Finding Connections via Satisfiability Solving

Clemens Eisenhofer[1]([✉]) [ID], Michael Rawson[2] [ID], and Laura Kovács[1] [ID]

[1] TU Wien, Vienna, Austria
{clemens.eisenhofer,laura.kovacs}@tuwien.ac.at
[2] University of Southampton, Southampton, UK
michael@rawsons.uk

Abstract. Commonly used proof strategies by automated reasoners organise proof search either by ordering-based saturation or by reducing goals to subgoals. In this paper, we combine these two approaches and advocate a SAT-based method with symmetry breaking for connection calculi in first-order logic, with the purpose of further pushing the automation in first-order classical logic proofs. In contrast to classical ways of reducing first-order logic to propositional logic, our method encodes the structure of the proof search itself. We present three distinct SAT encodings for connection calculi, analyse their theoretical properties, and discuss the effect of using SAT/SMT solvers on these encodings. We implemented our work in the new solver UPCoP and showcase its practical feasibility.

1 Introduction

Search strategies employed by *automated theorem provers for first-order logics* can be divided into two broad classes [16]: *ordering-based* and *subgoal-reduction*. The first class, which contains saturation-based theorem provers including VAMPIRE [35], E [54], and SPASS [60], work by continuously deducing new facts from an existing set of formulas and expanding the search space with these new facts. The second class, containing systems such as SETHEO [38] or leanCoP [45], works by manipulating a partial proof and implementing backtracking if necessary.

The subgoal-reduction class has the disadvantage that redundant, "useless", formulas in the search space may be explored in duplicate manner, unless special care is taken to "remember" where the prover has already been before. Avoiding such redundant cases for the purpose of efficient reasoning is a subject of great interest. Therefore, *global refinement* within the subgoal-reduction approach to theorem proving has been proposed and investigated [11]. In general, such refinements can contain non-trivial propositional structure; for example, the information "if clauses C and D are in the current proof attempt, and the current substitution binds $x \mapsto t$ and $y \mapsto s$, we are in a dead-end and have to backtrack" is a hard formula to reason with.

Backtracking mechanisms are routinely implemented in Boolean satisfiability (SAT) solvers [13], for the purpose of refining the (partial) proof. Modern SAT

solvers *learn* relevant information as they go, and even allow users to add constraints during the solver's search for a model, in response to the solver's current (partial) assignment. When a SAT solver cannot find a satisfying assignment, an *explanation* in the form of an unsat core is given. In this paper, we examine these learning and explanation features in the context of first-order theorem proving, which usually makes SAT solvers an ideal vehicle for managing global information and their refinements.

Here we are interested in the integration of SAT solving and subgoal-reduction, focusing on Bibel's *connection method* [10]. Connection methods yield powerful calculi for automation of expressive logics, for example, for classical first-order logic [10,38], higher-order logic [2], linear logic [31], and several modal logics [46]. Our work directly encodes the search for connection proofs as a Boolean satisfiability problem, allowing the solver to dictate search decisions and respond by asserting constraints, such that when a satisfying assignment is reached, it represents a complete proof. To this end, we introduce three encodings of connections (Sections 3–5) and show their complementary nature and power (Section 7).

First, we present our method applied to connection *tableaux* (Section 3) and highlight some unfortunate properties of this setting. Therefore, we refine our encoding of the connection calculus using a matrix form (Section 4). Further, we describe how unsat cores can be used to guide iterative deepening during SAT solving (Section 5). The resulting encodings allow many global refinements that are usually not feasible within other methods (Section 6).

Our work intends for the SAT solver to return a satisfying assignment of our constraints, where the model encodes a finished proof: matrix or tableau. *In other words, we represent a first-order proof as a propositional model.* This contrasts with most other uses of SAT solvers in theorem proving in which ground unsatisfiability is the aim [59], often witnessing Herbrand-style refutation by instantiation of first-order clauses.

The practical use of our approach is showcased by evaluating our work on the TPTP repository of first-order problems [57]. To this end, we provide a proof-of-concept implementations of our techniques in our new solver UPCoP, where UPCoP uses either the CaDiCaL SAT solver [12] and the Z3 SMT solver [43]. When evaluating UPCoP against the state-of-the-art meanCoP solver [28], our experimental findings demonstrate that our work can solve 179 problems that meanCoP cannot (Section 7).

Summary of Contributions.

1. We encode the existence of first-order connection calculi proofs as Boolean satisfiability problem, using connection tableaux (Section 3), a matrix representation (Section 4), and iterative deepening via unsat core refinement (Section 5). We show soundness and completeness of our encodings.
2. We explore optimisations to reduce redundancy and symmetry in the encodings (Section 6).
3. We discuss our implementation of the encodings in the new prototype solver UPCoP and evaluate it on the TPTP benchmark set (Section 7).

2   Preliminaries

We use standard syntax and semantics of classical first-order logic [56]. Logical objects such as terms t may be indexed: t^i_j. We assume the input problem has been negated and converted to conjunctive normal form (CNF) by a satisfiability-preserving transformation [4]. However, this transformation is optional in theory [10].

2.1   Satisfiability Solving

We assume familiarity with Boolean satisfiability (SAT) solving [14] and satisfiability modulo theories (SMT) [58]. In addition to the basic decision procedure for Boolean formulas, many SAT solvers support solving *under assumptions* and *unsatisfiable cores*. Solving under assumptions allows fixing some literals temporarily for the duration of a solving run: afterwards, the solver "forgets" them and their consequences. If the solver detects that the problem is unsatisfiable under assumptions, it may extract a subset of the assumptions used to derive inconsistency: the so-called "unsat core". Cores may not be *minimal*, so inconsistency can be derived with a strict subset of the core. Minimal cores can be generated at additional computational cost [21,40].

Some SAT and SMT solvers, including CaDiCaL [30] and Z3 [15], allow the user to intervene *during* search by a variety of means, often under the title "user propagation". Such mechanisms allow employing a solver to tackle a broad class of problems efficiently. For our purposes, we assume we can be notified when a SAT variable is assigned true or false, and respond by asserting additional constraints, potentially containing fresh SAT variables. We write $J_1, \ldots, J_n \Vdash F$ to represent that we added (*propagated*) the constraint $J_1 \wedge \ldots \wedge J_n \Rightarrow F$ for some formula F to the solver, given that the solver's current model satisfies all antecedent literals $J_1, \ldots, J_n$. This feature allows us to avoid eagerly generating an extensive set of all possible constraints and add only the currently relevant parts of the encoding. This kind of lazy generation is desirable in our case.

2.2   Connection Tableaux

Connection tableaux are essentially clausal tableaux [25] with the additional constraint that each clause added to a branch must have at least one literal *connected* to the current leaf literal [39]. *Two literals are connected if they have the same atom but opposite polarity: they are dual.* Recall that in the first-order case, clauses in the tableau have their variables renamed apart from any other, and a global substitution σ is applied to the entire tableau to connect literals (unification). The connection tableau calculus is not *confluent* and requires both backtracking and a fair enumeration of tableaux for completeness.

