Lazy Paramodulation in Practice

Grzegorz Prusak[1], Cezary Kaliszyk[2,*]

[1]Near Inc., Zug, Switzerland
[2]University of Innsbruck, Austria

Abstract

Tableaux-based provers work well for certain classes of first-order formulas and are better suited for integration with machine learning techniques. However, the tableaux techniques developed so far perform way worse than saturation-based techniques on the first-order formulas with equality. In this paper, we present the first implementation of Lazy Paramodulation which allows applying the ordering constraints in connection tableaux proof search without exponential blowup of the number of clauses (characteristic for Brand's modifications). We implement the original Lazy Paramodulation calculus and propose a variant of the Brand's modification (called LP-modification), which is based on the same ideas as Lazy Paramodulation, avoids exponential blowup and is just a preprocessing step for the standard Connection Tableaux calculus. We demonstrate the impact of both the Lazy Paramodulation and LP-modification on the proof search on a benchmark of TPTP library problems.

Keywords

Connection Tableaux, Lazy Paramodulation, First-order logic with equality, TPTP

1. Introduction

Model elimination, a proof algorithm created by Loveland [**?**], was initially a resolution-based method operating on clauses of a special form. Later it has been reformulated in terms of connections between clauses, which were guiding the proof procedure [**? ? ?**]. In this form, the method is today referred to as Connection Tableaux.

Connection Tableaux are a powerful goal-directed refinement of the general clause tableaux. They are of practical and theoretical interest, since tableaux-based proofs are relatively easy to translate to human-readable proofs and because they can be applied to the analysis of open problems. Even a partially closed connection tableaux can give significant insight into a problem, by exposing useful lemmas or determining the core of the problem. It has been shown that Connection Tableaux is a well-suited framework for Machine Learning-based automated deduction [**?**], thanks to the fact that the number of possible actions at every step of the Connection Tableaux construction is mostly bounded, in contrast with the saturation-based proving where the number of pairs of clauses available e.g. in the resolution calculus often grows quadratically with the number of the inferences performed. Finally, the Connection

PAAR 2022 submission
*Corresponding author.
cezary.kaliszyk@uibk.ac.at (C. Kaliszyk)
http://cl-informatik.uibk.ac.at/cek/ (C. Kaliszyk)
0000-0002-8273-6059 (C. Kaliszyk)

Tableaux method developed for classical logic generalizes well to many non-classical logics: intuitionistic logic, linear logic, modal logics, higher-order logics, and others [**?** **?**].

The most successful technique for handling equality in automated theorem proving — paramodulation (especially in combination with completion which gives rise to superposition [**?**]) — although widely used in the saturation-based theorem provers (like E-prover [**?**]) is unfortunately not compatible with the Connection Tableaux methods. Instead, the competitive Connection Tableau provers today (like leanCoP [**?**]) apply trivial equality handling by appending the equality axioms to the problem. Another popular equality handling technique in the Connection Tableaux-based provers is the Brand's modification method [**?** **?** **?**] (implemented for example in SETHEO [**?**]), which (similarly to paramodulation) severely limits the search space explosion caused by hard-to-control applications of the transitivity and monotonicity equality axioms. What is however missing, is the property of paramodulation which allows it to "orient" the equations according to a reduction order, so that they can be applied only in one direction. This property has been experimentally shown to give a major advantage to the paramodulation method. To overcome this deficiency in the Connection Tableaux setup, a Lazy Paramodulation technique has been introduced [**?** **?**], based on general E-unification [**?**].

In this paper, we present our implementation, lpCoP, of the Lazy Paramodulation calculus by Paskevich [**?**] and derive a variant of Brand's modification (which we call "LP-modification") - a preprocessing algorithm, which allows for emulating Lazy Paramodulation calculus within the standard Connection Tableau calculus. Lazy Paramodulation attempts to utilize a reduction order in the Connection Tableaux setup, while, LP-modification is designed to avoid some of the deficiencies of the Lazy Paramodulation. We implement both a Lazy Paramodulation calculus prover and an LP-modification prover. We use the FOL problems with equality from the TPTP library [**?**] for an experimental evaluation. We compare lpCoP against the leanCoP prover (version 2.1) in 2 setups: core strategy setup[1] and the full strategy schedule.

1.1. Lazy Paramodulation vs Axiomatic Equality

It is straightforward to take an arbitrary FOL prover and add equality support to it by appending the equality axioms to the input formula. An example of a connection tableaux produced by applying this method to the CNF (Conjunctive Normal Form) formula

$$a_1 \approx a_2 \wedge \ldots \wedge a_{n-1} \approx a_n \wedge a_1 \not\approx a_n$$

may look as follows

$$\dfrac{\mathbf{a_1 \not\approx a_n}}{\dfrac{a_1 \approx a_n}{\bot} \quad \dfrac{a_1 \not\approx a_2}{\dfrac{\mathbf{a_1 \approx a_2}}{\bot}} \quad \dfrac{\dfrac{a_2 \not\approx a_n}{\dfrac{a_2 \approx a_n}{\bot} \quad \dfrac{a_2 \not\approx a_3}{\dfrac{\mathbf{a_2 \approx a_3}}{\bot}} \quad \dfrac{a_3 \not\approx a_n}{\dfrac{\ldots}{\dfrac{\mathbf{a_{n-1} \approx a_n}}{\bot}}}}}$$

