

# Connection method

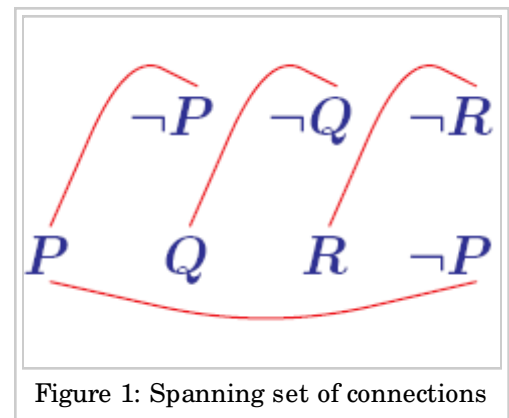
## From Scholarpedia

Wolfgang Bibel and Christoph Kreitz (2009), Scholarpedia, 4(1):6816.

revision #56298 [[link to/cite this article](#)]

Curator: Dr. Wolfgang Bibel, Professor emer., Darmstadt University of Technology, Germany  
Curator: Dr. Christoph Kreitz, Professor, Institut fuer Informatik, Universitaet Potsdam, Germany

The **connection method (CM)** provides a general framework for automated deduction. It differs from similar deductive frameworks, (such as resolution or tableaux), through its unique feature of establishing the truth of a statement by a thorough analysis of the syntactic structure of the statement, without cost-intensive search for some derivation in the traditional sense of logic. It may thus be termed a "derivationless" deductive method. From the result of the analysis a variety of possibly different derivations may be extracted if desired.



## Contents

- 1 Deductive background
- 2 CM's features
- 3 References
- 4 Recommended reading
- 5 External links
- 6 See also

## Deductive background

The automation of deduction is pursued in the subarea of the field of Artificial Intelligence (**AI**) called **Automated Deduction (AD)**. Deduction itself is a form of reasoning as done by humans. Reasoning is one of the fundamental capabilities associated with the notion of human intelligence. It is fundamental for any communication, for prediction and explanation, notably for any scientific exploration. An automation of reasoning is thus a prerequisite for achieving an artificial intelligence at a human level of performance, a fact which illustrates the central importance of

AD.

Human reasoning consists of a mental activity which from the knowledge available to the reasoning person may generate new knowledge, not available before. Any attempt to formalize, and eventually automate, reasoning has thus to start with a formalization of knowledge. Fortunately this is an endeavor pursued in logic already for more than two thousand years culminating in the work of Gottlob Frege in the 19<sup>th</sup> century who laid the basis for powerful logical languages for the representation of knowledge such as first-order or higher-order logic.

Assume we have formalized, say in first-order logic (**fol**), some knowledge  $K$ . As an example we may think of the two facts that humans are mortal, or  $H(x) \rightarrow M(x)$ , and Socrates is human, or  $H(s)$ . Then by logical reasoning we also come to know the additional (or new) knowledge  $L$  that Socrates is mortal,  $M(s)$ , although this was not part of  $K$  initially. In other words, buried in any explicitly available knowledge  $K$  there is some latent knowledge  $L$  which may be made explicit by logical reasoning. This capacity establishes a relationship between chunks of knowledge (like  $K$  and  $L$ ) which, following Frege, is denoted by  $\models$ . In formal notation, we thus have  $K \models L$ , or, in the case of our example,  $H(s) \wedge \forall x (H(x) \rightarrow M(x)) \models M(s)$ . It formally expresses that if  $K$  is true then logically  $L$  must also be true. The quantifier  $\forall x$  in the example formula symbolizes "for all  $x$ ".

The relation  $\models$  is defined by what humans consider to be logical. It would be helpful if it could be made computable in a formal way. It is one of the fundamental discoveries in logic that this relation, or rather some formalized version of it, can in fact be computed exclusively on the basis of the syntax without any regard to the semantics of the represented knowledge (at least for standard logics such as fol). This is done by introducing formal systems with rules of inference. In such a formal system knowledge  $L$  can then formally be derived from  $K$  following exclusively the rules of inference in the system. Since this is a purely mechanical task, it can as well be performed by a machine. So formal systems form the basis for AD.

