From Schütte's Formal Systems to Modern Automated Deduction

Wolfgang Bibel and Jens Otten

Abstract The paper traces the development of a family of high-performance proof systems out of formal first-order logical systems due to Kurt Schütte. In a first step Schütte's basic classical system is compressed by eliminating its redundancy leading to syntactic characterizations of validity exclusively in terms of formula features including connections. In a second step connection calculi for various logics are developed on the basis of this characterization. Their implementations, leanCoP for classical clausal logic, nanoCoP for classical non-clausal logic, ileanCoP for intuitionistic logic, and MleanCoP for various modal logics, turn out to be among the best performing proof systems internationally. Schütte's influences to Automated Deduction are pointed out by way of this exposition.

1 Introduction

In the history of mankind there has never been a period with such an unprecedented acceleration of changes for humans and human societies as the one of the last eighty years. The driving force behind these changes is the development of Information Technology (IT). It has been argued many times that one of the strongest roots of IT is Mathematical Logic (see eg. [24]).

In the 19th and the beginning 20th century the discipline of logic itself underwent a fundamental change causing a rise in its relevance for other disciplines. Among many others Boole [22] and Frege [28] have contributed substantially to make this change toward a well-founded mathematical discipline happen. In the course of these changes the discipline caused what is known as the foundational crisis of mathematics (Grundlagenkrise der Mathematik) and led to three different

Wolfgang Bibel
Darmstadt University of Technology, e-mail: bibel@gmx.net

Jens Otten
University of Oslo, e-mail: jeotten@ifi.uio.no

logical attempts to overcome it: logicism, intuitionism, and formalism [5]. Hilbert was the main representative of formalism.

Kurt Schütte, the last doctoral student of Hilbert although advised in his PhD work mainly by Hilbert's collaborator Paul Bernays, thereby became an important representative of formalism throughout his career. He interpreted this approach as a structural theory about mathematical proofs and founded the Munich School for Proof Theory. In its initial three years the first author was one of its members and got his PhD with a thesis on cut elimination in Higher-Order Logic during that time under Schütte's supervision.

Recognizing the enormous relevance for the emerging new discipline Computer Science (CS) the first author, after completing his PhD thesis, redirected his research work towards CS, especially on the special subject of Automated Deduction (AD), also known as Automated Theorem Proving (ATP). This area aims at determining the truth of statements represented as formulas in some logic. More specifically, in one line of research within AD this is achieved by proving the validity of these formulas. Determining the validity of formulas, in turn, is realized by proof search procedures evolved from so-called formal systems. This whole endeavour has far-reaching consequences for science and its applications in general and for IT in particular. In the first author's work within this area the formal basis acquired from Schütte turned out to be an excellent framework for the development of powerful deductive systems.

The present paper traces this development, starting with Schütte's formal system GS for First-Order Logic (FOL) and thereafter presenting several internationally leading deductive systems. In more detail, we begin by introducing GS in sections 2.1 and 2.2, thereby pointing out the remarkable compression achieved in comparison with Gentzen's original system. GS's particular features can yet be found in modern AD though under different names, notably the polarities in formulas and the classification into formula types α, β, γ and δ reducing the number of rules drastically in comparison with Gentzen.

In a sequence of steps we further compress GS in order to obtain, through intermediate systems GS_1 (Sect. 2.3) and GS_2 (Sect. 2.4), a formal system GS_3 in Sect. 2.6 with the following remarkable feature: it has no rules at all and its axioms are any valid formulas of First-Order Logic. The validity of its axioms, ie. the axiomhood, is syntactically characterized by structural features inherent in the formulas to be proved (like multiplicities, paths, connections, substitutions). In GS_3 the popular operation of skolemization becomes superfluous through an ordering approach to unification which in a certain sense models the order of inferences in GS.

We show in Sect. 2.5 that from the intermediate system GS_2 there is only a trivial step toward the standard matrix representation or clausal form of formulas which forms the basis for most deductive systems in use. The discussion of such systems will be left for Section 3.

Further in Sect. 2.6 we point out the extra feature of splitting-by-need which can be integrated into GS_3 in order to further enhance its proof power. Similarly, in Sect. 2.7 we point out the importance of the cut for future deductive systems. These

are two of the many topics for future research in AD with the long term goal of yet more powerful proof systems.

Finally, in Sect. 2.8 we briefly cover non-classical formal systems and point out that only on the basis of the ordering approach to unification involved in GS_3 became it feasible to develop proof systems for modal and intuitionistic logics with a performance comparable with those for FOL.

A formal system like GS is generative in nature. It allows to generate valid formulas from its axioms by way of applying its rules while our goal rather is to analyse given formulas. For that purpose the entire Sect. 2 extracts from GS in a stepwise fashion a formal syntactic characterization of the validity of formulas by compressing GS into a practically redundance-free formal system such as GS_3. The resulting characterization then forms the theoretical basis for the development of analytic calculi and of proof systems based on them which, applied to some formula, test it for validity.

Such calculi along with implemented systems are the topic of Sect. 3. Thereby we restrict ourselves to the connection calculi which have proved to be particularly powerful. That is, in contrast to competitive methods the systems described here are based on the Connection Method (CM) [1, 11, 13] which means that they guide the proof search in a formula-oriented and goal-oriented manner determined by the given formula's connection structure.

Sect. 3.1 introduces the simplest such connection calculus which is based on the matrix characterization of logical validity presented in Sect. 2.5 and is restricted to matrices in normal form. A proof system, leanCoP, based on this calculus is described in Sect. 3.4. The PROLOG codes of its simplest version, consisting just of three PROLOG clauses, as well as of an advanced and more powerful version, consisting of four clauses, are presented. Comparable systems are mentioned in passing.

Typically, formulas derived from applications turn out not to be in normal form. Even the best transformations of formulas into normal or clausal form introduce undesired redundancy and thus increase the search efforts needed for finding a proof. In order to avoid these unnecessary search efforts, a non-clausal connection calculus operating on non-normal form matrices has been developed which is described in Sect. 3.2. Its implementation, called nanoCoP, is briefly described in Sect. 3.4.

Following the same methodological approach as for classical logic, connection calculi for non-classical logics, briefly addressed in Sect. 2.8, have been developed which are the topic of Sect. 3.3. These include connection calculi for intuitionistic as well as for a variety of first-order modal logics which are outlined therein. The corresponding implementations are ileanCoP for Intuitionistic Logic and MleanCoP for several first-order modal logics, briefly described in Sect. 3.4. Their codes are so closely related to that of leanCoP that the latter can be obtained from the former by simply deleting parts of it which demonstrates the uniform development methodology underlying our approach to building proof systems. The conclusions in Sect. 4 end the present chapter in this volume.

With this exposition of the development of a range of powerful proof systems out of Schütte's formal systems we demonstrate the influence of Mathematical Logic to IT in a direct way since proof systems have become an essential part in IT. For

instance, the development of correct hardware and software, of the semantic web, and of knowledge systems would not be possible without these kinds of systems to mention but three out of the numerous roles they play in modern technology in one or the other way. The first author has carried this influence from Mathematical Logic into IT in person. He owes Schütte a lot in view of his entire career as this paper, the book [8] as well as his article with reminiscences in this volume show.

2 Schütte's Influences on the History of Automated Deduction

In the present section we outline some of the developments in the last century which eventually led to the deductive systems described in the section thereafter. This is of course not the place to give an outline of the history of the logics underlying those systems nor of the field of Automated Deduction (AD). There are excellent sources in the literature for this purpose. The book [36] gives a comprehensive historical account for the general field of Logic. The article [23] summarizes Logic's history with an emphasis on AD (as well as on its author's work in it). The chapter [16] describes the developments in the early years of AD. In order to point out Schütte's entrance into the course of events, we just mention here the following historical highlights.

Leibniz was the visionary for a computational mechanism to enhance the powers of reasoning [23, pp.2ff, p.14]. Frege's Begriffsschrift [28], with explicit reference to this vision, has laid the grounds for formal languages, logical or programming ones, as well as for logical calculi. Around the 1920's and early 1930's the works of Hilbert, Skolem, Herbrand, Gödel, Gentzen, and Jaśkowski as well as the book by Hilbert and Ackermann [33] clarified the most important logical concepts and issues such as logical calculi, consistency, completeness, decidability, Skolem functions, Skolem-Gödel-Herbrand theorem, Herbrand universe,[1] Gentzen calculi, cut elimination, and so forth. This short list leaves out many important names and contributions and is by no means to be seen as a comprehensive account of the roots of AD.

On this basis, in the mid 1950's, a proof procedure for classical First-Order Logic (FOL) by Quine [68] as well as four new and simplified completeness proofs for FOL by Beth [7], Hintikka [34], Kanger [35] and Schütte [60] were published independently. Each of these papers had an immediate and differing impact on the way early theorem provers were developed and designed. In particular, it was Prawitz et al. who succeeded in developing and implementing the first theorem prover for FOL [55]. While Beth's and Hintikka's systems used proof by contradiction, Kanger and Schütte pursued an affirmative approach. Although the difference between these two lines is completely irrelevant from a logical or deductive point of view, it is a historical coincidence that Prawitz and others decided to follow the contradictory approach which led most later researchers in AD to stay with this choice. The work

[1] For historical correctness the Herbrand universe should actually be named the "Skolem universe" [26].

reported in this chapter is one of the exceptions and has rather followed Schütte's affirmative approach. But this is only a minor point out of the many influences Schütte's work exerted on our work and on AD in general. This impact is outlined in the subsections of the current section, each covering a particular aspect relevant for AD. Most of these aspects are outlined for (classical) FOL and only the final Subsection 2.8 briefly addresses non-classical features.

2.1 The Formula Structure

Gentzen in [29] introduced the structuring of formulas in FOL in the form of "Sequenzen" (sequences). These are structures of the form $A_1,\ldots,A_m \to B_1,\ldots,B_n$ with the antecedents A_i and the succedents B_j, in which A_i and B_j are formulas. Such a sequence is meant to represent the formula $A_1 \wedge \ldots \wedge A_m \to B_1 \vee \ldots \vee B_n$ in FOL. The sequent structure led to a mathematically intuitive and symmetric set of 19 inference rules ("Schlußfiguren") in Gentzen's system LK [29, p.192f]. This structure had, however, also the disadvantageous consequence that proofs for theorems in proof theory (such as Gentzen's Hauptsatz) became rather lengthy because each of these 19 rules required a particular treatment in those.

One of Schütte's remarkable strengths consisted in his incredible ability to minimize proofs. For this purpose he first restricted the set of logical symbols in formulas to a minimal, yet functionally complete one such as $\neg, \vee, \exists$. It is well-known that any other known logical symbol in FOL can be defined with these so that this restriction is without any loss of generality. Then he introduced a different and generalized formula structure, determined by what he called positive and negative formula parts, in the following way [60, 62, 61].

Definition 1. Inductive definition of positive and negative parts of a formula F.

1. F is a positive part of F.
2. If $\neg A$ is a positive part of F then A is a negative part of F.
3. If $\neg A$ is a negative part of F then A is a positive part of F.
4. If $A \vee B$ is a positive part of F then A and B are positive parts of F.

If A is a positive part of F occurring at a particular position in F then this is denoted by $F[A_+]$ and similarly for negative parts using the sign "$-$" instead. Note that the A_i's in Gentzen's sequences above are negative parts and the B_j's positive parts of the corresponding formula under this definition. This shows that Schütte's concepts of positive and negative parts generalize Gentzen's concepts of antecedents and succedents. Later Smullyan in [64] used the notions of signs $0,1$ and signed formulas $\langle F,k\rangle$, $k = 0,1$, for the same purpose and without reference to Schütte. Eventually, Wallen, eg. in [70, p.30], coined the notion *polarity* for denoting a positive or a negative occurrence of a given proposition which since then has become the standard name for pointing out this distinction of occurrences in a formula. But, except for the name, the concept positive/negative originates in Schütte's work for the first time. It has become an important conceptual feature in our field.

