INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67, D–80538 München

Ludwig
Maximilians–
Universität
München

A Confluent Connection Calculus

Peter Baumgartner, Norbert Eisinger, Ulrich Furbach

appeared in *Proc. 16th Int. Conf. on Automated Deduction (CADE)*, Springer LNAI, 1999
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1998-4, April 1998

A Confluent Connection Calculus

Peter Baumgartner[1], Norbert Eisinger[2], and Ulrich Furbach[1]

[1] Universität Koblenz-Landau
Institut für Informatik
{peter,uli}@uni-koblenz.de
[2] Universität München
Institut für Informatik
eisinger@informatik.uni-muenchen.de

Abstract. This work[1] is concerned with basic issues of the design of calculi and proof procedures for first-order connection methods and tableaux calculi. Proof procedures for these type of calculi developed so far suffer from not exploiting proof confluence, and very often unnecessarily rely on a heavily backtrack oriented control regime.
As a new result, we present a variant of a connection calculus and prove its *strong* completeness. This enables the design of backtrack-free control regimes. To demonstrate that the underlying fairness condition is reasonably implementable we define an effective search strategy.

1 Introduction

This work is concerned with basic issues of the design of calculi and proof procedures for first-order connection methods and tableaux calculi. Calculi we have in mind include connection calculi [6], first-order clausal tableaux with rigid variables [9], more recent developments like A-ordered tableaux [14, 12], and tableaux with selection function [13]. Let us refer to all these calculi by the term "rigid variable methods". Recently complexity issues for those kinds of calculi have been considered in [19].

We propose a new technique for the design of proof procedures for rigid variable methods. The proposed technique should also be applicable to calculi which *avoid* rigid variables in the first place, like SATCHMO [16], MGTP [10], hyper tableaux [4] and ordered semantic hyper linking [17]. Usually the price for getting around rigid variables in these approaches is that they involve some uninformed ground instantiation in special cases. These calculi are likely to profit from techniques enabling them to handle rigid variables as well.

A more recent development is the disconnection method [7]. Unlike the calculi mentioned above, and like our calculus below, it avoids blind instantiation by the use of unification. Unlike our method and unlike the free variable methods mentioned above, the disconnection method does not deal with "free vari-

[1] A full version of this article, which includes all proofs and other details omitted here, is available as [3]

ables"[2]. In free variable methods, a branch or a path is considered as "solved" if it contains a pair (K, L) of literals that are complementary; in the disconnection method, the respective "\$-closed" condition on (K, L) is that K and L are equal after instantiating all variables with the same (new) constant \$. For instance, $(P(X, Y), \neg P(Z, Z))$ is considered as complementary. The disconnection method has inference rules to further instantiate clauses derived so far, but the parent clauses of an inference step have to be kept. This instantiation is driven by connections that are not \$-closed. Since substitutions are applied locally to clauses, a previously \$-closed connection may become open again. If the clause containing $\neg P(Z, Z)$ is further instantiated to, say, $\neg P(a, a)$ this implies that the connection $(P(X, Y), \neg P(a, a))$ is no longer \$-closed. In free variable methods this effect is avoided because, clearly, complementary literals *remain* complementary after instantiation; there is no need to check previously closed paths after instantiation whether they are still closed. On the other hand, for the disconnection method only one variant per clause needs to be kept; this is not possible in free variable methods.

In sum, we feel that theses approaches are rather different. A more rigid and systematic comparison would be interesting future work.

Our approach is based on the observation that current proof procedures for rigid variable methods rely on the following weak completeness theorem:

> *Weak completeness:* a clause set S is unsatisfiable if and only if *there is* a derivation from S which is also a refutation.

The search space thus is the space of *derivations*; it requires a tentative control regime such as backtracking, which explores *all* possible derivations.

Proof confluent calculi like the ones mentioned at the beginning of the introduction, however, should admit a *strong* completeness theorem of the form:

> *Strong completeness:* a clause set S is unsatisfiable if and only if *every* (fair) derivation from S is a refutation.

Consequently, proof procedures following this theorem can do with an irrevocable control regime that needs to develop only *one single* derivation and may safely ignore alternatives as it proceeds. They can thus *reuse* information which would be lost in a backtracking intensive approach. Typically they enumerate *models* but not *derivations* (the hyper tableaux calculus [4] is an example that enumerates models, model elimination [15] is an example for the enumeration of derivations).

Put abstractly, the source to gain efficiency is that there are usually many derivations for the *same* model, and all but one derivation can be avoided. In this paper, we will develop a strong completeness result for a modified connection calculus (the CCC calculus, Section 2) together with first steps towards a respective proof procedure.

[2] This is not meant to disqualify the disconnection method; merely, we want to point out differences.

This result closes a strange gap: the connection calculus in [6] is proof confluent on the propositional level, but its first-order version is not. This is the only calculus we are aware of having this property. Although other free-variable methods, such as the first-order tableaux calculus in Fitting's book [9] *are* proof-confluent, they are *implemented* as if they were non-confluent. This yields unnecessary inefficiencies, and the main motivation for the work presented in this paper is to find a cure for them.

Prawitz Procedures. Let us first briefly recall the basic idea of current proof procedures for rigid variable methods and identify the tackled source of inefficiency.

