating automated reasoning even in contexts that mimic higher-order constructs when suitably encoded.

Furthermore, ATP systems often integrate with proof assistants such as Coq, Isabelle, and Lean (5), enabling users to combine automated proof search with human-guided verification. This synergy provides a balanced approach, merging automation with transparency and control over the proof process.

2.2 SMT Solvers

Satisfiability Modulo Theories (SMT) solvers, including cvc5, Z3, and Yices, extend traditional SAT solvers by integrating specialized decision procedures for a broad spectrum of background theories, such as arithmetic, bit-vectors, arrays, datatypes, and uninterpreted functions. Most SMT

solvers are constructed upon the DPLL(T) architecture, which synergistically combines Boolean satisfiability solving (SAT) with modular theory solvers in a highly optimized and cooperative fashion (2).

In particular, cvc5 stands as one of the most advanced and modular SMT solvers available. Developed as the successor to cvc4, it introduces sophisticated capabilities and enhanced support for quantified reasoning, as evidenced in the work by Barbosa et al. (1). cvc5 encompasses a wide range of theories and employs efficient quantifier-handling techniques, including E-matching, Model-Based Quantifier Instantiation, and Counterexample-Guided Quantifier Instantiation (CEGQI). These strategies enable cvc5 to reason about quantified formulas without necessitating full clause-level saturation, a trait more characteristic of Automated Theorem Provers (ATPs).

Notwithstanding their robust capabilities, SMT solvers are predominantly tailored towards theory-specific reasoning, rendering them less suitable for purely symbolic logic problems that require exhaustive Skolemization, unification, or resolution-based inference (1). Their architecture is optimized for performance within particular theory domains, notably fragments such as EUF (Equality with Uninterpreted Functions), bit-vector logic, and linear integer arithmetic.

The significance of SMT solvers in the domains of software and hardware verification is substantial. They serve as fundamental components in bounded model checkers, software verifiers, and synthesis tools, providing powerful theory solvers for assessing the satisfiability of path conditions, invariants, or symbolic executions (2). In these applications, SMT solvers are valued primarily for their deep reasoning within theories rather than for general logical deduction, thereby illustrating their complementary relationship with ATPs.

2.3 Comparison SMT Solvers and in general ATPs

Although both Satisfiability Modulo Theories (SMT) solvers and classical Automated Theorem Provers (ATPs) endeavor to establish the validity or satisfiability of logical formulas, they exhibit substantial differences in methodology, strengths, and domains of application.

SMT solvers, such as cvc5, Z3 and Yices, constitute extensions of propositional satisfiability (SAT) solvers incorporating decision procedures for specific logical theories, including arithmetic, bit vectors, arrays, uninterpreted functions, and data types (3). The predominant architecture employed by SMT solvers is the DPLL(T) framework, which integrates theory propagation with Boolean SAT solving. This configuration enables SMT solvers to effectively address problems originating from fields such as program verification, hardware design, and constraint satisfaction, particularly when the logical formulas are predominantly quantifier-free and involve heavy theoretical content (1).

Conversely, traditional ATPs such as Vampire, E, and Prover9 are grounded in first-order logic reasoning and prioritize completeness and clause-level inference techniques over explicit treatment of theories (12). These provers typically transform input formulas into conjunctive normal form (CNF), employing saturation-based inference methods—such as superposition, resolution, or paramodulation—to derive contradictions (refutations) necessary to establish theorems (7). Automated theorem provers demonstrate particular efficacy in problems characterized by complex chains of implications, unification processes, or quantifier manipulation, which are prevalent in pure mathematics and logic puzzles. For example, consider a problem from the TPTP library involving nested existential and universal quantifiers over functions. Vampire can systematically saturate the clauses and apply unification rules recursively to derive a contradiction, thereby resolving the

problem. An SMT solver like cvc5, however, may time out or return an ”unknown” status due to its lack of aggressive saturation mechanisms and general-purpose quantifier instantiation strategies inherent in theory-specific decision procedures (1).

The differences between SMT solvers and ATPs become even more pronounced in the context of higher-order logic. Most SMT solvers are not designed to handle full higher-order reasoning; they primarily rely on heuristics for quantifier instantiation and lack the capacity for deep clause-level reasoning necessary to handle higher-order entities effectively (1). In contrast, although not all ATPs are inherently equipped for higher-order reasoning, some—such as extensions of Vampire—can address such problems through reduction to first-order logic or via specialized techniques and extensions (5).

Practically, these two classes of solvers are complementary. SMT solvers often outperform ATPs in code verification tasks due to their theory-aware optimizations. Conversely, ATPs are indispensable in domains involving mathematics, logic, and complex reasoning tasks that require comprehensive logical inference capabilities (4). The objective of this thesis is to examine the limitations of each approach—specifically, to identify scenarios where cvc5-type SMT solvers are insufficient, and Vampire-type ATPs demonstrate superiority in handling higher-order reasoning.

2.4 Benchmarking in Automated Theorem Proving

Benchmarking occupies a vital position in shaping the trajectory of automated theorem proving (ATP) systems. It provides researchers with a rigorous, empirical framework for measuring, comparing, and enhancing the performance of various provers. Competitions such as the CADE ATP System Competition (CASC), which utilize the TPTP problem set, have been instrumental in establishing standardized testing protocols and fostering equitable, meaningful comparisons within the field (10).

More recently, benchmarking practices have expanded to encompass emerging methodologies, including the application of machine learning techniques to guide proof searches. Approaches such as reinforcement learning and statistical models have begun to enable provers to navigate complex problems with greater sophistication (8). These developments demonstrate that benchmarking is not solely a metric of system performance but also a catalyst for innovation, drawing attention to the most challenging, unresolved problems.

This thesis is centered on an in-depth benchmarking analysis of automated theorem provers (ATPs), with particular emphasis on cvc5 and Vampire. The study critically relies on the Thousands of Problems for Theorem Provers (TPTP) library as the foundational dataset. The TPTP repository is a reputable resource, offering an extensive collection of test problems spanning various domains and levels of logical complexity. Its well-structured format and widespread adoption render it an ideal benchmark for evaluating and comparing the capabilities of different theorem proving systems (11).

The selection of cvc5 and Vampire was deliberate. CVC5 is a contemporary SMT (Satisfiability Modulo Theories) solver designed to address complex theories such as bit-vectors, arrays, and arithmetic with robustness (1). Conversely, Vampire is a well-established logic solver renowned for its consistent and reliable performance across numerous competitive evaluations. By conducting a comparative analysis of these two systems, the intent is to investigate their respective strengths and limitations, as well as to identify potential areas of overlap and synergy in their problem-solving capabilities.

3 Main Contribution

This chapter delineates the principal contributions of the present thesis, situated at the nexus of empirical benchmarking and formal proof-theoretical analysis. The primary objective was to assess the performance of two automated theorem provers—cvc5 and Vampire—on a subset of higher-order logic problems derived from the TPTP library and check the limitations of cvc5 in comparison to Vampire. The work endeavors to furnish not only a comparative account of the solvers' behavior but also comprehensive reconstructions and elucidations of the reasoning processes involved. The contribution is multifaceted, encompassing both implementation-level scripting and theoretical investigations in logic, which are elaborated upon in the subsequent sections.

3.1 Project Structure and Setup

The project was conducted within a dual-track framework, comprising both empirical and theoretical components. Empirically, a benchmarking pipeline was established to execute two theorem provers, cvc5 and Vampire, on hundreds of.p problem files sourced from the TPTP library. The environment employed was a Linux-based system, with both provers compiled from source code. Automated scripts written and executed on problem files, enforced timeouts, and enabled classification of results. Each solver was allocated identical resource limits to ensure a fair comparative assessment, and all outcomes were systematically recorded in structured spreadsheets for subsequent analysis.

On the theoretical front, select benchmark problems were scrutinized through manual reconstruction of the proof processes utilized by Vampire. This process entailed a detailed examination of the original formula contained within each problem file, followed by the application of logical transformations—namely negation, conversion to Negation Normal Form (NNF), Skolemization, translation into Conjunctive Normal Form (CNF), and clause-level resolution. The aim was to identify the precise logical inferences that facilitated Vampire's derivation of a contradiction or the establishment of a proof. These reconstructions not only served to verify Vampire's success but also provided insights into potential improvement areas within cvc5's reasoning strategies.

3.2 Comparative Analysis of cvc5 and Vampire

This thesis compares two widely used and highly regarded automated reasoning systems—cvc5 and Vampire—with a focus on how their differing architectures and inference mechanisms influence their behavior on logic-heavy benchmark problems from the TPTP library. While both provers are advanced and continually evolving, they adopt fundamentally different reasoning philosophies that make them suited to different types of problem domains.

cvc5 belongs to the class of SMT (Satisfiability Modulo Theories) solvers, and is designed to integrate SAT solving with modular decision procedures for background theories such as arithmetic, arrays, and bit-vectors. Its architecture allows it to combine propositional reasoning with efficient theory propagation and conflict resolution (1). This makes cvc5 exceptionally powerful in solving problems that involve constraints over interpreted domains and theory-driven logic, particularly in software verification and formal methods applications (2).