We say that two literals L, K *can* be connected and write $L \bowtie K$ if there exists some substitution ρ such that $\rho(L)$ is connected to $\rho(K)$. A subset of input clauses are considered potential roots of the tableau [39]: we assume these *start* clauses (conjectures) have been chosen in a way that at least one conjecture
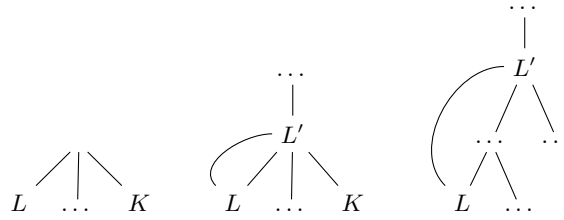
Fig. 1: Connection tableau rules, left-to-right: *start*, *extension*, and *reduction*. In *start* and *extension*, $L \vee \ldots \vee K$ is a freshly-renamed copy of a clause from the input problem. In *extension* and *reduction*, L is connected to L' using σ.

can be used for finding a proof. Equality is not handled by the basic connection calculus, and it is either axiomatised [44] or preprocessed away by some variation of Brand's modification [18]. We sometimes write C^k to distinguish the kth copy of the input clause C, indexing its variables as x^k.

Conventionally, three operations manipulate connection tableaux, shown in Figure 1. *Start* operations pick a start clause and add it at the root of the tableau. The chosen clause's literals are the initial leaves of the tableau. *Extension* operations add a copy of a clause from the input clause set below a leaf literal of the tableau, connecting at least one of the newly added clause's literals with the leaf. *Reduction* operations connect a leaf literal with another literal on the path from the literal toward the tableau's root. A branch of the tableau is *closed* in case the leaf of that branch is connected to some other literal. The whole tableau is closed, thus representing a proof if all of its branches are closed. In general, all these operations must be backtracked over to achieve completeness.

2.3   The Connection Method, Matrices, and Spanning Connections

Connection tableaux are an instance of the connection method [10]. While connection calculi are a rich topic with many facets, we are primarily interested in the following *matrix* representation. We consider matrices in *normal form* and thus define a matrix as a non-empty set of rigid clauses; that means, clauses containing rigid variables. The matrix can contain an arbitrary number of rigid copies of the same input clause, where variables are renamed apart, and a global substitution σ is applied. A *path* through a matrix is a set that contains exactly one literal from each copy of the clause in the matrix and is called *closed* if it contains at least one connected pair of literals, otherwise *open*. A *matrix proof*, or a matrix with a *spanning set of connections*, is a matrix for which there are no open paths.

Further, a matrix is *fully connected* with respect to a set of connections if each literal in the matrix is connected to at least one other literal of a different clause [37]. A matrix proof M is *minimal* if there is no proof using only a strict subset of M. Although we can require that there is at least one conjecture in the
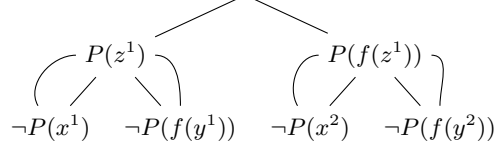
(a) Tableau proof with curved lines indicating connections. σ is computed such that, e.g., $\sigma(z^1) = f(y^1)$.
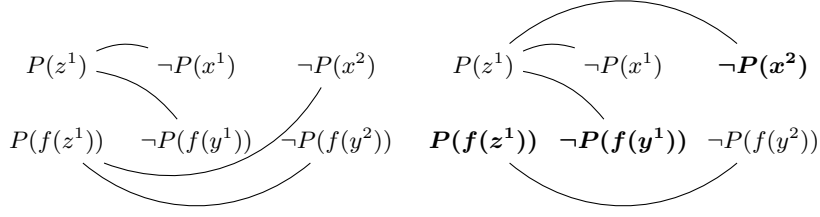
(b) Same proof in matrix form   (c) A matrix that does not represent a proof. An open path is shown in **bold**.

Fig. 2: Connection calculus derivations

matrix we could consider as a start clause, there is, in contrast to connection tableau proofs, no inherent tree structure in matrix proofs.

Example 1. To illustrate the two representations, consider the unsatisfiable set

$$\{ \ \forall x \forall y. \ \neg P(x) \vee \neg P(f(y)), \ \ \forall z. \ P(z) \vee P(f(z)) \ \}$$

and compare matrix and tableau refutations thereof in Figure 2a and 2b[3]. Figure 2c shows a fully connected matrix that is not a proof as it has an open path.

3   Encoding Connection Tableaux

We first encode the search for closed connection tableaux (Section 2.2) in a SAT solver. A closed connection tableau is explicitly constructed from a satisfying assignment. We encode that a literal L is a leaf of the tableau at path U using a SAT variable $\langle L; U \rangle$, where U is a set of literals labelling the nodes from L towards the root. L and U are used only to determine the corresponding variable. For example, $\neg P(x^2)$ in Figure 2a is represented by a variable $\langle \neg P(x^2); \{P(f(z^1))\} \rangle$. The substitution σ and unification of connected literals are handled with another family of variables we discuss later.

---

[3] the literals of each clause are written vertically in columns and the order of the clause copies does not matter

Connection tableau rules as SAT. We begin by asserting that at least one start clause S must be present in the tableau. Therefore, all literals $L \in S$ must be in the tableau at the root:

$$\bigvee_{S} \bigwedge_{L \in S} \langle L; \emptyset \rangle \tag{1}$$

The SAT solver is free to choose any start clause S, but all literals in the chosen S must be present at the root of the tableau. As the solver assigns variables $\langle L; U \rangle$ true we respond by propagating additional requirements. We demand that each literal has either an *extension* $E_{C,K}$ or a *reduction* R_K applied, in order to close the corresponding branch in the final tableau:

$$\langle L; U \rangle \Vdash \bigvee_{C,K} E_{C,K} \vee \bigvee_{K \in U} R_K \tag{2}$$

Each formula $E_{C,K}$ represents applying an extension operation at L using a fresh copy of a clause C containing a literal $K \bowtie L$, which yields

$$E_{C,K} := \left(\langle L \sim K \rangle \wedge \bigwedge_{\substack{K' \in C \\ K' \neq K}} \langle K'; \{L\} \cup U \rangle \right) \tag{3}$$

i.e. that if an extension step $E_{C,K}$ is taken, L and K are connected and other literals $K' \in C$ must be in the tableau with path $\{L\} \cup U$. We write $\langle L \sim K \rangle$ for the SAT variable representing that L and K are connected modulo σ. Similarly,

$$R_K := \langle L \sim K \rangle \tag{4}$$

where $K \bowtie L$ and $K \in U$. The steps $E_{C,K}$ and R_K are computed based only on the *possible* connection relation $\bowtie$: the current global substitution σ maintained by the SAT solver is ignored, otherwise relevant cases may be missed on backtracking. Iterative deepening may be applied as usual [45], for example, by offering no $E_{C,K}$ options if the path length $|U|$ exceeds the depth limit.

