COMPLETENESS OF THE CONNECTION GRAPH PROOF PROCEDURE FOR UNIT-REFUTABLE CLAUSE SETS

Gert Smolka
Institut für Informatik I, Universität Karlsruhe
7500 Karlsruhe 1, West Germany
September 1982

ABSTRACT. In this paper it is shown that R. Kowalski's connection graph proof procedure /Ko75/ terminates with the empty clause for every unit-refutable clause set, provided an exhaustive search strategy is employed. This result holds for unrestricted tautology deletion, whereas subsumption requires certain precautions.

The results are shown for an improved version of the connection graph resolution rule which generates fewer links than the original one. The new inference rule not only leads to a smaller search space but it also permits a more efficient implementation.

The proofs are based on refutation trees as in /HR78/ and are applied immediately at the general level. Hence the unsolved problems resulting from the classical lifting techniques in the context of connection graphs are avoided.

Finally a counterexample is presented at the propositional level which shows that unrestricted deletion of tautologies destroys completeness for non-unit-refutable clause sets.

1. INTRODUCTION

R. Kowalski's connection graph proof procedure /Ko75/ is an inference system for clause graphs based on resolution /Ro65/. Each link (i.e. edge) of a clause graph indicates a possible application of the clause graph resolution rule which replaces a link by its resolvent. There is no search for new links connecting the resolvent, but they are obtained from the links connecting the parent clauses by a mechanism called inheritance. Consequently, a link that does not exist for a parent clause cannot be inherited by the resolvent. This is but one of the major sources of search space reduction, but also a source of the theoretical problems. Once the resolvent is generated, the link resolved upon is deleted and the π-reduction (a generalisation of the purity principle in /Ro65/) is performed on the graph. The removal of the links resolved upon, the subsequent π-reduction, and the inheritance mechanism lead to the deletion of many clauses. This considerable search space reduction is enhanced even more by two rules allowing the elimination of tautologies and subsumed clauses.

In Proceedings of the 6th German Workshop on Artificial Intelligence, Informatik Fachberichte 58, pages 191-204, Springer-Verlag, Berlin, 1982.

A logical calculus is said to be complete, if there *exists* a proof for every valid formula. In automated theorem proving however, we are interested in the stronger requirement that a search strategy for the calculus will actually *find* this proof. For that reason a search strategy is said to be complete, if it terminates with a proof whenever there exists one.

Now, *the completeness problem of the connection graph proof procedure* is to show that each non-backtracking and exhaustive search strategy for the *Kowalski-calculus* (the clause graph inference system described above) is complete, i.e. terminates with the empty clause whenever the initial clause set was unsatisfiable. A search strategy for the Kowalski-calculus is *non-backtracking*, if it does not require the reconstruction of a graph obtained earlier in the derivation sequence. And it is called *exhaustive*, if each link generated in the inference process is deleted again after a finite number of steps. Note that a link can be deleted either because it is resolved upon or as a consequence of the other deletion rules (e.g. the π-reduction).

In spite of the attempts of Brown /Br76/, Siekmann and Stephan /SS76, SS80/, and Bibel /Bi81a, Bi81b/, the connection graph proof procedure completeness problem is still unsolved. The difficulties arise from the fact that the Kowalski-calculus is not commutative. An inference system is *commutative*, if an inference step β remains applicable after the application of an inference step α, provided β was applicable before the application of α. Contrary to most classical refinements for resolution (e.g. Merge or Linear), the Kowalski-calculus is not commutative, since resolution upon a link can cause further deletions of links by the subsequent π-reduction. Thus even an exhaustive search strategy cannot actually resolve upon every link generated in the inference process.

To solve the completeness problem of the connection graph proof procedure, a new proof technique had to be developped in order to handle the dynamic aspects of an inference process running on a non-commutative inference system. Our proof is based on the following idea first mentioned in /SS76, SS80/: The initial graph of an unsatisfiable clause set contains an unsatisfiable subgraph, called the *kernel*. A non-negative integer *complexity* is defined for the kernel. If this complexity is zero, the graph contains the empty clause. Resolution upon a link within the kernel transforms the kernel and strictly reduces its complexity, whereas resolution upon a link not belonging to the kernel does not affect the kernel at all.

But how can we define the kernel? How is the kernel transformed by the inference rules and what is a convenient measure for the kernel complexity? A first attempt may be to define the kernel as the projection of its resolution refutation, i.e. every link and every clause of the initial graph used in this refutation constitute the kernel. However, this idea turned out to be rather unsuccessful, since it leads to very complicated transformations which are not manageable.

The difficulties can be overcome by condensing the resolution refutations to a more simple structure representing a whole class of resolution refutations at once. This is achieved using Shostak's refutation graphs /Sh76/. Since the Shostak refutation graph for the kernel consists only of variants of clauses occurring in the kernel, its projection onto the kernel is a simple superposition. It turns out that one resolution step within the kernel results in one or more resolution steps upon the corresponding refutation graph.

A major advantage of this new proof technique is that it leads to a clearer understanding of the effects of the clause graph resolution and of the deletion rules for tautologies and subsumed clauses. For example it turns out that Bibel's conditions for the removal of tautologies and subsumed clauses (see /Bi81c/) are also adequate in the light of the new proof technique.

In addition the new proof technique led to a significant improvement of the clause graph resolution rule. The new rule presented in this paper generates fewer links than the versions defined by Kowalski /Ko75/, Bruynooghe /Br75, Ko79/, and Bibel /Bi81a, Bi81c/. It has the additional advantage that its implementation is more efficient than the rules previously proposed.