"Usual" proof procedures like the one used in the 3TAP prover [11], the LeanTAP prover [5]), the connection procedure proposed in [6], and the one in Fitting's Book [9] follow an idea suggested by Prawitz [18] and can be seen as more or less direct translation of the following formulation of the Herbrand-Skolem-Gödel theorem (we can restrict our attention to clause logic here):

> A clause set S is unsatisfiable if and only if there is a finite set S' of variants of clauses of S and there is a substitution δ such that $S'\delta$ is unsatisfiable, where $S'\delta$ is viewed as a propositional clause set.

Now, in accordance with the theorem, proof procedures for rigid variable methods typically realise the following scheme to prove that given clause set $S = \{C_1, \ldots, C_n\}$ is unsatisfiable. Following Voronkov [19], we call it *The Procedure*:

Procedure 1 (The Procedure).

(i) Let $\mu = 1$ called *multiplicity*.
(ii) Let $S^\mu = \{C_1^1, \ldots, C_1^\mu\ , \ldots,\ C_n^1, \ldots, C_n^\mu\}$ be a set of pairwise variable-disjoint clauses, such that C_i^j is a variant of C_i for $1 \le i \le n$, $1 \le j \le \mu$. It is usual to call S^μ an *amplification* of S.
(iii) Check if there is a substitution δ such that $S^\mu\delta$ is propositionally unsatisfiable.
(iv) If such a δ exists, then stop and return "unsatisfiable"; otherwise let $\mu = \mu{+}1$ and go to step (ii).

Completeness of *The Procedure* is achieved by, first, a fairness condition in the generation of the amplifications S^μ in step (ii), namely by uniformly taking $1, 2, 3, \ldots$ variants of every clause in S in round $1, 2, 3, \ldots$, and, second, by exhaustively searching for substitutions in step (iii).

Connection Methods. How is step (iii) in *The Procedure* realised? Our primary interest is in connection methods (also called matrix methods), hence we will briefly recall the idea: define a matrix to be any set of quantifier free formulae. For our purposes it suffices to consider clause sets only. Notice that S^μ is a matrix. A *path* through a matrix is obtained by taking exactly one literal from every clause. A connection is a pair of literals, which can be made complementary by application of a substitution; with each connection we associate a most general unifier σ achieving this. In order to realise step (iii), proof procedures for connection calculi search for a substitution δ such that every path through $S^\mu\delta$ contains a pair of complementary literals (we say that δ *closes* S^μ). If such a δ exists S must be unsatisfiable: take some arbitrary ground instance of $S^\mu\delta$ and observe that this set is propositionally unsatisfiable, because any possible way to satisfy a conjunction of disjunctions is excluded by virtue of the complementary literals.

For δ, it is sufficient to search through the *finite* space of most general unifiers making literals along paths (or branches) complementary (this guarantees the termination of step (iii). See e.g. [6] for a concrete procedure to decide if a δ exists which renders all paths complementary. For our purposes it suffices to rephrase the underlying idea: let $p_1, p_2, \ldots, p_n$ be any enumeration of all paths through the current amplification. It can be shown by usual lifting techniques that there is δ which simultaneously renders all paths as complementary if and only if there is a sequence $p_1\delta_1,\ p_2\delta_1\delta_2, \ldots, p_n\delta_1\cdots\delta_n$, where δ_i is a most general unifier associated with a connection in $p_i\delta_1\cdots\delta_{i-1}$. In other words, by defining $\delta := \delta_1\cdots\delta_n$ one recognises that δ can be computed incrementally. Notice, however, that the "there is" quantification is to be translated in a backtracking-oriented procedure.

Example 1 (Connection method). Consider the following unsatisfiable clause set:

$$S = \{P(X) \vee Q(X),\ \ \neg P(c),\ \ \neg P(a) \vee \neg P(b),\ \ \neg Q(a),\ \ \neg Q(b)\}$$

We write a matrix as a vertical sequence of clauses. Hence S^1 and S^2 look like this[3]:

S^1:

$$\begin{array}{c} P(X^1) \vee Q(X^1) \\ \neg P(c) \\ \neg P(a) \vee \neg P(b) \\ \neg Q(a) \\ \neg Q(b) \end{array}$$

S^2:

$$\begin{array}{c} P(X^1) \vee Q(X^1) \\ P(X^2) \vee Q(X^2) \\ \neg P(c) \\ \neg P(a) \vee \neg P(b) \\ \neg Q(a) \\ \neg Q(b) \end{array}$$

A path is obtained by traversing the matrix from top to bottom, picking up one literal from every clause.

The Procedure starts with $\mu = 1$, i.e. with S^1. By looking at the path through S^1 which passes through $P(X^1)$ one recognises that there are three candidate MGUs $\delta_1^1 = \{X^1/a\}$, $\delta_2^1 = \{X^1/b\}$ and $\delta_3^1 = \{X^1/c\}$. Since none of $S^1\delta_1^1$, $S^1\delta_2^1$ and $S^1\delta_3^1$ is propositionally unsatisfiable, *The Procedure* has to consider S^2. An incremental computation of a substitution δ which closes S^2 might proceed as follows: it starts by considering the connection $(P(X^1), \neg P(c))$ which results in $\delta_1^2 = \{X^1/c\}$. The next connection would be $(Q(X^2), \neg Q(a))$ with MGU $\delta_2^2 = \{X^2/a\}$. The combined substitution $\delta_1^2\delta_2^2 = \{X^1/c, X^2/a\}$ does not close the matrix, neither will $\delta_1^2\delta_3^2$, where $\delta_3^2 = \{X^2/b\}$. Hence, backtracking occurs

[3] In the literature, matrices are also written horizontally.

until eventually the "right" substitution $\delta = \{X^1/a,\ X^2/b\}$ is computed, which closes S^2.