[1]The first strategy in the leanCoP schedule [cut,comp(7)] applies restricted backtracking proof search until depth 7, after which it switches off the restriction to recover completeness

In this simple case, $n-2$ instances of the transitivity axiom have been used. There are exponentially many different Connection Tableaux of the same size for this formula, which differ only by the choice of the literals which have been extended with a transitivity axiom. In general, since most of the literals of the equality axioms are of the form $x = y$ or $x \neq y$, one may extend an arbitrary node of a Connection Tableaux with an arbitrarily large proof subtree consisting solely of the transitivity and/or monotonicity axioms. Also, many of such subtrees are equivalent. Lazy Paramodulation strongly restricts such constructions: one has to directly choose the clause with an equation literal that will be used to rewrite the subterm of the literal. For the example formula above the Lazy Paramodulation Connection Tableaux may look as follows (assuming that $a_1 > ... > a_n$ in the chosen reduction ordering, LPO or KBO in lpCoP).

$$\dfrac{\dfrac{\dfrac{\mathbf{a_1} \not\approx \mathbf{a_n}}{\mathbf{a_1} \approx \mathbf{a_2}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{a_2 \not\approx \hat{u}}{\mathbf{a_2} \approx \mathbf{a_3}}}{a_3 \not\approx \hat{u}}}{\cdots}}{a_n \not\approx \hat{u}}}{\bot \cdot (a_n = \hat{u})}}{a_n \not\approx \hat{u}}}{\bot \cdot (a_n = \hat{u})}$$

where $\hat{u}$ is a fresh variable and $s = t$ represents the unification of terms s and t. Lazy Paramodulation Connection Tableau for this formula is unique up to the choice of the first subterm to reduce (either a_1 or a_n).

Alternatively, to avoid explicit use of the equality axioms one can apply the Brand modification [**?**] to the problem instance. In this case, however, we obtain an exponential blowup of the number of clauses (every clause is converted to 2^n clauses, where n is the number of the positive equality literals in that clause), which is especially expensive for the Connection Tableaux methods (there are exponentially many clauses to choose from for an extension step, which just differ in the order of arguments in the equality literals). Lazy Paramodulation as defined in [**?**] is derived from a constrained variant of the Brand modification (CEE — Constrained Equality Elimination [**?**]), which avoids this blowup by incorporating the symmetry axiom into the inference rules. It defines a new set of inference rules (which are different than the ones of the standard Connection Tableaux), which has a side effect that optimizations known to be complete for the standard Connection Tableaux are not immediately applicable to the Lazy Paramodulation framework.

In this paper, we propose a yet another variant of the Brand modification, which we refer to as "LP-modification". It preserves the ideas behind Lazy Paramodulation, while being easier to implement and as such is also immediately compatible with the various optimization techniques of the standard Connection Tableaux. We compare the proposed method experimentally with the original Lazy Paramodulation, Axomatic Equality handling, and leanCoP core strategy. LP-modification can be thought of as a "middle ground" between CEE and Paskevich's Lazy Paramodulation [**?**], which on the one hand allows for handling the equality with delayed constraints, but on the other hand allows directly using all major known optimizations for the Connection Tableaux (like regularity, lemmas, etc.).

The remainder of the paper is organized as follows. In Section 2 we list the required preliminary material. In Section 3 we define our LP-modification, present motivation behind the design choices and prove its properties. In Section 4 we describe the implementation details of lpCoP (Connection Tableaux techniques used and data structures). In Section 5 we describe our experimental setup and present the results of the experiment.

2. Preliminaries

We work in the first-order logic with equality in CNF. We use the same notation as Paskevich [**?**] for clauses, literals, predicates, terms, variables, inference rules, constraints, constrained clauses and Connection Tableaux. Instead of the "pseudoequality" predicate $\simeq$, we use a "reduction ordering" predicate $\rightarrow$, to underline its relation to the reduction ordering constraints. We use $\mathrm{Var}\,(s)$ to denote the set of variables occurring in a term s. We use $Lit(\mathcal{S})$ to denote the set of literals occurring in the set of clauses $\mathcal{S}$. For the full definition of Connection Tableaux (CT), refer to [**?**], the full definition of Lazy Paramodulation (LP) Connection Tableaux can be found in Paskevich [**?**].

A (constrained) tableaux is a finite tree. The root of the tree contains the initial clauses to be refuted. Non-root nodes contain literals (in constrained tableaux the literals in the nodes are accompanied by constraints). A branch that contains false (denoted $\bot$) is considered closed. A tableaux is closed, when each branch is closed (in the constraint setting, additionally the overall set of constraints is satisfiable). Inference steps expand some branch in the tableaux, adding new leaves under the given leaf node.

Standard connection-tableaux, apart from the start rule that sets up the root node with the initial clauses in the problem, additionally includes an *expansion* rule unifying one of the newly attached literals with the complement of the literal to solve (this unification can be considered a constraint) and the *reduction* rule, which connects a literal on the current path with the complement of the literal to solve.