2.2 The Formal System GS

Besides the formula structure, discussed in the previous subsection, Schütte's second component for achieving short proofs consisted in his formal system for FOL, a variant of which is presented in the following definition adapted from [60, 62, 61].

Definition 2. Inductive definition of the derivability relation $\vdash_{GS}$, or $\vdash$ for short, in the formal system GS.

1. $\vdash F[P,\neg P]$ holds.
   Thereby F is a well-formed formula of FOL restricted to the logical operators $\neg,\vee,\exists$, in which the *prime* (or atomic) formula P (ie. without any logical symbols) occurs as a positive part of F. By generalizing the brackets notation in an obvious way for more than just one occurrence, the same is true for $\neg P$. Because the definition employs only positive and no negative parts, the index $+$ is omitted for simplicity. Prime formulas and their negated forms are also called *literals*. $\{P,\neg P\}$ is called a *complementary* pair of literals. Formulas of the particular structure depicted are called *axioms* (of GS).
2. If the premises of the following three proof rules are derivable then also the conclusion, depicted as follows.

$$\frac{F[\neg A] \quad F[\neg B]}{F[\neg(A\vee B)]}\ \wedge \qquad \frac{F[\neg A]}{F[\neg\exists a\, A]}\ \forall \qquad \frac{F[\exists x\, A\vee A\{x\backslash t\}]}{F[\exists x\, A]}\ \exists$$

   The attached rule names are chosen to reflect the underlying semantic intention. Rule ($\forall$) is subject to the *variable condition* that *eigenvariable* a must not occur in F except within the occurring part A. $\{x\backslash t\}$ denotes the standard function for (term) *substitution*, where x is substituted by t. We follow the standard notation that $A\sigma$ denotes the application of the substitution σ to the formula A.

Figure 1 shows the derivation of a simple formula in GS for illustration. The complementary pair of literals in each of the two axioms is indicated by a boldface font.

$$\dfrac{\dfrac{\dfrac{\neg\mathbf{Pa}\vee\exists z\,Pz\vee\mathbf{Pa}\vee\neg\exists b\,Qfb}{\neg Pa\vee\exists z\,Pz\vee\neg\exists b\,Qfb}\ \exists}{\neg\exists a\,Pa\vee\exists z\,Pz\vee\neg\exists b\,Qfb}\ \forall \qquad \dfrac{\dfrac{\dfrac{\neg\neg(\exists y\,Qy\vee\mathbf{Qfb})\vee\exists z\,Pz\vee\neg\mathbf{Qfb}}{\neg\neg\exists y\,Qy\vee\exists z\,Pz\vee\neg Qfb}\ \exists}{\neg\neg\exists y\,Qy\vee\exists z\,Pz\vee\neg\exists b\,Qfb}\ \forall}{\neg(\exists a\,Pa\vee\neg\exists y\,Qy)\vee\exists z\,Pz\vee\neg\exists b\,Qfb}\ \wedge$$

Fig. 1 A derivation in GS.

GS is sound and complete, as are similar variants of Schütte's system; see eg. [62, 9, 14]. The denotation GS refers to Gentzen and Schütte.[2] In GS the number of rules

[2] The name GS has first been introduced in [14, Def.7.1]. [67] uses the same name for the same system without giving reference to that much earlier publication.

has been reduced from 19 to just three (or rather four, see Sect. 2.7), simplifying proofs as well as proof programs drastically. This achievement by Schütte has many more advantageous consequences as we will see in the sequel.

Although Schütte himself focused in his research more or less exclusively on proof theoretic issues, implicitly he thereby exerted a strong influence on the work in AD. We already mentioned the line of development from positive/negative parts to polarities in this context. Another important point worth mentioning in the context of GS is the close relationship between Schütte's three rules of inference and Smullyan's uniform notation for signed formulas and their classification into the types α, β, γ and δ [64]. Three of these types coincide exactly with Schütte's three rules of inference while the fourth is incorporated in Schütte's formula structure. In other words, Schütte implicitly introduced these types in the form of his formal system in [60] twelve years, and in the first edition of [62] eight years, before the first edition of Smullyan's book. It is most unlikely that Smullyan had not been influenced directly or indirectly by Schütte's work (although he does not cite it).

It is common in AD to eliminate existential quantifiers with polarity 1 through skolemization (in terms of our restriction to the three logical operators $\neg$, $\vee$, $\exists$). This is because of the derivability of a formula being implied by that of its skolemized form (see eg. [14, Lemma 4.3, p.81]). If we apply this simplification to the system GS then its rule $\forall$ becomes obsolete, leaving GS with just two remaining rules. Figure 2 shows the derivation for the skolemized form of our previous example formula.

$$\dfrac{\dfrac{\neg\mathbf{Pa} \vee \exists z\, Pz \vee \mathbf{Pa} \vee \neg Qfb}{\neg Pa \vee \exists z\, Pz \vee \neg Qfb}\,\exists \qquad \dfrac{\neg\neg(\exists y\, Qy \vee \mathbf{Qfb}) \vee \exists z\, Pz \vee \neg\mathbf{Qfb}}{\neg\neg\exists y\, Qy \vee \exists z\, Pz \vee \neg Qfb}\,\exists}{\neg(Pa \vee \neg\exists y\, Qy) \vee \exists z\, Pz \vee \neg Qfb}\,\wedge$$

Fig. 2 The derivation of the skolemized form of the formula from Figure 1.

These examples demonstrate that in comparison with Gentzen's original systems featuring sequences and 19 rules we have arrived at a much more compact and transparent formalism with just two remaining rules and a simple criterion for testing axioms. Having achieved this much, the natural question arises whether in view of proof search further compressions are possible which will be the topic of the following subsection.

2.3 Compressing the Formal System Further into GS$_1$

Gentzen's motivation in developing his formal systems was the modelling of human reasoning. Schütte optimized Gentzen's systems in view of short human proofs in Proof Theory. AD in contrast aims for systems supporting the search for proofs whereby different criteria apply. Especially, computers have no problems with ap-

plying rules and criteria even if these are too complicated for human use. These are the reasons why the natural next step after – but on the basis of – Schütte's contribution was done no more in the field of Mathematical Logic but rather in AD. This next step can be interpreted in the following way.

As we will see now the rule ($\wedge$) in GS can be dispensed with at the price of complicating the criterion for the axioms. We first generalize the notions of positive and negative part and thereby rename them as 0-part and 1-part for distinction.

Definition 3. Inductive definition of 0-parts and 1-parts of a formula F.

1. F is a 0-part of F.
2. If $\neg A$ is a 0-part of F then A is a 1-part of F.
3. If $\neg A$ is a 1-part of F then A is a 0-part of F.
4. If $A \vee B$ is a 0-part of F then A and B both are 0-parts of F.
5. If $A \vee B$ is a 1-part of F then A and B both are 1-parts of F.

It is obvious that any positive part is a 0-part and any negative part is a 1-part. For readers familiar with the notion of polarity, to be introduced in the next subsection, it is also obvious that any k-part has polarity k, $k = 0, 1$. In other words, this generalized notion has lead us a step closer to the general notion of polarity in AD.

In order to specify a more general notion of an axiom we need the following notion of a path through a formula.

Definition 4. A path through a formula F is a set of literals of F defined inductively as follows.

1. If F is a literal then $\{F\}$ is the only path through F.
2. If $F = \exists x A$ then the empty path $\{\}$ is the only path through F.
3. If $F = \neg\neg A$ then any path through A is also a path through F.
4. If $F = A \vee B$ then $p \cup q$ is a path through F for any path p through A and any path q through B.
5. If $F = \neg(A \vee B)$ then any path through $\neg A$ or through $\neg B$ is a path through F.

A path through F which contains literals of the form P and $\neg P$ is called *complementary*. F is called *complementary* if all paths through it are complementary.

Now let GS_1 be the formal system obtained from GS by the following three changes. First, axioms are any complementary formulas. Second, the notion $F[A]$ in the rules of GS in GS_1 refer to 0-parts rather than positive parts as in GS. Third, only the single rule $\exists$ remains.

For illustration, Fig. 3 shows the derivation of our running example from Fig. 2 in GS_1. There are two paths $\{\neg Pa, Pa, \neg Qfb\}$ and $\{Qfb, Pa, \neg Qfb\}$ through the formula in the top line, each of which contains a complementary pair of literals $\{\neg Pa, Pa\}$ and $\{Qfb, \neg Qfb\}$, respectively, indicated in boldface. Hence the formula is complementary and therefore an axiom in GS_1.

The soundness and completeness of GS_1 for FOL is a straightforward consequence of those properties holding for GS.

$$\dfrac{\dfrac{\neg(\mathbf{Pa} \vee \neg(\exists y\, Qy \vee \mathbf{Qfb})) \vee \exists z\, Pz \vee \mathbf{Pa} \vee \neg\mathbf{Qfb}}{\neg(Pa \vee \neg(\exists y\, Qy \vee Qfb)) \vee \exists z\, Pz \vee \neg Qfb}\ \exists}{\neg(Pa \vee \neg\exists y\, Qy) \vee \exists z\, Pz \vee \neg Qfb}\ \exists$$

Fig. 3 The derivation in GS_1 of the skolemized form of the formula from Fig. 1.

The step from GS to GS_1 may be seen under a different aspect which is elaborated by Guglielmi in [30] under the term "deep inference". Gentzen's rules were applicable only at the surface of each formula in the sequent. Schütte allowed the rules to be applied to positive parts inside the formula and this way from the viewpoint of AD took a first step in the right direction. GS_1 goes a step further in the same direction and allows inferences to be applied to 0-parts. Guglielmi allows his rules to be applied anywhere inside the formula, hence the term "deep inference".

From the point of view of proof search the derivation depicted in Fig. 3 seems to be extremely redundant. This is because at each step backwards from the formula to be proved just a term is associated with an existentially quantified variable while everything else is left unchanged and is therefore carried along as a completely unnecessary burden. This obviously extreme redundancy begs for an improvement by which we get rid of the superfluous burden. This is the task for the following subsection.

2.4 Connections and the System GS_2

The improvement just announced is based on the simple observation that all informations contained in the axioms of the derivation of Fig. 3 can be located already in the formula to be proved. In other words, at least in the case of skolemized formulas considered so far, we may dispense with rules altogether and just redefine the criterion for axioms to apply for general skolemized formulas in FOL right away. This is the topic of the current subsection. We begin by introducing the general notion of polarity.

Definition 5. Inductive definition of the polarity of the occurrence of any subformula of a formula F.

1. F itself has polarity 0.
2. If $\neg A$ is a subformula of F with polarity 0 then A has polarity 1.
3. If $\neg A$ is a subformula of F with polarity 1 then A has polarity 0.
4. If $A \vee B$ is a subformula of F with polarity 0 then A and B both have polarity 0.
5. If $A \vee B$ is a subformula of F with polarity 1 then A and B both have polarity 1.
6. If $\exists xA$ is a subformula of F with polarity 0 then A has polarity 0.
7. If $\exists aA$ is a subformula of F with polarity 1 then A has polarity 1.

Polarities are indicated in formulas by an upper index, in the case of an atom attached either to the parenthesised atom or to its predicate symbol.

As we see, polarities are a straightforward generalization of 0- and 1-parts, and hence also a generalization of Schütte's original positive and negative parts. They simply count the number of negation signs dominating the subformula occurrence in the formula and assign 0 to the subformula if this number is even, otherwise 1.