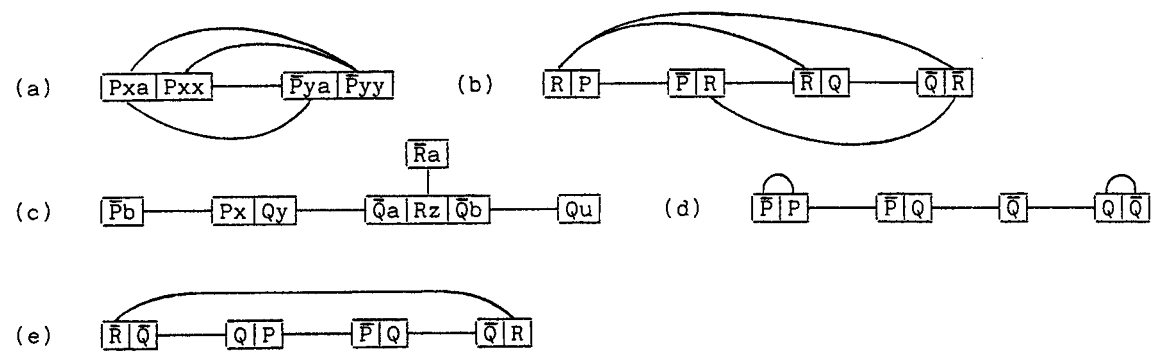
In this paper we consider only the case of unit-refutable clause sets. A separate investigation of this subcase appears to be indispensable, since several of the stronger results shown here do not hold for the general case. For example the omission of factoring, the restriction of resolution to unit-links, and suprisingly the unrestricted deletion of tautologies preserve completeness for unit-refutable sets but not for the general case.

Unit-refutable clause sets are of great practical importance in automated theorem proving, since on the one hand there are very efficient refutation techniques known for this special class (see /Ch70, Ku72, HW74, Oh82/), and on the other hand they are general enough to include the well-investigated Horn sets, within which it is possible to express for example the axioms of group theory or of Boolean algebras. Unit-refutable clause sets are distinguished from general clause sets by the fact that their unsatisfiability can be characterized by refutation trees instead by refutation graphs containing cycles.

The reader is expected to be familiar with the basic notions of resolution-based theorem proving (see e.g. /Lo78/), and with the main ideas and notions concerning confluent relations (see /Hu80/). For space limitations the proofs are only sketched or omitted entirely. The complete proofs are contained in /Sm82/. The author is currently working on completeness results concerning the general case and these will be presented in a later paper.

2. CLAUSE GRAPHS

A clause graph is a data structure for a set of clauses, where the edges indicate possible resolutions. As the examples on the next page suggest, clause graphs are not graphs



at all in the conventional sense. Each node represents a clause and is drawn as a chain of contiguous cells labeled with the literals of the clause. The edges in clause graphs are called links. Each link connects two literals which are potentially complementary, i.e. the literals can be made complementary by applying some substitution, after renaming the variables. The following definitions formalize these intentions:

2.1 Nodes. The following symbols are nodes: k_1, k_2, k_3, ...

2.2 Literal Occurrences. A literal occurrence L^a is an ordered pair $<L,a>$ consisting of a literal L and a node a.

2.3 Links. A link L^aK^b is an unordered pair $\{L^a,K^b\}$ consisting of two literal occurrences L^a and K^b such that L and K are potentially complementary. An autolink is a link L^aK^b such that $a=b$.

2.4 Clause Graphs. A clause graph is a tripel <NODES,C,LINKS> such that:

(a) NODES is a finite, possibly empty set of nodes.

(b) C is a mapping NODES $\longrightarrow$ Clauses. Instead of C(a) we write C_a.

(c) LINKS is a set of links such that each $L^aK^b \in$ LINKS satisfies the following conditions: $a \in$ NODES, $L \in C_a$, $b \in$ NODES, and $K \in C_b$.

(d) For any two different nodes $a,b \in$ NODES the clauses C_a and C_b have no variables in common.

Let $\mathcal{G}$ = <NODES,C,LINKS> be a clause graph. A node a [literal occurrence L^a, link ℓ] is called a node in $\mathcal{G}$ [literal occurrence in $\mathcal{G}$, link in $\mathcal{G}$], if $a \in$ NODES [$a \in$ NODES and $L \in C_a$, $\ell \in$ LINKS]. A unit-link in $\mathcal{G}$ is a link L^aK^b in $\mathcal{G}$ such that $|C_a| = 1$ or $|C_b| = 1$. A τ-link in $\mathcal{G}$ is a non-autolink $\ell = L^aK^b$ in $\mathcal{G}$ such that the binary resolvent indicated by ℓ (the clause $(\sigma C_a - \sigma L) \cup (\sigma C_b - \sigma K)$ where σ is an mgu of L and $\bar{K}$) is a tautology. A link L^aK^b in $\mathcal{G}$ is said to be incident with a literal occurrence M^e [a node c, a clause C] in $\mathcal{G}$, if $M^e = L^a$ or $M^e = K^b$ [$c = a$ or $c = b$, $C = C_a$ or $C = C_b$]. The number of nodes in $\mathcal{G}$ is denoted by $|\mathcal{G}|$:= |NODES|. $S(\mathcal{G}) := \{C_a |\ a \in \text{NODES}\}$ denotes the set of all clauses occurring in $\mathcal{G}$. $\mathcal{G}$ is called total, if for any two potentially complementary literal occurrences L^a and K^b in $\mathcal{G}$ L^aK^b is a link in $\mathcal{G}$. $\mathcal{G}$ is called satisfiable [unsatisfiable], if $S(\mathcal{G})$ is satisfiable [unsatisfiable]. The clause graph with no nodes is called the empty clause graph and denoted by (). Each clause graph $\mathcal{G}$ such that $|\mathcal{G}| = 1$ and $S(\mathcal{G}) = \{\Box\}$ is denoted by $(\Box)$.

2.5 Subgraphs. Let $\mathcal{G}$ = <NODES,C,LINKS> and $\mathcal{G}'$ = <NODES',C',LINKS'> be two clause graphs.

$\mathcal{G}$ is said to be a subgraph of $\mathcal{G}'$, if NODES $\subset$ NODES', LINKS $\subset$ LINKS', and if C is the restriction of C' to NODES. The relation $\subset$ defined by $\mathcal{G} \subset \mathcal{G}'$:<==> $\mathcal{G}$ is a subgraph of $\mathcal{G}'$ is a partial order on the set of all clause graphs.

