

Embedding the Connection Calculus in Satisfiability Modulo Theories

Clemens Eisenhofer*, Laura Kovács and Michael Rawson

TU Wien, Austria

Abstract

We investigate embedding a broad class of deduction systems in satisfiability solvers such as Z3. One such deduction system is the connection calculus. Using Z3's support for user-propagation, proofs in a user-specified calculus can be found automatically via Z3's internal satisfiability procedures. The approach places few constraints on the deduction system, yet allows for domain-specific optimisations if known. We discuss ramifications for proof search in the connection calculus.

Keywords

Connection Calculus, Sequent Calculus, SMT, User-Propagation, Deduction System

1. Introduction

Satisfiability Modulo Theories (SMT) is the problem of finding models for formulas over a predetermined set of first-order theories, such as integer/real arithmetic, arrays, and strings. SMT solvers are nearly always implemented by combining propositional reasoning with (mostly-separate) theory-specific decision procedures [1]. Adding support for a new theory in an existing SMT solver is not easy for end-users: the new theory could be axiomatised in the solver's input, but this typically requires universal quantification [2, 3] and is unlikely to be as efficient as a built-in theory reasoning engine. Alternatively, the solver could be modified directly to support the desired theory, but this demands significant, solver-specific knowledge from the user and incurs a maintenance overhead.

User-propagation [4, 5] is a recent development in satisfiability solving that avoids direct solver modification while maintaining efficient reasoning. End-users supply a *propagator* as a loadable module which interacts with the solver in response to solver actions. With the benefit of user-propagation, adding reasoning in new *classical* logics is usually straightforward, but *non-classical* logics are harder to support directly as they require a complete redesign of the SMT solver's internals, or inventing sophisticated embeddings to incorporate the intended non-classical semantics [6, 7].


AReCCa 2023: Automated Reasoning with Connection Calculi, 18 September 2023, Prague, Czech Republic

*Corresponding author.

✉ clemens.eisenhofer@tuwien.ac.at (C. Eisenhofer); laura.kovacs@tuwien.ac.at (L. Kovács); michael@rawsons.uk (M. Rawson)

ORCID 0000-0003-0339-1580 (C. Eisenhofer); 0000-0002-8299-2714 (L. Kovács); 0000-0001-7834-1567 (M. Rawson)

© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

Contributions. In contrast to model-based approaches that depend heavily on the semantics of a specific logic [8, 9, 10, 6, 11], in this paper we suggest a completely syntactic approach to proof, implemented via user-propagation. We search for *derivations* in a *calculus* for some logic by reasoning over the space of all possible derivations. In other words, *we advocate theorem proving in arbitrary calculi via SMT solving*. One such supported calculus, which is the focus of our paper, is the connection calculus. Our approach, while unusual, brings the following advantages in general and to connection calculus in particular:

1. The implementation burden with respect to SMT is significantly reduced, as the considered meta-logic is classical in nature: either some postulate is provable or not.
2. With very few restrictions, a large number of calculi can be supported in a relatively uniform manner.
3. Meta-logical properties, such as monotonicity or permutation of assumptions, can be generically handled and exploited during proving in order to improve efficiency.
4. Even logics with no known semantics can be supported. Moreover, logics for which decision procedures are not known (or are too complex) can be embedded.
5. Built-in theories can be supported within even non-classical logics, as long as theories may behave classically.
6. Ordinary propositional SAT proofs generated by SMT solvers can be replaced by a wide variety of proofs we can represent within our system.
7. In some calculi, we can even extract explicit counterexamples if no proof exists.

2. Preliminaries

We assume a given calculus has a finite number of *rules*. Each rule makes exactly one *conclusion* from a finite set of *premises*. For any given conclusion, there is a known finite, but possibly empty set of *applicable rules* that form the conclusion from their premises. A derivation of a certain conclusion is a rooted tree such as shown below:

$$\frac{\frac{\frac{A \quad B}{C} \quad \frac{D \quad E \quad F}{G}}{H} \quad I}{R}$$

where the root R concludes the theorem and leaves are *axioms*. Recall that axioms are rules with no premises. Deductions like the above include a variety of calculi, including (but not limited to) sequent calculi [12] such as that in Figure 1, the connection calculus [13] in Figure 2, or other deduction systems such as Hindley-Milner type inference [14].