Adding paramodulation to connection tableaux directly, results in an incomplete calculus [**?**]. Completeness can be recovered by allowing paramodulation into variables, which results in a quite inefficient calculus. Lazy paramodulation aims to solve this inefficiency by postponing the unification (and keeping it as a constraint) in the cases where it is required for completeness. In particular, the lazy paramodulation inferences are only performed in expansion steps (paramodulation and reduction/termination do not postpone unification). Furthermore, unifications are only postponed when two non-variable subterms start with the same functional symbol. Finally, ordering constraints reduce the number of possible paramodulations. For details see [**?**].

For the definition of the Constrained Equality Elimination (CEE), refer to [**?**]. CEE is a variant of the Brand modification, which takes a set of unconstrained clauses $\mathcal{S}$ as an input, and returns an equisatisfiable set of constrained clauses.

3. LP-modification: Lazy Paramodulation calculus revisited

In this section, we propose an alternative, LP-modification, which is a transformation taking a set of unconstrained clauses as an input, and producing a set of **constrained** clauses. The

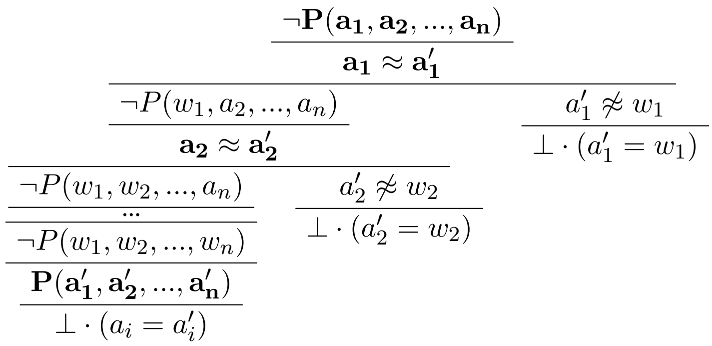
output will be equisatisfiable with the input clause set together with the equality axioms. It is based on the same principles as Lazy Paramodulation but avoids deficiencies described below, and admits an easier implementation, because LP-modification is a preprocessing step for a more standard Connection Tableau calculus with constraints.

Even starting with a single conjecture clause, one can find multiple proofs. To improve the performance of the prover, one should prune such redundant proofs from the search space. The tableaux-based provers use backtracking to traverse the search space. The state of such backtracking is a partial proof, therefore the prover should try to avoid processing partial proofs equivalent to the ones that it already processed.

In the Lazy Paramodulation calculus, as defined in [**?**], the subterms of a literal on a single branch can be paramodulated out of it in an arbitrary order. For example, for the CNF formula

$$\neg P(a_1, a_2, ..., a_n) \wedge P(a'_1, a'_2, ..., a'_n) \wedge a_1 \approx a'_1 \wedge a_2 \approx a'_2 \wedge ... \wedge a_n \approx a'_n$$

there are $n!$ closed LP Connection Tableaux which just differ by the order in which a_i are paramodulated out. One of the possibilities is as follows:



CEE on the other hand, suffers from an exponential clause blowup (see [**?**]) because for every unconstrained input clause it produces 2^n constrained clauses, where n is the number of distinct positive equality literals in the input clause. We propose a combination of CEE and LP (more precisely: a variant of CEE which allows for emulating LP in the standard Connection Tableaux calculus), called LP-modification which has the following properties:

- the output of LP-modification is a set of constrained clauses equisatisfiable with the original clause set with equality axioms (no new inference rules needed, however, constraints restrict inferences).
- Paramodulation of the independent subterms is emulated on the separate branches of a Connection Tableaux.
- Paramodulation into and under variables is not allowed (basicness property)
- For a flat clause (all non-variable subterms are arguments of the $\approx$ predicate) set $\mathcal{S}$, LP-modification generates $O(n)$ extra clauses, introduces $O(n)$ new literals and creates $O(n^2)$ new connections, where n is the number of equality literals in $\mathcal{S}$ (no blowup).

We will now define the steps of LP-modification. The reader may notice that LP-modification is very similar to CEE, with only "Transitivity and Symmetry elimination" step redefined. We

define a normal form of a clause C (wrt a set of transformation rules) to be the result of an application of any maximal chain of transformations to C. It is easy to observe that all the rule sets below uniquely define a normal form for an arbitrary clause C (up to variable renaming).

Flattening (aka E-modification). We say that a clause is flat iff all its non-variable subterms are arguments of the $\approx$ predicate. It is known that dropping monotonicity axioms of equality does not change the satisfiability of a set of flat clauses [**?**]. In this step we convert $\mathcal{S}$ to an equisatisfiable set of flat clauses. Flattening transformation rules are defined as follows:

$$\frac{P(\vec{s}[p]) \vee C}{P(\vec{s}[\hat{u}]) \vee p \not\approx \vec{\hat{u}} \vee C} \qquad \frac{\neg P(\vec{s}[p]) \vee C}{\neg P(\vec{s}[\hat{u}]) \vee p \not\approx \vec{\hat{u}} \vee C}$$