2.6 Basic Operations on Clause Graphs. Let $\mathcal{G}$ be a clause graph, ℓ a link, a a node, and C a clause. Then the addition or the deletion of a link or a node is defined as follows:

- $\mathcal{G} - \ell$ denotes the graph which results from $\mathcal{G}$ by the deletion of ℓ.
- $\mathcal{G} + \ell$ denotes the graph which results from $\mathcal{G}$ by the addition of ℓ. The operation is only defined, if the literal occurrences of ℓ are literal occurrences in $\mathcal{G}$.
- $\mathcal{G} - a$ denotes the maximal subgraph of $\mathcal{G}$ not containing a.
- $\mathcal{G} + \langle a, C\rangle$ denotes the graph which results from $\mathcal{G}$ by the addition of a such that $C_a = C$. The operation is only defined, if a is not a node in $\mathcal{G}$ and if $\mathcal{G}$ and C have no variables in common.

2.7 π-Reduced Clause Graphs. A literal occurrence L^a in a clause graph $\mathcal{G}$ is said to be pure in $\mathcal{G}$, if no non-autolink in $\mathcal{G}$ is incident with L^a. A clause graph $\mathcal{G}$ containing no literal occurrences which are pure in $\mathcal{G}$ is said to be π-reduced. The binary relation $\xrightarrow[\pi]{}$ defined by

$$\mathcal{G} \xrightarrow[\pi]{} \mathcal{G}' \text{ :<==> There is a literal occurrence } L^a \text{ in } \mathcal{G} \text{ such that } L^a \text{ is pure in } \mathcal{G} \text{ and } \mathcal{G}' = \mathcal{G} - a$$

is obviously Noetherian. Since $\xrightarrow[\pi]{}$ is locally confluent, $\xrightarrow[\pi]{}$ is also confluent and the unique $\xrightarrow[\pi]{}$-normal form $\pi(\mathcal{G})$ exists for each clause graph $\mathcal{G}$. It is easy to show that $\pi(\mathcal{G})$ is always the maximal π-reduced subgraph of $\mathcal{G}$.

2.8 The Initial Clause Graph $\mathcal{G}_S$ of a Clause Set S. Let S be a clause set. Let S' be the clause set obtained from S by the deletion of all tautologies and by renaming variables so that different clauses contain different variables. Let $\mathcal{G}'$ be a total clause graph with $S(\mathcal{G}') = S'$ and $|S(\mathcal{G}')| = |S'|$. Then $\pi(\mathcal{G}')$ is called an initial clause graph of S. Since two initial clause graphs of S are equal modulo the renaming of variables and nodes, each initial clause graph of S is said to be *the* initial clause graph of S and is denoted by $\mathcal{G}_S$.

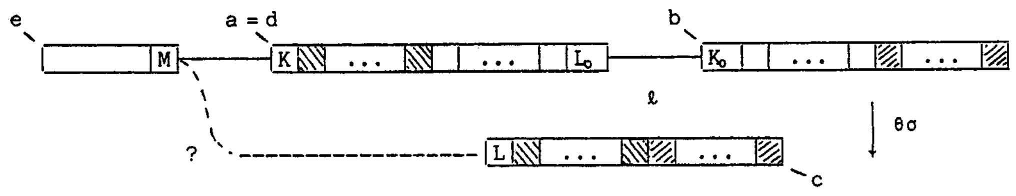
3. RESOLUTION AND THE ELIMINATION RULES FOR CLAUSE GRAPHS

In order to get an inference system, we now introduce three inference rules for clause graphs. The ρ-rule replaces a non-autolink by the corresponding resolvent. The τ-rule deletes a node if its clause is a tautology. The σ-rule deletes a node which is subsumed by another node, i.e. by its clause and by its links.

3.1 The ρ-Rule. The ρ-rule consists of an adding and of a deletion part. The adding part replaces a non-autolink ℓ by the corresponding resolvent. Thereby, the links connecting the resolvent are obtained exclusively from the links connecting the parent clauses (i.e. the clauses being incident with ℓ) by a mechanism called inheritance. The deletion part first deletes the link ℓ and then performs a π-reduction on the graph. The adding part of the clause graph resolution rule is defined by:

```
Algorithm    $\rho_0$;
Input:       $\mathcal{G}_0$ : clause graph;
             $\ell = L_0^a K_0^b$ : non-autolink in $\mathcal{G}_0$ ;
Constants:   $\sigma$ := most general unifier of $L_0$ and $\bar{K}_0$ ;
             R := $(\sigma C_a - \sigma L_0) \cup (\sigma C_b - \sigma K_0)$;       -- resolvent corresponding to $\ell$
             $\theta$ := renaming substitution so that
                 $\theta R$ and $\mathcal{G}$ have no variables in common;
             c := node that does not occur in $\mathcal{G}_0$;
Variables:   $\mathcal{G}$ :  clause graph := $\mathcal{G}_0$;
             L, K, M : literals;  d, e : nodes;
1:           $\mathcal{G} := \mathcal{G} + \langle c, R\rangle$;
2:           For all $L \in C_c$
2.1:           Choose K, d                                   -- significant ancestor
               such that $d \in \{a,b\}$ and $K \in C_d$ and $L = \theta\sigma K$;
2.2:           For all M, e
               such that $M^e K^d$ is a link in $\mathcal{G}$ and
               L and K are potentially complementary
2.2.1:           $\mathcal{G} := \mathcal{G} + M^e L^c$;
Output:      $\mathcal{G}$.
```

The diagram below illustrates the effect of ρ_0:



Each literal of the resolvent is obtained from one or several literal occurrences (the so-called ancestors) in the parent clauses. In step 2.1 the algorithm ρ_0 indeterministically chooses one of these ancestors, the so-called significant ancestor. Only links connecting significant ancestors are relevant for the inheritance mechanism. We call this mono-inheritance. Kowalski's original rule performs multi-inheritance, i.e. each link connecting an ancestor is relevant for the inheritance mechanism. Obviously, mono-inheritance not only generates fewer links in case of a merge, but also leads to a more efficient implementation. All completeness results known for the connection graph proof procedure hold also when mono-inheritance is used instead of multi-inheritance.