Example 2. Consider the input clauses from Example 1 and assume the solver assigned literal $\langle \neg P(x^2); \{P(f(z^1))\} \rangle$ true, so the solver considers literal $\neg P(x^2)$ to be within the tableau, below $P(f(z^1))$. The solver propagates

$$\langle \neg P(x^2); \{P(f(z^1))\} \rangle \Vdash$$
$$\left(\langle P(f(z^2)); \{P(f(z^1)), \neg P(x^2)\} \rangle \wedge \langle \neg P(x^2) \sim P(z^2) \rangle \right) \vee$$
$$\left(\langle P(z^2); \{P(f(z^1)), \neg P(x^2)\} \rangle \wedge \langle \neg P(x^2) \sim P(f(z^2)) \rangle \right) \vee$$
$$\langle \neg P(x^2) \sim P(f(z^1)) \rangle$$

where the first two disjuncts represent applications of the extension rule and the last an application of reduction.

Unification Constraints. Variables $\langle L \sim K \rangle$ constrain σ such that $\sigma(L)$ is connected to $\sigma(K)$. When the SAT solver assigns such a literal, we check whether this is consistent with the existing set of constraints. This can be done by applying a unification algorithm, perhaps using an efficient data structure such as the *variable trail* [39] to handle backtracking. We note in passing that algebraic datatype solvers [7], which are supported by many SMT solvers, implement a similar decision procedure. If the constraints are not satisfiable, we produce a *conflict clause* containing the reasons as Boolean assignments. For example, if we have $\langle L \sim K \rangle$, $\langle J \sim K \rangle$ and $\langle L \sim J \rangle$, but $\langle L \sim K \rangle \wedge \langle L \sim J \rangle$ is already unsatisfiable, we add the conflict

$$\neg\langle L \sim K \rangle \vee \neg\langle L \sim J \rangle \tag{5}$$

causing the solver to backtrack. This approach also allows a uniform treatment of refinements such as *regularity* based on *disequation constraints* [39].

SAT Encoding of Closed Connection Tableaux. We now have all the ingredients for our SAT encoding, which we denote by $\mathcal{E}_T$. By asserting that (i) a start clause must be present (1), (ii) each literal in the tableau must have a reduction or extension rule applied to it (2) and (iii) connections must have a consistent unifier, enforced by unification constraints, our encoding $\mathcal{E}_T$ is finished. Each propositional model of $\mathcal{E}_T$ represents a closed connection tableau.

Pathological Behaviour. Our SAT encoding $\mathcal{E}_T$ has drawbacks. Importantly, extension adds a *fresh instance* from the clause set to the tableau, and so the number of different SAT variables $\langle L; U \rangle$ grows rapidly. In turn, this means the resulting SAT problem has only limited propositional structure between variables that the solver can exploit. Search tends to degrade towards the kind of exhaustive enumeration of possible derivations that systems such as leanCoP [45] efficiently implement, but with the added overhead of a SAT solver.

4   Encoding Matrix Proofs

To avoid the previously described problems of $\mathcal{E}_T$, we encode matrix proofs (see Section 2.3). We denote our matrix-based encoding $\mathcal{E}_M$. Most search routines for spanning sets of connections presented in literature [10,45] restrict connections such that proofs are, or could be, simulated by a connection tableau proof [36]. In our $\mathcal{E}_M$ encoding, we allow arbitrary connections between clauses in the matrix. In this way, a single matrix proof can correspond to numerous tableau proofs [37]. In any event, our new representation $\mathcal{E}_M$ produces a combinatorial problem of finding connections between a set of clauses, which we argue is much more suitable for SAT solvers than $\mathcal{E}_T$.

4.1   Encoding Overview

We find a matrix proof with a given *resource limit* and split reasoning in:

1. We encode constraints for a fully-connected matrix (Section 4.2).

2. We constrain that the result has a set of spanning connections (Section 4.3).

We use the following result to motivate our encoding.

Theorem 1 (Fully Connected Matrix). *Suppose M is a minimal matrix proof with a spanning set of connections. Then M is fully connected.*

Proof. First, note that this is similar but not identical to Proposition 1 in Letz's work on matings pruning [37]. Suppose, towards contradiction, there is a literal $L \in C \in M$ that is not connected to any other K. Now consider the rest of the matrix $M' = M \setminus \{C\}$. Since M is minimal, there is an open path U through M', as otherwise M' would have a spanning set of connections. As L is not connected to any literal, $U \cup \{L\}$ is an open path for M, which conflicts with our assumption of M being a proof.

Theorem 1 allows us to restrict our work to fully-connected matrices. This restriction is a good approximation, as few fully-connected matrices do not have a spanning set of connections in practice (see Section 7). We use SAT variables of the form S_C to denote that clause C appears in the matrix, sometimes superscripted S_C^k to indicate selecting C^k, the kth copy of C. We call these S_C *selectors* and call C *selected* if S_C is assigned true. At least one of the start clauses C must be selected, cf. (1):

$$\bigvee S_C^1. \tag{6}$$

In Section 3, we apply iterative deepening on the maximum length of a branch. This kind of resource limit cannot be used here, as there is no obvious notion of a *branch*, so we must devise alternatives. We first apply iterative deepening on the matrix's number d of clauses. We immediately see that we need to introduce at most d selectors for each clause. As we always refer to the same copies, the solver can more easily learn sensible conflicts from $\mathcal{E}_M$ than from $\mathcal{E}_T$. We discuss a further enhanced encoding later in Section 5.

4.2   Fully Connected Matrices

By Theorem 1, we may constrain that each literal in the matrix must connect to at least one other literal. Similarly to (2), we respond to a selection S_C by propagating that each literal must be connected to some other literal in another clause in the matrix by enumerating all possible connections. This other clause could be selected or require selection, but there is no distinction between extension and reduction. Suppose C is selected. For each $L \in C$, we propagate

$$S_C \Vdash \bigvee_{D} \bigvee_{1 \leq k \leq d} \bigvee_{K \in D^k} S_D^k \wedge \langle L \sim K \rangle \tag{7}$$

where $K \bowtie L$ is a literal in the input clause D we connect to, and k indicates which copy D^k of that clause is used.

There are several possible options to enforce that at most d clauses are selected for the matrix. We suggest using pseudo-Boolean constraints [20] or a direct encoding [5,14,55] to constrain that "there are no more than d selector variables assigned". We can strengthen this to *exactly* d as we apply iterative deepening, so the less-than-d case was encountered already.

Example 3. Assume the situation

$$\{ \ S^1_{P(z)\vee P(f(z))}, \ \neg S^1_{\neg P(x)\vee\neg P(f(y))}, \ \langle P(z^1) \sim \neg P(x^1)\rangle, \ \langle P(z^1)\sim \neg P(f(y^1))\rangle \ \}$$

which corresponds to the situation in Figure 2c without the 3$^{\text{rd}}$ clause. Assuming that there are at most two copies of each clause, we would propagate that $P(f(z^1))$ needs to connect somehow. That means,

$$S^1_{P(z)\vee P(f(z))} \quad \Vdash \quad \left(S^1_{\neg P(x)\vee\neg P(f(y))} \wedge \langle P(f(z^1))\sim \neg P(x^1)\rangle\right) \vee$$
$$\left(S^1_{\neg P(x)\vee\neg P(f(y))} \wedge \langle P(f(z^1))\sim \neg P(f(y^1))\rangle\right) \vee$$
$$\left(S^2_{\neg P(x)\vee\neg P(f(y))} \wedge \langle P(f(z^1))\sim \neg P(x^2)\rangle\right) \vee$$
$$\left(S^2_{\neg P(x)\vee\neg P(f(y))} \wedge \langle P(f(z^1))\sim \neg P(f(y^2))\rangle\right)$$

These are all possible options given the available clauses and their copies.