Vampire, on the other hand, is a saturation-based automated theorem prover operating primarily in first-order logic (12). Its proof search strategy relies on the exhaustive application of inference rules to a clause set, including resolution, skolemization, unification and superposition (12). These

methods allow Vampire to navigate complex symbolic relationships and deduce conclusions through saturation and refutation, particularly in domains that involve rich logical structure.

One notable difference lies in the treatment of quantifiers and symbolic formulas. Vampire converts input formulas into clause normal form (CNF) through steps such as Skolemization and negation normal form (NNF) conversion (9). It then applies symbolic inference rules that allow for general-purpose logical reasoning. Superposition, in particular, plays a key role in Vampire's success in handling equalities by enabling the replacement of terms in clauses with their equals—a process not typically applied in SMT solving.

In contrast, cvc5 applies heuristic quantifier handling techniques, including E-matching and Counterexample-Guided Quantifier Instantiation (CEGQI) (1). These methods work especially well when theory reasoning can guide the search space, but they do not rely on Skolemization, superposition or unification. Its architecture favors model construction over refutation, which can sometimes make symbolic exploration of complex quantified formulas less direct than in saturation-based approaches.

Despite these distinctions, it is important to emphasize that both systems are highly capable, and their observed differences in performance across the TPTP benchmarks primarily reflect their intended design goals. Vampire is optimized for logical deduction over purely symbolic formulas, whereas cvc5 is engineered to efficiently handle decidable fragments of logic enriched by background theories. In practice, these paradigms are not mutually exclusive but complementary (9).

The empirical results of this thesis suggest that certain symbolic reasoning features, such as clause-level resolution, controlled Skolemization, unification or selective superposition, may be useful additions to future SMT solver frameworks like cvc5—not as replacements for its strengths, but as potential extensions that could increase its generality and performance on complex logical problems. At the same time, the insights from SMT frameworks—especially regarding efficient theory integration and conflict-driven clause learning—may inform future developments in saturation-based provers.

This analysis supports a broader view of automated theorem proving as an evolving and collaborative field. Rather than promoting one paradigm over the other, this thesis highlights the value of cross-pollination—where techniques traditionally associated with ATPs might enhance SMT solvers, and vice versa. The comparison between cvc5 and Vampire thus serves not to rank these tools, but to explore how their complementary strategies can collectively advance the capabilities of automated reasoning systems.

3.3 Proof Transformation Process

Prior to examining specific examples, it is essential to comprehend the structure of the proof transformation process employed to reconstruct how Vampire addressed certain problems. The process adheres to a classical first-order reasoning approach, adapted to correspond with Vampire's internal logical pipeline.

Beginning with the original formula extracted from the TPTP problem file, the formula is first negated, thereby converting the conjecture into a statement whose refutation would substantiate the original assertion. The negated formula is subsequently transformed into Negation Normal Form (NNF), ensuring that all negations are directly applied to atomic formulas. The next stage involves Skolemization, which eliminates existential quantifiers through the introduction of Skolem

functions, thereby simplifying the quantifier structure of the formula. Following Skolemization, the formula is converted into Conjunctive Normal Form (CNF), yielding a set of disjunctive clauses amenable to individual manipulation.

Ultimately, this set of clauses is submitted to the resolution process. Unification is employed to match terms across clauses, and the resolution rule is applied to derive new clauses. In many instances, techniques such as superposition and paramodulation are also integral, particularly when equality reasoning is central. This reconstruction not only enhances our understanding of certain concepts present in Vampire but absent in cvc5, but also facilitates the combination of related problem files that Vampire can solve in a similar manner—tasks that cvc5 was unable to accomplish. Consequently, this approach may contribute to more targeted improvements in cvc5.

3.4 Benchmark Problem Examples

To complement the theoretical comparison and architectural discussion, this section provides specific examples of TPTP benchmark problems analyzed throughout the course of this thesis. These examples include instances solvable by cvc5 but not by Vampire, as well as sets of problems exhibiting similar characteristics. The purpose of presenting these examples is to illustrate how Vampire was able to generate complete proofs for certain problems, whereas cvc5 faced difficulties in making progress toward a resolution.

3.4.1 Example 1

Given Formula

$$\forall r, \exists g, \forall x(\exists y : r(x,y)) \Rightarrow r(x,g(x)) \Rightarrow$$
$$\forall s, (\forall a : s(a) \Rightarrow \exists t : a(t) \Rightarrow$$
$$\exists f, \forall a : s(a) \Rightarrow a(f(a))$$

This formula articulates a sequence of quantified logical dependencies involving both relational and functional constructs. The predicate $r(x,y)$ is a binary relation, while g is a function symbol introduced existentially, applied to the variable x. The initial segment of the implication asserts that, for all relations r, if for every x there exists a y such that $r(x,y)$ implies $r(x,g(x))$, then a subsequent implication ensues. In this subsequent part, $s(a)$ is a unary predicate, and $a(t)$ is a predicate or function application involving the variable t. The structure stipulates that, if for all a, $s(a)$ implies the existence of a t such that $a(t)$ holds, then there exists a function f such that, for all a, if $s(a)$ is true, then $a(f(a))$ is false. This formula intricately combines universal and existential quantification with implications and nested functions, rendering it a suitable test case for evaluating the symbolic reasoning capabilities of automated theorem provers.

Negation of the Entire Formula

In logic, a formula is in Negation Normal Form (NNF) if the negation operator ($\neg$) is applied only to atomic propositions, and the only allowed logical connectives are conjunction $\wedge$) and disjunction

($\vee$). This form is pivotal because it simplifies the logical structure, making it more amenable to automated theorem proving techniques(13).

To convert a formula into NNF, we systematically eliminate implications ($\rightarrow$) and biconditionals by expressing them in terms of $\neg$, $\wedge$, and $\vee$.

Subsequently, we apply De Morgan's Laws to push negations inward, ensuring they only affect atomic propositions. Double negations are also eliminated during this process.

To apply resolution-based theorem proving, we begin by negating the original formula. This is a standard approach in proof by contradiction, where we assume the negation of the statement we aim to prove and derive a contradiction.

Negating the original formula transforms it into:

$$\begin{aligned}&\neg(\forall r, \exists g, \forall x(\exists y : r(x,y)) \Rightarrow r(x,g(x)) \Rightarrow \\ &\quad \forall s, (\forall a : s(a) \Rightarrow \exists t : a(t) \Rightarrow \\ &\quad \exists f, \forall a : s(a) \Rightarrow a(f(a)))\end{aligned}$$

Simplification

De Morgan's Laws

De Morgan's Laws constitute essential principles within the fields of logic and set theory, delineating the relationships between conjunctions and disjunctions via the application of negation.

$$\begin{aligned}\neg(a \wedge b) &\equiv (\neg a) \vee (\neg b) \\ \neg(a \vee b) &\equiv (\neg a) \wedge (\neg b)\end{aligned}$$

These laws play a crucial role in the process of transforming logical formulas into Negation Normal Form (NNF), as they facilitate the systematic movement of negations inward.(6)

In propositional logic, the implication $a \rightarrow b$ can be transformed into logically equivalent expressions using other logical connectives. This transformation is useful for simplifying logical expressions and is particularly beneficial in automated theorem proving.

- **Disjunctive Form:** $a \rightarrow b \equiv \neg a \vee b$
- **Contrapositive Form:** $a \rightarrow b \equiv \neg b \rightarrow \neg a$

These equivalences are essential in logical reasoning and are extensively utilized across diverse applications, encompassing computer science and mathematics.

$$\begin{aligned}&(\forall r,\ \exists g,\ \forall x,\ (\exists y,\ r(x,y) \Rightarrow r(x,g(x))) \\ &\wedge (\exists s,\ \exists a,\ (s(a) \Rightarrow \exists t,\ (a(t)) \\ &\wedge (\forall f,\ \exists a,\ (s(a) \wedge \neg(a(f(a)))\end{aligned}$$

Assign variables: To facilitate comprehension of the formula, descriptive variable names have been assigned.

$$r = x_0, \quad g = x_1, \quad x = x_2, \quad y = x_3, \quad s = x_4$$
$$a = x_5, \quad t = x_6, \quad f = x_7$$

$$\forall x_0,\ \exists x_1,\ \forall x_2 :\ (\exists x_3 :\ x_0(x_2,x_3) \Rightarrow x_0(x_2,x_1(x_2))$$
$$\wedge\, (\exists x_4,\ \exists x_5 :\ x_4(x_5) \Rightarrow \exists x_6 :\ x_5(x_6)$$
$$\wedge\, (\forall x_7,\ \exists x_5 :\ x_4(x_5) \wedge \neg x_5(x_7(x_5))$$

Skolemization

Skolemization is the procedure employed to remove existential quantifiers from a logical formula through the introduction of Skolem functions, thereby transforming the formula into Skolem Normal Form (15). This transformation is of fundamental importance in the domain of automated theorem proving, especially within resolution-based methodologies, as it simplifies the quantifier structure of the original formula.

Skolemization Process

The process involves the following steps:

- **Skolem Function Introduction**: Replacing each existential quantifier with a Skolem function that depends upon the universally quantified variables preceding it.
- **Quantifier Removal**: Removing the existential quantifiers, as their impact is now represented by the Skolem functions.
- The resulting formula is logically equivalent in terms of satisfiability to the original, indicating that it is satisfiable if and only if the original formula is satisfiable.(15).