Note that a graph resulting from the application of ρ_0 to a total graph is total again. In cases where e = a or e = b (a, b, and c are displayed in the diagram above) the effect of the inheritance mechanism is rather sophisticated. For a further discussion see /Ko79/. The complete clause graph resolution rule is defined by:

```
*Algorithm*   $\rho$;
*Input:*      $\mathcal{G}_0$ : clause graph;
              $\ell$ : non-autolink in $\mathcal{G}_0$;
*Variable:*   $\mathcal{G}$ : clause graph := $\mathcal{G}_0$;
*1:*          $\mathcal{G} := \rho_0(\mathcal{G}, \ell)$;        -- adding part
*2:*          $\mathcal{G} := \pi(\mathcal{G} - \ell)$;          -- deletion part
*Output:*     $\mathcal{G}$.
```

The deletion part of ρ can delete substantial parts of the graph. As an example consider graph (e) in section 2, which is π-reduced and unsatisfiable. An application of ρ to the link $P\bar{P}$ reduces graph (e) to the empty graph, regardless of the choice of the significant ancestors. Note that Kowalski's original rule deletes only the two parent nodes, because it generates a new link to each of the two occurrences of $\bar{Q}$.

An important point of the clause graph resolution rule is that a deleted link is not inherited by the resolvent. Thus, one deleted link can cause several missing links later on. This reduction effect is increased by the subsequent π-reduction. In addition, mono-inheritance allows the choice of that ancestor as significant that is incident with the fewest number of links.

3.2 <u>The τ-Rule</u>.

```
*Algorithm*   $\tau$;
*Input:*      $\mathcal{G}$ : clause graph;
              a : node in $\mathcal{G}$ such that $C_a$ is a tautology;
*Output:*     $\pi(\mathcal{G} - a)$.
```

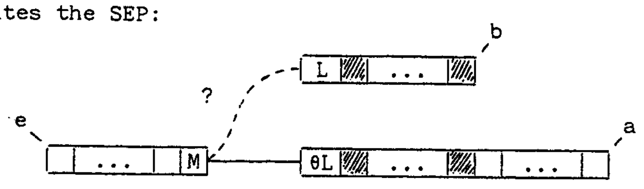
3.3 <u>The σ-Rule</u>. Let C and D be clauses. An <u>S-substitution from D to C</u> is a substitution θ such that $\theta D \subset C$ and $|\theta D| = |D|$. <u>D subsumes C</u>, if there exists an S-Substitution θ from D to C. <u>D subsumes C strictly</u>, if D subsumes C, and if D is not a variant of C. Our definition of subsumption differs from θ-subsumption as defined in /Lo78/. The reason is:

3.3.1 <u>Lemma</u>. There is no infinite sequence $C_1C_2C_3\dots$ of clauses such that C_{n+1} strictly subsumes C_n for all $n \geq 1$.

Proof. The complexity of a clause $C = \{L_1, \dots, L_n\}$ may be defined by $[C] := |L_1| + \dots + |L_n| - |var(C)|$, where $|L_i|$ denotes the length of the literal L_i as a string and $|var(C)|$ is the number of different variables occurring in C. It is not hard to show that $[C] > [D]$ holds for any two clauses C and D such that D subsumes C strictly. Hence an infinite sequence $C_1C_2C_3\dots$ as above is impossible. []

The links in a clause graph are of significant importance because of the inheritance mechanism and the π-reduction. In particular, a node a cannot be eliminated just because there exists another node b such that C_b subsumes C_a. It was first pointed out by Bibel /Bi81c/ that b has to "subsume" a also by its links.

3.3.2 The Subsumption Elimination Property. Let $\mathcal{G}$ be a clause graph. A node a in $\mathcal{G}$ is said to satisfy the subsumption elimination property (SEP) in $\mathcal{G}$, if there exists a node b in $\mathcal{G}$ different from a such that C_b subsumes C_a with an S-substitution θ, and if for each literal $L \in C_b$ and for each link $M^e \theta L^a$ in $\mathcal{G}$ also $M^e L^b$ is a link in $\mathcal{G}$. The diagram below illustrates the SEP:



A node a in $\mathcal{G}$ is said to satisfy the SEP strictly, if a satisfies the SEP in $\mathcal{G}$ and C_a is strictly subsumed by C_b. The subsumption elimination rule for clause graphs is given by:

Algorithm σ;
Input: $\mathcal{G}$: clause graph;
a : node that satisfies the SEP in $\mathcal{G}$;
Output: $\pi(\mathcal{G} - a)$.

3.4 The Inference Relations $\xrightarrow[\rho]{}$, $\xrightarrow[\tau]{}$, $\xrightarrow[\sigma]{}$, and $\longrightarrow$. The relations $\xrightarrow[\rho]{}$, $\xrightarrow[\tau]{}$, $\xrightarrow[\sigma]{}$, and $\longrightarrow$ are binary relations on the set of all clause graphs and are defined as:

- $\mathcal{G} \xrightarrow[\rho]{\ell} \mathcal{G}'$ iff $\mathcal{G}$ and ℓ are proper input data for ρ and $\mathcal{G}'$ is a possible output.
- $\mathcal{G} \xrightarrow[\rho]{} \mathcal{G}'$ iff there exists a link ℓ such that $\mathcal{G} \xrightarrow[\rho]{\ell} \mathcal{G}'$.
- $\mathcal{G} \xrightarrow[\sigma]{a} \mathcal{G}'$ iff $\mathcal{G}$ and a are proper input data for σ and $\mathcal{G}'$ is a possible output.
- $\mathcal{G} \xrightarrow[\sigma]{} \mathcal{G}'$ iff there exists a node a such that $\mathcal{G} \xrightarrow[\sigma]{a} \mathcal{G}'$.
- $\mathcal{G} \xrightarrow[\tau]{a} \mathcal{G}'$ iff $\mathcal{G}$ and a are proper input data for τ and $\mathcal{G}'$ is a possible output.
- $\mathcal{G} \xrightarrow[\tau]{} \mathcal{G}'$ iff there exists a node a such that $\mathcal{G} \xrightarrow[\tau]{a} \mathcal{G}'$.
- $\mathcal{G} \longrightarrow \mathcal{G}'$ iff $\mathcal{G} \xrightarrow[\rho]{} \mathcal{G}'$ or $\mathcal{G} \xrightarrow[\sigma]{} \mathcal{G}'$ or $\mathcal{G} \xrightarrow[\tau]{} \mathcal{G}'$.

4. REDUCED REFUTATION GRAPHS

This paper is concerned with confluence and completeness properties of the inference system defined by $\longrightarrow$. In order to show these properties we introduce a second inference system which consists of two rules applicable to special clause graphs, called reduced refutation graphs.