If we look at the axioms of the derivation in Fig. 3 we see that they still contain existential subformulas so that they could be subject to yet another, in fact to arbitrarily many more inverse applications of rule (∃), each possibly with a different term associated. In order to illustrate this important feature Fig. 4 shows a derivation of the formula $\neg N0 \vee \exists x \neg(\neg Nx \vee Nfx) \vee Nff0$ where a proof can only be found with two such rule applications. The formula expresses that 2 is a natural number if this holds for 0 and any successor. The prime formulas with attached polarities in the axioms are embraced by parentheses for clarity; in the examples thereafter these unnecessary parentheses will be left out.

$$\dfrac{\dfrac{\neg(\mathbf{N0})^{\mathbf{1}} \vee \exists x \neg(\neg Nx \vee Nfx) \vee \neg(\neg(\mathbf{Nf0})^{\mathbf{0}} \vee (\mathbf{Nff0})^{\mathbf{1}}) \vee \neg(\neg(\mathbf{N0})^{\mathbf{0}} \vee (\mathbf{Nf0})^{\mathbf{1}}) \vee (\mathbf{Nff0})^{\mathbf{0}}}{\neg N0 \vee \exists x \neg(\neg Nx \vee Nfx) \vee \neg(\neg N0 \vee Nf0) \vee Nff0}\ \exists}{\neg N0 \vee \exists x \neg(\neg Nx \vee Nfx) \vee Nff0}\ \exists$$

Fig. 4 Derivation with double application of the rule (∃).

As we see from the example, the derived formula after the first inverse inference step is not an axiom, whatever term is chosen. A second step, ie. a second copy of the existentially quantified subformula, is needed to derive a proof for this theorem. In fact, if we would want to prove Nn for an arbitrarily large n, we would need n copies. In order to cope for this necessity we introduce the multiplicity function on formulas in the following definition.

Definition 6. For any formula F of FOL (built with the logical symbols $\neg, \vee, \exists$) a *multiplicity* μ is a function which assigns to each occurrence of a subformula $\exists xA$ of polarity 0 a natural number $m \geq 0$. F along with μ is also written F^{μ}.

With multiplicities and polarities we are now in a position to generalize also the intermediate notion of path from Def. 4. Since the intermediate notion will not be used any further and turns out to become a special case, we use the same name for the generalized notion.

Definition 7. A *path* through a skolemized formula F^{μ} is a set of prime (or atomic) subformulas of F along with their polarities and is defined inductively as follows.

1. If F is a prime formula with polarity k then $\{F^k\}$ is the only path through F.
2. If $F = \neg A$ then any path through A is also a path through F.
3. If $F = A \vee B$ with polarity 0 then $p \cup q$ is a path through F for any path p through A and any path q through B.
4. If $F = A \vee B$ with polarity 1 then any path through A or through B is a path through F.

5. If $F = \exists xA$ with polarity 0 and multiplicity m then $p_1 \cup \ldots \cup p_m$ is a path through F for any paths p_i through $A\{x\backslash x_i\}$, $i = 1, \ldots, m$.

An unordered pair of elements in a path through F^μ with identical predicate symbols but differing polarities is called a *connection*. A set U of connections is called *spanning* for F^μ if each path through F^μ contains at least one connection of U. If a connection consists of two identical prime formulas with differing polarities, then it is called *complementary*. A skolemized formula F is called *complementary* if there is a multiplicity μ and a substitution σ such that $p\sigma$ contains a complementary connection for any path p through F^μ.

Let GS_2 be the formal system for skolemized FOL which has complementary formulas as its axioms and no rules at all.

We first note that the concept of path in this definition generalizes that of Def. 4 in the following way. First, the more special concept from Def. 4 only applies to formulas with a multiplicity which is 0 throughout. Second, here we use polarities to identify connections rather than pairs of literals as before. From now on the notion of path will refer exclusively to the present definition unless stated otherwise.

A proof of a formula F in GS_2 is characterized by a multiplicity μ, a substitution σ, and a set U of connections, referred to as a *connection proof*. Figure 5 shows such a connection proof for our running example. The multiplicity for each existential subformula is chosen to be 1, in which case it is not noted explicitly in the proof by default, but only by way of the index 1 attached to the variables y and z in the substitution. As before there are two paths through F each containing exactly one of the two depicted connections, ie. these are spanning. Along with the given substitutions these connections are complementary. Hence the formula is an axiom, thus establishing the proof. The reader might wish to compare this proof with the equivalent proofs given in the figures 1, 2, and 3. What has remained unchanged in all four versions is of course the derived formula, but also the two connections are exactly the same in all of them. This tells us that the formula along with a multiplicity, a substitution, and a spanning and complementary set of connections, ie. a connection proof, is all that is needed for a full-fledged guarantee for validity. The connections form the propositional, the multiplicity and the substitution the first-order part of the proof.

$$\neg(P^1 a \vee \neg\exists y\, Q^0 y) \vee \exists z\, P^0 z \vee \neg Q^1 fb \text{ with } \sigma = \{y_1\backslash fb, z_1\backslash a\}$$

Fig. 5 The connection proof for the skolemized formula from Fig. 1.

Thus we have achieved the complete elimination of any redundancy from the proof representation without any loss of information. This way we have *compressed* the formal system in the sense defined in [15, Sect. 6]. Validity of a formula is now characterized in terms of its structural features. All theorems have become axioms.

What needs to be done in view of finding a proof in GS_2 is, first, locating the set of connections in the formula which can be done fast. From this set and for a chosen multiplicity then a spanning subset of connections and a substitution has to be determined which renders the subset complementary, a task which is trivial in the present example but may be exponentially difficult in general since proof search is co-NP-complete even in the restriction to propositional logic, as is well-known. Any calculus which solves this task with a focus on connections or connection-structures is called a *connection calculus*. The general approach to determine validity through some connection calculus is termed the *connection method* (or connection principle). Apart from the lack of redundancy the main advantage of the connection method in comparison with other proof methods is its formula-oriented approach in a search space which is clear-cut by the very structure of the given formula (rather than by many additionally generated formula parts as in competitive methods, such as resolution or instance-based methods). In consequence, the space required in larger examples amounts only to a tiny fraction of that required by other methods.

Figure 6 shows the connection proof for our second example. The multiplicity of the existential subformula is 2, depicted by the upper index 2 attached to the existential quantifier. We therefore get two different variables x_1 and x_2 originating from x. The formula has 4 connections of which one connects terms which are not unifiable under any substitution, hence can be ignored. The remaining three are depicted in the figure. There are four paths through the formula and each contains one of these connections, hence these are spanning. The depicted substitution renders all three connections complementary, so that the formula is in fact an axiom and thus the formula is provable in GS_2.

$$\neg N^1 0 \vee \exists^2 x \neg(\neg N^0 x \vee N^1 fx) \vee N^0 ff0 \quad \text{with} \quad \sigma = \{x_1 \backslash 0, x_2 \backslash f0\}$$

Fig. 6 The connection proof for the formula from Fig. 4.

Recall that the formula in the figure expresses that 2 is a natural number. Had we chosen 3 instead of 2 then for a proof we would have to choose multiplicity 3 and we would get a second instance of the middle connection with the multiplicity indices of the connected literals increased by 1 in comparison with the first instance. This illustrates the fact that the original connections of a formula may lead to arbitrarily many instances of any of these in order to achieve a proof.

The following theorem is now a straightforward consequence of the soundness and completeness of GS. In fact, any proof for a skolemized formula in GS can easily be reduced to a connection proof in GS_2 as illustrated with our examples. Conversely, any connection proof in GS_2 can immediately be extended to a proof in GS. Our way of illustrating the development of GS_2 in this section has traced the close relationship between GS and GS_2.

Theorem 1. *GS_2 is sound and complete for skolemized FOL.*

We are not yet done completely because of the restriction to skolemized formulas, a restriction to be lifted in Sect. 2.6. Before turning to this alternative technique let us simplify matters somewhat in the next subsection.

2.5 The Matrix Representation

Quantifiers let formulas look complicated. That is why in AD for the daily practice they have been eliminated completely. We already achieved the first step in this process by eliminating (existential) quantifiers with polarity 1 through skolemization. Here we now take the second step in this process.

A skolemized formula (with the restricted variety of logical operators considered here) has only quantifiers with polarity 0 if any. Without restricting generality we can assume that for any two different quantifiers in the formula the variables bound by them are denoted differently. Otherwise this can be achieved by separating them apart through renaming of bound variables. Let us assume throughout this section that bound variables are separated apart in this way.

Now we transform any given skolemized formula into its equivalent prenex form.[3] In the case of our example formula $\neg(P^1a \vee \neg\exists y\, Q^0y) \vee \exists z\, P^0z \vee \neg Q^1fb$ this transformation results in $\exists y\exists z(\neg(P^1a \vee \neg Q^0y) \vee P^0z \vee \neg Q^1fb)$ where the ordering of the quantifiers is chosen arbitrarily. As we can see from this example the quantifiers have become redundant in this form since they are uniquely determined through the bound variables in the formula (ie. y and z in the example). Therefore the quantifiers can be deleted by default. Thus we obtain $\neg(P^1a \vee \neg Q^0y) \vee P^0z \vee \neg Q^1fb$.

Recall from the previous subsection that we are looking for a spanning set of connections in such a formula, a notion which is based on that of paths through it. It would be nice to have a representation of formulas which exhibits paths more visibly. This motivation has led to representing formulas as matrices in which disjunction is displayed horizontally and conjunction vertically (in our affirmative approach while in the contractory approach it is just the other way around).

In our formal system conjunction is represented as disjunction with polarity 1. For the purpose of more comfortable visibility aimed at in the present subsection we will reintroduce conjunction explicitly, thus yielding for our example $(\neg Pa \wedge Qy) \vee Pz \vee \neg Qfb$ (whereby polarities are dropped as obvious). As a matrix this formula looks like shown in Fig. 7.

All notions from Def. 7 carry over from formula to matrix in a straightforward way. If the represented formula is in disjunctive normal form, which happens to be the case in our example, then a path through such a matrix can be regarded as a walk through it strictly from left to right, thereby collecting exactly one literal from each column. If this is done for our example matrix, we see that it has exactly

[3] In view of efficiency of proof search care has to be taken in how to carry out this transformation in detail, an issue not pursued any further here.

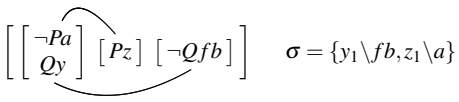

Fig. 7 The connection proof from Fig. 5 in matrix representation.

two different paths, $\{\neg Pa, Pz, \neg Qfb\}$ and $\{Qy, Pz, \neg Qfb\}$. Each of them contains a connection which is complementary under the attached substitution. Hence the two depicted connections are spanning and complementary; thus the matrix is complementary; hence the formula from which it derives is valid.

Eliminating the quantifier and replacing the disjunction with polarity 1 by conjunction in our second example $\neg N0 \vee \exists x \neg(\neg Nx \vee Nfx) \vee Nff0$ similarly as before yields $\neg N0 \vee (Nx \wedge \neg Nfx) \vee Nff0$. It again happens to be in disjunctive normal form. Its matrix representation and a connection proof for it is shown in Fig. 8.

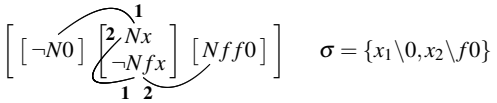

Fig. 8 The connection proof from Fig. 6 in matrix representation.

For the resulting quantifier-free formulas in disjunctive normal form each disjunct is called a *clause*. Thus in this special kind of formula a clause is a conjunction of literals in our affirmative approach (while in the popular contradictory approach it is a disjunction of literals). If formulas are restricted to this special case then we say that these as well as their matrix representations are in *clausal form*. Most popular theorem provers in AD operate on formulas in clausal form. This is without restriction of generality since any formula can equivalently be transformed into clausal form. This transformation may affect the efficiency of the prover, however.

In order to cope with this disadvantage we note that Def. 7 applies to arbitrary skolemized formulas, not only to those in disjunctive normal form, ie. clausal form. In general, each conjunct might therefore itself be a more involved formula rather than simply a literal. This leads to nested matrices by way of the transition to the matrix representation. A path through such a matrix is then still visualized by a literal collecting walk through the matrix from left to right, although a more complicated one observing the nesting structure of the matrix (see Sect. 3.2). Since the transformation to clausal form introduces redundancy into the formulas, engaging it is contrary to our main objective of excluding redundancy as much as possible, a point to be kept in mind for Sect. 3 where we will present proof systems both for clausal form (Sect. 3.1) as well as for non-clausal form (Sect. 3.2).