Application in Example 1

In our scenario, following the conversion of the formula to Negation Normal Form (NNF), Skolemization is employed to eliminate existential quantifiers. For example, an existential quantifier such as ∃x1 is replaced with a Skolem function f1(x1), where Xr is a universally quantified variable that precedes X0. Similarly, other existential quantifiers are substituted with suitable Skolem functions or constants, contingent upon their scope. This transformation streamlines the quantifier structure of the formula, rendering it appropriate for resolution-based proof procedures.

- $\exists x_1 : \quad (x_1 \rightarrow f_1(x_0) \quad (\textit{Since } x_1 \textit{ is dependent on } x_0)$
- $\exists x_4,\ \exists x_5,\ \exists x_6 : \quad x_4 = c_4,\ x_5 = c_5,\ x_6 = c_6$
- here $\exists x_5$ *in the last line depends on* $x_7 : x_5(x_7) \rightarrow f_2(x_7)$

So the formula is

$$\begin{aligned}\forall x_0 \forall x_2 \forall x_3 \, (&\neg x_0(x_2, x_3) \vee x_0(x_2, f_1(x_0)(x_2)) \\ &\wedge (\neg c_4(c_3) \vee c_5(c_6)) \\ &\wedge \forall x_7 \, (c_4(f_2(x_7)) \wedge \neg (f_2(x_7) \, x_7 \, f_2(x_7)))\end{aligned}$$

CNF Form

A formula is said to be in Conjunctive Normal Form (CNF) if it constitutes a conjunction (AND) of one or more clauses, each of which is a disjunction (OR) of literals. A literal is defined as either an atomic proposition or its negation (14).

CNF serves as a standardized representation that facilitates the application of automated theorem proving methods, such as the resolution technique. CNF Clauses: **Transformation Process** To convert a formula into CNF:

- Eliminate biconditionals and implications: Replace expressions like a→b with ¬avb.
- Move negations inward: Apply De Morgan's laws to push negations down to the level of literals.
- Standardize variables: Ensure that each quantifier uses a unique variable to avoid confusion.
- Skolemize: Eliminate existential quantifiers by introducing Skolem functions.
- Drop universal quantifiers: After Skolemization, all variables are universally quantified by default.
- Distribute disjunctions over conjunctions: Use distributive laws to achieve a conjunction of disjunctions.

Application in Example 1

In the case of example 1, following the implementation of the aforementioned transformations, the formula is transformed into a collection of clauses in Conjunctive Normal Form (CNF), with each clause representing a disjunction of literals. This standardized form is fundamental for the subsequent application of the resolution methodology.

1. $\neg x_0(x_2, x_3) \vee x_0(x_2, f_1(x_0)(x_2))$ (C1)
2. $\neg c_4(c_5) \vee c_5(c_6)$ (C2)
3. $c_4(f_2(x_7))$ (C3)
4. $\neg f_2(x_7)(x_7(f_2(x_7))$ (C4)

Detailed Unification and Resolution Process

In the domain of automated theorem proving, unification is defined as the procedure of determining a substitution that renders distinct logical expressions identical. This process constitutes a fundamental component of the resolution method, which is employed to demonstrate the unsatisfiability of a collection of clauses, consequently confirming the validity of the original formula.

Step-by-Step Unification and Resolution

Step 1: Unify Clause 2 and Clause 3

- Clause 2: ¬c4(c5) v c5(c6)
- Clause 3: c4(f2(x7))

Unification Process:

- Identify the complementary literals: c4(c5) in Clause 2 and c4(f2(x7)) in Clause 3.
- To unify these, we need a substitution that makes c5 equal to f2(X7).
- Substitution: c5 $\rightarrow$ f2(x7)

Resulting Clause (C5):

- Applying the substitution to Clause 2: ¬c4(f2(x7)) v f2(x7)(c6)
- Since c4(f2(x7)) is present in Clause 3, their negation and affirmation resolve, leaving: f2(x7)(c6)

Step 2: Unify Clause 4 and Clause C5

- Clause 4: ¬f2(x7)(x7(f2(x7)))
- Clause C5: f2(x7)(c6)

Unification Process:

- Identify the complementary literals: f2(x7)(x7(f2(x7))) in Clause 4 and f2(x7)(c6) in Clause C5.
- To unify these, we need a substitution that makes x7(f2(x7)) equal to c6. Substitution: c6 $\rightarrow$ x7(f2(x7))

Resulting Clause:

- Applying the substitution to Clause C5: f2(x7)(x7(f2(x7)))
- Now, Clause 4 contains the negation of this literal: ¬f2(x7)(x7(f2(x7))). Resolving these two clauses leads to an empty clause (contradiction).

Summary of Example 1 Proof

Through meticulous application of negation, Skolemization, transformation into conjunctive normal form (CNF), unification, and resolution, we have demonstrated that the Example 1 constitutes a theorem. This analysis was conducted in strict accordance with the proof methodology employed by the Vampire theorem prover, which successfully resolved the problem. In contrast, the cvc5 solver was unable to reach a solution, thereby highlighting a pronounced disparity in their respective capabilities.

The proof initiates with the original formula— a logically complex statement composed of nested quantifiers (both universal and existential), functional dependencies, and conditional implications. Such a high degree of logical complexity typifies challenging automated reasoning problems. The initial approach employed by cvc5 to establish a proof was unsuccessful, primarily due to its limited capacity to handle intricate logical transformations— including efficient Skolemization, advanced clause management, and sophisticated unification and resolution strategies.

Vampire's success can be largely ascribed to its systematic transformation of the problem utilizing traditional logical methods. It begins with negation, transforming the original problem into a refutation problem; this approach is foundational, permitting a proof by contradiction whereby the validity of the original statement is established by demonstrating that its negation leads to a contradiction.

Following negation, Skolemization is performed to eliminate existential quantifiers by substituting existentially quantified variables with Skolem functions. This step is vital, as it reduces the complexity of the formula by removing dependence on existential quantifiers. Vampire executes Skolemization efficiently, replacing each existentially quantified variable with a function dependent on the relevant universally quantified variables. This approach presents a challenge for cvc5, which lacks similarly sophisticated Skolemization techniques.

Subsequently, Vampire transforms the formula into conjunctive normal form (CNF). This multiphase process involves eliminating implications, distributing disjunctions over conjunctions, and standardizing variables. CNF is a fundamental representation in automated theorem proving because it represents formulas as sets of clauses—disjunctions of literals—facilitating the application of resolution.

With the formula expressed in CNF, Vampire proceeds with the resolution process. Thanks to its highly optimized resolution strategies— including clause selection, redundancy elimination, and saturation— Vampire performs this step effectively. For example 1, Vampire systematically applies resolution to the clause set, ultimately deriving an empty clause, which signifies a logical contradiction. This confirms the unsatisfiability of the negation and thereby establishes the validity of the original formula.

By way of contrast, cvc5 failed to arrive at this conclusion. This failure can be attributed to its limited resolution capabilities, inefficient management of clause interactions, and inadequate redundancy removal strategies. These limitations prevent cvc5 from effectively navigating the complex search space of dependencies inherent in the problem.

This proof exemplifies the advantages of Vampire as a saturation-based theorem prover capable of managing the logical intricacies of complex formulas through advanced proof strategies. Conversely, it also reveals fundamental deficiencies in cvc5 related to Skolemization, clause management, unification, and resolution.

Furthermore, this analysis underscores the critical importance of essential transformations—

such as Skolemization, CNF conversion, and resolution— in facilitating successful automated theorem proving. Vampire's adept handling of these transformations underpins its success in resolving problems like example 1, which remain beyond cvc5's capabilities.

The demonstrated success of Vampire in proving this theorem, juxtaposed with cvc5's failure, provides a significant benchmark for evaluating and enhancing the effectiveness of automated theorem proving systems. By identifying the specific conditions under which cvc5 falters while Vampire succeeds, we gain valuable insights into the strengths and limitations of current methods. This understanding can guide future developments, particularly in improving cvc5's performance through more sophisticated strategies in Skolemization, CNF conversion, unification, and resolution, thereby advancing its proficiency in handling complex logical problems.

3.4.2 Example 2

Initial Formula

$$\begin{aligned} &\forall a \forall b \left((\forall x \exists y \forall w : (a(x,y,w) \vee b(x,y,w))) \right. \\ &\Rightarrow (\exists f \forall x \forall w \left(a(x, f(x), w) \vee b(x, f(x), w) \right)) \\ &\Rightarrow \exists j \forall p : (\neg \forall z : p(z) \Rightarrow p(j(p))) \end{aligned}$$

This formula encodes a hierarchical structure of quantification involving predicates and functions within a sequence of logical implications. The predicates $a(x,y,w)$ and $b(x,y,w)$ are ternary relations over three variables, and the function symbols f and j are introduced within existential scopes. The formula begins with a universally quantified premise over a and b, asserting that for all x, there exists a y, such that for all w, the disjunction $a(x,y,w) \vee b(x,y,w)$ holds. This asserts that a certain relational property between these variables is perpetually satisfied.