Proof Search with Tableaux. Typically, the search for δ in step (iii) is organised as the construction of a free-variable tableau where a branch in a tableau stands for a path through the current amplification.

For space efficiency reasons it is prohibitive to explicitly represent in step (iii) all paths through S^μ in memory. For example, only 15 clauses consisting of 3 literals result in more than 14 million paths. A respective tableau thus has in the worst case the same number of branches. Hence, one possibility for step (iii) is to only keep in memory one path p (or branch in a tableau) at a time. A closing substitution δ_p is guessed, and if δ_p cannot be extended to a substitution δ which simultaneously closes all paths, then backtracking occurs and a different candidate δ_p is guessed. If all that fails, then a completely new tableau construction is started when entering step (iii) in the next round.

We emphasise that this is a common idea in all proof procedures for free-variable tableaux and connection calculi we are aware of. It is an intrinsic weakness and it seems that it is not solved by the many improvements that are around for tableau and connection calculi now.

2 A Confluent Connection Calculus

In this section we indroduce the confluent version of a connection calculus. For this we briefly have to set up the usual prerequisites. After indroducing the calculus together with an abstract notion of fairness, we show how this fairness can be realised and finally we prove strong completeness.

2.1 Preliminaries

For a literal L we denote by $|L|$ the atom of L, i.e. $|A| = A$ and $|\neg A| = A$ for any atom A. Literals K and L are *complementary* iff K and L have different sign and $|L| = |K|$.

By var(*object*) we denote the set of variables occurring in *object*, where *object* is a term, a literal, a clause or a set of one of these. Our notion of a *substitution* is the usual one [1], and, as usual, we confuse a substitution with its homomorphic extension to terms, literals, clauses and clause sets. For a substitution σ, we denote by $\mathrm{dom}(\sigma)$ the (finite) set $\{X \mid X\sigma \neq X\}$, by $\mathrm{cod}(\sigma)$ the set $\{X\sigma \mid X \in \mathrm{dom}(\sigma)\}$ and by $\mathrm{vcod}(\sigma)$ the set $\mathrm{var}(\mathrm{cod}(\sigma))$. Substitution γ is a *ground* substitution iff $\mathrm{vcod}(\gamma) = \emptyset$; it is a *ground substitution for object* iff additionally $\mathrm{var}(\textit{object}) \subseteq \mathrm{dom}(\gamma)$. A substitution σ is idempotent, if $X\sigma\sigma = X\sigma$ for each variable X. This is the case iff $\mathrm{dom}(\sigma) \cap \mathrm{vcod}(\sigma) = \emptyset$.

A clause is a finite disjunction of literals. In the following, S is the given finite input clause set, and M is a matrix for S, i.e. a set of clauses, each of which is an instance of a clause from S. It is worth emphasizing that in a matrix we consider clauses *not* to be individually universally quantified. That is, all variables are free, and thus applying a substitution, say, $\{Y/a\}$ to the matrix $\{\neg P(Y),\ P(Y) \vee Q(Y)\}$ would affect all occurrences of Y.

A *connection with substitution* σ is a pair of literals (L, K) such that $L\sigma$ and $K\sigma$ are complementary. A *connection* is a pair of literals that is a connection with some substitution. In these definitions, we replace "substitution σ" by "MGU σ" only if additionally $\sigma = \mathrm{unify}(\{|K|, |L|\})$, where unify returns any most general unifier of its argument. Below we make use of the following assumption:

Assumption 1. If a set Q of atoms is unifiable, then $\mathrm{unify}(Q)$ returns an idempotent most general unifier σ with (i) $\mathrm{dom}(\sigma) \subseteq \mathrm{var}(Q)$ and (ii) $\mathrm{vcod}(\sigma) \subseteq \mathrm{var}(Q)$.

Notice that this is a very mild assumption: (i) says that σ operates on the variables of Q only, and (ii) says that σ must not introduce new variables. Clearly, this is satisfied by any "standard" unification procedure.

A *path* through a matrix M is a set of literals, obtained by taking exactly one literal from each clause of M. A path is *closed* iff it contains a pair of complementary literals, otherwise it is *open*. A matrix is *open* iff there is a (at least one) path through it that is open, otherwise it is *closed*.

Notice that there is exactly one path through the "empty" matrix $\{\}$, which is the empty set $\{\}$ of literals; notice further that this path is open. On the other hand, if a matrix M contains the empty clause, then there is no path through M, in particular no open path, and the matrix is closed.

2.2 The Calculus

We are going to define the calculus *CCC* (Confluent Connection Calculus). The constituents are the inference rules, the notion of a derivation, and a fairness condition for derivations.

Definition 1 (CCC inference rules and derivation). The inference rule *variant step on* M is defined as follows:

$$\frac{M}{M \cup \mathsf{new}(S)} \quad \text{if} \begin{cases} \text{1. } \mathsf{new}(S) \text{ contains exactly one variant of each clause of } S\text{, and} \\ \text{2. the members of } \mathsf{new}(S) \text{ are pairwise variable disjoint and each} \\ \quad \text{of them is variable disjoint with each clause in } M. \end{cases}$$

The inference rule *connection step on* (K, L) *in* M is defined as follows:

$$\frac{M}{M \cup M\sigma} \quad \text{if} \begin{cases} \text{1. there are clauses } C \in M \text{ and } D \in M \text{ such that } C = K \vee R_K, \\ \quad D = L \vee R_L, \text{ for some clauses } R_K \text{ and } R_L\text{, and} \\ \text{2. } \{K, L\} \subseteq p \text{ for some open path } p \text{ through } M\text{, and} \\ \text{3. } (K, L) \text{ is a connection with MGU } \sigma. \end{cases}$$

The set $(M \cup M\sigma) \setminus M$ is called the set of *new clauses* (of this inference step). If conditions 1 to 3 above hold for M and (K, L), we say that a connection step is *applicable* to (K, L) in M or that (K, L) is a *candidate* for a connection step in M.