4.3   Spanning Sets of Connections

Once we have a fully connected matrix, we check for open paths (see Section 2.3). If there are none, we are done and can use the resulting SAT model to output the proof, consisting of the selected clauses, the connections, and the global substitution. Suppose instead there is an open path U – given by the set of literals – through the matrix M. At least two literals along U must connect to make M a proof. Let $\bar{S}$ be the set of selectors assigned true. Propagating

$$\bar{S} \Vdash \bigvee_{\{L,K\}\subseteq U} \langle L \sim K\rangle \tag{8}$$

forces the solver to "fix" M, likely via backtracking, by requiring that U is not an open path.

Example 4. Continuing Example 3, assume the SAT solver further assigns

$$\{ \ S^2_{\neg P(x)\vee\neg P(f(y))}, \ \langle P(f(z^1))\sim \neg P(f(y^2))\rangle, \langle P(z^1)\sim \neg P(x^2)\rangle \ \}$$

true. The result is the fully connected matrix, which is shown in Figure 2c. We cannot propagate further clauses that are not already satisfied. However, we can find the open path $\{ \ P(f(z^1)), \ \neg P(f(y^1)), \neg P(x^2) \ \}$, which we exclude with

$$S^1_{P(z)\vee P(f(z))}, S^1_{\neg P(x)\vee\neg P(f(y))}, S^2_{\neg P(x)\vee\neg P(f(y))} \ \Vdash$$
$$\langle P(f(z^1))\sim \neg P(f(y^1))\rangle \vee \langle P(f(z^1))\sim \neg P(x^2)\rangle.$$

As neither $\langle P(f(z^1)) \sim \neg P(f(y^1))\rangle$ nor $\langle P(f(z^1)) \sim \neg P(x^2)\rangle$ results in a consistent unifier, the SAT solver is required to backtrack.

4.4   Correctness and Complexity of Matrix Encodings

Encoding $\mathcal{E}_M$ consists of (6), (7), (8), and constraints for the depth limit. Its models represent fully-connected matrix proofs. We show soundness, completeness, and termination for a given size d in $\mathcal{E}_M$, and describe the respective complexity class of $\mathcal{E}_M$.

Theorem 2 (Soundness). *A propositional model of $\mathcal{E}_M$ represents a matrix with a spanning set of connections.*

Proof. Whenever the SAT solver finds a propositional model, we first check that it represents a proof, adding constraints if not (Section 4.3).

Theorem 3 (Completeness). *If a matrix M together with a spanning set of connections exists, there is a propositional model of $\mathcal{E}_M$ at depth $d = |M|$.*

Proof. Such a matrix proof M can be represented by setting S_C^k true iff there are at least k copies of C in M. The spanning set of connections is represented by setting $L \sim K$ iff L is connected to K in the proof. This model of $\mathcal{E}_M$ and all its submodels are consistent modulo the semantics of $\sim$ and all possible instances of (7). Furthermore, the final model satisfies the depth constraints and contains at least one start clause. Also, we do not block the model with the final check in Section 4.3 as the set of connections are spanning.

Theorem 4 (Complexity Bound). *Solving our particular encoding $\mathcal{E}_M$ is in the complexity class Σ_2^P with respect to the input size and the matrix proof size.*

Proof. There are polynomially many SAT variables. To see this, let c be the number of clauses in the input, containing a total of l literals. We have at most $d \cdot c$ selectors S_C^k. We also have $O(d^2 l^2)$ possible connection literals $\langle L \sim K \rangle$. Hence, there are only polynomially many instantiations of (7). After adding in the worst case, all of them, the problem is in NP. We can non-deterministically guess an assignment for all polynomially many selectors and unification atoms. Checking the model can be done clearly in deterministic polynomial time. Checking whether a matrix represents a matrix proof is co-NP complete. It can be solved by a separate SAT solver, which checks if the matrix $\sigma(M)$ represented by the SAT model is satisfiable. As we can solve $\mathcal{E}_M$ in NP with a co-NP oracle, the problem of solving our encoding for some fixed limit d is in Σ_2^P.

As checking the satisfiability of a set of clauses over rigid variables is Σ_2^P-complete [33], our approach's complexity coincides with its theoretical bound.

Corollary 1 (Termination). *A run for solving $\mathcal{E}_M$ at fixed d terminates.*

5   Iterative Deepening via Unsat Core Refinement

A downside of our encoding $\mathcal{E}_M$, especially of its constraints from Section 4.2, is that we eagerly introduce and use selectors for clause instances that are not required. If there is more than one input clause and the matrix is of size d, not all clauses can have d copies in the matrix for arithmetic reasons. Therefore, creating d instances of each clause is overkill. This section addresses this challenge and improves iterative deepening via unsat cores, resulting in a refined encoding $\mathcal{E}_U$.

We use an abstraction-refinement [23] approach to approximate the number of copies required for each clause. This way, we avoid polluting the search space with likely unnecessary clause instances. Instead of a coarse global limit d, we estimate how many copies of each clause are required with a *multiplicity* μ [9]. Initially, we have $\mu(C) = 1$ for start clauses and $\mu(C) = 0$ otherwise. The multiplicity is monotonically increased based on the unsat core of the following encoding. We refine constraint (7) to

$$S_C \Vdash \bigvee_D \bigvee_{1 \le k \le \boldsymbol{\mu(D)+1}} \bigvee_{K \in D^k} S_D^k \wedge \langle L \sim K \rangle \tag{9}$$

as we have $\mu(D)$ copies of D. Note that k ranges up to $\mu(D) + 1$. We add temporary assertions[4] $\kappa_D := \neg S_D^{\mu(D)+1}$ so that the solver cannot select $D^{\mu(D)+1}$, but *can* report that finding a proof failed in part due to a lack of copies of D. We revise (8), as we can no longer assume that a fully connected matrix has exactly d clauses. A matrix can be fully connected, but the desired matrix proof may in fact be a strict *superset* of the current matrix. As (8) is now too strong, we weaken it to

$$\bar{S} \Vdash \left[\bigvee_{\{L,K\} \subseteq U} \langle L \sim K \rangle \right] \vee \bigvee_{L \in U} \boldsymbol{F_L} \tag{10}$$

where F_L is a formula indicating that L could also be connected to another literal in a clause *not yet in the matrix* and $\bar{S}$ a set of selectors as before in (8).