The connection method applied to formulas in matrix representation is often also called *matrix method*. The matrix method has become popular because the two-

dimensional representation of formulas is very helpful for the human intuition. From a technical viewpoint, however, the difference between the representation of formulas in a one-dimensional form and that in the form of two-dimensional matrices is completely irrelevant. What counts technically is exclusively the focus on the connection structure determined by the given formula represented in whatever form. It is for this reason that we prefer "connection method" to "matrix method" as the characterizing term for this approach.

2.6 Alternative to Skolemization (GS_3) and Splitting by Need

The previous subsection sort of interrupted our line of development in this section in order to link it with the standard clausal form which is in common use in AD. This is also to say that the community in AD has got stuck at this form of representation although there is plenty of room for further optimization. The clause form is far simpler for developers of proof systems to work with than the formulas as they derive directly from applications in mathematics, program verification etc. However, it would be much better for the proof systems to work right with the original formulas rather than a version which just provides more comfort to the developers. This is because the clause form possibly increases the search space and thus renders the search efforts unnecessarily more complex.

Therefore, after this interruption, we resume here the line of our development up to Sect. 2.4 which has resulted there in the formal system GS_2. Recall that this system is already defined for arbitrary skolemized formulas. Could this result be generalized by doing even without skolemization? This is the topic for the present subsection.

Unfortunately, the technical details for this generalization of GS_2 are too involved to be presented in all details in this chapter. We refer to [12, Theorem H_3 and H_4], [14, Corollary IV.8.5 and Theorem IV.10.4] and [2, 3] for more details and for proofs. Here we try to present the gist of the solution.

Recall our example formula $\neg(\exists a\, Pa \vee \neg\exists y\, Qy) \vee \exists z\, Pz \vee \neg\exists b\, Qfb$ from Fig. 1 in its original form. How could a connection proof like that for its skolemized form in GS_2, shown in Figure 5, be obtained right away for the original formula?

In order to solve this problem we have to identify the feature in GS which takes care of the role played by skolemization. This is in fact the variable condition accompanying Rule ($\forall$) in Def. 2 which applies to existential quantifiers with polarity 1. It restricts the choice of a free variable as we apply the rule in an inverse direction. This restriction in view of proof search has exactly the same effect as skolemization [9]. So the problem just stated reduces to the problem of incorporating this variable condition into the transition from GS to GS_2 in order to arrive at a formal system GS_3 with connection proofs for arbitrary formulas.

Figure 9 shows what could be a connection proof for our example formula. Is it a correct proof according to our envisaged GS_3? Yes, it happens to be one which gives us a first feel for where we are heading to. In order to be a correct connection proof

the shown substitution σ has to fulfill a certain criterion to be discussed shortly which happens to be the case in this simple example right away.

$$\neg(\exists a\, P^1 a \vee \neg\exists y\, Q^0 y) \vee \exists z\, P^0 z \vee \neg\exists b\, Q^1 f b \quad \text{with} \quad \sigma = \{y_1 \backslash f b_1, z_1 \backslash a_1\}$$

Fig. 9 The connection proof for the formula from Fig. 1.

Due to the (eigen-)variable condition of Rule ($\forall$) in GS variables bound by existential quantifiers with polarity 1 play a completely different role from those bound by existential quantifiers with polarity 0. In order to visualize this distinction the former are denoted by $a, b, \ldots$ in GS and are called *constants* (or constant variables) while the latter are denoted by $x, y, \ldots$ and are called (free) *variables*.

If we now compare Fig. 1 with Fig. 9 we first note that the former is condensed in the latter into just two connections along with a substitution without any loss of information relevant for the proof. Where then has the relevant information, that in Fig. 1 the two rules ($\exists$) are above the corresponding ($\forall$) rules, gone in the connection proof? This information is indeed relevant since only then the variable condition is fulfilled. This information is hidden in the already mentioned criterion for the substitution which we now introduce with the help of the following definition.

Definition 8. Let $<$ denote the tree ordering among the occurrences of the subformulas of a formula which itself denotes the root of $<$. This ordering induces (via the corresponding quantified subformulas) also an ordering among the constants and variables in the formula which is denoted also by $<$.

For any substitution σ an equivalence relation $\sim$ on the set of variables in σ and a partial ordering relation $<\cdot$ on the set of constants and variables in σ is defined as follows.

1. If $x \backslash y \in \sigma$ then $x \sim y$ holds.
2. If $x \backslash t \in \sigma$ where t is not a variable, then $y \sim x$ holds for any variable y occurring in t and $c <\cdot x$ holds for any constant c occurring in t.
3. If $c <\cdot x$ and $x \sim y$ then also $c <\cdot y$.

Let $\lhd$ denote the transitive closure of the union of $<$ and $<\cdot$. We say that σ has no cycles, ie. is irreflexive, if this is the case for $\lhd$.

A formula in FOL with multiplicity μ is called *complementary* if there is a substitution σ without cycles such that $p\sigma$ contains a complementary connection for any path p through F^μ.

Let GS_3 be the formal system for FOL which has complementary formulas as its axioms and no rules at all.

For the substitution in Fig. 9 we get $b_1 \lhd y_1$ and $a_1 \lhd z_1$ according to this definition. In this ordering we find the relevant information that we mentioned just before the definition which expresses that the ($\exists$) rules must occur above the ($\forall$) rules in the

corresponding GS derivation. It is obvious that $\lhd$ in this example does not have cycles so that the formula is complementary. Hence it is also valid. This is because GS_3 is sound and complete according to the references given above, eg. [14, Coroll. 8.5]. As a matter of fact, the equivalence relation $\sim$ can be dispensed with if only idempotent substitutions are considered, a restriction that does not affect completeness.

In order to illustrate the cycle condition with a non-theorem consider the formula $\exists x \neg \exists a \neg (\neg P^1 ax \vee P^0 xa)$ with multiplicity 1. The only possible connection leads to the substitution $\sigma = \{x \backslash a\}$ (or rather to $\sigma = \{x_1 \backslash a_1\}$, but we drop the indices for simplicity here). Hence we get $a <\cdot x$, and therefore $a \lhd x$. The tree ordering of the formula gives $x < a$, and therefore $x \lhd a$. By transitivity of the ordering we get $x \lhd a \lhd x \lhd a$, ie. the ordering, and thus the substitution, is cyclic so that this connection along with the substitution does not constitute a "proof" for this obviously invalid formula.

Once again we note that this *ordering approach* to unification realized in GS_3 via the relation $\lhd$ plays exactly the same role as the interplay of skolemization with unification. The latter has dominated the work in AD since shortly after its beginnings. But only through the insight obtained by the ordering approach became it possible much later to develop effective proof systems for non-classical logics, a fact which Wallen in [70, p. 3] phrases as follows: *"Of most note is the proof-theoretical analysis of unification that emerges ... It is this rediscovery of the rôle of unification that forms the foundation for the matrix characterisations of non-classical logics in the sequel."*.

The ordering approach was introduced by Prawitz in [54] for the first time and refined in [9] and in later papers of the first author of the present paper. The influence of Schütte's paper from 1956 on this early work by Prawitz is clearly recognizable in it and the paper is of course also cited in Prawitz's paper. The same is of course the case for the cited paper of the first author of the present paper. So there is a direct line of influences from Schütte to the ordering approach to unification.

By the way, the test for cycles required in GS_3 is no extra burden in comparison with other proof methods since any unification algorithm needed in any such a method also requires the analogue cycle test [14, Sect. IV.9], a fact which again illustrates the close relationship between these two approaches.

There is an additional possibility to minimize connection proofs and thus enhance their compactness which is known as *splitting by need*. Consider the simple formula $\exists \neg P^1 x \vee \neg (\neg P^0 a \vee_1 \neg P^0 b)$ with multiplicity 1 in which we distinguish the occurrence of one of the disjunction symbols by attaching an index 1. There are two paths through it, each containing a connection. But x_1 is not unifiable with both a and b at the same time. Thinking in terms of finding a proof in GS, however, we could first apply the inverse of Rule ($\wedge$) leading to two independent subformulas each provable immediately. In terms of GS_3 this can be simulated by including positions of disjunction symbols with polarity 1 like $\vee_1$ in the example into the relation $<\cdot$ and by considering it in the cycle test. In our example the two connections would become complementary through the additional requirement $\vee_1 <\cdot x$ generated "by need" upon noticing the conflict between a and b in the unification attempt.

In full generality this technique is quite involved. It has first been explored in [21] and in more details in [14, Sect. IV.10]. Further work on it has been done in [2, 3, 32], but in a simpler context than that provided by GS_3. If the technique is incorporated into GS_3, it allows, for instance, a straightforward 4-connection proof for the formula $\neg\exists b\neg(\neg\exists a\neg\exists z\neg(\neg\exists x\,Paxz \vee_1 \neg\exists y\,Pbyz) \vee \neg(\exists c\neg\exists u\,\neg Puuc \vee_2 \exists d\neg\exists v\,\neg Pvbd))$ which is taken from [21]. If its multiplicity is restricted to 1 then the formula has four paths and four connections resulting in a straightforward substitution. By need the requirements $\vee_2 <\cdot\, z$, $\vee_1 <\cdot\, u$ and $\vee_1 <\cdot\, v$ are generated with which the resulting substitution becomes free of cycles thus establishing the formula's validity. The reader might wish to compare this 4-steps proof search with that of his/her favorite proof method. Of course, the formula is artificially constructed in order just to demonstrate the effects in a small example.

2.7 Cuts

In Sect. 2.1 we referred to Gentzen's formal system LK featuring 19 rules of inference. These rules include the cut (or Schnitt) rule which has not been included in GS of Sect. 2.2. This omission is justified by Gentzen's Hauptsatz which states that any proof in GS (or in LK for that matter) with cut can be reduced to a proof without cut. This cut elimination leads to longer proofs, in fact exponentially longer proofs in the worst case [67, Sect. 5.2]. This increase of the proof lengths is therefore an important issue for proof search since searching for exponentially long proofs is hopeless to begin with. Can we expect any improvement of this situation from the point of view of GS_3? The present section is concerned with this question.

The cut rule in GS may be written in the following way.

$$\frac{F[\neg C] \quad G[C]}{F[] \vee G[]}\ \mathit{cut}$$

Here $F[]$ denotes the formula obtained from $F[\neg C]$ by deleting $\neg C$ and with it any symbols which in view of a well-formed formula have become superfluous thereby, and similarly for $G[]$.

In order to view the cut rule in terms of GS_3 let us first assume for simplicity that C is a prime formula. Now assume further that we have connection proofs in GS_3 for the two premises of the rule. Then we could construct a connection proof in GS_3 also for the conclusion in the following way. If the cut formula $\neg C$ is not contained in a connection of the connection proof for the left premise then this connection proof is also a connection proof for the conclusion (and similarly for the right premise). Otherwise, if the cut formula is contained in a connection of the connection proof for each premise, then let $c_1 = \{C_1, \neg C\}$ be such a connection in the left premise and $c_2 = \{C, \neg C_2\}$ one in the right premise. For any pair of such connections in the premises we add the connection $\{C_1, \neg C_2\}$ to the union of all connections in the premises not involving the cut formula resulting in a set of connections to be considered in the conclusion. This set along with the union of the substitutions for

the premises is a connection proof of the conclusion in GS_3, as can be shown in a straightforward way by looking at the paths through F and G. This cut elimination step can be generalized to cut formulas other than literals. In this manner cuts may be eliminated from a formal system GS_3^c that includes the cut rule, thus resulting back in GS_3. (Formally, this cut elimination in GS_3^c is proved by induction on the number of cuts in the derivation and by a subinduction on the length of the cut formula; when the cut formula is a disjunction with negative polarity the cut is replaced by two cuts on the disjunctive parts; when it is an existential formula with negative polarity, it is replaced by m cuts where m denotes the multiplicity of the quantifier.)