$$\frac{f(\vec{s}[p]) \approx t \vee C}{f(\vec{s}[\hat{u}]) \approx t \vee p \not\approx \vec{\hat{u}} \vee C} \qquad \frac{f(\vec{s}[p]) \not\approx t \vee C}{f(\vec{s}[\hat{u}]) \not\approx t \vee p \not\approx \vec{\hat{u}} \vee C}$$

where C is an arbitrary clause, P is an arbitrary predicate different than $\approx$, $\hat{u}$ is a fresh variable, p is a non-variable term, $s[p]$ is a non-flat subterm, $\not\approx \vec{s}$ is a set of $\not\approx$ literals in a clause, t is an arbitrary term and $\vec{s}$ is an arbitrary sequence of terms. For every clause $C \in \mathcal{S}$ we define $flat(C)$ to be the normal form of C wrt the flattening transformation rules (the normal form with respect to the flattening is unique, modulo order and naming). Finally, we define $flat(\mathcal{S}) := \{flat(C) : C \in \mathcal{S}\}$.

Transitivity and symmetry elimination in LP-modification. This step consists of replacing the $\approx$ predicate with

- "reduction ordering" predicate $\rightarrow$. The intended meaning of this predicate is: $t_1 \rightarrow t_2$ means that t_1 equals t_2 while at the same time $t_1 \succ t_2$ under the reduction order; and
- a family of new predicates $X_{s,t}$, indexed by an arbitrary pair of terms s and t (indicating that s and t are equal, but without specifying which one is smaller in the reduction ordering).

We define the following transformation rules:

$$\frac{s \not\approx t \vee C}{s \not\rightarrow x \vee t \not\rightarrow x \vee C} \qquad \frac{s \approx t \vee C}{X_{s,t}(\overrightarrow{Var(s)}, \overrightarrow{Var(t)}) \vee C}$$

for an arbitrary clause C, arbitrary terms s, t, and fresh new variable x. We define $ST(C)$ to be the normal form of C wrt these rules. We also define 2 families of clauses (indexed by arbitrary terms s and t)

$$\alpha_{s,t} := \neg X_{s,t}(\overrightarrow{Var(s)}, \overrightarrow{Var(t)}) \vee s \rightarrow x \vee t \not\rightarrow x$$

$$\beta_{s,t} := \neg X_{s,t}(\overrightarrow{Var(s)}, \overrightarrow{Var(t)}) \vee s \not\rightarrow x \vee t \rightarrow x$$

Then we define ST on the sets of **flat** clauses:

$$ST(\mathcal{S}) := \{ST(C) : C \in \mathcal{S}\} \cup \{\alpha_{s,t}, \beta_{s,t} : s \approx t \in Lit(\mathcal{S})\}$$

Exceptionally, in this definition we treat $s \approx t$ as an **ordered** pair — for every literal $s \approx t$ in $\mathcal{S}$ we generate $\alpha_{s,t}, \beta_{s,t}$ but not $\alpha_{t,s}, \beta_{t,s}$.

Constraints generation. So far the transformations presented operated on the unconstrained clauses (no constraints were associated with the literals). In this step explicit constraints are generated as opposed to the $\rightarrow$ predicate. This step is what distinguishes LP-modification (and CEE) from the basic Brand modification: we associate the ordering constraints with the $\rightarrow$ literals which truncates the possible search space, while preserving completeness. The transformation rules of this step are defined as follows:

$$\frac{y \not\rightarrow x \vee C}{(\bot, y = x) \vee C} \qquad \frac{p \not\rightarrow x \vee C}{(p \not\rightarrow x, p \geq x) \vee C} \qquad \frac{p \rightarrow x \vee C}{(p \rightarrow x, p > x) \vee C}$$

where x, y are variables, p is a non-variable term, C is an arbitrary clause. We define $constr(\mathcal{S})$ to be the set of normal forms of clauses in $\mathcal{S}$ wrt the rules above.

3.1. Definition of LP-Modification

We define the LP-modification of a set of unconstrained clauses $\mathcal{S}$ as

$$LPmod(\mathcal{S}) := constr(ST(flat(\mathcal{S}))) \cup \{x \rightarrow x\}$$

where x is a variable.

Example: $LPmod(\mathcal{S})$ and $CEE(\mathcal{S})$ for the example formula $\mathcal{S}$:

$$\mathcal{S} = \neg P(a_1, a_2, ..., a_n) \wedge P(a'_1, a'_2, ..., a'_n) \wedge a_1 \approx a'_1 \wedge a_2 \approx a'_2 \wedge ... \wedge a_n \approx a'_n$$