2.1. Restrictions on Calculi

While the aforementioned deduction formalism is extremely general, we do have some requirements in order to support a calculus. We expect derivations to be as shown above: in particular, there must be a finite branching factor, both in the number of applicable rules and in the number of premises to rules. Infinite branching in a calculus can often be avoided by various kinds of indirection.

$AX-1 : \overline{F_1 \vdash F_1}$	$AX-2 : \overline{\perp \vdash}$	$AX-3 : \overline{\top \vdash}$
$\neg\text{-left} : \frac{\Gamma_1 \vdash A; \Delta_1}{\neg A; \Gamma_1 \vdash \Delta_1}$	$\neg\text{-right} : \frac{A; \Gamma_1 \vdash \Delta_1}{\Gamma_1 \vdash \neg A; \Delta_1}$	
$\wedge\text{-left} : \frac{F_1; F_2; \Gamma_1 \vdash \Delta_1}{F_1 \wedge F_2; \Gamma_1 \vdash \Delta_1}$	$\wedge\text{-right} : \frac{\Gamma_1 \vdash F_1; \Delta_1 \quad \Gamma_1 \vdash F_2; \Delta_1}{\Gamma_1 \vdash F_1 \wedge F_2; \Delta_1}$	
$\vee\text{-left} : \frac{F_1; \Gamma_1 \vdash \Delta_1 \quad F_2; \Gamma_1 \vdash \Delta_1}{F_1 \vee F_2; \Gamma_1 \vdash \Delta_1}$	$\vee\text{-right} : \frac{\Gamma_1 \vdash F_1; F_2; \Delta_1}{\Gamma_1 \vdash F_1 \vee F_2; \Delta_1}$	
$Perm\text{-left} : \frac{\Gamma_1; F_2; \Gamma_2; F_1; \Gamma_3 \vdash \Delta_1}{\Gamma_1; F_1; \Gamma_2; F_2; \Gamma_3 \vdash \Delta_1}$	$Perm\text{-right} : \frac{\Gamma_1 \vdash \Delta_1; F_2; \Delta_2; F_1; \Delta_3}{\Gamma_1 \vdash \Delta_1; F_1; \Delta_2; F_2; \Delta_3}$	
$Weak\text{-left} : \frac{\Gamma_1; \Gamma_2 \vdash \Delta_1}{\Gamma_1; F_1; \Gamma_2 \vdash \Delta_1}$	$Weak\text{-right} : \frac{\Gamma_1 \vdash \Delta_1; \Delta_2}{\Gamma_1 \vdash \Delta_1; F_1; \Delta_2}$	

Figure 1: Some propositional rules of the sequent calculus system LK [12]. Here, F_1, F_2 are formulas, whereas $\Gamma_1, \Gamma_2, \Gamma_3, \Delta_1, \Delta_2, \Delta_3$ denote possibly-empty sequences of formulas. $\Gamma_1; \dots; \Gamma_m \vdash \Delta_1; \dots; \Delta_n$ can be understood as “at least one Δ can be derived from all the Γ s”.

Our approach proceeds “upwards” from a goal via backwards chaining. Therefore, for any given conclusion, it must be possible to compute the set of applicable rules from the conclusion alone. Furthermore, each conclusion in the calculus is considered independently: note that this differs from systems like analytic tableaux [15] or the usual presentation of natural deduction [12], where context matters. Identical subgoals occurring at different locations in proof search can be assumed to have the same proof.

While soundness and completeness of a considered calculus is beneficial, neither property is essential: a satisfiable result may not be valid if the calculus is unsound, whereas an unsatisfiable result might not hold if the calculus is incomplete (as discussed later).

2.2. User-Propagation and a Theory of Goals

User-propagation as implemented in $Z3$ allows asserting additional constraints to the solver in response to solver decisions [4]. *Justifications* for the additional constraints are important such that the solver can detect where to backjump to when conflicts arise. As an example, if our theory contains a symmetric relation R and the solver decides $R(s, t)$, we can propagate $R(t, s)$, justified by $R(s, t)$, which we write $R(s, t) \Vdash R(t, s)$. We use this mechanism to implement our SMT solving approach via user-propagation.