The subsequent segment introduces a function f, stating that if such a property holds, then there exists a function f such that for all x and w, either $a(x, f(x), w)$ or $b(x, f(x), w)$ holds—effectively selecting a specific witness $f(x)$ for y that preserves the original property. Finally, the last implication asserts the existence of a function j such that, for every unary predicate p, if p holds for all z, then it must also hold for the specific term $j(p)$. This encodes a meta-logical condition, often associated with fixed-point behavior or self-application, where j selects a representative element for which the property p holds. The overall structure demonstrates an intricate interleaving of universal and existential quantification, implications, and functional abstraction, rendering it an effective benchmark for testing the depth and flexibility of automated reasoning systems.

Negation of the Formula

$$\begin{aligned} &\neg \left(\forall a \forall b \left((\forall x \exists y \forall w : (a(x,y,w) \vee b(x,y,w))) \right. \right. \\ &\Rightarrow (\exists f \forall x \forall w \left(a(x, f(x), w) \vee b(x, f(x), w) \right)) \\ &\Rightarrow \exists j \forall p : (\neg \forall z : p(z) \Rightarrow p(j(p))) \end{aligned}$$

Simplification

$$\begin{aligned}\neg(\forall x_0 \forall x_1 \big(\forall x_2 \exists x_3 \forall x_4 \, (x_0(x_2,x_3,x_4) \vee x_1(x_2,x_3,x_4)))\\ \Rightarrow \exists x_5 \forall x_6 \forall x_7 \, (x_0(x_6,x_5(x_6),x_7) \vee x_1(x_6,x_5(x_6),x_7))\\ \Rightarrow \exists x_8 \forall x_9 \, (\exists x_{10} x_9(x_{10}) \Rightarrow x_9(x_8(x_9))))\end{aligned}$$

$$\begin{aligned}(\forall x_0 \forall x_1 (\exists x_2 \forall x_3 \forall x_4 \, (\neg x_0(x_2,x_3,x_4) \wedge \neg x_1(x_2,x_3,x_4))\\ \vee (\exists x_5 \forall x_6 \forall x_7 \, (x_0(x_6,x_5(x_6),x_7) \vee x_1(x_6,x_5(x_6),x_7)))\\ \wedge (\forall x_8 \exists x_9 \, (\exists x_{10} x_9(x_{10}) \wedge \neg x_9(x_8(x_9))))\end{aligned}$$

Choice Axiom and Skolemization

Assuming:

- $x_2 = f_1(x_0, x_1)$
- $x_5 = f_2(x_0, x_1)$
- $x_9 = f_3(x_8)$
- $x_{10} = f_4(x_8)$

$$\begin{aligned}\forall x_0 \forall x_1 \forall x_3 \forall x_4 (\neg x_0(f_1(x_0,x_1),x_3,x_4) \wedge \neg X_1(f_1(X_0,X_1),X_3,X_u)\\ \vee (X_0(X_6, f_2(X_0,X_1)(X_6),X_7 \vee X_1(X_6, f_2(X_0,X_1)(X_6),X_7)))\\ \wedge \forall x_8 (f_3(x_8)(f_4(x_8)) \wedge \neg f_3(x_8)(x_8(f_3(x_8))))\end{aligned}$$

CNF Form

$$\begin{aligned}&(\neg x_0(f_1(x_0,x_1),x_3,x_4) \vee x_0(x_6, f_2(x_0,x_1)(x_6),x_7) \vee x_1(x_6, f_2(x_0,x_1)(x_6),x_7))\\ \wedge\ &(\neg x_1(f_1(x_0,x_1),x_3,x_4) \vee x_0(x_6, f_2(x_0,x_1)(x_6),x_7) \vee x_1(x_6, f_2(x_0,x_1)(x_6),x_7))\\ \wedge\ &\forall x_8 \, (f_3(x_8)(f_4(x_8)) \wedge \neg f_3(x_8)(x_8(f_3(x_8))))\end{aligned}$$

Resolution and Unification

Assume:

- $x_6 = f_1(x_0, x_1)$
- $x_5 = f_2(x_0, x_1)(x_6)$
- $f_4(x_8) = f_3(x_8)$

Then we derive a contradiction (false), hence the original formula is Theorem.

3.4.3 Example 3

Initial Formula

$$(\forall r(\forall x \exists y\,(r(x,y)) \Rightarrow \exists f \forall x(r(x(f(x)) \Rightarrow \exists j \forall p(\exists z\, p(z) \Rightarrow p(j(p))))$$

This formula comprises a sequence of implications involving nested quantifiers, relational predicates, and higher-order functions. It initiates with a universally quantified relation $r(x,y)$ and posits that if, for all x, there exists a y such that $r(x,y)$ holds (i.e., $\forall x \exists y\, r(x,y)$), then there exists a function f such that, for every x, $r(x,f(x))$ holds—effectively replacing the existential quantifier with a Skolem function. The formula proceeds to assert the existence of another function j, such that, for every unary predicate p, if there exists some z with $p(z)$ true, then $p(j(p))$ must also be true. In this context, f and j are functions introduced through existential quantification, while x, z, and p are universally quantified variables. The overall structure tests the theorem prover's capacity to handle dependencies between quantifiers and functions, integrating both first- and second-order reasoning within a concise framework.

Negation of the Formula

$$\neg\,(\forall r(\forall x \exists y\,(r(x,y)) \Rightarrow \exists f \forall x(r(x(f(x)) \Rightarrow \exists j \forall p(\exists z\, p(z) \Rightarrow p(j(p))))$$

Simplification

$$(\forall r(\exists x \forall y \neg(r(x,y)) \vee \exists f \forall x(r(x(f(x)) \wedge \forall j \exists p(\exists z\, p(z) \wedge \neg p(j(p))))$$

Skolemization

First part:

- $\exists x$ *can be replaced with* $f_4(r)$
- $\exists f$ *can be replaced with* $f_1(r)$

So, first part is: $\neg X_r(f_4(r),y) \vee (r(x,f_1(r)(x))$

Second part:

- $\exists p$ *is dependent on* $\forall j$ *so,* p *can be replaced with* $f_2(j)$
- $\exists z$ *is dependent on* $\forall j$ *so,* z *can be replaced with* $f_3(j)$

So,second part is: $f_2(j)(f_3(j)) \wedge \neg f_2(j)(j(f_2(j)))$

CNF Conversion

C1: $\neg r(f_4(r), y) \vee (r(X_x, f_1(r)(x))$
C2: $f_2(j)(f_3(j))$
C3: $\neg f_2(j)(j(f_2(j))$

Resolution

Assumtion and Unification:

- From C2 and C3, unify $f_3(j)$ with $j(f_2(j))$
- C2 and C3 leads to contradiction.

Since the negation leads to a contradiction, the original formula is a theorem.

Grouping and Interpretation of Similar Benchmark Problems

The benchmark problems Example 1, Example 2, and Example 3 exhibit a notably similar logical architecture. Each problem involves extensively nested quantifier alternations, functional abstractions, and logical implications that necessitate transformation into clause-based representations to facilitate resolution-based inference. In all three instances, the Vampire prover successfully addressed these problems through a consistent and systematic methodology: initiating with negation, proceeding to Skolemization, converting into conjunctive normal form (CNF), and culminating in resolution via unification and clause saturation. This predictable and reproducible success indicates that Vampire is particularly well-suited to handle this class of structurally complex logical problems.

In contrast, cvc5 was unable to resolve any of these problems. Despite the logical similarity among the problems and the relatively uniform transformation processes involved, the solver failed to reach a solution in any case.

Grouping these problems yields valuable insights. It highlights a pattern of solver performance whereby instances sharing certain structural characteristics consistently yield success in Vampire but remain unresolved in cvc5. This observation suggests that the current strategies employed by cvc5—particularly regarding Skolemization, clause-level inference, and symbolic rewriting—may benefit from targeted enhancements when addressing such formulae.

Furthermore, this grouping-based perspective offers a strategic pathway for future developments in solver technology. By identifying families of problems with common logical structures that diverge in solvability, we can more precisely identify areas for the extension of cvc5's capabilities. Rather than generalizing about broad performance deficiencies, this analysis points to specific transformation and reasoning stages—such as optimized Skolemization and clause saturation—that could improve cvc5's efficacy on logic-intensive problems of this nature.

In sum, analyzing these problems collectively rather than in isolation enables a clearer characterization of the boundaries between current SMT techniques and classical automated theorem proving strategies, thereby guiding the design of more versatile and hybrid reasoning systems in future research.

3.4.4 Example 4

Initial Formula

$$\forall x_0, x_3, x_2, x_1, x_5, x_4 \ ((cP(x_0, x_1, x_3) \wedge cP(x_1, x_2, x_4) \Rightarrow (cP(x_3, x_2, x_5) \Leftrightarrow cP_x(x_4, x_5)) \wedge (\forall x_6 \ cP(e, x_6, x_6)) \wedge (\forall x_7 \ cP(x_7, e, x_7)) \wedge (\forall x_8 \ cP(x_8, x_8, e))) \Rightarrow (cP(a, b, ab) \Rightarrow cP(b, a, ab))$$

This formula employs universal quantification over multiple variables and expresses a series of logical constraints involving a ternary predicate cP. It states that for all variables x0 through x5, the following conditions hold: if certain compositional chains over cP are satisfied—specifically, that cP(x0, x1, x3) and cP(x1, x2, x4) imply an equivalence between cP(x3, x2, x5) and cP(x4, x5)—and if particular axioms involving a constant e are met—namely, identity-like conditions concerning the predicate cP—then a final symmetry property logically follows: namely, that if cP(a, b, ab) holds, then cP(b, a, ab) must also hold. All variables within the formula are universally quantified. This formula models properties analogous to those encountered in algebraic structures or category theory, serving as a test of a theorem prover's capacity to handle extensive quantifier scopes, equivalences, and chained relational reasoning.