$\boldsymbol{LPmod(\mathcal{S})}$	$\boldsymbol{CEE(\mathcal{S})}$
$\neg P(x_1, x_2, ..., x_n) \vee a_1 \not\rightarrow x_1 \vee ... \vee a_n \not\rightarrow x_n$	$\neg P(x_1, x_2, ..., x_n) \vee a_1 \not\simeq x_1 \vee ... \vee a_n \not\simeq x_n$
$P(x_1, x_2, ..., x_n) \vee a'_1 \not\rightarrow x_1 \vee ... \vee a'_n \not\rightarrow x_n$	$P(x_1, x_2, ..., x_n) \vee a'_1 \not\simeq x_1 \vee ... \vee a'_n \not\simeq x_n$
$\neg X_{a_i, a'_i} \vee (a_i \rightarrow w, a_i > w) \vee (a'_i \not\rightarrow w, a'_i \geq w)$	$(a_i \simeq w, a_i > w) \vee (a'_i \not\simeq w, a'_i \geq w)$
$\neg X_{a_i, a'_i} \vee (a'_i \rightarrow w, a'_i > w) \vee (a_i \not\rightarrow w, a_i \geq w)$	$(a'_i \simeq w, a'_i > w) \vee (a_i \not\simeq w, a_i \geq w)$
X_{a_i, a'_i}	
$x \rightarrow x$	$x \simeq x$

A Connection Tableaux for $LPmod(\mathcal{S})$ (constraints have been omitted):

$$\dfrac{\dfrac{\dfrac{\neg P(x_1, .., x_n)}{P(x_1, ..., x_n)}{\bot} \quad \dfrac{\dfrac{a'_1 \not\rightarrow x_1}{a'_1 \rightarrow a'_1}}{\bot} \quad ...}{\dfrac{a_1 \not\rightarrow x_1}{\dfrac{\neg X_{a_1, a'_1}}{X_{a_1, a'_1}}{\bot} \quad \dfrac{a_1 \rightarrow x_1}{\bot} \quad \dfrac{a'_1 \not\rightarrow x_1}{a'_1 \rightarrow a'_1}{\bot}} \quad ...}$$

3.2. Properties of LP-Modification

We conjecture that LP-modification is sound and complete. However, full proofs of this are left as future work.

Conjecture 1. *$flat(\mathcal{S})$ with equality axioms is satisfiable $\Rightarrow$ $LPmod(\mathcal{S})$ is satisfiable $\Rightarrow$ $CEE(\mathcal{S})$ is satisfiable*

Justification for the first implication. Take a model of $flat(\mathcal{S})$ with equality axioms and let $I : Atoms \to Bool$ be the interpretation function. We can extend interpretation of that model to atoms $X_{s,t}(\vec{v}, \vec{w})$ and $x \to y$ for all terms s, t and all elements $x, y, \vec{v}, \vec{w}$ of the model as follows

$$I(X_{s,t}(\vec{v}, \vec{w})) := I(s[\vec{v}] \approx t[\vec{w}]) \qquad I(x \to y) := I(x \approx y)$$

where $s[\vec{v}]$ denotes s with its variables substituted by $\vec{v}$ ($t[\vec{w}]$ analogically).

Then we can see that $LPmod(\mathcal{S})$ is true in the extended interpretation:

- The instances of $x \to x$ are true, since the corresponding instances of the reflexivity axiom are true: $I(x \to x) = I(x \approx x) = \top$
- The instances of $\alpha_{s,t}$ and $\beta_{s,t}$ are true, since the corresponding instances of the transitivity axiom are true:

$$I(\alpha_{s,t}) = I(s \not\approx t \vee s \approx x \vee t \not\approx x) = \top$$

$$I(\beta_{s,t}) = I(s \not\approx t \vee s \not\approx x \vee t \approx x) = \top$$

- The instances of $ST(C)$ for $C \in flat(\mathcal{S})$ are true, since the corresponding instances of C are true.

Justification for the second implication. It is possible to infer from $LPmod(\mathcal{S})$ an arbitrary clause of $CEE(\mathcal{S})$ derived from $C \in flat(\mathcal{S})$ by resolving $\alpha_{s,t}$ or $\beta_{s,t}$ clauses with C and adding constraints to it. Therefore $LPmod(\mathcal{S}) \Rightarrow CEE(\mathcal{S})$ so every model of $LPmod(\mathcal{S})$ is a model of $CEE(\mathcal{S})$.

We additionally conjecture that LP-modification is able to avoid the exponential blowup.

Conjecture 2 (LP-modification output size bounds)**.** *For a flat clause set $\mathcal{S}$, LP-modification generates $O(n)$ extra clauses, introduces $O(n)$ new literals total and creates $O(n^2)$ new connections, where n is the number of equality literals in $\mathcal{S}$.*

As $\mathcal{S}$ is assumed to be already flat, the flattening step is trivial and the constraint generation step does not create any new clauses, literals or connections. Transitivity and symmetry elimination replaces every $s \not\approx t$ literal with at most 2 literals, replaces every $s \approx t$ literal with a single $X_{s,t}$ literal and add adds 2 clauses ($\alpha_{s,t}$,$\beta_{s,t}$) with at most 6 literals total. Each literal $X_{s,t}$ has only 2 connections (with one literal from $\alpha_{s,t}$ and one from $\beta_{s,t}$). Each $\approx$ literal has its arguments separated by LP-modification and may correspond to up to 4 positive and negative $\to$ literals, so in the worst case up to $O(n^2)$ new connections may arise. Connections for the remaining predicates stay the same.

4. Implementation

Lazy Paramodulation and LP-modification operate on the constrained clauses, and up to the authors' knowledge, there exists no Connection Tableau prover which accepts clauses together with constraints corresponding to an order as an input (ordering and constraints are usually an internal implementation detail of the prover). Therefore to evaluate LP-modification in practice we had to engineer a custom prover. The original Lazy Paramodulation defines an entirely

different calculus than the standard connection tableau, so it required developing a dedicated prover anyway.