Our theory consists of a “goal” sort and a provability predicate \mathcal{F} . We write $\mathcal{F}(G)$ to mean that “ G has a proof”, $\mathcal{F}(C, G)$ for “ G has a proof with immediate consequence C ”, and $\mathcal{F}(d, C, G)$ for “ G has a proof of depth at most d with immediate consequence C ”. Visually, $\mathcal{F}(d, C, G)$ represents the following piece of information:

$$\begin{array}{c}
\text{AXIOM} \\
\hline
\{\}; M; Path
\end{array}
\qquad
\begin{array}{c}
\text{START} \\
\hline
C; M; \{\} \\
\epsilon; M; \epsilon
\end{array}
\qquad
\begin{array}{c}
\text{REDUCTION} \\
\hline
C; M; Path \cup \{\neg L\} \\
C \cup \{L\}; M; Path \cup \{\neg L\}
\end{array}$$

$$\begin{array}{c}
\text{EXTENSION} \\
\hline
C' \setminus \{\neg L\}; M; Path \cup \{L\} \quad C; M; Path \\
\hline
C \cup \{L\}; M; Path
\end{array}$$

Figure 2: Propositional connection calculus as a deduction system, adapted from [16]. Here, C, C' are clauses in M and L is a literal. The matrix M is valid if there is a derivation for $\epsilon; M; \epsilon$.

$$\frac{\frac{\frac{\{\}; M; \{\neg B, \neg A\}}{\{\neg A\}; M; \{\neg B\}} \quad \frac{\{\}; M; \{\neg B\}}{\{\}; M; \{\}}}{\{\neg B\}; M; \{\}}}{\epsilon; M; \epsilon}$$

Figure 3: Connection proof for $M := \{A, \neg A \vee B, \neg B\}$.

$$\frac{\dots \frac{\dots [\leq d]}{G} \quad \dots [\leq d]}{C} \dots$$

3. Theorem Proving in Arbitrary Calculi via SMT Solving

We now describe our approach for syntactic theorem proving, in particular in the connection calculus, via user-propagation in SMT solving. While our approach can be applied to any SMT engine and deduction system, we discuss our work in the context of the Z3 SMT solver [17] and the sequent calculus LK .

We begin by asserting that the goal is provable, and then propagate applicable rules for this goal resulting in new expressions stating that some subgoals are provable. Exploiting the internal satisfiability procedure of Z3, we already have a kind of idiosyncratic proof search, as follows.

Example 1 (Proof Search by Propagating Applicable Rules)

Suppose we use system LK from Figure 1 and Z3 assigns $\mathcal{F}(A; B \vee C \vdash A)$ to true. We can then apply weakening on both the left and right sides of the sequent, permutation on the left, or the \vee -left rule, and therefore propagate

$$\begin{array}{lcl}
\mathcal{F}(A; B \vee C \vdash A) & \Vdash & \mathcal{F}(B \vee C \vdash A) & \vee \\
& & \mathcal{F}(A \vdash A) & \vee \\
& & \mathcal{F}(A; B \vee C \vdash) & \vee \\
& & \mathcal{F}(B \vee C; A \vdash A) & \vee
\end{array}$$

$$(\mathcal{F}(A; B \vdash A) \wedge \mathcal{F}(A; C \vdash A))$$

We leave it up to the solver to handle backtracking search, but clearly the second disjunct leads to a proof. Once Z3 assigns $\mathcal{F}(A \vdash A)$ to true, we can apply weakening or the axiom rule:

$$\begin{array}{l} \mathcal{F}(A \vdash A) \Vdash \mathcal{F}(\vdash A) \qquad \qquad \qquad \vee \\ \qquad \qquad \qquad \mathcal{F}(A \vdash) \qquad \qquad \qquad \qquad \qquad \vee \\ \qquad \qquad \qquad \top \end{array}$$

The last disjunct represents the lack of premises of the axiom rule, and therefore the whole propagation is a tautology. If the goal was $A; B \vee C \vdash A$, then Z3 can report a model satisfying $\mathcal{F}(A; B \vee C \vdash A) \wedge \mathcal{F}(A \vdash A)$, from which we can extract a proof in LK.