4.1 Clause Trees. A trail of a clause graph $\mathcal{G}$ is an alternating sequence $a_0 \ell_1 a_1 \ldots a_{n-1} \ell_n a_n$, $n \geq 0$ of nodes and links in $\mathcal{G}$, such that all links are distinct and each link is incident with the two nodes immediately preceding and following it. This trail joins a_0 and a_n and may also be denoted $a_1 a_2 \ldots a_n$ (the links being evident by context). It is called a cycle if $a_0 = a_n$ and $n \geq 1$. A clause graph is connected if every pair of nodes is joined by a trail. A clause graph is acylic if it has no cycles. Obviously, an acyclic clause graph contains no autolinks. A clause tree is a connected acyclic clause graph, in which each literal occurrence is incident with exactly one link. Obviously, any clause tree is π-reduced. As an example, graph (c) in section 2 is a clause tree.

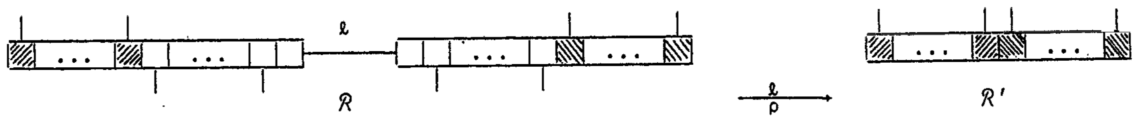
4.2 Refutation Trees. A non-empty clause tree $\mathcal{R}$ is called a refutation tree, if there exists a substitution ξ such that for each link L^aK^b in $\mathcal{R}$ ξ is a unifier of L and $\bar{K}$. ξ is called a unifier for $\mathcal{R}$. For each refutation tree there exists a most general unifier. Graph (c) in section 2 is a refutation tree with {x/b, y/a, z/a, u/b} as a most general unifier. Any refutation tree with only one node contains the empty clause.

4.3 τ-Trails. Let $\mathcal{R}$ be a refutation tree. A trail $a_0\ell_1a_1\ldots a_{n-1}\ell_na_n$, $n \geq 0$ of $\mathcal{R}$ is called a τ-trail of $\mathcal{R}$, if there exists a unifier ξ for $\mathcal{R}$ and two literals $L \in C_{a_0}$ and $K \in C_{a_n}$ such that ξ is a unifier of L and $\bar{K}$, L^{a_0} is not incident with ℓ_1, and K^{a_n} is not incident with ℓ_n. A τ-trail indicates a potential tautology.

4.4 Reduced Refutation Trees (RR-Trees). An RR-tree is a refutation tree containing no τ-trails. An RR-tree $\mathcal{R}$ is said to be an RR-tree for a clause set S, if each clause in $\mathcal{R}$ is a variant of a clause in S. Obviously, an RR-tree contains no tautologies and no τ-links. Graph (c) in section 2 is an RR-tree.

4.5 Theorem. Let $\mathcal{R}$ be an RR-tree, and $\mathcal{R} \xrightarrow[\rho]{\ell} \mathcal{R}'$. Then $\mathcal{R}'$ is an RR-tree which contains the resolvent corresponding to ℓ.

The diagram below illustrates the effect of the application of ρ to an RR-tree $\mathcal{R}$:



The significant ancestors are drawn hatched. Resolution upon ℓ results in merging the two parent clauses incident to ℓ and in deleting the lower parts of the tree $\mathcal{R}$.

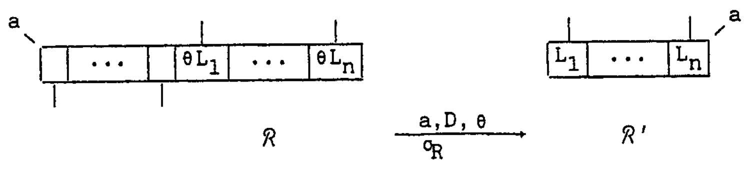
Note that the restriction of $\xrightarrow[\rho]{}$ to RR-trees is Noetherian and confluent. The $\xrightarrow[\rho]{}$-normal form of each RR-tree is ($\square$).

4.6 Theorem. The following statements about a set S of clauses are equivalent:
(a) S has a unit-refutation.
(b) S plus its factors has a unit-refutation without factoring and without tautologies.
(c) S has an input-refutation.
(d) S plus its factors has an input-refutation without factoring and without tautologies.
(e) S plus its factors has an RR-tree.

The equivalence of (a) and (c) was shown by Chang /Ch70/. A weaker version of theorem 4.6 using refutation trees only is proved in /HR78/. A detailled proof of theorem 4.6 is included in /Sm82/.

Let $\mathcal{R}$ be an RR-tree. Then each deduction $\mathcal{R} \xrightarrow[\rho]{*} (\square)$ corresponds to a resolution refutation of $S(\mathcal{R})$ without factoring and without tautologies. Thus $\mathcal{R}$ represents a whole class of refutations for $S(\mathcal{R})$. This class always contains a unit- and an input-refutation. A unit-refutation is obtained by a deduction $\mathcal{R}\xrightarrow[\rho]{*}(\square)$ in which only unit-links are resolved upon. Note that each RR-tree different from $(\square)$ contains at least one unit-link. An input-refutation is obtained by a linear deduction $\mathcal{R}\xrightarrow[\rho]{*}(\square)$, i.e. after the first ρ-step only links connecting the newly generated resolvent are resolved upon.

4.6 The σ_R-Rule. Let $\mathcal{R}$ be an RR-tree, a a node in $\mathcal{R}$, $D = \{L_1,\dots,L_n\}$ a clause, and θ an S-substitution from D to C_a (i.e. $\theta D \subset C_a$ and $|\theta D| = |D|$). The application of the σ_R-rule to $<\mathcal{R},a,D,\theta>$ is defined according to the diagram below:



The graph $\mathcal{R}'$ results from $\mathcal{R}$ by replacing C_a by D and by deleting the lower part of $\mathcal{R}$. In /Sm82/ the following theorem is proved:

4.7 Theorem. Let $\mathcal{R}$ be an RR-tree, and $\mathcal{R}\xrightarrow[\sigma_R]{}\mathcal{R}'$. Then $\mathcal{R}'$ is an RR-tree.