We have implemented (in C++) a customizable Connection Tableaux prover, which takes as an input a FOL formula in a CNF form and outputs a proof of the formula in a form of an unsatisfiable set of ground instances of the input clauses (SAT-solvers can be used to check such proofs efficiently).

It can be also used to just compute an LP-modification of the input. The proof search engine (accepting a set of constrained clauses) is a separate library module and can be reused. The prover is configurable in the following dimensions:

- the choice of the transformation that converts the initial clause set into a (possibly) constrained clause set (LP-modification, appending equality axioms or none)
- the choice of the reduction ordering (KBO [**?**] or LPO [**?**]). In comparison with the state-of-the-art superposition provers, lpCoP lacks advanced heuristics for selecting the ordering weights, resorting to constant weights currently. The precedence of the functors is configurable, defaults to the order in which functors occur in the input.
- the choice of the Connection Tableau calculus (either standard Connection Tableau, or Lazy Paramodulation Connection Tableau)
- the choice of the search strategy. The implementation allows for iterative deepening over branch length bound (including the leanCoP bounds), and MESON bound distribution [**?** **?**]. Additionally, restricted backtracking [**?**] can be activated. Following Otten's terminology, we refer to restricted backtracking as "cut" in the remainder of the paper.

The TPTP syntax allows labeling certain clauses as conjectures. This is normally used to indicate that at least one of the conjectures is required to prove unsatisfiability. However, as certain problems may be unsatisfiable due to contradictory axioms, for completeness our implementation can use all all-negative clauses as candidates for the root clause, rather than just the conjecture clauses (we do not use that option in lpCoP in the evaluations, however it is a part of the leanCoP schedule).

We have developed a basic index for selecting clauses for extension steps, similar to the Prolog built-in one, used by leanCoP.

We have developed a custom generic memory allocator optimized for proof search backtracking. It is basically a stack on which all the data created during the Connection Tableaux construction is pushed. Deallocation is done only when a step back is performed in the backtracking procedure, and it consists solely of restoring the stack pointer. This allocation semantics is equivalent to putting everything on the execution stack, however, it allows for more flexible implementation of the backtracking logic itself as it does not force the programmer to use tail-recursion throughout the implementation. Our allocator is also a nice alternative to having a memory pool for each structure type used (which makes code less flexible and readable, as it is the case for example in the implementation of the BMTP [**?**]).

Input conversion: lpCoP takes as input a set of CNF clauses in a compressed machine-readable format, so we need a way to convert human-readable TPTP FOF problems to it. In our experimental setup, we use the standard TPTP tools to convert from FOF to CNF. Most provers have such functionality, including for example E-prover. For parsing TPTP syntax, the TPTP

module imported from the Vampire [**?**] project is used (for evaluation and research purposes only). We used the existing (non-tableau specific) features of these state-of-the-art provers, as our experiment focuses solely on comparing the Connection Tableau calculus efficiency. The time needed for the conversion is included in the timings given in the evaluation section, but it is negligible for the solved problems.

Output validation: In order to test the correctness of lpCoP implementations, we collect the substitution as part of the proof. We use this substitution to ground the original set of clauses together with equality axioms. We then check the satisfiability of the grounded set of clauses by using a SAT solver (similar to e.g. Zombori et al. [**?**]).

5. Experimental results

In this section we describe the problem set that we used to benchmark lpCoP, setup of an external connection tableau-based prover (leanCoP) that we have compared our implementation against, the execution environment of the benchmark, and present the results.

Experimental setup: A benchmark problem set has been selected from the TPTP library (v7.3.0). It consists of all FOF problems satisfying all of the following conditions:

- The problem has status Theorem, Unsatisfiable or ContradictoryAxioms.
- The input conversion (as described in the previous section) fits into (an arbitrarily chosen) timeout of $3s$.
- The problem does not contain numbers or distinct objects[2].
- The problem contains at least 1 equality literal
- Vampire (version 4.3) is unable to solve the problem with equality replaced with a new predicate in 30s (which means that most likely the problem is not solvable without equality axioms).

We intentionally avoid the problems with status Unknown (which means that no known prover can solve that problem), as we do not expect our single strategy prover to outperform competitive provers at this point of development. The input conversion timeout has been chosen so that the prover can spend most of the available time in the proof search phase, which is the focus of this experiment. Numbers and distinct objects require additional axioms, which lpCoP does not support yet. If a formula does not contain equality literals, then the equality axioms are not needed to prove it, so they have been removed from the benchmark as a potential source of noise. Our experiments have also indicated that TPTP contains a large block of problems that are solvable without equality axioms (so uninteresting wrt improving equality handling). We have found and filtered out (most of) them from the benchmark, by running the Vampire prover on all the problems with the equality predicate replaced with a new 2-argument predicate.

2362 TPTP-7.3 problems have been excluded from 38 of the TPTP domains, e.g. predicate logic puzzles without equality such as `PUZ031+2`. An archive with all the benchmark problems is available along with the prover[3].