Negation of the Formula

To validate this formula via refutation, the Vampire ATP negates the conjecture and attempts to derive a contradiction.

$$\neg(\forall x_0, x_3, x_2, x_1, x_5, x_4 \ ((cP(x_0, x_1, x_3) \wedge cP(x_1, x_2, x_4) \Rightarrow (cP(x_3, x_2, x_5) \Leftrightarrow cP_x(x_4, x_5)) \wedge (\forall x_6 \ cP(e, x_6, x_6)) \wedge (\forall x_7 \ cP(x_7, e, x_7)) \wedge (\forall x_8 \ cP(x_8, x_8, e))) \Rightarrow (cP(a, b, ab) \Rightarrow cP(b, a, ab))$$

Simplification

$$((\neg cP(x_1, x_2, x_4) \vee \neg cP(x_0, x_1, x_3)) \vee (cP(x_3, x_2, x_5) = cP_x(x_4, x_5))) \wedge (\forall x_6 \ cP(e, x_6, x_6)) \wedge (\forall x_7 \ cP(x_7, e, x_7)) \wedge (\forall x_8 \ cP(x_8, x_8, e)) \wedge (cP(a, b, ab)) \wedge \neg(cP(b, a, ab))$$

CNF Conversion

Each simplified statement is converted into Conjunctive Normal Form (CNF), which consists of a conjunction of clauses, with each clause being a disjunction of literals.

C1: $(\neg cP(x_1,x_2,x_4) \vee \neg cP(x_0,x_1,x_3)) \vee (cP(x_3,x_2,x_5) = cP_x(x_4,x_5))$
C2: $cP(e,x_6,x_6)$
C3: $cP(x_7,e,x_7)$
C4: $cP(x_8,x_8,e)$
C5: $cP(a,b,ab)$
C6: $\neg cP(b,a,ab)$

Binary Proxy Clausification für C1

Binary Proxy Clausification is a technique used to handle equivalences or biconditionals ($\leftrightarrow$) in a clause, typically of the form: $(a \leftrightarrow b)$ This is rewritten as:

- $(a \wedge b)$
- $(b \vee a)$

However, more significantly, in automated theorem provers such as Vampire, when an equality is involved—such as cP(...) = cPx(...)—the automated theorem prover may introduce proxy variables or partition cases to effectively manage disjunctions.

- Set the left-hand side (e.g., cP(...)) to false
- Or the right-hand side (e.g., cPx(...)) to false

(12)

In C1 the clause was rewritten into two alternative forms — by assuming cP(x3, x2, x5) or cPx(x4, x5) to be false and resolving accordingly:

$$\textit{Set } cP(x_3,x_2,x_5) \textit{ to } false:$$
$$\text{C1: } \neg cP(x_3,x_2,x_5) \vee cP_x(x_4,x_5) \vee \neg cP(x_1,x_2,x_4) \vee \neg cP(x_0,x_1,x_3)$$

$$\textit{or set } cP_x(x_4,x_5) \textit{ to } false:$$
$$\text{C1}': cP(x_3,x_2,x_5) \vee \neg cP_x(x_4,x_5) \vee \neg cP(x_1,x_2,x_4) \vee \neg cP(x_0,x_1,x_3)$$

Superposition

Superposition constitutes one of the most potent inference rules employed in logical theorem proving with equality. It extends the classical resolution rule to accommodate domains that encompass

equations and equalities, which are prevalent in mathematical and logical reasoning processes. Unlike basic resolution, which solely relies on the logical structure of clauses—such as implications and disjunctions—superposition enables the reasoner to directly manipulate and reason about equalities between terms. Consequently, it serves as a fundamental mechanism within saturation-based automated theorem provers, such as Vampire.

Fundamentally, superposition facilitates the substitution of terms utilizing established equalities. Specifically, when two terms are known to be equal— for example, if a clause asserts s = t—the superposition rule permits the replacement of an occurrence of s within any other clause with t, provided certain conditions are satisfied. These conditions may include constraints related to term ordering, unifiability, and the selection functions employed by the prover. This process effectively incorporates the known equality into the broader set of clauses, propagating its influence and enabling the derivation of further inferences.

To understand this more concretely, consider two clauses:

- A clause that includes an equality, such as s = t;
- Another clause in which s appears as a subterm, say in a literal like P(..., s, ...).

The superposition rule allows for the substitution of the subterm s within the second clause with t, resulting in the formation of a new clause. This resultant clause may subsequently be utilized in additional inference procedures, thereby expanding the set of clauses until either a contradiction is encountered or no further information can be derived. The underlying principle is that, since s and t are equivalent, the logical meaning of the clause remains intact following the substitution. However, unlike naive substitution, superposition operates with full formal rigor; it involves unification, ensures soundness, and adheres to the syntactic constraints stipulated by resolution-based calculi.

- Using Superposition for C1 and C3:(assume $x_0 = x_7, x_1 = e, x_3 = x_7$)
  C7: $\neg cP(x_7, x_2, x_5) \vee cP_x(x_4, x_5) \vee \neg cP(e, x_2, x_4)$

- Using Superposition for C7 and C5:(assume $x_7 = a, x_2 = b, x_5 = ab$)
  C8: $cP_x(x_4, ab) \vee \neg cP(e, b, x_4)$

- Using Superposition for C1´ and C8:(assume in C1´ $x_3 = e, x_2 = b, x_5 = x_4$)
  C9: $\neg cP_x(x_4, x_4) \vee cP_x(x_4, ab) \vee \neg cP(x_1, b, x_4) \vee \neg cP(x_0, x_1, x_3)$

- Using Superposition for C9 and C3:(assume $x_1 = x_7, b = e, x_4 = x_7$)
  C10: $\neg cP_x(x_4, x_7) \vee cP_x(x_7, ab) \vee \neg cP(x_7, e, x_7) \vee \neg cP(x_0, x_7, x_3)$

- Using Superposition for C10 and C4:(assume $x_0 = x_8, x_7 = x_8, x_3 = e$)
  C11: $\neg cP_x(x_4, x_8) \vee cP_x(x_8, ab)$

- Using Superposition for C8 and C4:(assume $e = x_8, b = x_8, x_4 = e$)
  C12: $cP_x(e, ab)$

- Using Superposition for C11 and C12:(assume $\mathrm{x}_4 = e, x_8 = ab$)
  C13: $cP_x(ab, ab)$

- Using Superposition for C1´ and C6:(assume $\mathrm{x}_3 = b, x_2 = a, x_5 = ab$)
  C14: $\neg cP_x(x_4, ab) \vee \neg cP(x_1, a, x_4) \vee \neg cP(x_0, x_1, b)$

- Using Superposition for C14 and C3: (assume $\mathrm{x}_0 = x_7, x_1 = e, b = x_7$)
  C15: $\neg cP_x(x_4, ab) \vee \neg cP(e, a, x_4)$

- Using Superposition for C15 and C2:(assume $\mathrm{a} = \mathrm{x}_6, x_4 = x_6$)
  C16: $\neg cP_x(x_6, ab)$

- Using Superposition for C13 and C16: (assume $\mathrm{ab} = \mathrm{x}_6$)
  C13: $cP_x(x_6, ab)$

- From looking at C13 and C16, C13 is negation of C16
  So, this leads to contradiction.

Hence, the original formula is Theorem

Summary of Example proof

Through a meticulous application of negation, Conversion to Conjunctive Normal Form (CNF), Binary Proxy Clausification, and superposition, it is demonstrated that example 4 constitutes a theorem. This proof adhered closely to the methodologies implemented by the Vampire theorem prover, which successfully resolved the problem. Conversely, the cvc5 solver failed to establish the theorem, thereby underscoring the divergence in capabilities between saturation-based automated theorem proving systems and SMT-based frameworks.

The original formula exhibits a complex logical architecture characterized by nested universal quantifiers, ternary function symbols, and implications involving biconditional subformulae. Notably, the conjecture posits a form of commutativity for the operation cP, predicated upon certain associativity-like and identity properties. This intricate structure presents substantial challenges to automated reasoning tools, especially when equalities, compositions, and chained inferences are deeply embedded within the logical formulation.

The proof commences with the negation of the conjecture to facilitate a refutation-based strategy. This critical transformation shifts the problem from direct proof to demonstrating the inconsistency of the negated statement. Subsequently, the negated formula undergoes simplification and is transposed into conjunctive normal form (CNF). This process involves eliminating implications and restructuring the logical expression into a conjunction of disjunctions, conforming to standard practices in clause-based automated reasoning.

A distinctive aspect of this proof, setting it apart from more conventional approaches, is the employment of Binary Proxy Clausification. This technique is utilized to handle equivalences such as $(a \leftrightarrow b)$ within the CNF. Instead of expanding this biconditional directly into two implications, Vampire introduces propositional flexibility by considering cases where either side may be false, thereby generating multiple clauses that facilitate more efficient reasoning. This approach allows the inference engine to split the biconditional into multiple reasoning paths, which can be explored in parallel during clause saturation.