Example 5. Assume we have $\mu(P(z) \vee P(f(z))) = \mu(\neg P(x) \vee \neg P(f(y))) := 2$ and let us consider Example 3 again. In this propagation, we would have to additionally list the case that $S_{\neg P(x) \vee \neg P(f(y))}^3$ is true. The case $S_{P(z) \vee P(f(z))}^3$ is not needed, as there is no way to connect it anyway. As we assumed that $S_{\neg P(x) \vee \neg P(f(y))}^3$ is false, the SAT solver will not assign it true, but might report it in its unsat core in case there are feasible ways to proceed, given a higher depth limit. In Example 4, we would have to list the additional cases:

$$S_{P(z) \vee P(f(z))}^2 \wedge \langle \neg P(f(y^1)) \sim P(z^2) \rangle, \; S_{P(z) \vee P(f(z))}^2 \wedge \langle \neg P(f(y^1)) \sim P(f(z^2)) \rangle,$$

$$S_{P(z) \vee P(f(z))}^2 \wedge \langle \neg P(x^2) \sim P(z^2) \rangle, \; S_{P(z) \vee P(f(z))}^2 \wedge \langle \neg P(x^2) \sim P(f(z^2)) \rangle,$$

$$S_{P(z) \vee P(f(z))}^3, \; S_{\neg P(x) \vee \neg P(f(y))}^3$$

---

[4] named κ because it indicates that a clause needs more "κ-city"

As the respective clause selector is assumed false anyway, we do not need to introduce connectedness atoms for clause copies beyond the depth limit. Slight variations and optimisations of the encoding, however, would benefit from encoding those aspects as well. Further, listing both the second and third copies of clause $P(z) \vee P(f(z))$ shows one source of symmetry discussed in Section 6.

Whenever the SAT solver reports unsatisfiability, we retrieve the *unsat core* representing a potentially non-minimal subset of κ assertions sufficient to yield unsatisfiability. We may increase one or more $\mu(C)$ if the corresponding assertion occurs in the unsat core. However, to retain completeness, we need to ensure that we eventually increment the multiplicity of every clause appearing repeatedly in the unsat core: in other words, we require *fairness*. In case the core is empty, we can conclude that no proof exists. As a result, our SAT encoding $\mathcal{E}_U$ with improved iterative deepening is given by (6), (9), and (10).

Example 6. Consider the input problem

$$C := P(a) \qquad D := \forall x.\ \neg P(x) \vee P(f(x)) \qquad E := \forall y.\ \neg P(y)$$

with C as the start clause. κ_D will always be contained within the unsat core, no matter its multiplicity. However, a fair enumeration eventually includes κ_E.

Our improved encoding $\mathcal{E}_U$ remains sound and terminating by similar arguments to Theorems 2 and 4. Completeness requires an adjusted argument.

Theorem 5 (Completeness). *If a matrix M together with a spanning set of connections exists, there is a corresponding propositional model of $\mathcal{E}_U$.*

Proof. In addition to Theorem 3, we show that if there is a proof using M which our current μ does not permit, at least one relevant κ_C is contained in the unsat core. Fairness then ensures we will eventually find the proof. Consider a maximal subset $M' \subset M$ representable at μ.

1. If M' cannot be fully connected, it contains at least one literal L with no connections. Since M' is maximal and M can be fully connected, L should be connected to some literal in a clause D that is not yet in the matrix. This option is offered in (9), but fails because the respective κ_D assumption is forced false. κ_D is therefore in the unsat core.
2. If M' can be fully connected, we would have failed to close some open path U and propagated some instance of (10). Some $L \in U$ must connect to at least one literal of a clause not yet in M' by the right disjunct of (10). As M' is maximal, we can add no clauses, so the constraint fails because of the κ assumption.

A side effect of encoding $\mathcal{E}_U$ is that it also terminates on some non-theorems.

6   Redundancy Elimination

Restricting the SAT solver's search space is beneficial when solving the SAT encodings of Sections 3–5. In addition to standard techniques, such as tautology elimination [39], we propose some specialised redundancy eliminations.

6.1   Multiplicity Symmetry

Our encodings from Sections 3–5 contain *several symmetries* [1], which we now *avoid*, rather than *break* [51]. The first symmetry is that copies of clauses are interchangeable. Suppose we connect some literal L to literal K in a copy of C not yet in the matrix, and subsequently fail to find a proof in that direction. Nothing prevents the SAT solver from selecting another copy of C and failing for the same reasons. We avoid this by propagating

$$S_C^{i+1} \Vdash S_C^i, \tag{11}$$

enforcing that C^i is selected only if C^j with $j < i$ are selected.

6.2   Subsumption and Instance Symmetry

Saturation systems often delete a clause C because it is *subsumed* [47] by some more-general clause D. Dynamics in connection systems are somewhat different as new first-order clauses are not deduced, but we can profit by applying some subsumption. If two different clauses C and D are in the current matrix, we can enforce that neither becomes a subset of the other, modulo σ^5. This restriction preserves completeness, by Bibel's Lemma 6.8 [10].

An obvious extension of this idea is removing clauses from the matrix that are subsumed by other clauses from the input set. This fails, however.

Example 7. Consider the four input clauses

$$C := P(a) \quad D := Q(a) \quad E := \forall x.\ \neg P(x) \vee Q(x) \quad F := \forall y.\ \neg Q(y)$$

with C being the only start clause. There is a proof without subsumption via C, E, and finally F; this is the only minimal proof using C. However, putting E in the matrix with $\sigma(x) = a$ makes it be subsumed by D.

Subsumption in the usual sense of smaller clauses representing any usage of larger clauses fails. This failure occurs because we might lose the *reason to connect* a clause to our current matrix. Keeping larger clauses instead also does not work, as we might not be able to connect all literals of the larger clause. Nonetheless, we can motivate additional symmetry avoidance this way. Define an arbitrary total order $\prec$ on input clauses such that start clauses are the least elements. We assume that the order of each clause in the matrix is the same as the order of the clauses in the input set from which they are a copy.

Lemma 1 (Instance Symmetry). *Suppose there is a matrix proof M containing a clause D with $D \succ C$, and that there is a ρ such that $\rho(C) = \sigma(D)$. Then M with D exchanged for C also has a spanning set of connections.*

Proof. As all variables in C and D are fresh, we can adapt σ according to ρ. This way, C may be connected to the same literals as D. As $\rho(C)$ has the same literals as $\sigma(D)$, we neither add additional paths that must be closed, nor do we prevent other clauses connecting to C because we dropped the respective literal.

---

[5] we do not apply an additional substitution to either side

Corollary 2 (Instance Symmetry Completeness). *Pruning models containing such clauses D remains complete.*

6.3 Substitution Symmetry

Symmetry appears in substitutions applied to different copies of the same clause.

Example 8. Consider a literal in two copies of the same clause, $L[x]$ and $L[y]$. Assume that all attempts with $\sigma(x) = a$ and $\sigma(y) = b$ fail. Nothing prevents trying again with all connections "flipped" to the other clause and $\sigma(x) = b$ and $\sigma(y) = a$, introducing an exponential number of branches in the worst case.

We enforce an ordering on the substitutions applied to the *variables in copies of the same clause.* This ordering of terms should be stable under substitution and orient as many terms as possible, but need not have the subterm property and therefore may not be a reduction ordering [3]. We suggest the following order. Assume an arbitrary total ordering $\prec$ over function symbols. Define $f(\bar{t}) \prec g(\bar{s})$ iff (i) $f \prec g$ or (ii) $f = g$ and $\bar{t} \prec \bar{s}$. Sequences of terms $\bar{t} \prec \bar{s}$ are compared lexicographically. Let $\bar{x}$ be the variables occurring left-to-right in clause C. Given two instances C^i and C^j of the same clause with $i < j$, we may enforce that $\sigma(\bar{x}_i) \not\succeq \sigma(\bar{x}_j)$ to avoid symmetries over clause substitutions.