5. KERNELS AND THEIR TRANSFORMATIONS

5.1 Projections. Let $\mathcal{G} = $<NODES,C,LINKS> and $\mathcal{G}' = $<NODES',C',LINKS'> be two clause graphs. A projection of $\mathcal{G}$ into $\mathcal{G}'$ is an ordered pair $<\hat{\ },\pi>$ such that:

(a) $\hat{\ }$ is a function NODES $\longrightarrow$ NODES' such that for each node a in $\mathcal{G}$ $C'_{\hat{a}}$ is a variant of C_a. We write $\hat{a}$ instead of $\hat{\ }(a)$.

(b) π is a substitution such that $\pi C_a = C'_{\hat{a}}$ for all nodes a in $\mathcal{G}$.

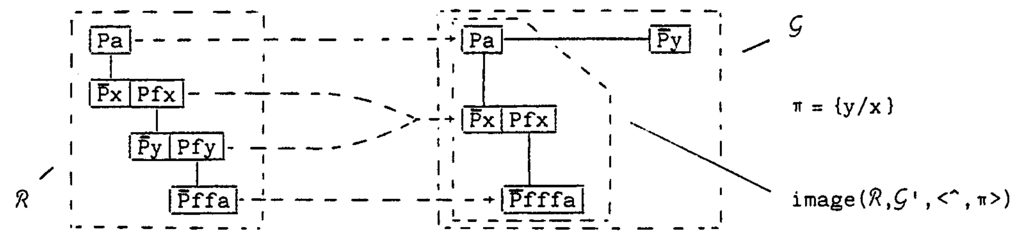
(c) For each link $\ell = L^aK^b$ in $\mathcal{G}$ the link $\hat{\ell} = \pi L^{\hat{a}}\pi K^{\hat{b}}$ is a link in $\mathcal{G}'$.

We write $<\hat{\ },\pi>: \mathcal{G}\longrightarrow\mathcal{G}'$, if $<\hat{\ },\pi>$ is a projection of $\mathcal{G}$ into $\mathcal{G}'$. In a context where $\hat{\ }$ and π are not relevant, $<\hat{\ },\pi>$ is abbreviated to $<>: \mathcal{G}\longrightarrow\mathcal{G}'$. If $<>: \mathcal{G}\longrightarrow\mathcal{G}'$, then image($\mathcal{G},\mathcal{G}'$,<>) denotes the subgraph of $\mathcal{G}'$ which consists of the nodes $\{\hat{a}\mid$ a is node in $\mathcal{G}\}$ and the links $\{\hat{\ell}\mid \ell$ is link in $\mathcal{G}\}$. We write image($\mathcal{G}$) instead of image($\mathcal{G},\mathcal{G}'$,<>), if $\mathcal{G}'$ and <> are evident from the context.

5.2 Kernels. A kernel of a clause graph $\mathcal{G}$ is an ordered pair $<\mathcal{R},\pi>$ consisting of an RR-tree $\mathcal{R}$ and a projection $<>: \mathcal{R}\longrightarrow\mathcal{G}$. The complexity of a kernel $<\mathcal{R},\pi>$ is defined as $|\mathcal{R}|$ (i.e. the number of nodes in $\mathcal{R}$). Obviously, if $<\mathcal{R},\pi>$ is a kernel of $\mathcal{G}$ with complexity 1, then $\mathcal{G}$ contains the empty clause, because $\mathcal{R} = (\square)$. The diagram on the next page gives an example of a kernel $<\mathcal{R},<\hat{\ },\pi>>$ of a clause graph $\mathcal{G}$.

5.3 Theorem (Kernel Properties). Let $<\mathcal{R},<>>$ be a kernel of a clause graph $\mathcal{G}$. Then image($\mathcal{R}$) is an unsatisfiable and π-reduced subgraph of $\mathcal{G}$ containing no tautologies and no τ-links. If image($\mathcal{R}$) $\neq (\square)$ then image($\mathcal{R}$) contains a unit-non-τ-link.

5.4 Theorem (Initialisation). For any unit-refutable clause set S that contains its factors, there exists a kernel of the initial graph $\mathcal{G}_S$.



Theorem 5.4 is a direct consequence of theorem 4.6. Since image($\mathcal{R}$) is π-reduced (theorem 5.3), the application of the ρ-, σ-, and τ-rule to a node or a link not in image($\mathcal{R}$) does not affect the kernel at all. If a link ℓ in image($\mathcal{R}$) is resolved upon, the kernel is transformed by resolving upon each link in $\mathcal{R}$ which is mapped onto ℓ by <>:

5.5 Theorem (ρ-Transformation). Let $\mathcal{G}$ and $\mathcal{G}'$ be clause graphs with $\mathcal{G} \xrightarrow[\rho]{\ell} \mathcal{G}'$, and let $<\mathcal{R},<>>$ be a kernel of $\mathcal{G}$. Then there exists a kernel $<\mathcal{R}',<>'>$ of $\mathcal{G}'$ such that:

$$\begin{tikzcd} \mathcal{G} \arrow[r, "\ell", "\rho"'] & \mathcal{G}' \\ \mathcal{R} \arrow[u, "<>"] \arrow[r, "*", "\rho"'] & \mathcal{R}' \arrow[u, "<>'"'] \end{tikzcd}$$

If ℓ is in image($\mathcal{R}$), then $|\mathcal{R}| > |\mathcal{R}'|$. If ℓ is not in image($\mathcal{R}$), then $\mathcal{R} = \mathcal{R}'$ and $<> = <>'$.

Since <> may map more than one link in $\mathcal{R}$ onto ℓ, the resolution step at the $\mathcal{G}$-level may result in several resolution steps at the $\mathcal{R}$-level. Each resolution step at the $\mathcal{R}$-level decreases the complexity of the kernel, i.e. the number of nodes of the corresponding RR-tree. Since a kernel contains no tautologies (see theorem 5.3), it is not affected by the τ-rule:

5.6 Theorem (τ-Transformation). Let $\mathcal{G}$ and $\mathcal{G}'$ be clause graphs with $\mathcal{G} \xrightarrow[\tau]{} \mathcal{G}'$. Then, each kernel of $\mathcal{G}$ is also a kernel of $\mathcal{G}'$.