The most pivotal advancement in this proof is the utilization of superposition—a rule extending classical resolution by enabling equalities to be used for rewriting terms. Superposition allows the system to replace terms within clauses based on established equalities, which is essential when dealing with functional terms and nested compositions. In example 4, superposition is employed repeatedly to propagate equality information, instantiate variables systematically, and derive new clauses linked to earlier inferences. This iterative reasoning process ultimately yields a contradiction, thereby establishing the unsatisfiability of the negated formula.

Vampire's capacity to systematically apply superposition within a saturation-based framework is instrumental to the proof's success. The intricate clause interactions involve deeply nested structures, multiple inference pathways, and equality-driven substitutions requiring precise variable unification. Vampire's internal mechanisms for clause selection, simplification, and ordering optimize the management of this complexity, contributing significantly to the proof's efficiency.

In contrast, cvc5 fails to resolve this problem, primarily due to its lack of support for full saturation-based superposition. Although efficient in quantifier instantiation and certain theory reasoning tasks, cvc5's inference engine does not implement advanced equality handling techniques such as equational clause saturation. This deficiency hampers its ability to navigate the heavily equational and nested functional structures characteristic of this problem. Additionally, cvc5's limited capacity for managing chained implications and proxy clausification constrains its applicability to problems rich in equalities and complex logical constructs.

In conclusion, the successful proof of example 4 by Vampire exemplifies the potency of saturation-based theorem proving, especially when augmented with Binary Proxy Clausification and superposition techniques. The inability of cvc5 to solve the problem highlights current limitations within SMT technology regarding sophisticated equality reasoning and the employment of saturation-based inference mechanisms. This case serves as a valuable benchmark for evaluating the comparative strengths and limitations of different automated reasoning paradigms. Furthermore, it underscores potential avenues for enhancement in SMT solvers, notably in handling equality reasoning, supporting saturation-based inference strategies, and implementing advanced clause transformation methodologies.

4 Experiments

The primary objective of this study is to identify and thoroughly analyze instances in which cvc5 is unable to resolve specific problems from the TPTP library, whereas Vampire is capable of doing so. Such instances are of particular significance, as they illuminate potential deficiencies in the reasoning mechanisms employed by cvc5 or in its overall design and implementation. Through meticulous documentation of these cases, the research aims to investigate the underlying causes contributing to cvc5's difficulties in these scenarios.

To carry this out, a clear plan is followed:

- **Execution of the Tests:** Both cvc5 and Vampire are configured to operate on a carefully selected subset of problems from the TPTP library, ensuring diversity in problem type and difficulty.
- **Result Comparison:** Upon completion of the tests, the study examines instances where cvc5 fails to find a solution whereas Vampire succeeds.
- **Detailed Analysis:** For each discrepancy, an in-depth investigation is conducted. The team analyzes the structure of the problems, the tactics employed by Vampire, and the reasons behind cvc5's difficulties.
- **Manual Reconstruction of Proofs:** In certain cases, proofs are reconstructed manually to gain deeper insights into the reasoning processes and to verify the results.

Through meticulous adherence to this methodology, the study aims to identify valuable strategies to enhance cvc5's performance and to advance the broader field of automated theorem proving.

4.1 The TPTP Problem Library

The Thousands of Problems for Theorem Provers (TPTP) library, compiled by Geoff Sutcliffe, serves as the primary benchmark for research in automated theorem proving. It provides a meticulously organized collection of problems, categorized by domain, logic type, and difficulty level, thereby offering researchers a reliable foundation for assessing and comparing various ATP systems in a fair and consistent manner (10). The TPTP library is regularly updated to ensure its continued relevance and to serve as a challenging test environment for contemporary theorem provers. Its structure, which encompasses classifications such as algebra, geometry, set theory, and logic puzzles, delivers a comprehensive overview of diverse logical domains (10).

In addition to performance evaluation, the TPTP framework has influenced the development and refinement of new theorem-proving methodologies. Its broader infrastructure—such as the SZS ontology for annotating problem outcomes—has facilitated greater reproducibility of results and enhanced transparency in evaluations across competitions and scholarly research. Notably, the TPTP library underpins the CADE ATP System Competition (CASC), an annual event designed to evaluate prominent ATP systems on a representative subset of TPTP problems. This competitive platform has significantly contributed to the advancement of ATP efficiency and robustness over the years (3).

4.2 Experimental Setup

The experiments conducted in this study were carried out within a Linux environment, utilizing Ubuntu as the chosen operating system. Ubuntu was selected due to its optimal balance of stability and extensive support for the tools required in automated theorem proving. Within this environment, the theorem provers Vampire, cvc4, and cvc5 were installed and configured meticulously to ensure seamless operation. The entire TPTP (Thousands of Problems for Theorem Provers) library was

downloaded directly from its official website (www.tptp.org) and employed as the comprehensive set of benchmark problems for all testing procedures.

The TPTP library categorizes its problems systematically into directories, each associated with a specific logical domain or a defined level of difficulty. Within these directories, individual problem files are stored with a .p extension, each representing a distinct logical claim that necessitates proof. This organized structure facilitated a structured, step-by-step approach to processing the benchmark problems and provided a systematic framework for the experiments.

4.3 Scripting and Test Execution

To automate the testing process, custom shell scripts were developed to manage the operational workflow. These scripts sequentially processed all .p files within each designated TPTP folder, utilizing the cvc5 solver. During each processing step, the output of cvc5 was captured and categorized based on the returned result. Particular attention was given to cases where cvc5 either resulted in a timeout or returned an "Unknown" status. A "Timeout" indicated that the solver was unable to complete the problem within the predefined time limit, whereas an "Unknown" status signified that cvc5 was unable to determine the validity of the problem within its reasoning capabilities.

For each batch of problems, a new Excel worksheet was generated to maintain an organized record. Each row corresponded to a different problem, documenting the filename and the outcome provided by cvc5.

Following the completion of cvc5 runs, the same procedures were repeated using Vampire. Vampire was selected due to its well-established performance and proven track record in theorem proving competitions. Similar automated scripts processed each folder with Vampire, and the results were systematically recorded for subsequent comparative analysis.

4.4 Benchmark Filtering and Comparison

After gathering all the results from both cvc5 and Vampire, the next step was a detailed comparison. The main goal was to spot problems where cvc5 couldn't give a clear answer—whether it "timed out" or reported "Unknown"—but Vampire managed to solve the problem. Vampire's successful outcomes showed up as statuses like "Theorem," "Satisfiable (SAT)," or "Unsatisfiable (UNSAT)."

Each case was double-checked against the Excel sheets to make sure the match was correct. Whenever a problem fit the pattern—cvc5 struggling and Vampire succeeding—the file was marked for closer study. This careful filtering trimmed down the huge TPTP collection into a smaller, more meaningful set of tough problems that really showed the practical differences between the two provers.

This comparative analysis was crucial in identifying the strengths and weaknesses of each prover. By focusing on the cases where Vampire succeeded and cvc5 did not, insights were gained into the specific problem characteristics that posed challenges for cvc5. Such characteristics included the presence of complex quantifier structures, intricate equality reasoning, or particular theory combinations.

Furthermore, this filtering process enabled the categorization of problems into distinct groups based on their logical features, facilitating a more targeted investigation into the limitations of cvc5's reasoning capabilities.

4.5 Data Collection and Validation

To systematically record the benchmarking outcomes, a dedicated Excel worksheet was created for each benchmark directory within the TPTP library. Within each worksheet, all individual benchmark files from the respective directory were cataloged. For each problem, the result generated by cvc5 was documented, with particular emphasis on cases where the solver either exceeded the time limit (timed out) or returned an ”Unknown” status. These instances were of primary interest, as they indicated scenarios wherein cvc5 was unable to either complete the problem within the allocated time or determine the satisfiability status conclusively.

Subsequently, these challenging benchmarks were re-evaluated utilizing the Vampire theorem prover. The results from Vampire were recorded in a manner consistent with the initial data collection, focusing on cases where Vampire successfully provided definitive outcomes, such as ”Theorem,” ”Satisfiable (SAT),” or ”Unsatisfiable (UNSAT).” In the respective Excel worksheets, benchmarks in which Vampire succeeded but cvc5 failed were distinctly highlighted. These cases constituted the core dataset for subsequent in-depth analysis.

To ensure the integrity and reliability of the collected data, a validation procedure was implemented. This involved cross-verifying the automated results with manual re-executions for a randomly selected subset of benchmark problems. Any discrepancies identified were thoroughly investigated and rectified to maintain dataset accuracy.

Furthermore, relevant metadata—including problem size (quantified by the number of clauses and literals), domain categorization, and logical complexity—was annotated for each benchmark. This enriched dataset facilitated a more detailed analysis of factors influencing the performance of the theorem provers.

4.6 Benchmark Results

4.6.1 Key Findings and Problem Grouping

Through this two-stage evaluation, distinct patterns emerged across problem types that elucidated the comparative strengths of the two provers. A substantial proportion of the problems that cvc5 failed to solve—yet Vampire successfully resolved—exhibited specific structural characteristics. These problems typically demonstrated a high degree of symbolic density, including deeply nested quantifiers, layered implications, and sequences of equalities. Addressing such problems often necessitates the application of inference strategies such as Skolemization, resolution, and unification, which are fundamental components of Vampire’s saturation-based architecture. In these instances, cvc5 either exceeded the allotted time or returned an ”unknown” result, whereas Vampire was able to derive complete proofs. This category constituted the majority of the benchmarks where the performance of the two solvers diverged.