Lemma 2 (Spanning Order). *Suppose M has a spanning set of connections and contains two copies C^i and C^j of the same clause. Then there is a spanning set of connections that satisfies $\sigma(\bar{x}_i) \not\succeq \sigma(\bar{x}_j)$.*

Proof. If this condition does not already hold, we have $\sigma(\bar{x}_i) \succeq \sigma(\bar{x}_j)$. Duplicate clauses are already eliminated, so in fact $\sigma(\bar{x}_i) \succ \sigma(\bar{x}_j)$. Now "swap" C^i and C^j by exchanging their connections to obtain a new spanning set of connections and consistent substitution σ'. Necessarily, $\sigma'(\bar{x}_i) \prec \sigma'(\bar{x}_j)$.

By iterated application of Lemma 2, it is possible to "reorder" any spanning set of connections into another that respects the order.

Corollary 3 (Substitution Symmetry Completeness). *Enforcing an ordering on substitution of variables in copies of the same clause remains complete.*

7 Implementation and Experiments

Implementation. We implemented the encodings and optimisations of Sections 3–6 in our new prototype UPCoP[6]. UPCoP uses either the user-propagator of the CaDiCaL SAT solver [30] or of the Z3 SMT solver [15].

Due to our encoding via SAT variables, UPCoP can choose an arbitrary atom to be assigned in case of a decision. However, such an arbitrary atom

---

[6] at https://github.com/CEisenhofer/UPCoP

Solver	UPCoP$_{\mathsf{SAT}}$	UPCoP$_{\mathsf{SMT}}$	UPCoP$_{\mathsf{SAT}}$	UPCoP$_{\mathsf{SMT}}$	UPCoP$_{\mathsf{SAT}}$	UPCoP$_{\mathsf{SMT}}$	meanCoP
Enc.	$\mathcal{E}_M$		$\mathcal{E}_U$		$\mathcal{E}_H$		
Solved	928	855	1,152	1,055	1,272	1,264	1,972
Unique	27	20	109	93	105	76	551

Fig. 3: Experimental summary. UPCoP$_{\mathsf{SAT}}$ and UPCoP$_{\mathsf{SMT}}$ denote the UPCoP versions based on CaDiCaL (SAT) and Z3(SMT). "Unique" lists the number of problems solvable by UPCoP, but not meanCoP, and vice versa.

selection conflicts with the notion of goal-directedness. Consider, for example, the initial constraint (6) that at least one selector of a conjecture clause has to be taken. However, this constraint does not require that the solver start by choosing one of those clauses, but only that one start clause is selected in the final model. As a result, the solver might pick an arbitrary selector and derive a submatrix from it that is not connectable to any conjecture. A similar situation may occur when propagating all possible candidates in (7). We therefore force UPCoP always to pick one of the currently relevant options, rather than leaving the choice to the internal heuristics of UPCoP. Further, we make UPCoP to assign currently irrelevant selectors false, rather than true.

We note that UPCoP might spend significant reasoning time upon term equalities and orderings not required by any selected clause in the matrix. To overcome this issue, UPCoP associates every non-ground term uniquely to the clause copy it originates from. This way, we process (in)equalities reported by those non-ground terms only in case the respective selector is assigned true.

Experimental Setup. We evaluated UPCoP on the TPTP benchmark set [57] of first-order problems using an AMD EPYC 7502 clocked at roughly 1.8GHz, a limit of 16GB memory per run, and a 30s timeout. We considered all 6468 provable first-order problems of the TPTP 8.2.0 problem repository by translating them upfront to the SMT-LIB input format [6].

We evaluated UPCoP with three encodings, as follows. First, we used the encodings $\mathcal{E}_M$ and $\mathcal{E}_U$ of Sections 4–5, respectively. Further, we considered a hybrid approach $\mathcal{E}_H$ which combines $\mathcal{E}_M$ and $\mathcal{E}_U$ in a way such that the capacity of every (non-ground) clause is increased on each failure, but the set of clauses in the matrix is limited by the number of copies available, rather than by a strict upper bound. For each encoding, we separately evaluated UPCoP with user-propagators of CaDiCaL and Z3. Our experimental results thus report on six different settings of UPCoP, shown in Figure 3. In addition, we compared UPCoP with the default (complete) mode of meanCoP [28].

Experimental Analysis. Our results are summarised in Figure 3, with all versions of UPCoP solving altogether 1,601 problems. While UPCoP solves less problems than meanCoP, we emphasize that UPCoP solves 179 problems from all different kinds of categories that meanCoP cannot. There are many different reasons for this. There are mainly two reasons apart from the "non-determinism" in the choice of which literals to connect. The first one is that some matrix proofs found by UPCoP are in fact decently smaller than the respective tableau-

shaped proof of meanCoP. The second one is that the unsat core-based encodings indeed sometimes deduce that certain clauses are completely irrelevant, and consequently ignore them, resulting in a reasonable speed-up.

Counter-intuitively, UPCoP usually does not get stuck in eliminating non-spanning connections. Most cases in which UPCoP does not terminate are because UPCoP is too slow to find a fully-connected matrix due to the previously discussed points. As soon as a fully-connected matrix is found, it becomes spanning after only a few rejected matrices. Although the spanningness check is co-NP complete, we did not encounter cases where it took a significant time.

We note that $\mathcal{E}_M$ terminates quickly on inputs with small (potentially non-tree-shaped) proofs. $\mathcal{E}_U$ is best suited for large problems with redundant unconnectable clauses, as we will not generate further copies of the redundant clauses. $\mathcal{E}_H$ best pays off on problems whose proofs use most clauses from the input, but each of them only a few times.

Discussion. When analysing our experimental results, we note the following aspects for further improvements in UPCoP:

- The SAT/SMT solvers used by UPCoP repeatedly learn additional facts about the underlying proof systems, mostly using new Boolean variables. Hence, the solvers might decide to split on these Boolean variables. This can slow down proof search, as the solver might put a lot of effort into learning facts about partial proofs that cannot be extended to a proper proof. Further, some of these Boolean assignments represent partial proofs that a naive proof enumeration cannot even reach.
- Learning clauses coming from those unfruitful proof spaces also do not necessarily combine: for example, the solver can consider multiple submatrices simultaneously, even though these submatrices can never be combined. The assumption that conflict learning and SAT heuristics will guide the solver towards a more focused search has not been vindicated. Postponing the propagation of possible connections from submatrices not attached to the main matrix helps to reduce the number of Boolean variables and the overhead of propagating potentially unused constraints, but requires additional expensive book-keeping.
- Adjusting the SAT/SMT solvers' decision heuristic to prefer Boolean variables relevant for the current proof state might be beneficial [41]. Yet, such adjustments harm efficient heuristics for variable selection within the solver.
- The encoding $\mathcal{E}_T$ suffers because the conflicts mainly learned depend on the precise paths in the tableau. Similarly, conflicts learned about the Boolean variables in the matrix encodings $\mathcal{E}_M$ and $\mathcal{E}_U$ do not generalise well from one copy to the other in a way such that multiple copies of the same clause are considered in an isomorphic way numerous times.
- Additionally, SAT/SMT solvers incur non-negligible overhead from propagation loops for processing clauses. While simulating a single rule via SAT, a naive proof enumeration solver can process multiple steps within that time.