The first observation from the point of view of this insight shows us how the exponential explosion may occur in this process of cut elimination. Namely, let m_1 be the number of connections containing the left cut formula and m_2 the one for the right one, assumed to be literals for simplicity. Then the elimination step just described yields $m_1 \cdot m_2 - (m_1 + m_2)$ additional connections in the connection proof of the conclusion as compared with the sum of connections in the two proofs of the premises. A corresponding result is obtained for non-literal cut-formulas.[4] If we have k cut rules in the derivation this amounts to an increase of the order of n^k additional connections for some n, in accordance with the exponential increase known from other formal systems discussed in the reference given above. But a precise result in this respect for the situation given by a system such as GS_3^c is currently not known.

This increase can be avoided if the set of connections containing a cut formula is handled as a connection structure in the sense of [19, Sect. 2.10]. Although there has been some work in this direction and generally on cuts in the context of proof search such as [25, 37] and the references therein, the issue has remained a major challenge for AD. We may, for instance, remind the community in this context of the conjecture posed in [15, Sect. 6.1].

2.8 Non-Classical Formal Systems

Up to now in this section we have focused exclusively on classical FOL. In the present subsection we will briefly extend the view also to include non-classical logics. Again, Schütte was among the first logicians who presented compact completeness and soundness proofs for intuitionistic and modal first-order logics [61], shortly after Kripke had introduced his uniform semantics for these logics. This book by Schütte seems to have had no direct influence on proof systems for non-classical logics. For instance, the book [70] does not cite it.

But since this work of Schütte had its roots firmly in his own work on classical FOL, it indirectly exerted its influence also on the later work on proof systems for non-classical logics through all his contributions pointed out in the preceding sub-

[4] Cut formulas may be regarded as mathematical lemmata. What is a good lemma? In fact, $m_1 + m_2$ may be taken as a concrete measure for the goodness of the lemma (in the sense "the bigger the better").

sections such as polarity, types of inferences and especially the ordering approach to unification. Recall the quotation of Wallen in Sect. 2.6 in which he pointed to the fundamental insight through the ordering approach leading to his genious solutions for intuitionistic and modal proof systems.

Namely, in Gentzen-type systems for non-classical logics the role of modal operators is analogous to that of quantifiers in terms of the rules of inference and their ordering. Therefore their treatment can be integrated into the unificational part by way of the ordering approach in full analogy with the treatment of the quantifiers described in Sect. 2.6. Thus the non-classical features affect solely the details involved in the relation $\lhd$ to be adapted appropriately. Further details about this integration as well as the resulting proof systems for non-classical logics will be presented in Sect. 3.3 of the present chapter.

Thus the theoretical basis for the research program outlined in the preceding subsections can be transferred to many non-classical logics in a uniform way on the basis of Wallen's work. As far as the formal systems for those logics were concerned Wallen was directly influenced mostly by the book [27] of Fitting who of course was closely familiar with Schütte's work.

3 Modern Connection Calculi

The previous section has presented the theoretical basis for proof systems in FOL. It has done so in a stepwise fashion whereby at each step the underlying formal system was compressed somewhat further by eliminating redundancy left over in its predecessor. The field AD so far has not succeeded in producing a proof system for any logic based on the most compressed system GS_3. It has up to this point in time produced systems which are based on the intermediate formalism described in Sect. 2.5 and which are mostly restricted to clause form. This intermediate formalism is still quite redundant in comparison with GS_3 so that there remains a major challenge ahead for the field to develop systems on the basis of GS_3 including the features discussed in sections 2.6 and 2.7.

Nonetheless, the systems obtained so far on this intermediate basis are already quite impressive in their performance. One of these systems, leanCoP, will be described in the present section. It has been developed very closely in line with the research program outlined in the previous section and insofar was influenced at least indirectly by Schütte's work as well.

The first step towards a proof system on the basis of the formalism presented in Sect. 2.5 consists in the development of a calculus which determines the search for a spanning set of connections. Calculi serving this purpose are called *connection calculi* as already pointed out in Sect. 2.4. Examples of this kind of calculi are those presented in [11, 13, 14], the connection tableau calculus [39], and the model elimination calculus [40]. Proof search in a connection calculus is guided by connections, the structural elements within the formula to be proved which were introduced in the previous section. This guidance along the formula's structural features, which

are crucial for establishing a proof, renders the search much more goal-oriented compared to the proof search in sequent, tableau or resolution calculi. Furthermore, compared to GS_3 presented in Sect. 2.5, these calculi contain, eg., representational redundancies that cannot be completely eliminated by using structure-sharing techniques. It has also been shown that a slight variation of the connection calculus in form of the *connection structure calculus* [19, Sect. 2.10] can simulate resolution in linear time. Thus future connection calculi incorporating all such additional features might have the potential to outperform any of its competitors.

This section first introduces a clausal connection calculus for classical FOL. Afterwards, a non-clausal connection calculus and connection calculi for several popular non-classical logics are presented. Finally, leanCoP and versions of leanCoP for classical and some non-classical logics that are based on the given calculi are described. leanCoP's core program consists of just four PROLOG clauses and yet features a performance competitive with some of the best provers worldwide.

3.1 The Clausal Calculus

The connection calculus for classical logic to be introduced now is based on the matrix characterization of logical validity presented in Sect. 2.5. Recall that this characterization established a *connection proof* by identifying a spanning set of unifiable connections for the given matrix as illustrated in Fig. 10.

$$\left[\,[P^0 z]\ \begin{bmatrix}P^1 a\\ Q^0 y\end{bmatrix}\ [Q^1 f b]\,\right] \quad \sigma = \{z\backslash a\}\ , \qquad \left[\,[P^0 z]\ \begin{bmatrix}P^1 a\\ Q^0 y\end{bmatrix}\ [Q^1 f b]\,\right] \quad \sigma = \{z\backslash a, y\backslash f b\}$$

Fig. 10 The connection proof for the formula $Pz \vee (\neg Pa \wedge Qy) \vee \neg Qfb$.

Determining a spanning set of unifiable connections in general is a computationally complex task. We describe the solution to this task by way of an indeterministic logical calculus which allows the flexibility for certain strategical choices in the search. The idea behind its derivational mechanism is to identify connections in a systematic order which guarantees that eventually all paths through the matrix are covered. Once a connection is selected then only paths that do not contain this connection are investigated afterwards. If every path contains a connection, the proof search succeeds and the given formula is valid. At each step the substitution is updated to make each selected connection complementary. For example, the proof of the previous matrix consists of two inferences, which identify two connections resulting in the shown substitution.

A formal description of a connection calculus was given by Otten and Bibel [51] and is presented in the following definition.

Definition 9. The axiom and the rules of this *clausal connection calculus* are given in Fig. 11, in which $\overline{L_2}$ denotes the complement of L_2, ie. $\{L_1, L_2\}$ is a complementary connection for the substitution σ. The words of the calculus are tuples of the form "$C, M, Path$", where M is a matrix, C and $Path$ are sets of literals or ε; C is the *subgoal clause*, $Path$ is the *active path*, and σ is a *rigid* term substitution (which is applied to the whole derivation). A *clausal connection proof* of a matrix M is a clausal connection proof of $\varepsilon, M, \varepsilon$.

Axiom (A)	$\dfrac{}{\{\}, M, Path}$	
Start (S)	$\dfrac{C_2, M, \{\}}{\varepsilon, M, \varepsilon}$	and C_2 is copy of $C_1 \in M$
Reduction (R)	$\dfrac{C, M, Path \cup \{L_2\}}{C \cup \{L_1\}, M, Path \cup \{L_2\}}$	and $\sigma(L_1) = \sigma(\overline{L_2})$
Extension (E)	$\dfrac{C_2 \setminus \{L_2\}, M, Path \cup \{L_1\} \quad C, M, Path}{C \cup \{L_1\}, M, Path}$	and C_2 is a copy of $C_1 \in M$, $L_2 \in C_2$, and $\sigma(L_1) = \sigma(\overline{L_2})$

Fig. 11 The clausal connection calculus for classical logic.

The proof of the matrix $M_1 = \{\{P^0 z\}, \{P^1 a, Q^0 y\}, \{Q^1 fb\}\}$ in the clausal connection calculus with $\sigma = \{z' \backslash a,\ y' \backslash fb\}$ is given in Fig. 12. The clausal connection calculus always uses *copies of clauses*, in which all variables are renamed. By this technique the concept of multiplicity introduced in Def. 6 is realized in a simplified way by expanding copies explicitly.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{\}, \{\{P^0 z\}, \{P^1 a, Q^0 y\}, \{Q^1 fb\}\}, \{P^0 z', Q^0 y'\}}\, A \qquad \dfrac{}{\{\}, M', \{P^0 z'\}}\, A}{\{\mathbf{Q^0 y'}\}, \{\{P^0 z\}, \{P^1 a, Q^0 y\}, \{\mathbf{Q^1 fb}\}\}, \{P^0 z'\}}\, E \qquad \dfrac{}{\{\}, M', \{\}}\, A}{\{\mathbf{P^0 z'}\}, \{\{P^0 z\}, \{\mathbf{P^1 a}, Q^0 y\}, \{Q^1 fb\}\}, \{\}}\, E}{\varepsilon, \{\{P^0 z\}, \{P^1 a, Q^0 y\}, \{Q^1 fb\}\}, \varepsilon}\, S$$

Fig. 12 A proof in the (formal) connection calculus.

The presented clausal connection calculus is *correct* and *complete*, ie. a formula is valid in classical logic if and only if there is a clausal connection proof for its matrix [14]. The proof is based on the matrix characterization for classical logic.

Proof search in the clausal connection calculus is carried out by applying the rules of the calculus in an *analytic* way, ie. from bottom to top, starting with $\varepsilon, M, \varepsilon$, in which M is the matrix of the given formula. At first a start clause is selected. Afterwards, connections are successively identified in order to make sure that all

paths through the matrix contain a σ-complementary connection. This process is guided by the active path, a subset of a path through M. During the proof search, backtracking might be required, ie. alternative rules or rule instances have to be considered if the chosen rule or rule instance does not lead to a proof. This might happen when choosing the clause C_1 in the start and extension rules or the literal L_2 in the reduction and extension rules. The term substitution σ is calculated step by step by one of the well-known *term unification algorithms* (see, eg. [58]) whenever a reduction or extension rule is applied.

3.2 The Non-clausal Calculus

Clausal connection calculi, such as the one presented in Sect. 3.1, require the input formula in disjunctive normal or clausal form. Formulas that are not in clausal form have to be translated into clausal form. The standard transformation translates a first-order formula F into clausal form by applying the distributivity laws. In the worst case, the size of the resulting formula grows exponentially with respect to the size of the original formula F. This increases the search space significantly. Even a definitional translation [53] that introduces definitions for subformulas introduces a significant overhead for the proof search [44] and both clausal form translations modify the structure of the original formula.

A *non-clausal* connection calculus [45] that works directly on the structure of the original formula does not have these disadvantages. Other non-clausal approaches [1, 14, 31] work only on ground formulas, ie., for formulas that do not contain variables. For first-order formulas, these approaches have to add copies of subformulas iteratively, which introduces a huge redundancy into the proof search. For a more efficient proof search, clauses have to be added dynamically during the proof search, similar to the approach used for copying clauses in clausal connection calculi. Hence, in the following the clausal connection calculus is generalized and its rules are carefully extended [45].

Definition 10. The *non-clausal matrix* $M(F^{pol})$ of a formula F^{pol} (where *pol* is a polarity) is a set of clauses, in which a clause is a set of literals and (sub-)matrices, and is defined inductively according to Table 1. In Table 1, x^* is a new variable, t^* is the Skolem term $f^*(x_1,\ldots,x_n)$ in which f^* is a new function symbol and $x_1,\ldots,x_n$ are the free variables in $\forall xG$ or $\exists xG$. The *non-clausal matrix* of a formula F is the matrix $M(F^0)$.