[2]For a definition of a distinct_object, see http://www.tptp.org/TPTP/SyntaxBNF.html
[3]http://cl-informatik.uibk.ac.at/cek/paar2022/

All experiments have been executed on the Google Cloud Platform, using Cloud Run service. Each prover has been executed on each problem with 1 vCPU available, 2GB of RAM and 30s timeout. The output validation mentioned in Section 4 is performed on the outputs returned by lpCoP (validation is not subject to the 30s timeout).

To provide a practical comparison against a state-of-the-art connection tableau-based prover, leanCoP (v2.1) has been included in our benchmark. LeanCoP has been evaluated with the following configurations:

- [cut,comp(7)], which is the core strategy of leanCoP
- full schedule[4] - to see if LP-modification has the potential of improving the state-of-the-art

The [cut,comp(7)] strategy consists of 2 phases:

- search for a Connection Tableau with branches of length at most 7, with "cut" enabled (an incomplete strategy)
- an unbounded search with "cut" disabled (a complete strategy)

The following configurations of lpCoP have been tested during the experiment.

- **LP-modification**: LP-modification transformation, standard Connection Tableau, LPO reduction ordering, MESON bound balancing
- **Lazy Paramodulation**: no transformation, Lazy Paramodulation Connection Tableau, LPO reduction ordering, MESON bound balancing
- **Axiomatic Equality**: appending equality axioms, standard Connection Tableau, LPO reduction ordering, MESON bound balancing

The choice of MESON bound balancing and LPO reduction order is based on a smaller experiment with 100 randomly selected TPTP problems with the same time limits. The differences with our implementation were minimally better for LPO and for MESON bound, so we fix this for all further larger experiments (this is contrary to KBO usually performing better in ordered resolution / superposition provers). Axiomatic Equality configuration is included as a baseline to assess basic implementation efficiency, which is orthogonal to the impact of LP-modification.

To make the comparison against the leanCoP more relevant, we have used a similar [cut,comp] strategy with all our prover configurations. Each prover has been first executed with "cut" enabled for 0.6 (incomplete strategy) of the total time available, then it has been restarted with the "cut" disabled for the remaining time (complete strategy). MESON bound balancing is incompatible with bounding the branch length, that is the reason why a time bound has been used instead. The choice of the time distribution has been determined experimentally.

Each table shows both the total number of the problems solved by each prover, as well as the number of unique problems solved (i.e. not solved by the opponent). The first column of each table contains the abbreviations of the TPTP problem categories. Only the categories for which the behavior of the compared provers was significantly different are presented in each table.

LP-modification vs Lazy Paramodulation. LP-modification has solved 491 problems and 136 more problems than Lazy Paramodulation (355), which proves it more efficient, as

[4] standard leanCoP 2.1 implementation uses a fixed strategy scheduling

expected. However Lazy Paramodulation solved 72 problems not solved by LP-modification, which means that their search spaces are significantly different, therefore we cannot conclude that LP-modification is strictly better than Lazy Paramodulation. More insight is needed to determine the root cause.

Table 1
LP-modification vs Lazy Paramodulation

	LP-modification		Lazy Paramodulation		number
	unique solved	total solved	unique solved	total solved	of problems
total	208	491	72	355	3820
ALG	15	19	0	4	222
KLE	18	32	2	16	221
NLP	7	7	0	0	7
NUM	31	71	11	51	507
RNG	13	27	8	22	117
SET	10	64	5	59	190
SEU	21	95	14	88	600
SWC	41	63	3	25	421

LP-modification vs Axiomatic Equality. LP-modification solved 73 problems more than Axiomatic Equality prover. It performed significantly better in SWC (Software creation) and KLE (Kleene algebra) categories. An important observation is that 30% of the problems solved by LP-modification are unique, which means that these strategies are significantly different not only in theory but also in practice in our experimental setting.

Table 2
LP-modification vs Axiomatic Equality

	LP-modification		Axiomatic Equality		number
	unique solved	total solved	unique solved	total solved	of problems
total	147	491	70	414	3820
ALG	7	19	1	13	222
KLE	13	32	1	20	221
NUM	26	71	14	59	507
SCT	0	0	6	6	70
SET	4	64	9	69	190
SEU	17	95	19	97	600
SWC	42	63	0	21	421

LP-modification vs leanCoP. Tables 3 and 4 compare results of our implementation of LP-modification with results of leanCoP in 2 schedule variants: core [cut,comp(7)] strategy and schedule (originally designed for the CASC competition but good for many kinds of TPTP problems). LP-modification solves 128 problems more than core leanCoP, and 219 problems not solved by the opponent. With the full schedule leanCoP outperforms lpCoP by 9% problems solved. Still LP-modification solves 133 problems not solved by leanCoP with the full schedule. An interesting observation is that in both comparisons, LP-modification solves significantly more problems in KLE (Kleene algebra) category.