As shown and discussed before, one way of tackling most of those problems is to implement a custom variant of conflict learning that does not have the

discussed behaviours that are mostly inherent to modern SAT solvers. Based on these results, we build a further system hopCoP [49] that implements such a custom conflict learning engine for the tableaux form of connection calculus.

8   Related Work

Most first-order theorem provers employ a variety of ground reasoning techniques, predominantly SAT and SMT solvers. Here we must mention the family of instance-based methods [8]: grounding a set of first-order clauses in the hope that they become unsatisfiable, which can be employed with a dedicated calculus [34] or alongside an existing system [50,53]. In the other direction, SMT solvers often integrate quantifier instantiation into satisfiability routines [32,42]. Ground reasoning can also be used for many other combinatorial tasks in first-order theorem provers [52], such as keeping track of clause splitting [59], detecting subsumption [48], or determining if inferences are applicable [24]. MACE-style finite model builders [22] employ SAT solving to determine whether a set of clauses is satisfiable, assuming a fixed-size finite model, using symmetry-breaking [51].

Apart from our other approach [49] using a custom conflict learning engine and restricting ourselves to the direct encoding of proof objects, the ChewTPTP system in both its SAT [26] and SMT [17] incarnations is the closest existing approach to theorem proving via satisfiability solving. ChewTPTP encodes constraints for a closed connection tableau completely ahead of time, then passes the resulting constraints to a SAT or SMT solver. The higher-order systems Satallax [29] and its fork Lash [19] also reduce proof search to the propositional level. However, in contrast to UPCoP, when the Satallax's SAT solver reports unsatisfiability, a proof has been found. We previously published an early version of our ideas in a more general setting [27].

9   Conclusion

We encode first-order connection calculi as a propositional problem, improve our SAT encodings for matrix proofs, and guide iterative deepening using unsat cores. We introduce several optimisations to prune symmetries and eliminate unnecessary branches during proof search. Initial experiments with our new UPCoP solver show that our encoding can be used to solve certain problems, other connection solvers fail to solve. Further improvements may come with applying custom garbage-collection and variable selection heuristic strategies in UPCoP due to the discussed problems encountered during solving.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Aloul, F.A., Sakallah, K.A., Markov, I.L.: Efficient symmetry breaking for Boolean satisfiability. IEEE Trans. Computers **55**(5), 549–558 (2006). https://doi.org/10.1109/TC.2006.75
2. Andrews, P.B.: On Connections and Higher-Order Logic. J. Autom. Reason. **5**(3), 257–291 (1989). https://doi.org/10.1007/BF00248320, https://doi.org/10.1007/BF00248320
3. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge university press (1998)
4. Baaz, M., Egly, U., Leitsch, A.: Normal form transformations. In: Handbook of Automated Reasoning (in 2 volumes), pp. 273–333. Elsevier and MIT Press (2001). https://doi.org/10.1016/B978-044450813-3/50007-2
5. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of Boolean cardinality constraints. In: CP. LNCS, vol. 2833, pp. 108–122 (2003). https://doi.org/10.1007/978-3-540-45193-8_8
6. Barrett, C., Fontaine, P., Tinelli, C.: The satisfiability modulo theories library (SMT-LIB). www.SMT-LIB.org (2016)
7. Barrett, C.W., Shikanian, I., Tinelli, C.: An abstract decision procedure for a theory of inductive data types. J. Satisf. Boolean Model. Comput. **3**(1-2), 21–46 (2007). https://doi.org/10.3233/SAT190028
8. Baumgartner, P., Thorstensen, E.: Instance based methods – A brief overview. Künstliche Intell. **24**(1), 35–42 (2010). https://doi.org/10.1007/S13218-010-0002-X
9. Bibel, W.: Matings in matrices. Commun. ACM **26**(11), 844–852 (1983). https://doi.org/10.1145/182.183
10. Bibel, W.: Automated theorem proving. Artificial intelligence, Vieweg, 2., rev. ed. edn. (1987)
11. Bibel, W.: Comparison of proof methods. In: AReCCa. pp. 119–132 (2023), https://ceur-ws.org/Vol-3613/
12. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions. pp. 51–53. Department of Computer Science Report Series B, University of Helsinki (2020)
13. Biere, A., Froleyks, N., Wang, W.: CadiBack: Extracting Backbones with CaDiCaL. In: SAT. LIPIcs, vol. 271, pp. 3:1–3:12 (2023). https://doi.org/10.4230/LIPICS.SAT.2023.3
14. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336. IOS Press (2021). https://doi.org/10.3233/FAIA336
15. Bjørner, N.S., Eisenhofer, C., Kovács, L.: Satisfiability modulo custom theories in Z3. In: VMCAI. LNCS, vol. 13881, pp. 91–105 (2023). https://doi.org/10.1007/978-3-031-24950-1_5
16. Bonacina, M.P.: A taxonomy of theorem-proving strategies. In: Artificial Intelligence Today: Recent Trends and Developments, LNCS, vol. 1600, pp. 43–84. Springer (1999). https://doi.org/10.1007/3-540-48317-9_3

17. Bongio, J., Katrak, C., Lin, H., Lynch, C., McGregor, R.E.: Encoding first order proofs in SMT. In: SMT. ENTCS, vol. 198, pp. 71–84 (2007). https://doi.org/10.1016/J.ENTCS.2008.04.081

18. Brand, D.: Proving theorems with the modification method. SIAM J. Comput. **4**(4), 412–430 (1975). https://doi.org/10.1137/0204036

19. Brown, C.E., Kaliszyk, C.: Lash 1.0 (system description). CoRR **abs/2205.06640** (2022). https://doi.org/10.48550/ARXIV.2205.06640

20. Chai, D., Kuehlmann, A.: A fast pseudo-Boolean constraint solver. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **24**(3), 305–317 (2005). https://doi.org/10.1109/TCAD.2004.842808

21. Cimatti, A., Griggio, A., Sebastiani, R.: Computing small unsatisfiable cores in satisfiability modulo theories. J. Artif. Intell. Res. **40**, 701–728 (2011). https://doi.org/10.1613/JAIR.3196

22. Claessen, K., Sörensson, N.: New techniques that improve MACE-style finite model finding. In: Proceedings of the CADE-19 Workshop: Model Computation-Principles, Algorithms, Applications. pp. 11–27 (2003)

23. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003). https://doi.org/10.1145/876638.876643

24. Coutelier, R., Kovács, L., Rawson, M., Rath, J.: SAT-based subsumption resolution. In: CADE. LNCS, vol. 14132, pp. 190–206 (2023). https://doi.org/10.1007/978-3-031-38499-8_11

25. D'Agostino, M., Gabbay, D.M., Hähnle, R., Posegga, J.: Handbook of tableau methods. Springer Science & Business Media (2013)