5.7 Theorem (σ-Transformation). Let $\mathcal{G}$ and $\mathcal{G}'$ be clause graphs with $\mathcal{G} \xrightarrow[\sigma]{a} \mathcal{G}'$, and let $<\mathcal{R},<>>$ be a kernel of $\mathcal{G}$. Then there exists a kernel $<\mathcal{R}',<>'>$ of $\mathcal{G}'$ such that:

$$\begin{tikzcd} \mathcal{G} \arrow[r, "a", "\sigma"'] & \mathcal{G}' \\ \mathcal{R} \arrow[u, "<>"] \arrow[r, "*", "\sigma_R"'] & \mathcal{R}' \arrow[u, "<>'"'] \end{tikzcd}$$

If a is in image($\mathcal{R}$), then $\mathcal{R} \xrightarrow[\sigma_R]{+} \mathcal{R}'$, whereby a strict σ-step projects to strict σ_R-steps. If a is not in image($\mathcal{R}$), then $\mathcal{R} = \mathcal{R}'$ and $<> = <>'$.

6. COMPLETENESS THEOREMS AND COUNTEREXAMPLES

A refutation of a clause graph $\mathcal{G}$ is a deduction $\mathcal{G} \longrightarrow \mathcal{G}_1 \longrightarrow \ldots \longrightarrow \mathcal{G}_n$ such that $\mathcal{G}_n$ contains the empty clause. Since a refutation of $\mathcal{G}$ corresponds to a resolution refutation of S($\mathcal{G}$), the following theorem holds:

6.1 Theorem (Soundness). Let S be a clause set. Then S is unsatisfiable if there exists a refutation of $\mathcal{G}_S$.

Obviously, the converse of the soundness theorem does not hold, because the inference system $\longrightarrow$ has no device to generate factors. We have not provided a factoring rule because it is not necessary for URFC-sets (unit-refutable clause sets containing their factors), as the following theorem shows:

6.2 Theorem (Confluence). Let S be a URFC-set, and let $\mathcal{G}$ be a graph with $\mathcal{G}_S \xrightarrow{*} \mathcal{G}$. Then there exists a refutation $\mathcal{G} \xrightarrow[\rho]{} \mathcal{G}_1 \xrightarrow[\rho]{} \dots \xrightarrow[\rho]{} \mathcal{G}_n$, $\square \in S(\mathcal{G}_n)$ of $\mathcal{G}$, in which only unit-non-τ-links are resolved upon.

Proof. Let S and $\mathcal{G}$ be as above. The existence of a kernel $\langle \mathcal{R}, <> \rangle$ of $\mathcal{G}$ follows from the theorems 5.4, 5.5, 5.6, and 5.7. If $\mathcal{G}$ does not contain the empty clause, it follows from theorem 5.3 that image($\mathcal{R}$) contains a unit-non-τ-link ℓ. Since resolution upon ℓ decreases the complexity of the kernel, after at most $|\mathcal{R}|$ steps the empty clause will be generated. []

This shows that the inference system $\longrightarrow$ is complete for URFC-sets, i.e. for each such S there exists a refutation of $\mathcal{G}_S$. In addition, we know that $\longrightarrow$ is confluent for URFC-sets modulo an appropriate equivalence relation, i.e. for any such S and for any graph $\mathcal{G}$ with $\mathcal{G}_S \xrightarrow{*} \mathcal{G}$ there exists a refutation of $\mathcal{G}$. The confluence property implies that backtracking is not necessary when searching for a refutation of $\mathcal{G}_S$. But since there exist infinite deductions $\mathcal{G}_S \longrightarrow \mathcal{G}_1 \longrightarrow \mathcal{G}_2 \longrightarrow \dots$ even for URFC-sets, we need a condition to guarantee the termination of the search:

6.3 Theorem (Termination). Let S be a URFC-set, and let $\mathcal{G}$ be a graph with $\mathcal{G}_S \xrightarrow{*} \mathcal{G}$. Then there exists no infinite deduction $\mathcal{G} \longrightarrow \mathcal{G}_1 \longrightarrow \mathcal{G}_2 \longrightarrow \dots$ satisfying the following conditions:

(a) None of the graphs $\mathcal{G}_i$ contains the empty clause.

(b) Each unit-non-τ-link is deleted again.

(c) Backward subsumption is applied only strictly.

In the context of a deduction a σ-step is called a backward subsumption step, if it does not delete a resolvent which was generated in the step preceding immediately.

Proof. By contradiction there exists an infinite deduction as above. As a consequence of the theorems in section 5, there exist RR-trees and projections such that

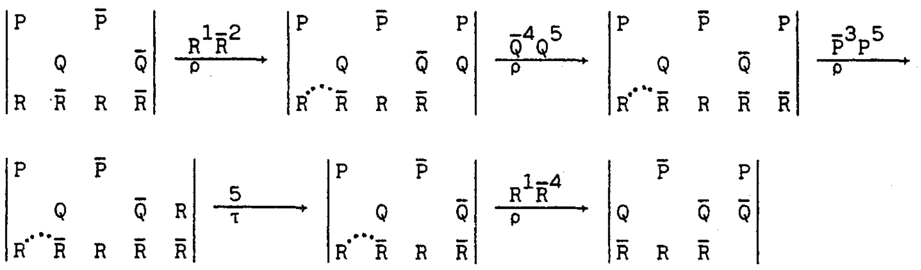
$$\begin{tikzcd}
\mathcal{G}_S \arrow[r] & \mathcal{G}_1 \arrow[r] & \mathcal{G}_2 \arrow[r] & \cdots\cdots \arrow[r] & \mathcal{G}_i \arrow[r] & \cdots\cdots \\
\mathcal{R}_0 \arrow[u, "<>_0"] \arrow[r, "*", "R"'] & \mathcal{R}_1 \arrow[u, "<>_1"] \arrow[r, "*", "R"'] & \mathcal{R}_2 \arrow[u, "<>_2"] \arrow[r, "*", "R"'] & \cdots\cdots \arrow[r, "*", "R"'] & \mathcal{R}_i \arrow[u, "<>_i"] \arrow[r, "*", "R"'] & \cdots\cdots
\end{tikzcd}$$

