TOWARDS AN ADVANCED IMPLEMENTATION OF THE CONNECTION METHOD

Wolfgang Bibel, Elmar Eder, Bertram Fronhoefer
Institut fuer Informatik
Technische Universitaet Muenchen
Postfach 202420
D-8000 Muenchen 2

ABSTRACT

This paper is intended to give a glance at some issues involved in implementing an advanced proof component based on the connection method. The material presented comprises contributions to the following problem domains: (a) Dealing with formulas In non-normal form, (b) The dynamic incorporation of unification into a proof procedure, (c) Controlling the generation of copies of clauses.

INTRODUCTION

In [1] and [4] the theory and architecture of LOPS, a system for logical program synthesis, has been presented. Its deductive component consists of a theorem prover based on the connection method (see [2] or [3]). The current work in the LOPS-project focuses on the replacement of this prover by a considerably advanced version.

There is theoretical [2] and experimental [10] evidence that the connection method has the potential of outperforming any other deductive method (such as resolution) due to its lack of redundancy, its applicability to non-normal-form formulas and several other unique features. In particular, the deep insight into the nature of proofs provided by this approach has opened the view of problems and their solutions which have not even been clearly identified in the context of previous methods. Most of the material presented here is of that nature. It is concerned with three major aspects of theorem proving, each presented in one of the following three sections.

With the results reported in the first section we achieve a refinement of all our previous work concerning the ground-level which is well-balanced in view of the present implementation. Although (usual first-order) unification may be regarded as a solved problem, its dynamic incorporation into the process of proof deserves further considerations presented in section 2. Section 3 gives first results on a promising way of controlling the generation of copies of clauses which has never been taken into account before.

Each section highlights a whole paper (viz. [8], [6], [7]). Thus the reader should be aware that this note cannot provide more than a few condensed excerpts from an extensive body of material. In particular, the important issue of combining these facets to a well-balanced system cannot be explained here in further detail.

* Work supported by Deutsche Forschungsgemeinschaft

1. GROUND-LEVEL FEATURES

While in [2] and [5] some techniques have been introduced to enhance the performance of the Connection Procedure for normal form matrices of propositional logic, their adaptation to non-normal form will be dealt with in the following paragraphs. A detailed discussion of these and further issues may be found in [8].

<u>**1.1. Definition:**</u>
Given an occurrence L^r of a literal L at some position r, we define the clausal core of L^r, in symbols $cc(L^r)$, as the set:
$cc(L^r) := \{K^s \mid K^s$ is an element of a clause in which L^r occurs$\}$

Note that in the non-normal form case a matrix is a set of clauses, which may contain submatrices, which in their turn may contain clauses and so on. Now it becomes important to distinguish between whether a literal is an element of a clause (in the set-theoretic sense) or only occurs within it, "to occur" referring to the transitive closure of the is-element-of relation.

As an example consider the clause $\{R,K,\{\{\{L,M\},N\},\{O,P\}\}\}$ of a main matrix. Here, for instance, R,K are elements of the clause while L,M, N,O,P only occur within it. Examples of clausal cores are $cc(K)=\{R,K\}$ and $cc(L)=\{R,K,L,N\}$.

Recall that in the course of a connection proof subgoals are stored to be handled later. If any such subgoal is a literal then it is contained in the clausal core of some path literal. This enables an easy access to a relevant part of the clausal core of path literals.

The following proposition together with the subsequent theorem have an immediate application to reducing the number of paths to be checked for complementarity.

<u>**1.2. Proposition:**</u>
Given a path p through an arbitrary matrix M.
(a) If there is an occurrence L^r of a literal s.t. the set $N(L^r) := \{K^s$ in $p \mid L^r$ in $cc(K^s)\}$ is not empty, then $(p \backslash N(L^r)) \cup \{L^r\}$ is a path through M.
(b) Furthermore, if p contains an occurrence L^t of

a riteral, then p is complementary if (p\N(L)) ^{u} {L } is complementary,
(c) If there are occurrences L and -L of literals then p is complementary if both (p\N(L)) U {L } and (p\N(-L)) U {-L } are complementary.

1.3. Theorem:
Given a partial path p and a clause c selected for extension, then (l)-(3) hold:
(1) The continuations of p through the occurrence L s of a literal in c need not be checked for complementarity if there is an occurrence L^r of the literal L which appears both among the stored subgoals and in the clausal core of a path literal.
(2) If p does not contain an occurrence of the literal L but c contains an occurrence -Lu and a subgoal L^r exists which is in the clausal core of a path literal, then only the continuations of p through $-L^u$ must be checked for complementarity.
(3) If c contains an occurrence K of a literal and p contains an occurrence K^s of the same literal then we can exclude c as a clause suitable for extension.