We say that a connection step on (K, L) in M is *progressive* if $M \cup M\sigma \neq M$ (i.e. at least one clause in the conclusion is new). If $M \cup M\sigma = M$ we say that it is *non-progressive*.

Note that a connection step on a connection (K, L) with $|K| = |L|$ is impossible and therefore neither progressive nor non-progressive – any path containing $\{K, L\}$ would be closed, contradicting condition 2. above.

Any sequence $D = (\{\} = M_0), M_1, \ldots, M_n, \ldots$ is called a *derivation from* S, provided that M_{i+1} is obtained from M_i, which is open, by a single application of one of the inference rules (for $i \geq 0$). A *refutation* is a derivation that contains (i.e. ends in) a closed matrix.

Notice that we do not have inference rules that delete clauses. Thus, every derivation has the following *chain property*: $(\{\} = M_0) \subseteq M_1 \subseteq \cdots \subseteq M_n \subseteq \cdots$. The inclusions are not strict, because non-progressive steps are allowed as well (though they are not needed at all).

2.3 Example

Suppose the given input clause set is $S = \{\neg P(X),\ \ P(Y) \vee Q(Y),\ \ \neg Q(Z)\}$. Clearly, S is unsatisfiable. We develop a refutation.

We have to start with the empty matrix M_0. Only a variant step can be applied to M_0, hence M_1 is just a copy of S (the left matrix in the figure below). For each matrix M_i we discuss the possibilities for one open path through M_i, which is indicated by underlining.

$$
\begin{array}{lcc|lcc}
M_1: & \underline{\neg P(X)} & (C_1) & M_2: & \underline{\neg P(X)} & (C_1) \\
 & \underline{P(Y)} \vee Q(Y) & (C_2) & & P(X) \vee \underline{Q(X)} & (C_2\sigma_1) \\
 & \underline{\neg Q(Z)} & (C_3) & & \underline{\neg Q(Z)} & (C_3) \\
 & & & & P(Y) \vee \underline{Q(Y)} & (C_2)
\end{array}
$$

The underlined path in M_1 is open, and we carry out a connection step on $(\neg P(X), P(Y))$. Suppose that unify returns $\sigma_1 = \{Y/X\}$. This step is progressive and results in the matrix M_2 (right side in the figure above).

Now there are two connections in the underlined open path in M_2 to which a connection step is applicable: $(Q(X), \neg Q(Z))$ and $(\neg Q(Z), Q(Y))$. By applying a connection step to the former we would obtain a closed matrix, thus ending the refutation. In order to make the example more illustrative, let us instead apply a connection step to $(\neg Q(Z), Q(Y))$ with MGU $\sigma_2 = \{Y/Z\}$. This gives us matrix M_3, which is depicted below.

The underlined path in M_3 offers three possibilities. First, the connection $(Q(X), \neg Q(Z))$ is still a candidate for a connection step, but we disregard it for the same reason as in the previous matrix. Second, the connection step on $(\neg Q(Z), Q(Y))$ with MGU $\{Y/Z\}$ would not be progressive and therefore not interesting (if unify returned $\{Z/Y\}$ instead, the step would be progressive, though). Third, a connection step on $(\neg P(X), P(Z))$ is progressive no matter which of the two MGUs unify returns. Suppose that this step is applied with $\sigma_3 = \{Z/X\}$. The resulting matrix M_4 is depicted in the figure below as well. Note that $P(X) \vee Q(X)$ occurs twice in M_4. Since matrices are sets, deleting one of these occurrences would represent the same set.

$$
\begin{array}{lcr|lcr}
M_3: & \underline{\neg P(X)} & (C_1) & M_4: & \neg P(X) & (C_1) \\
 & P(X) \vee \underline{Q(X)} & (C_2\sigma_1) & & P(X) \vee Q(X) & (C_2\sigma_1) \\
 & \underline{\neg Q(Z)} & (C_3) & & \neg Q(X) & (C_3\sigma_3) \\
 & \underline{P(Z)} \vee Q(Z) & (C_2\sigma_2) & & P(X) \vee Q(X) & (C_2\sigma_2\sigma_3) \\
 & P(Y) \vee \underline{Q(Y)} & (C_2) & & P(Y) \vee Q(Y) & (C_2) \\
 & & & & \neg Q(Z) & (C_3) \\
 & & & & P(Z) \vee Q(Z) & (C_2\sigma_2)
\end{array}
$$

M_4 is closed because either path through the first three clauses alone is closed. Hence we have found a refutation.

2.4 Fairness

Control strategies for any kind of rule-based system usually depend on some notion of fairness, when several rules or rule instances are applicable and one of them has to be selected for the next step. In matrix M_2 above, two connections were candidates for progressive connection steps, and one of them was selected to derive M_3, while the other was disregarded. In M_3 the disregarded connection was again a candidate for a progressive connection step and was again disregarded. Fairness is a condition on the choices made by the strategy to prevent that an applicable step is disregarded forever.