Another prominent group of benchmarks involved problems related to functional dependencies and equational reasoning, including recursive terms and algebraic expressions. These problems frequently benefit from techniques such as superposition and rewriting, which Vampire implements with efficiency. Such inference mechanisms enable Vampire to simplify complex functional relationships into forms that are more readily resolvable. In contrast, cvc5 lacks integrated saturation or superposition mechanisms and consequently encountered difficulties in progressing on these problems. This further underscores the advantage of Vampire’s clause-based symbolic handling in

equational reasoning domains.

A third classification encompassed problems of moderate logical complexity, often characterized by chains of universal quantifiers, uninterpreted symbols, and implication-rich formula structures. Although less complex than the previous categories, these problems still posed challenges for cvc5, primarily due to the absence of clause-level symbolic reasoning capabilities. Vampire's adaptable use of inference rules allowed it to consistently complete these problems, demonstrating its efficacy even in cases of intermediate logical complexity.

Additionally, it is important to note that a subset of the benchmark problems remained unsolved by both provers. These cases likely involve undecidable logical fragments or require inference steps beyond the capabilities of current solvers within feasible time and resource limits. These instances highlight the persistent challenges in automated theorem proving, particularly when addressing the full scope of higher-order logic.

4.6.2 Broader Implications

The divergence in performance observed between cvc5 and Vampire across the curated problem set offers valuable insights into solver specialization. cvc5 demonstrates superior capabilities in theory-intensive, quantifier-light contexts and is recognized as a leading performer in domains such as verification and constraint solving (1). Conversely, in purely symbolic logic tasks characterized by structural quantification, Vampire's resolution-based architecture, along with Skolemization and clause saturation techniques, provides it with a significant advantage (12).

Rather than interpreting this as a limitation of cvc5, these findings highlight targeted opportunities for enhancement and development of cvc5. Additionally, the observed grouping patterns of problems suggest avenues for designing adaptive or advance cvc5 capable of selecting the most appropriate approach on specific problem characteristics.

4.7 Identification of Problem Patterns

After aggregating the results of the comparisons, each instance in which Vampire succeeded while cvc5 did not was meticulously examined. This process involved opening each .p file, analyzing the structure of the problem, the construction of the logical expressions, and the complexity of the setup. Additionally, the outputs and proof trails of Vampire were reviewed to ascertain how it successfully resolved each problem.

This detailed examination revealed certain recurring patterns. The more challenging problems were categorized based on shared characteristics, such as:

- **Logical structure:** Extensive use of quantifiers or multiple layers of nested functions.
- **Problem size:** Large terms or numerous clauses to process.
- **Reasoning challenges:** Difficulties associated with handling equations or addressing Skolemization.
- **Theory-specific issues:** Problems related to aspects such as uninterpreted functions or particularly challenging axioms.

Breaking down the benchmarks into these groups provided a clearer understanding of where cvc5 tends to encounter difficulties. It also highlighted which proof strategies or heuristics may require further enhancement to bridge the existing gap.

5 Analysis

5.1 Overview of Comparative Results

The experiments conducted in this thesis have revealed significant and reproducible disparities between two widely utilized automated reasoning systems—cvc5, a solver for satisfiability modulo theories, and Vampire, a saturation-based logical theorem prover. The experiments encompassing 23 logical problem sets from the TPTP library demonstrated that, although both tools are effective within their respective scopes, cvc5 consistently underperformed in problems characterized by highly nested logical structures, alternation of universal and existential quantifiers, or abstraction involving predicates and functions.

In instances where cvc5 either timed out or yielded an 'unknown' response, Vampire frequently succeeded by establishing a complete proof trace leading to either a contradiction or affirmation of a conjecture. This divergence was particularly pronounced in higher-order problems or those with minimal theoretical content. This chapter presents a multi-faceted explanation for these results, beginning with the identification of failure patterns in cvc5, followed by an examination of the internal mechanisms of Vampire that contribute to its successes, and concluding with an analysis of design implications and potential architectural enhancements.

5.2 Patterns in cvc5 Failures

Although cvc5 is a general-purpose and exceptionally high-performance SMT solver, renowned for its proficiency in addressing theory-rich, quantifier-light problems, it has consistently underperformed with higher-order or purely logical problems within our benchmark suite. These failures were not incidental but reflected the intrinsic design architecture and inference strategies employed by cvc5. Unlike saturation-based provers such as Vampire, cvc5 lacks essential mechanisms including general resolution, unification, Skolemization, binary proxy clausification, and, crucially, superposition. The absence or limited implementation of these foundational automated theorem proving (ATP) techniques significantly contributes to cvc5's inability to prove challenging logical formulas from the TPTP library.

5.2.1 Skolemization Incompatibility

Skolemization is a fundamental preprocessing step in automated theorem proving that transforms existential quantifiers into function terms that depend on previously quantified variables. This transformation is critical for converting logical formulas into a form suitable for clause-based reasoning and resolution.

Vampire conducts comprehensive Skolemization early in its processing pipeline, introducing Skolem functions that capture logical dependencies among variables. For example, in problem

Example 1, Vampire introduces function symbols such as f1(X) to eliminate existential quantifiers, thereby rendering the resulting set of clauses amenable to resolution and unification.

Conversely, cvc5 does not employ traditional Skolemization across general first- or higher-order logic. Its treatment of quantifiers is heuristic and often incomplete. It utilizes E-matching and counterexample-guided quantifier instantiation (CEGQI) methods, which are heavily reliant on pattern matching. While these techniques are highly effective in various SMT contexts, particularly in verification tasks, they do not structurally eliminate existential quantifiers and fail to generate a fully ground, Skolemized clause set necessary for saturation-based reasoning.

As a result, cvc5 often remains impeded and is unable to transform the formula into a form from which a contradiction can be derived through logical inference.

5.2.2 Absence of Resolution Calculus

Resolution constitutes the foundational inference rule of first-order automated theorem proving. It enables the generation of a resolvent clause from two clauses with complementary literals, progressively narrowing down the clause space until either a contradiction (empty clause) is discovered or saturation is achieved.

Vampire excels at resolution-based reasoning, systematically applying resolution across all pairs of clauses while optimizing the search through redundancy elimination and clause prioritization. This approach is central to its success in problems such as Example 4, where the chaining of relational properties through resolution was crucial for establishing the conjecture.

By design, cvc5 does not incorporate general-purpose resolution. Its conflict-driven clause-learning (CDCL) SAT solving is focused on Boolean conflicts, with any resolution-like reasoning constrained to ground-level Boolean abstractions. When faced with quantified logical expressions that necessitate symbolic resolution across abstract terms, cvc5 lacks the inference machinery required to proceed, resulting in either a timeout or an "unknown" verdict.

This structural deficiency renders cvc5 unsuitable for reasoning chains that require the merging and simplification of clauses through general resolution.

5.2.3 Inability to Perform Unification

Unification—the process of identifying a substitution that renders two terms syntactically identical—is essential for effectively applying resolution. In higher-order or function-rich problems, unification becomes increasingly intricate yet critical.

Vampire extensively employs syntactic unification and rewriting. It can match variable terms across clauses, resolve them under substitutions, and maintain those substitutions throughout the proof tree. This capability enables it to chain inferences even when functions are nested or when arguments must be unified with Skolem terms.

In contrast, cvc5 does not support general symbolic unification. Its quantifier instantiation is heuristic and lacks a global substitution framework necessary for clause-level inference. Without unification, cvc5 is unable to reconcile the variable bindings essential for executing multi-step logical deductions. This limitation was evident in benchmarks such as Example 1 and Example 2, where function instantiations depended on unifying multiple argument structures—an operation that Vampire successfully managed, but cvc5 did not.

The absence of unification not only impedes resolution but also precludes the ability to track variable dependencies within Skolemized logic chains.

5.2.4 Lack of Binary Proxy Clausification

Binary proxy clausification is a transformation technique utilized to simplify complex formulas, particularly biconditionals and implications, into binary clauses with proxy terms. This approach facilitates manipulation of the formula through resolution and unification.

In Example 4, Vampire transforms a deeply nested biconditional into a collection of binary clauses utilizing proxy functions. These binary proxies serve as placeholders for subformulas, thereby allowing for the separation of clauses and targeted resolution. This technique enables Vampire to address compositional logic structures without becoming enmeshed in excessively large or overlapping clauses.

cvc5 does not implement binary proxy clausification. Its internal representation of formulas preserves more of the original logical structure, without introducing intermediate placeholders. This design choice simplifies integration with theory solvers but severely limits clause-level flexibility.

5.2.5 Absence of Superposition Calculus

A fundamental structural limitation of the cvc5 system is the absence of superposition, an inference rule that enhances resolution by facilitating reasoning with equalities. Superposition allows for the replacement of terms in one clause with equal terms from another, thereby enabling inference across functionally or algebraically equivalent expressions.