Example 2 (Connection-Driven Proof Search)

Similarly, we can simulate the connection calculus. Consider Figure 3. We start with $\mathcal{F}(\varepsilon; M; \varepsilon)$ and propagate all possible choices for the start clause

$$\begin{array}{l} \mathcal{F}(\varepsilon; M; \varepsilon) \Vdash \mathcal{F}(\{A\}; M; \varepsilon) \qquad \qquad \qquad \vee \\ \qquad \qquad \qquad \mathcal{F}(\{\neg A, B\}; M; \varepsilon) \qquad \qquad \qquad \vee \\ \qquad \qquad \qquad \mathcal{F}(\{\neg B\}; M; \varepsilon). \end{array}$$

Supposing the solver decides to assign $\mathcal{F}(\{\neg B\}; M; \varepsilon)$ to true, we can proceed by propagating

$$\mathcal{F}(\{\neg B\}; M; \varepsilon) \Vdash (\mathcal{F}(\{\neg A\}; M; \{\neg B\}) \wedge \mathcal{F}(\{\}; M; \{\})).$$

Note that adding a connection to $\neg A \vee B$ is the only feasible step. As with LK, we consider local goals like $\{\neg A, B\}; M; \varepsilon$ atomically and implement them as constants of our “goal” sort.

3.1. Preventing Cyclic Proofs

Unfortunately, the intuitive encoding from Example 1 is unsound for some calculi, as it permits cyclic derivations.

Example 3 (Cyclic Proofs)

Suppose we use the system LK from Figure 1 with the goal $A; B \vdash$, which is not provable in LK. From $\mathcal{F}(A; B \vdash)$, we propagate a disjunction containing $\mathcal{F}(B; A \vdash)$, and then propagate another disjunction containing $\mathcal{F}(A; B \vdash)$ once again. However, Z3 has already assigned $\mathcal{F}(A; B \vdash)$ true and hence reports a model, having discovered a cyclic “proof”.

To avoid soundness issues due to cyclic derivations, we must perform a cyclicity check. We introduce the binary form of \mathcal{F} , as it allows us to maintain during reasoning the transitive, asymmetric relation “ G has a proof containing an ancestor A ”. More precisely, we consider the relation graph of the binary relation induced by the propositional variables $\mathcal{F}(C, G)$. In case of a cycle, we eliminate it by propagating a conflict with respect to all propositional variables involved in the cycle.

Example 4 (Preventing a Cyclic Proof)

Considering Example 3, the chain of propagated atoms includes $\mathcal{F}(A; B \vdash, B; A \vdash)$ and subsequently $\mathcal{F}(B; A \vdash, A; B \vdash)$. However, we detect a cycle and prevent the cyclic “proof”, backtracking from this branch and attempting another. Since the goal is not a theorem in LK and the search space is finite, eventually the solver will indicate unsatisfiability.

3.2. Infinite Search Spaces

Many useful calculi have large, redundant, but ultimately finite search spaces. Here the procedure sketched for the propositional system LK is already complete. However, in the case of infinite search spaces a wrong decision will cause the solver to get lost in barren space, from which it may never return. Avoiding such scenarios is a challenging task.

Iterative deepening, as used in connection systems [18], may be employed here: we use the ternary form of \mathcal{F} to determine that there are no proofs of a certain size before trying larger sizes, perhaps re-using this information elsewhere as in failure caching [19]. Another option disables Z3’s support for relevancy propagation [20] and, assuming all atoms to be true by default, explores all branches simultaneously and fairly. It may also be possible to use complementary calculi to disprove a certain goal, causing a partial “restart” — see Section 4.5.

As well as merely infinite search spaces, there is a special case in which the search space has an infinite chain where only one rule is applicable, a well-known problem in tree search [21]. In this case, even unit propagation in the solver will never terminate. In practice, this happens relatively rarely, but solutions for infinite search spaces should take this into account.