In many cases, especially with control strategies for logical calculi, fairness can be defined very simply as *exhaustiveness*: every step that is applicable to any state will ultimately be performed. There are standard text book procedures to implement exhaustive strategies effectively; in particular, resolution calculi can be treated this way.

This simple approach is sufficient if the underlying rules are commutative in the following sense: whenever two steps are applicable in some state, each of them remains applicable in the successor state produced by the other. If, on the other hand, the rules are not commutative, then the application of one step might destroy the applicability of the other. Such a phenomenon makes exhaustiveness an inherent impossibility and turns fairness into a more difficult question.
Unfortunately, our system is of the non-commutative variety. As an example let

$$M_i = \{\underline{\neg P(a)} \vee R, \quad \underline{P(X)}, \quad \underline{\neg Q(a)}, \quad \underline{Q(Y)} \vee P(Y), \quad \ldots\}$$

where the underlined path through M_i is open. Both connections $(\neg P(a), P(X))$ and $(\neg Q(a), Q(Y))$ are candidates for progressive connection steps in M_i. Disregarding the former and applying the latter yields

$$M_{i+1} = \{\neg P(a) \vee R, \quad P(X), \quad \neg Q(a), \quad Q(a) \vee P(a), \quad Q(Y) \vee P(Y), \quad \ldots\} \ .$$

Now any path through M_{i+1} containing the disregarded connection $(\neg P(a), P(X))$ is closed. This is a consequence of the presence of the clauses $\neg Q(a)$ and $Q(a) \vee P(a)$ in M_{i+1}. Therefore a connection step on the first connection, which would introduce the new clause $P(a)$, is no longer applicable – the second condition in the definition of a connection step can no longer be satisfied for this connection. Note that other paths through M_{i+1} are still open, though.

By the way, had we selected the other applicable step in M_i, the next matrix would have been

$$M'_{i+1} = \{\neg P(a) \vee \underline{R}, \quad \underline{P(a)}, \quad \underline{P(X)}, \quad \underline{\neg Q(a)}, \quad \underline{Q(Y)} \vee P(Y), \quad \ldots\}$$

in which the disregarded connection $(\neg Q(a), Q(Y))$ is still a candidate for a progressive connection step as shown by the underlined open path through M'_{i+1}. Selecting this connection next we can achieve that the new clauses of both steps are present.

Should the presence of all of these clauses happen to be indispensable for closing the matrix, it might be that the sequence $M_0, \ldots, M_i, M_{i+1}$ cannot be extended to a refutation whereas the sequence $M_0, \ldots, M_i, M'_{i+1}$ can – this would be a typical case of non-confluence. Fortunately, it turns out that our calculus *is* confluent: in all such situations, either none or both sequences can be continued to a refutation. The example illustrates why this is not a trivial property.

Coming back to fairness, the simple exhaustiveness definition is not possible, and we need a more complex definition. With this definition we will prove below our main result, that any fair derivation from an unsatisfiable clause set is a refutation (strong completeness).

Definition 2 (Fairness). A derivation $D = M_0, M_1, \ldots, M_i, \ldots$ is *fair* iff it is a refutation or else the following two conditions are satisfied:

1. For every $i \geq 0$ there is a $j \geq i$ such that M_{j+1} results from M_j by a variant step.
2. For every $i \geq 0$ and every connection (K, L) with MGU σ that is a candidate for a progressive connection step in M_i there is a $j \geq i$ such that one of the following is true:
   (a) $M_i\sigma \subseteq M_j$.
   (b) Every path p through M_j with $\{K, L\} \subseteq p$ is closed.

Condition 1 simply requires variant steps to be performed "every now and then".

Condition 2.(a) formalises that any progressive connection step that remains applicable sufficiently long must ultimately be performed. More precisely, the condition does not enforce that this very step be performed, but only that its effect be achieved at some point – regardless whether by this step or by some other.

In a nice case, if Condition 2.(a) holds, the connection (K, L) is irrelevant from M_j onward, because any connection step on (K, L) in some later matrix would result in clauses that have been introduced up to M_j anyway.

However, perhaps contrary to the intuition, this is not always so. The reason is that the MGU σ of the connection, say $\{X/Y\}$, may be applicable to clauses containing the variable X *that are derived only later in the derivation.* Hence, applying σ at a later time may well be progressive. Condition 2.(a) covers this possibility: let the later time be i', then there must again be a time j' at which the clauses that are new for $M_{i'}$ have been introduced.

Condition 2.(b) captures the case that a connection step loses its applicability because of other steps.

2.5 Achieving Fairness

Our notion of fairness – Definition 2 – is defined as an abstract mathematical property derivations may or may not enjoy. In order to implement a proof procedure, however, we need not only the property, but an effective strategy for the construction of a derivation that is guaranteed to have this property.

Fortunately, the existence of such a strategy can be demonstrated with a fairly simple approach: use an iteratively increasing bound T that serves both as a limit for the term nesting depth and as a trigger for the application of variant steps.

More precisely, we call a progressive connection step *T-progressive*, if at least one (but not necessarily all) of its new clauses has a term nesting depth not exceeding T. Building on this, we define an effective strategy as follows (in contrast to *The Procedure*):

Procedure 2 (The CCC Procedure).