Part (1) and (3) of the theorem are dynamic applications of factorization, while (2) is a dynamic application of a generalized form of 6.1.T in [2], which expresses an idea due to Prawitz.

An unreflected combination of part (1) of the theorem and ditchmarking [2] would result in an erroneous proof procedure, as pointed out in 15] as an open problem. This has now been solved by storing the occurrence L^r together with all the information needed for a future checking for complementarity of the paths containing L^r.

2 THE CONCEPT OF WEAK UNIFICATION

In order to prove a formula F of first-order logic using the connection or resolution method it is, in general, necessary to consider, at least implicitly, several copies (obtained by renamings of variables) of each clause. Such a formula F which we shall, for simplicity, here assume to be in normal form is valid iff there exists a set of copies of its clauses which can be made propositionally complementary by substitution of variables. So if c and d are two different copies of clauses of F then we have to look for unifiable connections between literals of c' and d', respectively, where c' and d' are obtained from c and d by applying the current substitution generated so far in the proof process.

It is necessary for an efficient proof procedure to eliminate, before actually establishing the -proof, all searches for connections which can be recognized in advance as not unifiable by taking into account a single copy of each clause. This leads us to the concept of weak unification. A similar but weaker concept has been introduced in [11] under the name of weak substitution.

2.1 Definition:
Two terms or literals L and M are called weakly unifiable if there are substitutions $ and $ such that L$ M$.
A set {L,-M} of literals occurring in two (not necessarily different) clauses of a formula F is called a weakly unifiable connection, for short a w-connection, if L and M are weakly unifiable.

Obviously, L and M are weakly unifiable iff two copies L~ and M~ of L and M, respectively, with disjoint sets of variables are unifiable. Hence any connection between literals of clauses c' and d' as above must correspond to a w-connection of F. So we first determine, using a unification algorithm, the set of w-connections of F. In the main part of the proof procedure we then only need to consider connections corresponding to w-connections.

There is another advantage of weak unification. If we check two literals L and M for weak unifiability then the unification algorithm gives us a most general unifier uof L~ and M~ as above. Let $be the current substitution. In order to establish a new connection between L and M we have to find out whether L~$ and M~$ are unifiable. Let $ denote the most general unifier of these two literals. Then, after the proof procedure has taken into account the connection {L~,M~}, $ts the new current substitution. Now the point is that $ can be obtained as a supremum of $and u in a lattice of equivalence classes of substitutions by the following theorem the non-trivial proof of which is given in [6]. There are similarities with the lattice of equivalence classes of terms introduced in |9] although a direct relationship does not seem to exist.

2.2 Theorem:
For substitutions $and tlet $< t iff there is a substitution tsuch that $ r t We say that two idempotent substitutions $and tare equivalent to each other iff $< t and t< $.
Then the set of equivalence classes of idempotent substitutions together with an added greatest element is a complete lattice.

Determining the supremum of $and uin this lattice amounts to a unification of pairs of terms. It takes less time than a unification of {L$,M$}. Incidentally, if F is valid then its set of w-connections is spanning (in one copy of F), a fact which may be taken advantage of for the elimination of useless alternatives In the course of proof.

Another property which can be checked by looking at just one copy of the matrix is whether a clause contains a pure literal. The simplest case is that the clause contains a literal which does not have a w-connection with any literal of the matrix. Then we can delete the whole clause (with all its copies) from the matrix. But there are other cases in which we can also do this.

2.3 Definition:
Let c be a clause of F, c' an instance of c (i.e., c' is obtained from c by substituting consistently all variables of c by closed terms), and L a literal of c'. Then we call L a pure literal of c' in F if, for all literals M of F and substitutions $ {L,M$ is not complementary. Note that L is not a literal of F. If there is a pure literal of c' in F then we call c' a pure instance of c in F. c is called a p-clause of F if every possible instance

of c is a pure instance.

<u>2.4 Proposition:</u>
A formula F in normal form is valid iff the formula obtained from F by deleting all p-clauses of F is valid.

For example, the first clause of $\begin{matrix} Px \\ Qx \end{matrix} \quad -Pa \quad -Qb$

is a p-clause although each of its literals has a w-connection. Our concept of p-clauses is a generalization of pure-reduction (see [2]). It allows us to look ahead for all possible connections with this clause.