The superposition engine of Vampire is among its core advantages. It efficiently employs term indexing and ordering constraints to implement superposition. In problems that are heavily algebraic or in logical puzzles characterized by symmetry, superposition empowers Vampire to consolidate logically equivalent terms and deduce contradictions that would otherwise be inaccessible through resolution alone.

In contrast, cvc5 lacks any form of superposition. Its equality reasoning is addressed at the theory level, for instance, utilizing congruence closure in the theory of equality with uninterpreted functions; however, it does not engage in a clause-level, inference-driven approach. Consequently, cvc5 can resolve simple equalities within constrained theories, yet it is unable to rewrite symbolic equalities within the context of proof search. This limitation becomes particularly critical in higher-order problems, where equalities are often functional, nested, or recursive.

6 Conclusion

The primary objective of this thesis has been to conduct a rigorous benchmarking and theoretical analysis comparing two fundamentally different paradigms in automated reasoning: cvc5, a state-of-the-art SMT (Satisfiability Modulo Theories) solver, and Vampire, a traditional saturation-based automated theorem prover. The research was motivated by a striking and recurring observation—cvc5, despite its strong performance on theory-rich problems, often struggles with a variety of higher-order problems characterized by symbolic complexity from the TPTP (Thousands of Problems for Theorem Provers) corpus. In contrast, Vampire frequently excels on the same problem

set. This prompted a focused investigation not only to determine where these discrepancies arise but also to explore the underlying reasons and what these insights reveal about the capabilities and design limitations of each system.

To achieve this, a comprehensive benchmark framework was developed, which facilitated automated testing on problems from 23 distinct folders within the TPTP library, utilizing both cvc5 and Vampire. This framework examined a broad spectrum of logical domains. Each problem was subjected to identical memory and runtime conditions, with outputs categorized and documented for comparative analysis. A central focus remained on identifying those problems where cvc5 encountered failures—either through timeouts or returning an "unknown" result—when Vampire was able to successfully demonstrate a proof. The quantity and nature of such instances exhibited a consistent trend: cvc5 tends to perform poorly in domains characterized by minimal background theory and strong logical abstraction.

This benchmarking process transcended mere performance comparisons. It served as a lens through which deeper structural weaknesses of each solver could be illuminated. cvc5's failures were not typically random; rather, they displayed familiar patterns associated with the alternation of quantifiers, the depth of logical dependencies, and the lack of theory-dependent support. Conversely, the successes of Vampire were not coincidental either; they stemmed from the deliberate use of classical ATP mechanisms such as clause saturation, resolution, unification, Skolemization, and superposition.

One significant component of this thesis involved the reconstruction and formal transformation of specific benchmark problems including Example 1, Example 2, and Example 4. These problems were meticulously transformed through the complete proof pipeline of Vampire, encompassing negation, NNF translation, Skolemization, conversion into CNF, and concluding with clause-level resolution. Through this transformation process, critical features absent from cvc5 were brought to light.

Firstly, cvc5 does not apply standard Skolemization as practiced in traditional ATPs. Existential quantifiers are not eliminated; rather, they are managed through heuristic instantiation techniques, which fall short in problems necessitating robust logical transformation. Secondly, cvc5 does not possess general resolution mechanisms or the ability to achieve clause saturation. It lacks the capacity to systematically combine clauses in order to produce resolvents, a fundamental characteristic of classical proof search. Additionally, it does not incorporate unification, which involves finding substitutions that render different terms equivalent, within a more general symbolic context. cvc5 is unable to track substitutions between different clauses in a manner that permits chained deductions. Lastly, the absence of superposition—a potent equality reasoning method that is pivotal in Vampire's inference system—renders cvc5 largely insensitive to functional symmetries and equalities. Collectively, these structural shortcomings account for its inability to make progress on higher-order or proof-theorem-free logical problems.

In contrast, Vampire is architecturally designed for these particular problem classes. Skolemization is enacted early in the process, reducing existential quantifiers to functional terms and thus streamlining problem structure. Its resolution-based inference engine explores a wide array of candidate clause sets, employing saturation to ascertain logical consequences. Unification is utilized for the combination of symbolic terms and the propagation of equalities across proof states. Moreover, Vampire's application of binary proxy clausification, in addition to superposition, allows it to decompose large and complex formulas into manageable components and to rewrite terms in the context of equality logic.

These strategies enable Vampire to systematically navigate a proof space that remains largely inaccessible to cvc5. Its successes on examples specified in this theisi where equality reasoning and nested functions are essential—underscore its superiority over the newer SMT solver cvc5 in purely symbolic contexts.

The insights derived from this comparative study transcend mere descriptiveness; they carry significant implications for the future design of automated reasoning systems. The findings suggest that the existing dichotomy between SMT solvers and ATPs is inherently restrictive. SMT solvers such as cvc5 are optimized for specific classes of problems, particularly those involving arithmetic, data types, or arrays with shallow quantification. Conversely, ATPs provide unparalleled symbolic reasoning capabilities, especially in situations where saturation and general-purpose inference are required. Bridging this divide will necessitate a reevaluation of solver architecture to facilitate dynamic adaptation according to problem characteristics.

A hybrid system that amalgamates the modular theory reasoning capabilities of cvc5 with the ability to invoke saturation or resolution-based inference when theory solvers encounter difficulties could represent a promising avenue for future research. Likewise, the integration of symbolic unification, advanced Skolemization, or superposition reasoning into SMT solvers could considerably extend their applicability in logic-intensive domains.

Building upon this work, future investigations may take several directions. One natural extension is the incorporation of machine learning techniques into solver control strategies. Learning-based systems could optimize quantifier instantiation, clause selection, or the prioritization of inference rules. Initiatives like DeepMath, which explore premise selection, clause weight adjustment, and proof path prediction using neural networks, could be adapted for application in both ATP and SMT environments.

Furthermore, the benchmarking dataset produced in this thesis—particularly the subset of TPTP problems wherein cvc5 failed and Vampire succeeded—can serve as a valuable resource for solver development, regression testing, and optimization. These problems highlight the edge cases where current systems diverge and create a focused platform for experimentation and performance evaluation.

Another significant future endeavor could involve the integration of automated theorem provers with interactive proof assistants such as Coq or Isabelle/HOL. The manual proof transformations and logical decompositions presented in this thesis could serve as templates for automation modules within those environments, thereby reducing user overhead in the construction of formal proofs.

Lastly, there is ample opportunity for expanding the scope of benchmarking itself—by introducing mixed theory-logic problems, combining verification-like constraints with abstract logical reasoning, and comparing not only cvc5 and Vampire but also other tools such as Z3, E Prover, and iProver.

This thesis commenced with a straightforward inquiry: why does cvc5 falter on certain logical problems that Vampire successfully resolves? Through systematic benchmarking, rigorous proof reconstruction, and careful comparative analysis, this inquiry has been addressed in both theoretical and practical aspects. cvc5's reliance on theory solvers and its lack of general symbolic inference render it ill-suited for domains necessitating clause saturation, Skolemization, resolution, or equality-based rewriting. In contrast, Vampire's classical inference methods are precisely the tools required for such domains.

By elucidating this gap and providing a structured foundation for understanding it, this thesis not only augments the comprehension of contemporary reasoning systems but also opens avenues

for bridging this divide—by learning from the strengths of both paradigms and progressing toward more unified, intelligent, and adaptive automated theorem proving systems.

References

[1] Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, et al. cvc5: A versatile and industrial-strength smt solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 415–442. Springer, 2022.

[2] Clark Barrett, Leonardo De Moura, and Aaron Stump. Smt-comp: Satisfiability modulo theories competition. In *International Conference on Computer Aided Verification*, pages 20–23. Springer, 2005.

[3] Clark Barrett, Leonardo de Moura, and Aaron Stump. Smt-comp: Satisfiability modulo theories competition. pages 20–23, 01 2005.

[4] Clark Barrett, Aaron Stump, and Cesare Tinelli. The smt-lib standard: Version 2.0. *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (SMT '09)*, 2009.

[5] Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Encoding higher-order logic in first-order logic: On the correctness of translating hol to fol. *Journal of Automated Reasoning*, 56(3):237–259, 2016.

[6] Wikipedia Contributors. De morgan's laws - wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/De_Morgan%27s_laws`, 2025. Last accessed: May 18, 2025.

[7] John Harrison. *Handbook of practical logic and automated reasoning*. Cambridge University Press, 2009.

[8] Geoffrey Irving, Christian Szegedy, Alexander A Alemi, Niklas Eén, François Chollet, and Josef Urban. Deepmath-deep sequence models for premise selection. *Advances in neural information processing systems*, 29, 2016.

[9] Alan JA Robinson and Andrei Voronkov. *Handbook of automated reasoning*, volume 1. Elsevier, 2001.

[10] Geoff Sutcliffe. The tptp problem library and associated infrastructure. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[11] TPTP Organization. Thousands of problems for theorem provers (tptp), 2024.

[12] Andrei Voronkov. The anatomy of vampire: Implementing bottom-up procedures with code trees. *Journal of Automated Reasoning*, 2014.

[13] Wikipedia contributors. Negation normal form, 2024. Accessed: 2025-05-18.

[14] Wikipedia contributors. Conjunctive normal form, 2025. Accessed: 2025-05-18.

[15] Wikipedia contributors. Skolem normal form, 2025. Accessed: 2025-05-18.