In the *graphical representation* the clauses of matrices are arranged horizontally, literals and matrices of clauses are arranged vertically. For example, the formula

$$(Pz \vee (\neg Pa \wedge Qy) \vee \neg Qfb) \wedge ((N0 \wedge \forall x(Nx \Rightarrow Nfx)) \Rightarrow Nff0)$$

has the simplified non-clausal matrix (in which redundant parentheses are omitted)

$$\{\{\ \{\{P^0z\},\{P^1a,Q^0y\},\{Q^1fb\}\}\ ,\ \{\{N^10\},\{N^0x,N^1fx\},\{N^0ff0\}\}\ \}\}\ .$$

Table 1 The definition of the non-clausal matrix.

type	F^{pol}	$M(F^{pol})$	
atomic	P^0	$\{\{P^0\}\}$	(P is atomic)
	P^1	$\{\{P^1\}\}$	(P is atomic)
α	$(\neg A)^0$	$M(A^1)$	
	$(\neg A)^1$	$M(A^0)$	
	$(A \wedge B)^1$	$\{\{M(A^1)\},\{M(B^1)\}\}$	
	$(A \vee B)^0$	$\{\{M(A^0)\},\{M(B^0)\}\}$	
	$(A \Rightarrow B)^0$	$\{\{M(A^1)\},\{M(B^0)\}\}$	

type	F^{pol}	$M(F^{pol})$
β	$(A \wedge B)^0$	$\{\{M(A^0),M(B^0)\}\}$
	$(A \vee B)^1$	$\{\{M(A^1),M(B^1)\}\}$
	$(A \Rightarrow B)^1$	$\{\{M(A^0),M(B^1)\}\}$
γ	$(\forall xA)^1$	$M(A[x\backslash x^*]^1)$
	$(\exists xA)^0$	$M(A[x\backslash x^*]^0)$
δ	$(\forall xA)^0$	$M(A[x\backslash t^*]^0)$
	$(\exists xA)^1$	$M(A[x\backslash t^*]^1)$

The graphical representation of this (non-clausal) matrix is depicted in Fig. 13.

$$\left[\left[\begin{array}{c} \left[\left[P^0 z\right]\ \begin{bmatrix}P^1 a\\ Q^0 y\end{bmatrix}\ \left[Q^1 f b\right]\right] \\ \left[\left[N^1 0\right]\ \begin{bmatrix}N^0 x\\ N^1 f x\end{bmatrix}\ \left[N^0 f f 0\right]\right] \end{array}\right]\right]$$

Fig. 13 The graphical representation of a non-clausal matrix.

The definition of paths through a non-clausal matrix is generalized in a straightforward way, see also Def. 4. All other concepts used for clausal matrices, eg. the definitions of connections and term substitutions, remain unchanged.

The graphical non-clausal connection proof for the previous formula represented by the matrix in Fig. 13 is given in Fig. 14.

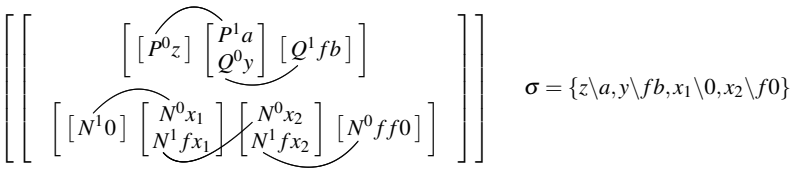

Fig. 14 A (graphical) non-clausal connection proof.

The formal non-clausal connection calculus has the same axiom, start rule, and reduction rule as the clausal connection calculus. The extension rule is slightly modified and a decomposition rule that splits subgoal clauses into their subclauses is added. For example, in the above non-clausal matrix, copies of the clause $\{N^0 x, N^1 f x\}$ need to be allowed. A few definitions are required to specify which clauses can be copied when the non-clausal extension rule is applied [45].

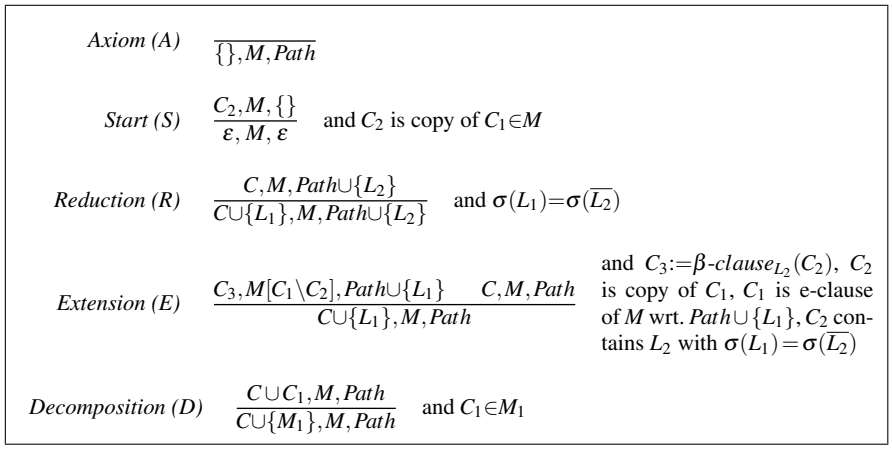

Fig. 15 The non-clausal connection calculus.

A clause C is α-*related* to a literal L, only if L occurs in some clause C' such that C and C' are different clauses of the same matrix. I.e., in the graphical matrix representation, C occurs besides L. In the clausal connection calculus, a copy of a clause C is made by simply renaming all variables in C. In the non-clausal connection calculus, a *copy of the clause* C in the matrix M is made by renaming all *free variables* in C. $M[C_1\backslash C_2]$ denotes the matrix M, in which the clause C_1 is replaced by the clause C_2. The *parent clause* of a clause C in a matrix M is the smallest clause in M that contains C (there are clauses that do not have a parent clause). The clause C in a matrix M is an *extension clause (e-clause) of* M *with respect to* a set of literals $Path$, only if either (a) C contains a literal of $Path$, or (b) C is α-related to all literals of $Path$ occurring in M and if C has a parent clause, it contains a literal of $Path$. In the β-*clause* of C_2 with respect to L_2, denoted by $\beta\text{-}clause_{L_2}(C_2)$, the literal L_2 and all clauses that are α-related to L_2 are deleted from C_2, as these clauses do not need to be considered in the new subgoal clause in the premise of the extension rule.

Definition 11. The axiom and the rules of the *non-clausal connection calculus* are given in Fig. 15. The words of the calculus are tuples "$C,M,Path$", where M is a non-clausal matrix, C is a clause or ε and $Path$ is a set of literals or ε, and σ is a rigid term substitution. A *non-clausal connection proof* of M is a non-clausal connection proof of $\varepsilon,M,\varepsilon$.

The non-clausal connection calculus for classical logic is *correct* and *complete*. The correctness proof is based on the non-clausal matrix characterization, completeness is proved by an embedding into the clausal connection calculus [45].

The *proof search* in the non-clausal connection calculus is carried out in the same way as in the clausal connection calculus, ie. the rules of the calculus are applied in an analytic way. Additional backtracking might be required when choosing the clause C_1 in the decomposition rule, but no backtracking is required when choosing

the matrix M_1 in the decomposition rule. Again, the term substitution σ is calculated by one of the well-known algorithms for term unification. On formulas that are in clausal form, the non-clausal connection calculus behaves just like the clausal connection calculus.

If the matrices that are used in the non-clausal connection calculus are slightly modified, the start and the reduction rule are subsumed by the decomposition and the extension rule, respectively. The resulting *simplified* non-clausal calculus consists only of the axiom, the extension rule, and the decomposition rule [45]. Optimization techniques, such as positive start clauses, regularity, and restricted backtracking, can be used in the non-clausal connection calculus as well [50]. Furthermore, the non-clausal calculus can be extended to some *non-classical logics* in the same way as done for the clausal connection calculus (see Sect. 3.3).

3.3 Calculi for Non-Classical Logics

By using the notion of *prefixes* the connection calculus for classical logic can be extended to intuitionistic logic and several modal logics.

To adapt the connection calculus to *intuitionistic logic*, a prefix is assigned to every subformula of a given formula. A *prefix* is a string, ie. a sequence of characters over an alphabet $\Phi \cup \Psi$, in which Φ is a set of *prefix variables* and Ψ is a set of *prefix constants*. Prefix constants and variables represent applications of the rules $\neg$-*right*, $\Rightarrow$-*right*, $\forall$-*right*, and $\neg$-*left*, $\Rightarrow$-*left*, $\forall$-*left* of the intuitionistic sequent calculus [29], respectively [69, 70]. Each subformula A is labelled with its prefix p, denoted $A:p$, which specifies the sequence of rules that have to be applied (analytically) to obtain the formula A in the sequent of a derivation. Two atomic formulas form an axiom in the intuitionistic sequent calculus if their prefixes unify. This is achieved by an *intuitionistic substitution* $\sigma_I : \Phi \to (\Phi \cup \Psi)^*$ that maps elements of Φ to strings over $\Phi \cup \Psi$. In the *intuitionistic matrix* of a formula, every literal is labelled with its prefix.

In the *matrix characterization for intuitionistic logic* it is additionally required that the prefixes of the two literals in every connection unify under σ_I [70]. For a combined term and prefix substitution (σ, σ_I), a connection $\{L_1 : p_1, L_2 : p_2\}$ is *complementary* only if $\sigma(L_1) = \sigma(\overline{L_2})$ and $\sigma_I(p_1) = \sigma_I(p_2)$. An additional *admissibility condition* ensures that σ and σ_I are mutually consistent [70].

The *skolemization* technique, originally used to eliminate *eigenvariables* in classical logic, is extended and also used for prefix constants in intuitionistic logic [42]. This allows the specification of a *clausal* matrix characterization, in which clause copies can simply be made by renaming all term and prefix variables [42].

For example, for the formula $(N0 \wedge \forall x(Nx \Rightarrow Nfx)) \Rightarrow Nf0$ the intuitionistic matrix is

$$\{\{N^1 0 : a_1 V_1\}, \{N^0 x : a_1 V_2 a_2(x), N^1 fx : a_1 V_2 V_3\}, \{N^0 f0 : a_1 a_3\}\} \ ,$$

in which a_1, $a_2(x)$, a_3 are prefix constants, and V_1, V_2, V_3 are prefix variables. A graphical intuitionistic connection proof of this matrix with $\sigma = \{x \backslash 0\}$ and $\sigma_J = \{V_1 \backslash a_2(0), V_2 \backslash \varepsilon, V_3 \backslash a_3\}$ (where ε is the empty string) is shown in Fig. 16.

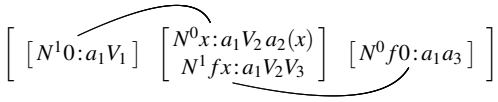

Fig. 16 The intuitionistic connection proof for the formula $(N0 \wedge \forall x(Nx \Rightarrow Nfx)) \Rightarrow Nf0$.

The intuitionistic matrix of the formula $N0 \vee \neg N0$ is $\{\{N^0 0 : a_1\}, \{N^1 0 : a_2 V_1\}\}$, As there is no intuitionistic substitution σ_J with $\sigma_J(a_1) = \sigma_J(a_2 V_1)$ and, hence, no connection proof of this matrix, this formula is *not* valid in intuitionistic logic.

The *clausal connection calculus for intuitionistic logic* is an extension of the clausal connection calculus for classical logic, in which a prefix has been added to each literal and an additional prefix unification is used to calculate the intuitionistic substitution σ_J. The *prefix unification* procedure takes the *prefix property* of all prefixes into account: for two prefixes $p_i = u_1 X w_1$ and $p_j = u_2 X w_2$ with $X \in \Phi \cup \Psi$, $u_1, u_2, w_1, w_2 \in (\Phi \cup \Psi)^*$, the property $u_1 = u_2$ holds. This reflects the fact that prefixes correspond to sequences of connectives and quantifiers within the same formula. Details of the intuitionistic calculus and the prefix unification algorithm can be found in [42].