Table 3
LP-modification vs leanCoP with [cut,comp(7)] strategy

	LP-modification		leanCoP [cut,comp(7)]		number
	unique solved	total solved	unique solved	total solved	of problems
total	219	491	91	363	3820
ALG	11	19	9	17	222
COM	10	11	0	1	41
KLE	17	32	1	16	221
NUM	45	71	9	35	507
RNG	21	27	2	8	117
SWC	19	63	8	52	421
SWV	15	20	4	9	142

Table 4
LP-modification vs leanCoP with full schedule

	LP-modification		leanCoP schedule		number
	unique solved	total solved	unique solved	total solved	of problems
total	133	491	175	533	3820
ALG	10	19	11	20	222
KLE	17	32	1	16	221
LAT	0	0	5	5	130
NUM	21	71	24	74	507
SET	3	64	17	78	190
SEU	18	95	48	125	600
SWC	17	63	15	61	421

6. Related work

Competitive saturation-based provers (E-prover [**?**], Vampire [**?**], SPASS [**?**]) implement variants of the ordered paramodulation techniques to handle equality. All these variants are derived from the superposition calculus described in [**?**]. This technique allows restricting the resolution of equality literals, so that always the largest term in the clause is rewritten (according to some given reduction ordering).

Existing implementations of the other goal-oriented provers use either the trivial equality handling by appending axioms (like leanCoP [**?**]) or apply variants of Brand modification [**?**] to eliminate equality axioms from the problem (like SETHEO [**?**] and SAD [**?**]).

Degtyarev and Voronkov [**?**] proposed a combination of saturation-based paramodulation with non-clausal Connection Tableaux. The idea here is to derive clauses by using a basic superposition saturation, append them to the problem and then execute a Connection Tableaux proof search. However, an additional restriction is introduced that the extension step using a clause from saturation cannot be followed by any other extension step.

LP-modification described in this paper emulates (within the standard Connection Tableaux calculus) the Lazy Paramodulation calculus proposed by Paskevich [**?**] which attempts to incorporate the strength of the ordering constraints into the Connection Tableaux.

The Handbook of Automated Reasoning [**? ?**] describes a handful of practical optimizations

for pruning the Connection Tableaux search space (for the tableaux methods in general, not just related to the equality handling). Our lpCoP incorporates path regularity and local lemma optimizations. Our implementation of the original Lazy Paramodulation incorporates basicness of the paramodulated terms.

In a recent work Rawson introduces the LazyCoP [**?**] prover, which is also an implementation of the Lazy Paramodulation calculus, designed toward Machine Learning integration [**?**]. It incorporates more of the standard Connection Tableaux optimizations than lpCoP, however, the resulting procedure is not known to be complete. It also adds an additional strong connection inference rule, which shortens the proofs in some cases.

7. Conclusions

In this paper, we have presented an efficient implementation of a customizable prover containing both the original Lazy Paramodulation (LP) Connection Tableau search, as well as LP-modification with the standard Connection Tableau search (with constraints). Our implementation, lpCoP, is a proof of concept which implements a single strategy with a hard-coded reduction ordering.

The implementation is available at:

http://cl-informatik.uibk.ac.at/cek/paar2022/

We have experimentally compared the performance of Lazy Paramodulation vs LP-modification vs Axiomatic Equality handling. LP-modification solved 38% more problems than the Lazy Paramodulation. We cannot conclude that we have found a strict optimization, however, because 20% of the problems solved by LP were not solved by LP-modification. We know at least 2 factors that contribute to that discrepancy:

- in some large test cases the LP-modification transformation causes too much overhead on the proof search, which Lazy Paramodulation is able to avoid at runtime (not every clause in the proof needs paramodulation)
- the implementation differences cause equivalent branches of LP-modification and Lazy Paramodulation to be processed in a different order in some cases.

We believe that both issues can be resolved with further engineering work and we plan to work on these in the future.

LP-modification solved 19% more problems than the Axiomatic Equality and 30% of the solved problems were not solved by the Axiomatic Equality. This means that combining both strategies might give up to 14% better results than using any of them separately. This is still behind superposition-based provers on FO problems with equality. A further comparison against the similar approaches [**? ?**] might be necessary to get further insights into the particular advantages of LP-modification.

We have also performed an experiment to compare LP-modification against the leanCoP prover. LP-modification solved 35% more problems than the leanCoP core strategy and 8% fewer problems than leanCoP with its full schedule. Still, LP-modification solved 133 problems not solved by leanCoP's strategy schedule. This suggests that augmenting leanCoP with LP-modification strategy might allow it to solve up to 25% more equality problems. This is likely

an overestimation, given that the schedule distributes available time among different strategies, yet the expected impact is still significant.

Paramodulation in saturation-based provers is known to be sensitive to the choice of the reduction ordering (for example in E-prover [**?**]). Following the intuition that the same might be true in the Connection Tableaux setup, we plan to investigate the impact of different reduction orderings on the performance of LP-modification. We also plan to experiment with using Machine Learning to guide the proof search of lpCoP. Furthermore, a comparison with other, more efficient techniques for preprocessing equality [**?**] in connection calculi is open.

Acknowledgements

We thank Andrei Paskevich for his explanation of the details of lazy paramodulation and the various tableaux optimizations. We thank the anonymous reviewers for their comments on a previous version of this paper submitted to Tableaux 2021. This paper has been supported by the ERC Starting Grant no. 714034.