where $\xrightarrow[R]{}$ is defined as $\xrightarrow[R]{} := \xrightarrow[\rho]{} \cup \xrightarrow[\sigma_R]{}$. Since $|\mathcal{R}_0| \geq |\mathcal{R}_1| \geq \dots \geq 2$ holds, there exists a number m such that $2 \leq |\mathcal{R}_m| = |\mathcal{R}_{m+1}| = \dots$. Thus, from $\mathcal{R}_{m+1}$ on, only strict σ_R-steps can be applied at the $\mathcal{R}$-level. However by lemma 3.3.1, only a finite number of successive strict σ_R-steps are possible. Hence there exists a number n such that $\mathcal{R}_n = \mathcal{R}_{n+1} = \dots$ and $<>_n = <>_{n+1} = \dots$. Thus for all $i \geq n$, image($\mathcal{R}_n$) $\subset \mathcal{G}_i$ holds. That contradicts the fact that image($\mathcal{R}_n$) contains a unit-non-τ-link (theorem 5.3). []

As a consequence of the theorems 6.2 and 6.3, every non-backtracking strategy terminates with the generation of the empty clause for any URFC-set, provided each unit-non-τ-link has a finite chance to be deleted. This result also holds if every tautology and every subsumed resolvent is immediately deleted. Furthermore, strict backward subsumption is possible.

Graph (a) in section 2 is the initial graph of a unit-refutable clause set that does not contain its factors. Each of the four links is a τ-link. Consequently, there exists a deduction starting with Graph (a) and ending with the empty graph. Thus, the Kowalski-Calculus is not confluent for unit-refutable clause sets which do not contain their factors.

6.4 Non-Confluence of the Kowalski-Calculus for Propositional Clause Sets. The theorems 6.2 and 6.3 do not hold even for unsatisfiable propositional clause sets, although at the propositional level factoring is not necessary. As an example consider the following deduction:



The propositional clause graphs in this deduction are given in a matrix representation similiar to the one used by Bibel in /Bi81c/. Each column represents a node of the graph. Instead of the existing links only the missing links are drawn by dotted lines. Note that the first matrix above represents graph (b) in section 2. In order to display the nodes and links of a graph in matrix representation, the columns of the matrix are numbered consecutively, such that $L^n\bar{L}^m$ denotes the link that joins the literal L in the nth column and the literal $\bar{L}$ in the mth column.

The first graph in the deduction above is the initial graph of the *unsatisfiable* clause set {PR, Q$\bar{R}$, $\bar{P}$R, $\bar{Q}\bar{R}$}. The last graph is *satisfiable* and the empty graph can be deduced by further applications of the ρ- and the τ-rule. Therefore, unrestricted deletion of tautologies as proposed in /Ko75, Ko79/ is admissable for URFC-sets, but even for general propositional clause sets it is inconsistent and causes the loss of completeness.

Bibel /Bi81c/ defines a restriction of the τ-rule similiar to the SEP (see 3.3.2). In /Sm82/ we have shown an equivalent of theorem 6.2 for propositional clause sets and the restricted τ-rule. But contrary to URFC-sets the equivalent of theorem 6.3 does not hold: In /Sm82/ we give a counterexample for an infinite deduction starting from the initial graph of an unsatisfiable propositional clause set which only uses the ρ-rule and the restricted τ-rule and in which each node is deleted after a finite number of steps. Hence the restriction of /Bi81c/ is too weak to ensure termination.

ACKNOWLEDGEMENT. I would like to thank Jörg Siekmann and Norbert Eisinger for many stimulating discussions and their invaluable assistance and advice in the preparation of this work.

REFERENCES

Bi81a W. BIBEL, A strong completeness result for the connection graph proof procedure. Technische Universität München, 1981.

Bi81b W. BIBEL, On the completeness of connection graph resolution. Proc. of GWAI-81, Springer Informatik Fachberichte 47, 1981, 246-247.

Bi81c W. BIBEL, On matrices with connections. J. ACM 28,4 (Oct. 1981), 633-645.

Br75 M. BRUYNOOGHE, The inheritance of links in a connection graph. Report CW2, Katholike Universiteit Leuven, 1975.

Br76 F. BROWN, Notes on chains and connection graphs. Personal Notes, Dep. of Computation and logic, Edinburgh University, 1976.

Ch70 C.L. CHANG, The unit proof and the input proof in theorem proving. J. ACM 17,4 (Oct. 1970), 698-707.

HR78 M.C. HARRISON and N. RUBIN, Another generalisation of resolution. J. ACM 25,3 (Oct. 1978), 341-351.

Hu80 G. HUET, Confluent reductions: abstract properties and applications to term rewriting systems. J. ACM 27,4 (Oct. 1980), 797-821.

HW74 L. HENSCHEN and L. WOS, Unit refutations and Horn-sets. J. ACM 22,4 (Oct. 1974), 590-605.

Ko75 R. KOWALSKI, A proof procedure using connection graphs. J. ACM 22,4 (Oct. 1975), 572-595.

Ko75 R. KOWALSKI, Logic for problem solving. North Holland, 1979.

Ku72 D. KUEHNER, Some special purpose resolution systems. 1972. In Meltzer and Michie (Eds), Machine intelligence 7. Edinburgh University Press, 1972.

Lo78 D.W. LOVELAND, Automated theorem proving: a logical basis. North Holland, 1978.

Oh82 H.J. OHLBACH, Terminator: an efficient proof procedure for unit-refutable clause sets. Universität Karlsruhe, Institut für Informatik I, to appear 1982.

Sh76 R.E. SHOSTAK, Refutation graphs. Artif. Int. 7 (1976), 823-835.

Sm82 G. SMOLKA, Einige Ergebnisse zur Vollständigkeit der Beweisprozedur von Kowalski. Diplomarbeit, Universität Karlsruhe, Institut für Informatik I, 1982.

SS76 J.H. SIEKMANN and W. STEPHAN, Completeness and soundness of the connection graph proof procedure. Universität Karlsruhe, Institut für Informatik I, 1976.

SS80 J.H. SIEKMANN and W. STEPHAN, Completeness and consistency of the connection graph proof procedure. Universität Karlsruhe, Institut für Informatik I, 1980.