4. Logic-Specific Optimization

While we envision our approach to be widely applicable without special-purpose modifications, there are some recurring themes among many calculi. In the sequel, we discuss our “box of tricks” to be applied whenever their prerequisite properties are met by the calculus.

4.1. Improving the Calculus

In some cases, the proof calculus is extremely ill-suited to automation. While the system LK as described initially by Gentzen [12] is not adequate for efficient, machine-supported theorem proving, a few well-known tweaks to the calculus can result in significantly better performance. Eliminating structural rules, treating each side of the sequent as a set, and strengthening the axiom rules to be applicable regardless of extraneous formulae on either side helps enormously. We also adapt our reasoning engine to reuse “goal” variables for elements that should be equivalent, such as $\mathcal{F}(A; A; B \vdash \perp) = \mathcal{F}(B; A \vdash \perp)$.

4.2. Acyclic Calculi

Notwithstanding the discussion in Section 3.1, some calculi either explicitly allow cyclic proofs, or more often cannot generate cycles by virtue of their rules. In either case, we can forgo the cycle-checking. For example, the system LK with incorporated structural optimizations as

described in Section 4.1 has this property and hence does not require tracking the ancestry of a derivation. The presented connection calculus also has this property, by a short inductive argument.

4.3. Monotonicity

Sequent calculi very often have the *monotonicity* meta-logical property: that is, if $A \vdash C$ then $A, B \vdash C, D$. This can be exploited to improve performance. Consider the solver already assigned $\mathcal{F}(A \vdash C)$ to true whereas $\mathcal{F}(A; B \vdash C; D)$ is assigned to false. By monotonicity, we already have a conflict, but the solver will not detect this on its own without lengthy search. We can also propagate monotonicity information. If we have $\mathcal{F}(\Gamma \vdash \Delta)$, we propagate all $\mathcal{F}(\Gamma' \vdash \Delta')$ such that $\Gamma \subseteq \Gamma'$ and $\Delta \subseteq \Delta'$. Naturally, we only propagate monotonicity information in case the respective atoms are already present in the search space, otherwise there would be infinitely many propagations.

An important special case is monotonicity with respect to the number of steps, which resembles failure caching. In case $\mathcal{F}(i, C, G)$ is true, we can also propagate $\mathcal{F}(j, C, G)$ if $j > i$.

4.4. Ordering

Another common problem in theorem proving is that there might be several (similar) ways to reach exactly the same proving state. Proof search can become highly symmetrical. To break this symmetry, we can order rule applications, cf. ordered resolution [22] or matings pruning [23].

Example 5 (Symmetric Rule Applications)

Consider the sequent $A_1 \wedge B_1, \dots, A_n \wedge B_n \vdash \perp$. There are $n!$ possible derivations using the \wedge -right rule, each failing in the same sequent $A_1, \dots, A_n, B_1, \dots, B_n \vdash \perp$. Nonetheless, the solver will fail to detect this symmetry, causing the solver to inspect all possible orderings.

In some cases, we may be able to prevent such unnecessary blow-ups by establishing a partial order on the rules applied by a variety of methods from theorem proving. The connection calculus, for instance, has complete “don’t-care non-determinism” with respect to which open goal is selected for the next connection, although of course it may be beneficial to choose one or another [24, 25]. The partial order should be as total as possible, but reachability must be preserved.

4.5. Complementary Calculi

Some logics have a useful *complementary* calculus that allows *disproving* statements in the logic. This duality allows a possible optimisation for logics with a complementary calculus. It may be possible to disprove a statement quickly, but exhausting all possible proofs is considerably harder or even impossible; equally, a statement may be proved quickly but disproving it can be difficult. Therefore, we can explore both the positive and the complementary calculi in a single proof attempt, exchanging information between the two.

4.6. Global Constraints

Note that the connection calculus from Figure 2 is the propositional version, rather than the arguably more interesting first-order variant. This is because the first-order variant requires that a global substitution satisfies a number of side conditions imposed by the application of rules (unification). Therefore, the first-order version of Figure 2 does not have independent goals in the manner that we require, as solving one subgoal in a certain way may prevent the solution of another. This property necessitates considerable backtracking or communication in connection and other free-variable systems [26, 27].