For *modal logic* the classical sequent calculus is extended by rules for the modal operators $\Box$ ("necessity") and $\Diamond$ ("possibility"). Again, prefixes are used to capture the properties of these rules. In the connection calculus for modal logic, a *prefix* over an alphabet of *prefix variables* $\mathcal{V}$ and *prefix constants* Π is assigned to every subformula of a given formula. Prefix variables and constants represent applications of the rules $\Box$-*left* or $\Diamond$-*right*, and $\Box$-*right* or $\Diamond$-*left*, respectively [69, 70]. Proof-theoretically, a prefix of a subformula A captures the modal context of A and specifies the sequence of modal rules that have to be applied analytically in order to obtain A in the sequent of a derivation. Semantically, a prefix denotes a specific world in a Kripke model [27, 70]. A modal substitution $\sigma_M : \mathcal{V} \to (\mathcal{V} \cup \Pi)^*$ is used to unify the prefixes of literals in each connection.

The core of the formal *clausal connection calculus for modal logic* [6, 46] is similar to the one for intuitionistic logic. The only difference to the intuitionistic calculus is the definition of the prefixes and the prefix unification. The prefix unification procedure depends on the specific modal logic and its domain condition. The *accessibility condition* has to be respected when calculating the modal substitution, eg. $|\sigma_M(V)| = 1$ for the modal logic D and $|\sigma_M(V)| \leq 1$ for the modal logic T for all prefix variables V; there is no restriction for the modal logics S4 and S5. The prefix unification for D is a simple pattern matching, for S4 the prefix unification for intuitionistic logic can be used, for S5 only the last character (or ε if the prefix is empty) of each prefix has to be unified [46, 47].

Both clausal calculi for intuitionistic logic and modal logic can be generalized to non-clausal form. By adding prefixes and a prefix unification, the non-clausal connection calculus presented in Sect. 3.2 is extended to first-order intuitionistic and first-order modal logic [49].

3.4 Implementations

Several automated theorem provers for classical logic that are based on clausal connection calculi have been implemented so far, such as KoMeT [18], METEOR [4], PTTP [65], and SETHEO [38].

leanCoP is a very compact PROLOG implementation of the connection calculus described in Sect. 3.1. leanCoP 1.0 [51] essentially implements the basic clausal connection calculus shown in Fig. 11. The source code of the core prover is given in Fig. 17 (sound PROLOG unification has to be used). PROLOG lists are used to represent sets, terms are used to represent atomic formulas, PROLOG variables represent term variables, and "−" is used to mark literals that have polarity 1. For example, the matrix $M = \{\{P^0 z\}, \{P^1 a, Q^0 y\}, \{Q^1 fb\}\}$ is represented by the PROLOG list `M=[[Pz],[-Pa,Qy],[-Qfb]]`.

```
prove(M,I) :- append(Q,[C|R],M), \+member(-_,C),
 append(Q,R,S), prove([!],[[-!|C]|S],[],I).
prove([],_,_,_).
prove([L|C],M,P,I) :- (-N=L; -L=N) -> (member(N,P);
 append(Q,[D|R],M), copy_term(D,E), append(A,[N|B],E),
 append(A,B,F), (D==E -> append(R,Q,S); length(P,K), K<I,
 append(R,[D|Q],S)), prove(F,S,[L|P],I)), prove(C,M,P,I).
```

Fig. 17 The source code of the leanCoP 1.0 core prover for classical first-order logic.

The prover is invoked by calling the predicate `prove(M,I)`, in which `M` is a matrix and `I` is a positive number. The predicate succeeds only if there is a connection proof for the matrix `M`, in which the size of the active path is smaller than `I`. The proof search starts by applying the start rule implemented in the first two lines. Afterwards, reduction and extension rules are repeatedly applied. These rules are implemented in the last four lines by the PROLOG predicate `prove(C,M,P,I)`, in which `C` is the subgoal clause, `M` is the matrix, `P` is the active path, and `I` is the maximum path limit. The path limit is used to perform iterative deepening on the size of the active path, which is necessary for completeness. When the extension rule is applied, the proof search continues with the left premise before the right premise is considered. The axiom is implemented in the third line. The term substitution σ is stored implicitly by PROLOG.

leanCoP 2.0 integrates additional optimization techniques into the basic connection calculus [43, 44]. The source code of the core prover is shown in Fig. 18.[5] The prover uses *lean PROLOG technology*, a technique that stores the clauses of the matrix in PROLOG's database. This integrates the main advantage of the "PROLOG technology" approach [65] into leanCoP by using PROLOG's fast indexing mechanism to quickly find connections. A *controlled iterative deepening* stops the overall proof search if the current path limit for the size of the active path is not reached in the current iteration. This yields a decision procedure for formulas without (free) variables and also allows for refuting some (not valid) first-order formulas. The *regularity* condition restricts the proof search such that no literal occurs more than once in the active path [39]. The *lemmata* technique reuses the subproof of a literal in order solve the same literal on other branches [39]. *Restricted backtracking* [44] cuts off alternative connections once the application of the reduction or extension rule has successfully solved a literal. A *definitional clausal form translation* is used in a preprocessing step to translate arbitrary first-order formulas into an equivalent clausal form by introducing definitions for certain subformulas [44]. Furthermore, leanCoP uses a *fixed strategy scheduling*.

The techniques described here improve the performance of leanCoP significantly, in particular for formulas containing many axioms [44]. In general, the performance of leanCoP is slightly better than the performance of Prover9 [41], the successor of the famous OTTER theorem prover.

```
prove(I,S) :- \+member(scut,S) -> prove([-(#)],[],I,[],S) ;
    lit(#,C,_) -> prove(C,[-(#)],I,[],S).
prove(I,S) :- member(comp(L),S), I=L -> prove(1,[]) ;
    (member(comp(_),S);retract(p)) -> J is I+1, prove(J,S).
prove([],_,_,_,_).
prove([L|C],P,I,Q,S) :- \+ (member(A,[L|C]), member(B,P),
    A==B), (-N=L;-L=N) -> ( member(D,Q), L==D ;
    member(E,P), unify_with_occurs_check(E,N) ; lit(N,F,H),
    (H=g -> true ; length(P,K), K<I -> true ;
    \+p -> assert(p), fail), prove(F,[L|P],I,Q,S) ),
    (member(cut,S) -> ! ; true), prove(C,P,I,[L|Q],S).
```

Fig. 18 The source code of the leanCoP 2.0 core prover for classical first-order logic.

nanoCoP [48] is a very compact PROLOG implementation of the non-clausal connection calculus specified in Fig. 15.[6] In a first step the input formula F is translated into a non-clausal (indexed) matrix $M(F)$ according to Table 1. In the second step this matrix is written into PROLOG's database.

[5] The full source code of the prover and additional information is available on the leanCoP website at `http://www.leancop.de`.

[6] The source code of nanoCoP is available at `http://www.leancop.de/nanocop/`.

An additional predicate `prove_ec` is used to calculate extension clauses. An extension clause has to fulfill the conditions described in Sec. 3.2: it has to be (a) large enough to contain a literal of the active path or (b) small enough to be α-related to all literals of the active path in the current matrix, and again large enough that in case it has a parent clause, this contains a literal of the active path; the extension clause also has to be large enough such that the literals in the connection can be unified. Additional optimization techniques are integrated that are already used in the classical (clausal) connection prover leanCoP [44], such as regularity, lemmata, and restricted backtracking.

As nanoCoP can simulate leanCoP (in linear time), the proof time and proof size for formulas in clausal form is very similar for both provers [20]. For formulas that are not in clausal form, the proof time and size of the non-clausal nanoCoP proofs can be significantly smaller than those of the clausal leanCoP proofs [20].

ileanCoP [42, 43] is a compact PROLOG implementation of the clausal connection calculus for first-order intuitionistic logic described in Sect. 3.3.[7] It extends the classical connection prover leanCoP by (a) prefixes that are added to the literals in the matrix in a preprocessing step, (b) a set of prefix equations that are collected during the proof search, (c) a set of term variables together with their prefixes in order to check the admissibility condition, and (d) an additional prefix unification algorithm that unifies the prefixes of the literals in each connection. First, ileanCoP performs a classical proof search. After a classical proof is found, the prefixes of the literals in each connection are unified and the admissibility condition is checked. If the prefix unification or the admissibility condition fails, the search for alternative connections continues via backtracking. ileanCoP 1.2 proves significantly more formulas of the TPTP problem library and the ILTP problem library [57] than any other fully automated theorem prover for first-order intuitionistic logic [43].

MleanCoP [46, 47] is a compact PROLOG implementation of the clausal connection calculus for several first-order modal logics, as described in Sect. 3.3.[8] The source code of the core prover is similar to the source code of ileanCoP. Only the prefix unification algorithm changes according to the specific modal logic under consideration. MleanCoP supports the constant, cumulative, and varying domain variants of the first-order modal logics D, T, S4, and S5. MleanCoP 1.3 proves more formulas from the QMLTP problem library [56] than any other prover for first-order modal logic [6, 47].

Extending the non-clausal connection prover nanoCoP by prefixes and prefix unification algorithms, we obtain non-clausal provers for intuitionistic logic and several modal logics. nanoCoP-i and nanoCoP-M are such compact PROLOG implementations of the *non-clausal* connection calculi for first-order intuitionistic logic and several first-order modal logics [49].[9]

---

[7] The source code of ileanCoP is available at `http://www.leancop.de/ileancop/`.

[8] The source code of MleanCoP is available at `http://www.leancop.de/mleancop/`.

[9] The source code of nanoCoP-i is available at `http://www.leancop.de/nanocop-i/`, the source code of nanoCoP-M is available at `http://www.leancop.de/nanocop-m/`.

4 Conclusions

Starting with Schütte's formal system GS for First-Order Logic we have derived characterizations of validity exclusively in terms of formula features such as its connections, paths and substitutions for a certain multiplicity. This characterization provides the basis for the development of proof calculi which determine a complementary set of connections spanning the formula thus establishing its validity. This approach guides the proof search in a formula-oriented and goal-oriented way focussed on nothing else than the given formula which is in stark contrast to approaches based on the still popular resolution or tableaux methods.

This general approach, termed Connection Method, is characterized by connection calculi which form the basis for a uniform family of implemented proof systems for a variety of different logics. Here we presented leanCoP for classical clausal logic, nanoCoP for classical non-clausal logic, ileanCoP for Intuitionistic Logic, and MleanCoP for various modal logics. Any of these turns out to be among the very best systems available for the respective logic, some of them even outperforming any competitive system by a wide margin. The implementations are in the high-level language PROLOG offering the potential for substantial further optimizations.

The line of research presented in this chapter, despite its remarkable success already in its present state of affairs, has plenty of room for further improvements. The present systems are all based on some form of GS_2 defined in Sect. 2.4 so that the step to GS_3 along with the possible improvements described in sections 2.6 and 2.7 are still not incorporated. In this context we refer the interested reader also to the respective literature, notably to [15, Sect. 2.7–2.8] and [14, Sect. IV.11]. Additional improvements can be achieved by special techniques for theories such as equational ones as well as for induction. The theoretical basis for work into such a direction can be found in [14, sect. V.3ff]. On top of all these potential improvements the resulting systems might be trained in their search behaviour by the deep learning approaches which resulted in such spectacular successes in other areas like the game Go. The papers [17, 20] contain, among other envisioned improvements, an elaboration of applying learning to connection provers.

As pointed out in the Introduction proof systems play an important role in modern IT. For the particular application mentioned there, namely provably correct systems, the authors have contributed a chapter concerning this application [52]. We still also see a bright future for program synthesis of the kind first outlined in [10], a topic which, however, would require a further chapter.