26. Deshane, T., Hu, W., Jablonski, P., Lin, H., Lynch, C., McGregor, R.E.: Encoding first order proofs in SAT. In: CADE. LNCS, vol. 4603, pp. 476–491 (2007). https://doi.org/10.1007/978-3-540-73595-3_35

27. Eisenhofer, C., Kovács, L., Rawson, M.: Embedding the connection calculus in satisfiability modulo theories. In: AReCCa. pp. 54–63 (2023), https://ceur-ws.org/Vol-3613/

28. Färber, M.: A curiously effective backtracking strategy for connection tableaux. In: AReCCa. pp. 23–40 (2023), https://ceur-ws.org/Vol-3613/

29. Färber, M., Brown, C.E.: Internal guidance for satallax. In: IJCAR. LNCS, vol. 9706, pp. 349–361. Springer (2016). https://doi.org/10.1007/978-3-319-40229-1_24

30. Fazekas, K., Niemetz, A., Preiner, M., Kirchweger, M., Szeider, S., Biere, A.: IPASIR-UP: user propagators for CDCL. In: SAT. LIPIcs, vol. 271, pp. 8:1–8:13 (2023). https://doi.org/10.4230/LIPICS.SAT.2023.8

31. Galmiche, D.: Connection Methods in Linear Logic and Proof Nets Construction. Theoretical Computer Science **232**(1), 231–272 (2000). https://doi.org/https://doi.org/10.1016/S0304-3975(99)00176-0, https://www.sciencedirect.com/science/article/pii/S0304397599001760

32. Ge, Y., de Moura, L.M.: Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: CAV. LNCS, vol. 5643, pp. 306–320 (2009). https://doi.org/10.1007/978-3-642-02658-4_25

33. Goubault, J.: The complexity of resource-bounded first-order classical logic. In: STACS. pp. 59–70. LNCS (1994). https://doi.org/10.1007/3-540-57785-8_131

34. Korovin, K.: Inst-Gen - A modular approach to instantiation-based automated reasoning. In: Programming Logics - Essays in Memory of Harald Ganzinger. LNCS, vol. 7797, pp. 239–270 (2013). https://doi.org/10.1007/978-3-642-37651-1_10

35. Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: CAV. pp. 1–35. LNCS (2013). https://doi.org/10.1007/978-3-642-39799-8_1

36. Kreitz, C., Otten, J., Schmitt, S., Pientka, B.: Matrix-based constructive theorem proving. In: Intellectics and Computational Logic (to Wolfgang Bibel on the occasion of his 60th birthday). Applied Logic Series, vol. 19, pp. 189–205 (2000)

37. Letz, R.: Using matings for pruning connection tableaux. In: CADE. LNCS, vol. 1421, pp. 381–396 (1998). https://doi.org/10.1007/BFB0054273

38. Letz, R., Schumann, J., Bayerl, S., Bibel, W.: SETHEO: A high-performance theorem prover. J. Autom. Reason. **8**(2), 183–212 (1992). https://doi.org/10.1007/BF00244282

39. Letz, R., Stenz, G.: Model elimination and connection tableau procedures. In: Handbook of Automated Reasoning (in 2 volumes), pp. 2015–2114. Elsevier and MIT Press (2001). https://doi.org/10.1016/B978-044450813-3/50030-8

40. Lynce, I., Marques-Silva, J.: On computing minimum unsatisfiable cores. In: SAT (2004), http://www.satisfiability.org/SAT04/programme/110.pdf

41. de Moura, L., Bjørner, N.: Relevancy propagation. Technical Report MSR-TR-2007-140, Microsoft Research, Tech. Rep. (2007)

42. de Moura, L.M., Bjørner, N.S.: Efficient e-matching for SMT solvers. In: CADE. LNCS, vol. 4603, pp. 183–198 (2007). https://doi.org/10.1007/978-3-540-73595-3_13

43. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24

44. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Handbook of Automated Reasoning (in 2 volumes), pp. 371–443. Elsevier and MIT Press (2001). https://doi.org/10.1016/B978-044450813-3/50009-6

45. Otten, J.: leanCoP 2.0 and ileanCoP 1.2: High performance lean theorem proving in classical and intuitionistic logic (system descriptions). In: IJCAR. pp. 283–291. LNCS (2008). https://doi.org/10.1007/978-3-540-71070-7_23

46. Otten, J.: Implementing connection calculi for first-order modal logics. In: IWIL. EPiC Series in Computing, vol. 22, pp. 18–32. EasyChair (2012). https://doi.org/10.29007/82M9

47. Ramakrishnan, I.V., Sekar, R., Voronkov, A.: Term Indexing. In: Handbook of Automated Reasoning (in 2 volumes), pp. 1853–1964. Elsevier and MIT Press (2001). https://doi.org/10.1016/b978-044450813-3/50028-x

48. Rath, J., Biere, A., Kovács, L.: First-order subsumption via SAT solving. In: FMCAD. pp. 160–169 (2022). https://doi.org/10.34727/2022/ISBN.978-3-85448-053-2_22

49. Rawson, M., Eisenhofer, C., Kovács, L.: Constraint learning for non-confluent proof search. In: TABLEAUX. LNCS (2025). https://doi.org/to-appear, appears in the same TABLEAUX'25 inproceeding

50. Rawson, M., Reger, G.: Eliminating models during model elimination. In: TABLEAUX. LNCS, vol. 12842, pp. 250–265 (2021). https://doi.org/10.1007/978-3-030-86059-2_15

51. Reger, G., Riener, M., Suda, M.: Symmetry avoidance in MACE-style finite model finding. In: FroCoS. LNCS, vol. 11715, pp. 3–21 (2019). https://doi.org/10.1007/978-3-030-29007-8_1

52. Reger, G., Suda, M.: The uses of SAT solvers in Vampire. In: Vampire. EPiC Series in Computing, vol. 38, pp. 63–69 (2015). https://doi.org/10.29007/4W68

53. Schulz, S.: Light-weight integration of SAT solving into first-order reasoners – first experiments. Vampire pp. 9–19 (2017)

54. Schulz, S., Cruanes, S., Vukmirovic, P.: Faster, higher, stronger: E 2.3. In: CADE. pp. 495–507. LNCS (2019). https://doi.org/10.1007/978-3-030-29436-6_29
55. Sinz, C.: Towards an optimal CNF encoding of Boolean cardinality constraints. In: CP. LNCS, vol. 3709, pp. 827–831 (2005). https://doi.org/10.1007/11564751_73
56. Smullyan, R.M.: First-Order Logic. Springer (1968)
57. Sutcliffe, G.: The TPTP problem library and associated infrastructure. from CNF to TH0, TPTP v6.4.0. J. Autom. Reason. **59**(4), 483–502 (2017)
58. Tinelli, C.: A DPLL-Based Calculus for Ground Satisfiability Modulo Theories. In: JELIA. pp. 308–319 (2002). https://doi.org/10.1007/3-540-45757-7_26
59. Voronkov, A.: AVATAR: the architecture for first-order theorem provers. In: CAV. LNCS, vol. 8559, pp. 696–710 (2014). https://doi.org/10.1007/978-3-319-08867-9_46
60. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: CADE. LNCS, vol. 5663, pp. 140–145 (2009). https://doi.org/10.1007/978-3-642-02959-2_10