Such difficulties could be remedied by “rephrasing” the calculus once more, perhaps carrying around an explicit substitution in the manner of Algorithm W [28] and requiring a single, compound premise for the extension rule as a result. Now we can handle the calculus once more, but there would be no shared sub-goals for Z3 to exploit. For example, variants of splitting [13, 29] could provide us efficient remedies.

However, SMT solvers have no problem maintaining a set of global constraints and backtracking over decisions that affect them – it is arguably what they do best, in fact. Having rules propagate equations, such as $t_1 = t'_1, \dots, t_n = t'_n$ when unifying two literals $L(t_1, \dots, t_n)$ and $\neg L(t'_1, \dots, t'_n)$ is a convenient way out. An algebraic datatype [30] over the signature provides the required semantics, with suitably-fresh uninterpreted constants representing rigid variables.

5. Conclusions

We advocate theorem proving in arbitrary calculi, in particular in connection calculus, by SMT solving via user-propagators. Our work is inspired by the application of user-propagators to simulate analytic tableaux in SMT solving [31]. However, we do not build a model by applying rules to get assignments to subformulas, but consider *rules themselves* as the relevant objects in the model to construct derivations. This makes our approach more generic, although we cannot usually generate explicit models from successful SMT solving runs.

The cyclicity check proposed in our work is vaguely related to the foundedness check in answer set programming [32], where atoms are required not to be only justified by cycles in a set of implications [10]. A different encoding of the first-order connection calculus as a satisfiability problem was implemented in ChewTPTP [33].

Techniques from satisfiability solving such as conflict-driven clause learning or local search may have a clear interpretation at the calculus level. This may provide inspiration for new search routines in a particular calculus.

Acknowledgments

We acknowledge funding from the ERC Consolidator Grant ARTIST 101002685, the TU Wien SecInt Doctoral College, the FWF SFB project SpyCoDe F8504, and the WWTF ICT22-007 Grant ForSmart.

References

- [1] D. Kroening, O. Strichman, *Decision Procedures - An Algorithmic Point of View - Second Edition*, 2016. doi:10.1007/978-3-662-50497-0.
- [2] L. M. de Moura, N. S. Bjørner, Efficient e-matching for SMT solvers, in: *CADE*, Springer, 2007, pp. 183–198. doi:10.1007/978-3-540-73595-3_13.
- [3] Y. Ge, L. M. de Moura, Complete instantiation for quantified formulas in satisfiability modulo theories, in: *CAV*, 2009, pp. 306–320. doi:10.1007/978-3-642-02658-4_25.
- [4] N. S. Bjørner, C. Eisenhofer, L. Kovács, Satisfiability modulo custom theories in Z3, in: *VMCAI*, Springer, 2023, pp. 91–105. doi:10.1007/978-3-031-24950-1_5.
- [5] K. Fazekas, A. Niemetz, M. Preiner, M. Kirchweger, S. Szeider, A. Biere, IPASIR-UP: user propagators for CDCL, in: *26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy, volume 271 of LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 8:1–8:13. doi:10.4230/LIPIcs.SAT.2023.8.
- [6] K. Claessen, D. Rosén, SAT modulo intuitionistic implications, in: *LPAR*, Springer, 2015, pp. 622–637. doi:10.1007/978-3-662-48899-7_43.
- [7] C. Fiorentini, R. Goré, S. Graham-Lengrand, A proof-theoretic perspective on smt-solving for intuitionistic propositional logic, in: *TABLEAUX*, Springer, 2019, pp. 111–129. doi:10.1007/978-3-030-29026-9_7.
- [8] J. P. M. Silva, K. A. Sakallah, GRASP: A search algorithm for propositional satisfiability, *IEEE Trans. Computers* 48 (1999) 506–521. doi:10.1109/12.769433.
- [9] M. Schmidt-Schauß, G. Smolka, Attributive concept descriptions with complements, *Artif. Intell.* 48 (1991) 1–26. doi:10.1016/0004-3702(91)90078-X.
- [10] F. Lin, Y. Zhao, ASSAT: computing answer sets of a logic program by SAT solvers, *Artif. Intell.* 157 (2004) 115–137. doi:10.1016/j.artint.2004.04.004.
- [11] Y. Kesten, Z. Manna, H. McGuire, A. Pnueli, A decision algorithm for full propositional temporal logic, in: *CAV*, volume 697, Springer, 1993, pp. 97–109. doi:10.1007/3-540-56922-7_9.
- [12] G. Gentzen, Untersuchungen über das logische schließen. i., *Mathematische zeitschrift* 35 (1935) 176–210.
- [13] W. Bibel, *Automated theorem proving*, 2nd Edition, Artificial intelligence, Vieweg, 1987.
- [14] R. Hindley, et al., The principal type-scheme of an object in combinatory logic, *Transactions of the american mathematical society* 146 (1969) 29–60.
- [15] M. D’Agostino, D. M. Gabbay, R. Hähle, J. Posegga, *Handbook of tableau methods*, 2013. doi:10.1007/978-94-017-1754-0.
- [16] J. Otten, Restricting backtracking in connection calculi, *AI Communications* 23 (2010) 159–182.
- [17] L. M. de Moura, N. S. Bjørner, Z3: An Efficient SMT Solver, in: *TACAS*, Springer, 2008, pp. 337–340.
- [18] J. Otten, W. Bibel, leancop: lean connection-based theorem proving, *J. Symb. Comput.* 36 (2003) 139–161. doi:10.1016/S0747-7171(03)00037-3.
- [19] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, K. Mayr, Setheo and e-setheo-the cade-13 systems, *Journal of Automated Reasoning* 18 (1997) 237–246.