Strictly speaking a formal system defines its own logical relation, which is usually denoted by  $\vdash$ , possibly indexed by the name of the system. For each such system it is then necessary to establish the fact that the two relations  $\models$  and  $\vdash$  coincide, which is done by way of mathematical proof. Once this is achieved we can leave the task to establish  $K \vdash L$  to any suitably programmed computer. These kinds of programs are developed in AD. Mostly they assume the represented knowledge to be presented in some more normalized form than illustrated so far.

First of all there is a general theorem in logic, called the deduction theorem, which allows the formula to the left of the symbol  $\vdash$  always to be empty. Intuitively, the (meta-level) symbol  $\vdash$  carries the same meaning as the (object-level) implication symbol  $\rightarrow$  used in our example formula. That is, instead of  $K \vdash L$  we may write equivalently  $\vdash K \rightarrow L$  and say that the formula is derivable.

$K$  may involve meta-knowledge such as knowledge how to inductively infer general knowledge from instances. This means that deductive reasoning can model also inductive or abductive reasoning. In this sense deduction is the most general mode of reasoning, thus covering other known modes (like those just mentioned) at the same time. This underlines the central importance of AD even further.

Finally, deductive systems in AD often operate on knowledge represented in some further normalized language such as the so-called *clause form*. They also preprocess the given formulas in order to simplify the subsequent deductive task if possible. We will not go into these more technical details here.

There are many formal systems, even if we restrict ourselves to fol. Among these there is a distinct kind originating from the system **NK** of Gerhard Gentzen (N for *natural* and K for – German – *classical*). NK simulates human (mathematical) reasoning as closely as possible. There are several variants of technically simplified, but otherwise closely related versions of NK. One of them, **LK**, was developed by Gentzen himself in parallel with NK, four others were published in the mid-fifties by E.W. Beth, K.J.J. Hintikka, S. Kanger and K. Schütte. Some are generative, ie. they start from axioms and derive the formula to be proved. Others are analytic, ie. they start from the formula and trace it back to the axioms. From a logical point this particular difference is irrelevant, although the analytic versions are more convenient for AD purposes.

A special feature of NK is that it has two rules of inference for each logical connective, one for introducing the connective and one for eliminating it. The successors of NK (and LK) have dispensed with this natural, although technically redundant symmetry, but have retained the focus on the connectives. For instance, Schütte's system features just three rules for the connectives  $\wedge$ ,  $\vee$  and  $\exists$ , respectively, together with an axiom scheme.

## CM's features

Establishing the derivability of some formula amounts to finding one of the possibly many derivations for it. In fact, one only has to show that such a derivation exists, ie. provide criteria guaranteeing the existence. This is exactly, what the **connection method (CM)** does in terms of the structure of the given formula. In other words, the CM – in contrast to most other deductive systems – does not carry out deductive steps of the usual kind, but rather analyzes the structure of the formula in a stepwise fashion.

Within this analysis the focus is on connections, which are pairs of occurrences of literals of the form  $\{L, \neg L'\}$ . More precisely, one occurrence of the literal  $L$  has to be positive, the other negative within the formula whereby "positive" and "negative" is defined in terms of the number of explicit or implicit negation symbols dominating the literal. For instance, in our example formula  $\vdash H(s) \wedge \forall x (H(x) \rightarrow M(x)) \rightarrow M(s)$  there are two connections,  $\{\neg H(s), H(x)\}$  and  $\{\neg M(x), M(s)\}$ , whereby the implicit negations involved in the implications are made explicit in the connected pairs. This also illustrates the nature of literals as possibly negated predicate symbols (like  $H$  and  $M$ ) along with their arguments.

The criteria for the existence of a derivation of the formula are fulfilled if a set of such connections exists such that the set is *spanning* the formula and the connected arguments are respectively *unifiable*, all with the same unifier. "Spanning" and "unifiable" are purely syntactic notions in terms of the given formula which are illustrated below and whose precise definition can be taken from the literature.

In our example the two connections are in fact spanning and unifiability is established by instantiating the variable  $x$  by  $s$  (as the common unifier), which makes the two connected literals identical, or *unified*. In other words, the proof by the CM of the derivability of our example formula is represented by the following annotated formula in which the two arcs represent the two spanning connections.

$$H(s) \wedge \forall x(H(x) \rightarrow M(x)) \rightarrow M(s)$$

In the normalized language of clause form this proof translates to a matrix of three clauses listed from left to right with connections between the appropriate literals (and without quantifiers).

$$\neg H(s) \quad \neg M(x) \quad H(x) \quad M(s)$$

This translation represents the quantifiers within the terms (with no visible effect in this simple example) and replaces the occurring propositional connectives such that only negation of literals, conjunction and disjunction remain. In the matrix representation of the resulting formula disjunctions are displayed horizontally, conjunctions vertically. In this representation "spanning" roughly means that each path through the matrix from left to right contains a connection. It is because of this simple intuition provided by the matrix representation that makes it useful for human understanding; otherwise the original formula can be used as well as its matrix representational form.