(i) Initialise M with the empty matrix and T with the maximum term nesting depth of the clauses in S.
(ii) While M is open repeat:
   (a) Modify M by applying one variant step.
   (b) Increment T.
   (c) Modify M by applying a T-progressive connection step.
   Repeat this until saturation, i.e., until no such step is applicable any more.

The sequence of values of M over time corresponds to the constructed derivation. Let us denote these values by $M_0, M_1, \ldots, M_i, \ldots$

Note that this strategy is indeed irrevocable in the sense described in the introduction. It never backtracks to previous values of M or makes any other provision for reconsidering alternatives at a later point in time. A subtle difference to *The Procedure* concerns the generated amplifications. One might think that each iteration through (a) creates the next amplification S^μ of S. However, during the application of T-progressive connection steps in phase (c), the current M_i is extended by new clauses yielding M_{i+1}, whereas S^μ in *The Procedure* would only be instantiated, but not extended. In a sense our connection steps combine instantiation with (partial) amplification.

Theorem 1. *Any derivation (from any finite input clause set) constructed by Procedure 2 is fair.*

Of course there are certainly much better strategies. The procedure above is only meant as a proof that fair derivations can be effectively constructed.

What remains, now, is to show that the fairness of derivations, no matter how achieved, ensures confluence. This result is established in the next section.

3 Completeness

Definition 3 (Bounded downward closed clause set). Let S be a finite clause set, and γ be a ground substitution for S. Define a set of sets $S \downarrow \gamma$ as follows: $S \downarrow \gamma = \{S\delta \mid S\delta\gamma = S\gamma \text{ for some substitution } \delta\}$.

That is, $S \downarrow \gamma$ is the set of instances of S, (including S, variants of S and $S\gamma$ itself) that can further be instantiated to $S\gamma$ by γ itself. Notice that if $S\delta' \in S \downarrow \gamma$ and $S\delta'\delta''\gamma = S\gamma$ for some δ'', then also $S\delta'\delta'' \in S \downarrow \gamma$ by simply taking $\delta = \delta'\delta''$. In this sense $S \downarrow \gamma$ is "downward closed".

Lemma 1. *$S \downarrow \gamma$ is a finite set of finite sets.*

Proposition 1. *Let M be a matrix, p be a path through M, and γ be a ground substitution for M. Suppose that $\{K, L\} \subseteq p$ (for some literals K and L), that (K, L) is a connection with substitution γ, and that (K, L) is a connection with idempotent MGU σ. Then $M\gamma = M\sigma\gamma$.*

Definition 4 (Ordering on Clause Sets). Let M and N be finite clause sets. Define $M \succ N$ iff $\mathrm{var}(M) \supset \mathrm{var}(N)$.

It is easy to see that $\succ$ is an irreflexive and transitive relation, hence a strict (partial) ordering. Obviously, since $\mathrm{var}(M)$ is always finite, it is well-founded as well.

Lemma 2 (Path Extension). *Let $D = M_0, M_1, \ldots, M_n, \ldots$ be an infinite derivation from clause set S. Let M be a finite clause set, and suppose that $M \subseteq M_k$ for some k. Then there is a path p through M such that for every $i \geq k$ there is an open path p_i through M_i with $p \subseteq p_i$.*

That is, we can find a path p through M such that p will be part of some open path as the derivation proceeds.

The main result of this paper is the following completeness theorem. Note that it assures completeness, whenever fairness is guaranteed; hence we have strong completeness and thus proof confluence.

Theorem 2 (Completeness). *Let S be the given input clause set. If S is unsatisfiable, then every fair derivation from S is a refutation.*

Proof. By Herbrand's theorem there is a finite set S^g of ground instances of clauses from S which is unsatisfiable. It can be presented as a finite set M of pairwise variable disjoint variants of clauses from S and a ground substitution γ such that $M\gamma = S^g$.

Now, let D be a fair derivation from S and assume contrary to the theorem that D is not a refutation.

Since D is fair, the variant rule must be applied infinitely often. Recall that we never delete clauses from matrices. Hence, at some time point, say l, it will thus be that $M \subseteq M_l$ (and hence also $M \subseteq \bigcup_{i \geq 0} M_i$). W.l.o.g. we can assume that each clause in M is syntactically identical to one of the variants introduced up to M_l; otherwise rename M (and γ) appropriately.

Now let $N \in M \downarrow \gamma$ be a minimal set wrt. $\succ$ such that $N \subseteq \bigcup_{i \geq 0} M_i$. We have to check that such a set N exists: since $M \subseteq \bigcup_{i \geq 0} M_i$ and it trivially holds that $M \in M \downarrow \gamma$ it is clear that M itself might be a candidate to be taken as N. Now, since $\succ$ is a well-founded ordering on finite clause sets, and the elements of $M \downarrow \gamma$ are finite (cf. Lemma 1), any chain $(N_0 := M) \succ N_1 \succ \cdots \succ N_{n-1} \succ N_n$ with $N_i \in M \downarrow \gamma$ and $N_i \subseteq \bigcup_{i \geq 0} M_i$ must be finite, and we set $N := N_n$, provided that this chain cannot be extended to the right. Thus, N is minimal wrt. $\succ$ restricted to the elements in $\bigcup_{i \geq 0} M_i$.

The proof technique now is to construct an N' with $N' \in M \downarrow \gamma$ and $N' \subseteq \bigcup_{i \geq 0} M_i$ and $N \succ N'$, contradicting the minimality of N.