The following proposition provides a way of testing for p-clauses as follows. Let us call a set k of w-connections of F a <u>brush of c</u> in F if $c=\{L_1,\ldots,L_n\}$ is a clause of F and k has the form $\{\{L_1,K_1\},\ldots,\{L_n,K_n\}\}$ where $K_1,\ldots,K_n$ are arbitrary literals of F. k is called <u>consistent</u> if the set $\{\{L_1,M_1\},\ldots,\{L_n,M_n\}\}$ is unifiable where $M_1,\ldots,M_n$ are obtained from $K_1,\ldots,K_n$ by suitable renamings of variables. Then c is a p-clause iff no brush of c is consistent. The number of unifications needed for a consistency check of a brush of c is equal to the number of literals in c and the number of brushes of c is the product of the numbers of w-connections of the literals of c. So the amount of time needed to determine whether a clause is a p-clause can be estimated in advance which in turn gives information for the control of the proof process.

<u>3. ON THE NUMBER OF COPIES NEEDED OF A CLAUSE</u>

In certain special cases bounds can be given for the number of copies needed of a clause. However, this is not true in general since first-order logic is undecidable. A few simple estimates are the following.

If F contains no function symbols, k is the maximum of 1 and the number of constants occurring in F, c is a clause of F and n is the number of variables occurring in c then at most k^n copies are needed of c. This fact is useful if the number n of variables occurring in c is small. However, already the case where F contains only the three terms x, y and fx (and no constants) is undecidable as can be seen by Skoleraization of the formulas of the undecidable (w.r.t. validity) prefix class $\exists\forall\exists$.

If F has only one clause with more than one literal and there are no w-connections <u>within</u> this clause then only <u>one</u> copy of this clause is needed. This is not the case if there is a weak connection within the clause as can be seen from

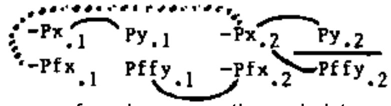
$$\begin{matrix} Pxyz \\ -Pzxy \end{matrix} \quad -Pcab \quad Pabc$$

If a valid matrix F has two clauses with two literals in each clause then it has the form

$$\begin{matrix} -Pq_1\ldots q_n & Ps_1\ldots s_n \\ -Pr_1\ldots r_n & Pt_1\ldots t_n \end{matrix}$$

As an example we give the matrix

$$\begin{matrix} -Px & Py \\ -Pfx & Pffy \end{matrix}$$



For its connection proof, shown on the right, we need two copies of each clause. The dotted line indicates factorization and the horizontal line in the fourth clause indicates splitting of this clause. This example illustrates the importance of factorization and splitting. Without these features we would have to consider four copies of the first clause and three of the second.

Since all these properties apply to very special cases of formulas only, and thus will not be applicable directly to the input formula in practice, their relevance lies in a dynamic application in the course of the proof process. Namely there will arise subproblems which may well belong to one of these simple cases.

REFERENCES

[I] W.Bibel: Syntax-directed, semantics-supported program synthesis. Artificial Intelligence 14,3-261 (1980).

[2] W.Bibel: Automated theorem proving. Vieweg Verlag, Wiesbaden, 293 S., (1981).

[3] W.Bibel: Matings in matrices. German Workshop on Artificial Intelligence, Bad Honnef, Januar 1981 (J. Siekmann, ed.), Inforraatik-Fachberichte 47 Springer, Berlin, 171-187 (1981). (Invited Paper)

[4] W.Bibel and K.M.Hoernig: LOPS - A system based on a strategical approach to program synthesis. In: Automatic program construction techniques (A. Biermann et al.), MacMillan, New York 1983.

[5] W.Bibel and K.M.Hoernig: Improvements of a tautology-testing algorithm. 6th Conference on Automated Deduction, New York, USA, June 1982 (D. Loveland, ed.), Lecture Notes in Computer Science <u>138</u>, Springer, Berlin, 326-341 (1982).

16] E.Eder: Properties of Substitutions and Unifications. Projekt Beweisverfahren, Inst. f. Inf. TUM, Report ATP-17-I-83 (1983).

[7] E.Eder: On the number of copies needed of a clause in a connection proof. Projekt Beweisverfahren, Inst. f. Inf. TUM, Report, (in preparation)

[8] B.Fronhoefer: A discussion of refinements of the connection method. Projekt Beweisverfahren, Inst. f. Inf. TUM, Report, (in preparation)

[9] G.Huet: Confluent Reductions. JACM 27, pp.797-821 (1980)

[10] A.Mueller: An implementation of a theorem prover based on the connection method. Projekt Beweisverfahren, Inst. f. Inf. TUM, Report ATP-12-XII-81 (1981).

(II] R.B.Stillman: The Concept of Weak Substitution in Theorem-Proving. JACM 20, pp.648-667. (1973)