A second example illustrates an additional aspect of connection proofs. They may take into account parts of the formula in various instances. Assume Al is a member of some family, or  $M(a)$ , and whenever  $x$  is a member then so is his or her father  $f(x)$ , or  $M(x) \rightarrow M(f(x))$ . Is the grandfather of Al,  $f(f(a))$ , a member of the family? Here is the connection proof establishing the answer "yes".

$$M(a) \wedge \forall x(M(x) \rightarrow M(f(x))) \rightarrow M(f(f(a)))$$

The corresponding matrix reads as follows.

$$\neg M(a) \quad \neg M(f(x)) \quad M(x) \quad M(f(f(a)))$$

The proof uses two instances of the rule in the premise, which are distinguished by the indices at the end of the connections (no index by convention means index 1). The set of these three

connections is spanning and the common unifier assigns the variables  $x_1$  and  $x_2$  the values  $a$  and  $f(a)$ , respectively.

While such a connection proof is not very intuitive for humans, it is obviously an extremely compact representation of the corresponding set of possible derivations in some formal system, which for machines turns out to be a striking advantage in terms of the required resources in time and storage space. Since the focus is exclusively on the given formula and its structure, which need not be in any special (eg. normal) form, no redundancy whatsoever due to the particular method is left in the search for a proof. Moreover, there is only a small (polynomial) number of connections in any formula relative to its size. So the search for a spanning set of connections can be done in a rather controlled way, although this search in general is of course an NP-hard problem in terms of complexity theory (wrt. the ground level part of the whole task). Also, the CM can be applied to a variety of different logics in exactly the same way except that the unifiability criterion has to be adapted to the logic in question (see further below). None of these distinctive features are available to any of the competitors of the CM such as the resolution principle.

As we said, identifying a spanning set of connections is one of the fundamental tasks in the CM. Algorithmically this task can be solved in a great variety of ways which cannot be discussed here in any detail. We just mention that usually the set of potential connections is preprocessed and some form of a linear chaining strategy is employed as the backbone of any more refined strategy. Thereby the connections are selected in the form of chains connecting sequences of clauses (as illustrated with the previous matrix proofs). For formulas in nonnormal form these procedures become rather intricate.

Unification is the other fundamental task in the CM. If we restrict ourselves to usual term unification then this statement is true also for any other deductive method. However, in the CM the unificational part can involve more than just term unification, for instance constraints codifying possible derivations in some formal system. This particular feature may substitute what is known as Skolemization and at the same time integrate advanced techniques such as variable splitting (Antonsen, 2008) etc. In non-classical logics such as intuitionistic, modal, or linear logics, the characteristics of quantifiers and operators may be coded in prefix strings attached to the literals, so the CM can easily be extended to these logics by adding a string-unification algorithm for prefixes (Otten and Kreitz, 1996) to the unification discussed so far.

The compactness of the CM makes it extremely difficult for humans to understand and develop the details of implementations, especially if such advanced features are taken into consideration. Nevertheless, a number of deductive systems have been successfully designed, implemented and tested, with the CM as the guiding principle. One of them is the system **SETHEO** (Letz et al. 1992), an extension of which (**E-SETHEO**) in 1996 was the overall winner in the international CASC competition of deductive systems and was in the league of leading systems as long as its creators were able to work in the area. Another one is **leanCoP** (Otten and Bibel, 2003) which is remarkable in various ways. First of all, it follows the CM's spirit closer than any other system before. It is programmed in Prolog and in its basic version comprises just three Prolog clauses with altogether 333 bytes (while systems with similar performance are several orders of magnitude larger). By simply adding a few extra parameters and literals the system becomes a system for intuitionistic logic, **ileanCoP**, demonstrating the feature concerning the wide applicability of the CM to a variety of logics. At the CADE System Competition in 2007, CASC-21, leanCoP 2.0 with its few lines of code won the "Best Newcomer" award by outperforming several large systems and by solving some problems that the winning prover could not solve within the given time limit.