Acknowledgements We would like to thank the editors Reinhard Kahle and Michael Rathjen for inviting us to contribute to this volume. We also appreciate the comments of the anonymous referee. Thanks are also due to Hannes Bibel and Florian Hauser for bailing the first author out of a disastrous computer problem which reestablished the prerequisites for writing this chapter.

References

1. P. B. Andrews. Theorem proving via general matings. *Journal of the ACM*, 28:193–214, 1981.
2. R. Antonsen. *The Method of Variable Splitting*. PhD thesis, University of Oslo, 2008.
3. R. Antonsen and A. Waaler. Liberalized variable splitting. *J. Automated Reasoning*, 38:3–30, 2007.
4. O. L. Astrachan and D. W. Loveland. Meteors: High performance theorem provers using model elimination. In R. S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Automated Reasoning Series, pages 31–60. Kluwer Academic Publishers, 1991.
5. P. Benacerraf and H. Putnam, editors. *Philosophy of Mathematics*. Cambridge University Press, 1964. ISBN 0-521-29648-X.
6. C. Benzmüller, J. Otten, and T. Raths. Implementing and evaluating provers for first-order modal logics. In L. De Raedt et al., editor, *20th European Conference on Artificial Intelligence (ECAI 2012)*, pages 163–168, Amsterdam, 2012. IOS Press.
7. E. W. Beth. Semantic entailment and formal derivability. *Mededelingen van de Koninklijke Nederlandse Akademie van Wetenschappen*, 18(13):309–342, 1955.
8. L. W. Bibel. *Reflexionen vor Reflexen – Memoiren eines Forschers*. Cuvillier Verlag, Göttingen, 2017.
9. W. Bibel. An approach to a systematic theorem proving procedure in first-order logic. *Computing*, 12:43–55, 1974. First presented to the GI Annual Conference in 1971; also available as Bericht Nr. 7207, Technische Universität München, Abteilung Mathematik (1972).
10. W. Bibel. Syntax–directed, semantics–supported program synthesis. *Artificial Intelligence*, 14:243–261, 1980.
11. W. Bibel. On matrices with connections. *Journal of ACM*, 28:633–645, 1981. Also available as Bericht 79, Universität Karlsruhe, Institut für Angewandte Informatik und Formale Beschreibungsverfahren.
12. W. Bibel. Computationally improved versions of Herbrand's theorem. In J. Stern, editor, *Proceedings of the Herbrand Symposium*, pages 11–28, Amsterdam, 1982. North–Holland.
13. W. Bibel. Matings in matrices. *Comm. ACM*, 26:844–852, 1983.
14. W. Bibel. *Automated Theorem Proving*. Vieweg Verlag, Braunschweig, second edition, 1987. First edition 1982.
15. W. Bibel. Research perspectives for logic and deduction. In O. Stock and M. Schaerf, editors, *Reasoning, Action, and Interaction in AI Theories and Systems – Essays dedicated to Luigia Carlucci Aiello*, volume 4155 of *LNAI*, pages 25–43. Springer, Berlin, 2006.
16. W. Bibel. Early history and perspectives of automated deduction. In J. Hertzberg, M. Beetz, and R. Englert, editors, *Proceedings of the 30th Annual German Conference on Artificial Intelligence (KI-2007)*, volume 4667 of *LNAI*, pages 2–18, Berlin, 2007. Springer.
17. W. Bibel. A vision for automated deduction rooted in the connection method. In R. Schmidt and C. Nalon, editors, *The 26th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2017)*, volume 10501 of *LNAI*, pages 3–21. Springer, 2017.
18. W. Bibel, S. Brüning, U. Egly, and T. Rath. KoMeT. In A. Bundy, editor, *Proceedings of the International Conference on Automated Deduction, CADE-94*, Lecture Notes in Artificial Intelligence, pages 783–787, Berlin, 1994. Springer.
19. W. Bibel and E. Eder. Methods and calculi for deduction. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1, chapter 3, pages 71–193. Oxford University Press, Oxford, 1993.
20. W. Bibel and J. Otten. Experiments with connection method provers. *Presented to the 4th Conference on Artificial Intelligence and Theorem Proving, AITP 2019, April 7-12, 2019, Obergurgl, Austria*, 2019.
21. W. Bibel and J. Schreiber. Proof search in a Gentzen–like system of first–order logic. In E. Gelenbe and D. Poitier, editors, *Proceedings of the International Computing Symposium*, pages 205–212, Amsterdam, 1975. North–Holland.

22. G. Boole. *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities*. Macmillan, 1854.
23. M. Davis. The Prehistory and Early History of Automated Deduction. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 1 – Classical Papers on Computational Logic 1957–1966*, pages 1–28. Springer, Berlin, 1983.
24. M. Davis. *Engines of Logic – Mathematicians and the Origin of the Computer*. Norton, New York, 2000.
25. E. Eder. The cut rule in theorem proving. In S. Hölldobler, editor, *Intellectics and Computational Logic*, volume 19 of *Applied Logic Series*, pages 101–123. Kluwer Academic Publishers, Dordrecht, 2000.
26. J. E. Fenstad and H. Wang. Thoralf Albert Skolem. In D. M. Gabbay and J. Woods, editors, *Logic from Russell to Church*, volume 5 of *Handbook of the History of Logic*, pages 127–194. Elsevier B.V., 2009.
27. M. C. Fitting. *Proof methods for modal and intuitionistic logics*. D. Reidel, Dordrecht, 1983.
28. G. Frege. *Begriffsschrift*. Louis Nebert, Halle, 1879.
29. G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210 and 405–431, 1935. Engl. transl. in [66].
30. A. Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 8(1):1–64, January 2007.
31. R. Hähnle, N. V. Murray, and E. Rosenthal. Linearity and regularity with negation normal form. *Theoretical Computer Science*, 328:325–354, 2004.
32. C. M. Hansen. *A Variable Splitting Theorem Prover*. PhD thesis, University of Oslo, 2012.
33. D. Hilbert and W. Ackermann. *Grundzüge der Theoretischen Logik*. Springer, 1928.
34. K. J. J. Hintikka. Form and content in quantification theory. *Acta Philosophica Fennica*, 8:7–55, 1955.
35. S. Kanger. *Provability in Logic*. PhD thesis, University of Stockholm, 1957.
36. W. Kneale and M. Kneale. *The Development of Logic*. Clarendon Press, Oxford, 1984.
37. R. Letz, K. Mayr, and C. Goller. Controlled integration of the cut rule into connection tableaux calculi. *Journal of Automated Reasoning*, 13:297–337, 1994.
38. R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO — A high-performance theorem prover for first-order logic. *Journal of Automated Reasoning*, 8(2):183–212, 1992.
39. R. Letz and G. Stenz. Model elimination and connection tableau procedures. In *Handbook of Automated Reasoning*, pages 2015–2112. Elsevier Science Publishers, Amsterdam, 2001.
40. D. Loveland. Mechanical theorem proving by Model Elimination. *J. ACM*, pages 236–251, 1968.
41. W. McCune. Release of Prover9. In *Mile high conference on quasigroups, loops and nonassociative systems*, Denver, 2005.
42. J. Otten. Clausal connection-based theorem proving in intuitionistic first-order logic. In *TABLEAUX 2005, International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, volume 3702 of *LNCS*, pages 245–261, Berlin, 2005. Springer.
43. J. Otten. leanCoP 2.0 and ileanCoP 1.2 – High performance lean theorem proving in classical and intuitionistic logic. In A. Armando, P. Baumgartner, and G. Dowek, editors, *IJCAR 2008*, volume 5195 of *LNCS*, pages 283–291, Heidelberg, 2008. Springer.
44. J. Otten. Restricting backtracking in connection calculi. *AI Communications*, 23:159–182, 2010.
45. J. Otten. A non-clausal connection calculus. In K. Brünnler and G. Metcalfe, editors, *TABLEAUX 2011*, volume 6793 of *LNAI*, pages 226–241, Heidelberg, 2011. Springer.
46. J. Otten. Implementing connection calculi for first-order modal logics. In E. Ternovska, K. Korovin, and S. Schulz, editors, *9th International Workshop on the Implementation of Logics (IWIL 2012)*, Merida, Venezuela, 2012.
47. J. Otten. MleanCoP: A connection prover for first-order modal logic. In S. Demri, D. Kapur, and C. Weidenbach, editors, *IJCAR 2014*, volume 8562 of *LNCS*, pages 269–276, Heidelberg, 2014. Springer.
48. J. Otten. nanoCoP: A non-clausal connection prover. In N. Olivetti and A. Tiwari, editors, *IJCAR 2016*, volume 9706 of *LNCS*, pages 302–312, Heidelberg, 2016. Springer.

49. J. Otten. Non-clausal connection calculi for non-classical logics. In R. Schmidt and C. Nalon, editors, *26th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, LNAI 10501, pages 209–227, Cham, Switzerland, 2017. Springer.
50. J. Otten. Proof search optimizations for non-clausal connection calculi. In B. Konev, J. Urban, and P. Rümmer, editors, *Practical Aspects of Automated Reasoning, PAAR 2018*, volume 2162 of *CEUR Workshop Proceedings*, pages 49–57. CEUR-WS.org, 2018.
51. J. Otten and W. Bibel. leanCoP: Lean connection-based theorem proving. *Journal of Symbolic Computation*, 36:139–161, 2003.
52. J. Otten and W. Bibel. Advances in connection-based automated theorem proving. In J. Bowen, M. Hinchey, and E.-R. Olderog, editors, *Provably Correct Systems*, NASA Monographs in Systems and Software Engineering, pages 211–241. Springer, London, 2017.
53. D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *J. Symbolic Computation*, 2(3):293–304, 1986.
54. D. Prawitz. An improved proof procedure. *Theoria*, 26:102–139, 1960. Also contained in [63, 159–199].
55. D. Prawitz, H. Prawitz, and N. Voghera. A mechanical proof procedure and its realization in an electronic computer. *J.ACM*, 7:102–128, 1960.
56. T. Raths and J. Otten. The QMLTP problem library for first-order modal logics. In B. Gramlich, D. Miller, and U. Sattler, editors, *IJCAR 2012*, volume 7364 of *LNCS*, pages 454–461. Springer, 2012.
57. T. Raths, J. Otten, and C. Kreitz. The ILTP problem library for intuitionistic logic. *Journal of Automated Reasoning*, 38:261–271, 2007.
58. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of ACM*, 12:23–41, 1965.
59. J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier Science Publishers, Amsterdam, 2001.
60. K. Schütte. Ein System des verknüpfenden Schließens. *Archiv f. Mathematische Logik und Grundlagen der Wissenschaften*, 2:55–67, 1956.
61. K. Schütte. *Vollständige Systeme Modaler und Intuitionistischer Logik*. Ergebnisse der Mathematik und ihrer Grenzgebiete, Band 42. Springer Verlag, Berlin, 1968.
62. K. Schütte. *Proof Theory*. Grundlehren der mathematischen Wissenschaften 225. Springer-Verlag, Berlin, 1977. English and revised version of: Beweistheorie (1960).
63. J. Siekmann and G. Wrightson, editors. *Automation of Reasoning — Classical Papers on Computational Logic 1957–1966*, volume 1. Springer, Berlin, 1983.
64. R. M. Smullyan. *First-Order Logic*. Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer-Verlag, Berlin, Heidelberg, New York, 1971.
65. M. E. Stickel. A Prolog technology theorem prover: Implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.
66. M. E. Szabo, editor. *The collected papers of Gerhard Gentzen*. North-Holland, Amsterdam, 1969.
67. A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, 2nd edition, 2000.
68. W. van Orman Quine. A proof procedure for quantification theory. *J. Symbolic Logic*, 20:141–149, 1955.
69. A. Waaler. Connections in nonclassical logics. In *Handbook of Automated Reasoning*, pages 1487–1578. Elsevier Science Publishers, Amsterdam, 2001. See [59].
70. L. A. Wallen. *Automated Deduction in Non-Classical Logics*. MIT Press, Cambridge, Mass., 1990.