- [20] L. de Moura, N. Bjørner, Relevancy Propagation, Technical Report MSR-TR-2007-140, Microsoft Research, Tech. Rep. (2007). URL: <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2007-140.pdf>.
- [21] L. Orseau, L. Lelis, T. Lattimore, T. Weber, Single-agent policy tree search with guarantees, *Advances in Neural Information Processing Systems* 31 (2018).
- [22] L. Bachmair, H. Ganzinger, Resolution theorem proving., *Handbook of automated reasoning* 1 (2001).
- [23] R. Letz, Using matings for pruning connection tableaux, in: *International Conference on Automated Deduction*, Springer, 1998, pp. 381–396.
- [24] O. Ibens, R. Letz, Subgoal alternation in model elimination, in: *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, Springer, 1997, pp. 201–215.
- [25] G. C. Kertész, G. Papp, P. Szeredi, D. Varga, Z. Zombori, Ordering subgoals in a backward chaining prover (2021).
- [26] M. Färber, A curiously effective backtracking strategy for connection tableaux, *arXiv preprint arXiv:2106.13722* (2021).
- [27] J. Cailler, J. Rosain, D. Delahaye, S. Robillard, H. L. Bouziane, Goéland: A concurrent tableau-based theorem prover (system description), in: *International Joint Conference on Automated Reasoning*, Springer International Publishing Cham, 2022, pp. 359–368.
- [28] R. Milner, A theory of type polymorphism in programming, *Journal of computer and system sciences* 17 (1978) 348–375.
- [29] E. N. Haga, Complexity of variable splitting, Master’s thesis, University of Oslo, 2008.
- [30] N. S. Bjørner, Integrating decision procedures for temporal verification, Ph.D. thesis, Stanford University, USA, 1998. URL: <https://searchworks.stanford.edu/view/4077712>.
- [31] C. Eisenhofer, R. Alassaf, M. Rawson, L. Kovács, Non-classical logics in satisfiability modulo theories, in: *TABLEAUX*, Springer, 2023, pp. 24–36. doi:10.1007/978-3-031-43513-3_2.
- [32] T. Eiter, G. Ianni, T. Krennwallner, *Answer set programming: A primer*, Springer, 2009.
- [33] J. Bongio, C. Katrak, H. Lin, C. Lynch, R. E. McGregor, Encoding first order proofs in SMT, in: *SMT*, 2007, pp. 71–84. doi:10.1016/j.entcs.2008.04.081.