leanCoP 1.2 outperforms all existing systems for intuitionistic first-order logic uniformly and with a wide margin. **randocoP** is a new version of leanCoP which adds a stochastic feature to it, through which the performance is increased remarkably; at CASC-22 randocoP ended third place in the most important FOF section among all provers which output proofs (behind Vampire and E, both resolution provers which are far larger systems by orders of magnitude and coded in a lower level programming language). Another remarkable CM system is TPS developed by Peter Andrews from the CMU which is specialized to proofs in higher-order logic.

Once a connection proof is established it can mechanically be expanded to some proof in a formal system such as LK (out of the set of those proofs coded by the connection proof). In fact there are even programs that may transform such a proof into one expressed in natural language, so that anyone can read and understand the proof generated by the CM.

Deductive systems like the ones just mentioned attempt to automate human reasoning, as we said at the outset. Since reasoning is involved in literally most human activities, deductive systems have practically a universal potential for applications. They are already used in Mathematics for establishing mathematical proofs, in Computer Science for verifying and synthesizing programs and in the form of programming languages (like Prolog), in various disciplines as an important component of so-called knowledge systems, as a component of natural language processing systems, to mention just a few out of numerous other applications. The CM may enhance the performance of this technology even further, if its potential will eventually be fully exploited.

The CM has been developed by Wolfgang Bibel (beginning in 1970 with a first publication in 1971), and independently by Peter Andrews, based on the work of Dag Prawitz, Kurt Schütte and other leading logicians of the previous century. Lincoln Wallen has contributed by applying the CM to non-classical logics. Related work has been carried out by S.J. Maslov, by N.V. Murray and E. Rosenthal, and many others. Recently, work in the context of the notion of deep inference has focused on ideas very closely related to those of the CM.

## References

Antonsen, R. (2008) The Method of Variable Splitting. Faculty of Mathematics and Natural Sciences, University of Oslo. ISSN 1501-7710.

Letz, R. et al. (1992) SETHEO: A High-Performance Theorem Prover. *Journal of Automated Reasoning* 8(2):183-212.

Otten, J. and Kreitz, C. (1996) T-String-Unification: Unifying Prefixes in Non-Classical Proof Methods. 5th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods, *Lecture Notes in Artificial Intelligence* 1071:244-260.

Otten, J. and Bibel, W. (2003) leanCoP: Lean Connection-Based Theorem Proving. *Journal of Symbolic Computation* 36:139-161. **Internal references**

- Olaf Sporns (2007) Complexity. *Scholarpedia*, 2(10):1623.

## Recommended reading

- P.B. Andrews (1976) Refutations by Matings, *IEEE Trans. Comput.* C-25:193-214.
- W. Bibel (1983) Matings in Matrices, *Communications of the ACM* 26:844-852.
- W. Bibel (1987) *Automated Theorem Proving*, 2nd ed., Vieweg Verlag, Braunschweig.
- W. Bibel (1993) *Deduction: Automated Logic*, Academic Press, London.

## External links

Wolfgang Bibel's website (<http://www.intellektik.de/index/WolfgangBibel.htm>)

Christoph Kreitz's website (<http://www.cs.uni-potsdam.de/ti/kreitz>)

## See also

Logic, Deduction, Reasoning

Wolfgang Bibel, Christoph Kreitz (2009) Connection method. *Scholarpedia*, 4(1):6816, (go to the first approved version)

Created: 12 March 2008, reviewed: 13 January 2009, accepted: 13 January 2009

Suggested by: Mr. Florian Hauser, Institute of Neural Information Processing, Ulm University, Germany

Invited by: Dr. Eugene M. Izhikevich, Editor-in-Chief of Scholarpedia, the peer-reviewed open-access encyclopedia

Action editor: Dr. Eugene M. Izhikevich, Editor-in-Chief of Scholarpedia, the peer-reviewed open-access encyclopedia

Retrieved from "[http://www.scholarpedia.org/article/Connection\\_method](http://www.scholarpedia.org/article/Connection_method)"

Categories: Computational Intelligence

- 
- This page was last modified 09:54, 13 January 2009.
  - Copyright (C)
  - ISSN 1941-6016