Since derivations have the chain property, the property $N \subseteq \bigcup_{i \geq 0} M_i$ implies by the fact that N is a finite set that

$$N \subseteq M_k \quad \text{, for some } k. \tag{1}$$

Therefore we can apply Lemma 2 and conclude that there is a path p through N such that for every $i \geq k$ there is an open path p_i through M_i with $p \subseteq p_i$. This result will be used further below – the immediate consequence that this p is open will be needed in a moment.

Clearly, $N\gamma$ is an unsatisfiable ground clause set, because $N \in M \downarrow \gamma$ means by definition that $N = M\delta$ for some δ such that $M\delta\gamma = M\gamma$, hence $N\gamma = M\gamma$, and $M\gamma$ was assumed to be ground and unsatisfiable above. Consequently, any path through $N\gamma$ contains a pair of complementary literals. In particular, $p\gamma$ contains a pair of complementary literals, say $K\gamma$ and $L\gamma$, where

$$\{K, L\} \subseteq p \ . \tag{2}$$

In other words, (K, L) is a connection with substitution γ. But then (K, L) is a connection with MGU σ, where σ is computed by unify. Then Proposition 1 is applicable and we conclude that $N\sigma\gamma = N\gamma$. The element $N\sigma$ has the form $N\sigma = M(\delta\sigma)$ because $N = M\delta$. For this element we obtain by the previous equation and by $N\gamma = M\gamma$ the properties $M(\delta\sigma)\gamma = N\gamma = M\gamma$, which is just the definition for

$$N\sigma \in M \downarrow \gamma \ . \tag{3}$$

The next subgoal is to show that $N\sigma$ is strictly smaller than N.

Since p is open and $\{K, L\} \subseteq p$ it trivially holds that $|K| \neq |L|$. On the other hand, σ is a most general unifier of $|K|$ and $|L|$. Hence there is at least one variable, say X, in $\{K, L\} \subseteq p$, for which $X\sigma \neq X$, that is, $X \in \mathrm{dom}(\sigma)$. Now p is a path through N, therefore $X \in \mathrm{var}(N)$.
Obviously, it holds that $\mathrm{var}(N\sigma) \subseteq \mathrm{var}(N) \cup \mathrm{vcod}(\sigma)$. By Assumption 1 $\mathrm{vcod}(\sigma) \subseteq \mathrm{var}(\{K, L\}) \subseteq \mathrm{var}(N)$, hence $\mathrm{var}(N\sigma) \subseteq \mathrm{var}(N)$.
The idempotence of σ implies $\mathrm{dom}(\sigma) \cap \mathrm{vcod}(\sigma) = \emptyset$, thus $X \notin \mathrm{vcod}(\sigma)$ and $X \notin \mathrm{var}(N\sigma)$. So we have $\mathrm{var}(N\sigma) \subset \mathrm{var}(N)$, which means nothing but

$$N \succ N\sigma \quad . \tag{4}$$

There remains to be shown that

$$N\sigma \subseteq \textstyle\bigcup_{i \geq 0} M_i \quad . \tag{5}$$

We distinguish two cases, each leading to the conclusion 5.

Case 1: For every $i \geq k$, it is not the case that (K, L) is a candidate for a progressive connection step in M_i.

As concluded above by the application of Lemma 2, for every $i \geq k$ there is an open path p_i through M_i with $p \subseteq p_i$. With the fact that $\{K, L\} \subseteq p$ (obtained in 2 above) it follows that $\{K, L\}$ is a candidate for a connection step in M_i, for every $i \geq k$.

Together with the assumption of this Case 1 we obtain that for every $i \geq k$ a connection step on (K, L) in M_i exists, but this connection step is non-progressive. Hence, for every $i \geq k$, $M_i \cup M_i\sigma = M_i$. But then with $N \subseteq M_k$ as obtained in 1 we get the following chain:

$$N\sigma \;\subseteq\; M_k\sigma \;\subseteq\; \textstyle\bigcup_{i \geq k}(M_i \cup M_i\sigma) \;=\; \textstyle\bigcup_{i \geq k} M_i \;\subseteq\; \textstyle\bigcup_{i \geq 0} M_i \quad .$$

Case 2: This is the complement of Case 1. Hence suppose that for some $i \geq k$ the connection (K, L) is a candidate for a progressive connection step in M_i. We are given that the given derivation D is fair (cf. Def. 2). In particular, Condition 2 in the definition of fairness holds wrt. the connection (K, L) and M_i. Let $j \geq i$ be the point in time claimed there.

Assume that Condition 2.(b) is satisfied (this will yield a contradiction). This means that every path p' through M_j with $\{K, L\} \subseteq p'$ is closed. But at the same time, however, observing that $j \geq k$ (since $j \geq i$ and $i \geq k$) we concluded further above by application of Lemma 2 that there is an open path p_j through M_j with $p \subseteq p_j$. Since $\{K, L\} \subseteq p$ (as by 2) and thus $\{K, L\} \subseteq p_j$, we arrive at a contradiction to the just assumed by setting $p' = p_j$.

Hence Condition 2.(b) cannot be satisfied, and consequently Condition 2.(a) must be satisfied. This means that $M_i\sigma \subseteq M_j$. Recall that derivations have the chain property. From $i \geq k$ we thus get $M_k \subseteq M_i$. Together with $N \subseteq M_k$ then $N\sigma \subseteq M_i\sigma$, and so $N\sigma \subseteq M_j$ follows immediately. Clearly, $N\sigma \subseteq \bigcup_{i \geq 0} M_i$ as well. This completes Case 2.

Notice that in both cases we concluded with 5. Altogether, 3, 4, and 5 contradict the minimality of N.

Hence, the assumption that D is not a refutation must be wrong, and the theorem holds. □

4 Conclusions

In this paper we discussed the drawback of naive connection methods, and, as an improvement, defined a confluent connection calculus and *The CCC Procedure* based on this calculus. We gave a strong completeness proof, and hence proved proof confluence as well.

The CCC Procedure treats matrices, which are sets of clauses. Consequently, the order of clauses does not play any role for fairness or completeness. Hence, more refined proof procedures are free to represent sets as (duplicate-free) lists. A good strategy then would be to append new instances of clauses derived during the saturation phase or during variant steps at the end of the list, because then it is cheap to identify and represent those new closed paths that simply extend previously closed paths[4].

Our calculus achieves confluence by taking into account only derivations which obey a fairness condition. This condition is formulated as an abstract formal property of derivations. It allows to formulate a strong completeness theorem, stating that *any fair* derivation leads to a proof, provided that a proof exists at all.

The difficulty was to define the calculus in such a way that an *effective* fairness condition can be stated. Defining an effective fair strategy is much less straightforward than in resolution calculi (CCC is not commutative, unlike resolution).

That it is not trivial was observed already in [8]. There, a rigid-variable calculus is defined and a strong completeness result is proven. However, the question how to define an *effective* fair strategy had to be left open. Thus, our new approach can be seen to address open issues there.

We came up with a strategy, which is based on a term-depth bound and we proved that this strategy indeed results in fair derivations.

We are aware of the fact, that this effective strategy is only a first step towards the design of an efficient proof procedure based on the CCC-calculus. We expect improvements over "usual" tableaux based implementations of connection calculi, which do not exploit confluence.

This article is a first step, a lot of work remains to be done. In particular the saturation step within our strategy for achieving fairness needs to be turned into a more algorithmic version. Another important topic is to avoid the generation of redundant clauses. To this end regularity, as it is implemented in clausal tableaux would be a first attempt. A further point would be to investigate under which

[4] This strategy resembles much a tableau procedure – one that takes advantage of *confluence*, however.

conditions the variant inference rule can be dispensed with, or how to modify the calculus so that new variants of input clauses are introduced more sparsely.

Acknowledgments. We are grateful to Donald Loveland for comments on an earlier version.

References

1. F. Baader and K. U. Schulz. Unification Theory. In W. Bibel and P. H. Schmitt, editors, *Automated Deduction. A Basis for Applications.* Kluwer, 1998.
2. P. Baumgartner. Hyper Tableaux — The Next Generation. In H. de Swaart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods.* LNAI 1397, Springer, 1998.
3. P. Baumgartner, N. Eisinger, and U. Furbach. A Confluent Connection Calculus. Fachberichte Informatik 23–98, Universität Koblenz-Landau, Universität Koblenz-Landau, D-56075 Koblenz, 1998.
4. P. Baumgartner, U. Furbach, and I. Niemelä. Hyper Tableaux. In *Proc. JELIA 96*, LNAI 1126. Springer, 1996.
5. B. Beckert and J. Posegga. lean$T^{A}P$: Lean tableau-based deduction. *Journal of Automated Reasoning*, 15(3):339–358, 1995.
6. W. Bibel. *Automated Theorem Proving.* Vieweg, 2nd edition, 1987.
7. J.-P. Billon. The Disconnection Method. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 1071. Springer, 1996.
8. F. Bry and N. Eisinger. Unit resolution tableaux. Research Report PMS-FB-1996-2, Institut für Informatik, LMU München, 1996.
9. M. Fitting. *First Order Logic and Automated Theorem Proving.* Texts and Monographs in Computer Science. Springer, 1990.
10. H. Fujita and R. Hasegawa. A Model Generation Theorem Prover in KL1 using a Ramified-Stack Algorithm. In *Proc.*8^{th} *of the Eighth International Conference on Logic Programming*, pages 535–548, Paris, France, 1991.
11. R. Hähnle, B. Beckert, and S. Gerberding. The Many-Valued Theorem Prover 3TAP. Interner Bericht 30/94, Universität Karlsruhe, 1994.
12. R. Hähnle and S. Klingenbeck. A-Ordered Tableaux. *Journal of Logic and Computation*, 6(6):819–833, 1996.
13. R. Hähnle and C. Pape. Ordered tableaux: Extensions and applications. In D. Galmiche, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, LNAI 1227, pages 173–187. Springer, 1997.
14. S. Klingenbeck and R. Hähnle. Semantic tableaux with ordering restrictions. In A. Bundy, editor, *CADE 12*, LNAI 814, pages 708–722. Springer, 1994.
15. D. Loveland. Mechanical Theorem Proving by Model Elimination. *JACM*, 15(2), 1968.
16. R. Manthey and F. Bry. SATCHMO: a theorem prover implemented in Prolog. In E. Lusk and R. Overbeek, editors, *CADE 9*, LNCS 310, pages 415–434. Springer, 1988.
17. D. A. Plaisted and Y. Zhu. Ordered Semantic Hyper Linking. In *Proceedings of AAAI-97*, 1997.
18. D. Prawitz. An improved proof procedure. *Theoria*, 26:102–139, 1960.
19. A. Voronkov. Strategies in rigid-variable methods. In *IJCAI 97*, Nagoya, 1997.