Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Artificial Intelligence 144 (2003) 59–93

Artificial Intelligence

www.elsevier.com/locate/artint

A relevance restriction strategy for automated deduction

David A. Plaisted [a,*], Adnan Yahya [b]

[a] *Department of Computer Science, UNC Chapel Hill, Chapel Hill, NC 27599-3175, USA*
[b] *Electrical Engineering Department, Birzeit University, Birzeit, Palestine*

Received 6 September 2001; received in revised form 15 August 2002

Abstract

Identifying relevant clauses before attempting a proof may lead to more efficient automated theorem proving. Relevance is here defined relative to a given set of clauses S and one or more distinguished sets of support T. The role of a set of support T can be played by the negation of the theorem to be proved or the query to be answered in S which gives the refutation search goal orientation. The concept of *relevance distance* between two clauses C and D of S is defined using various metrics based on the properties of paths connecting C to D. This concept is extended to define relevance distance between a clause and a set (or multiple sets) of support. Informally, the relevance distance reflects how closely two clauses are related. The relevance distance to one or more support sets is used to compute a *relevance set* R, a subset of S that is unsatisfiable if and only if S is unsatisfiable. R is computed as the set of clauses of S at distance less than n from one or more support sets; if n is sufficiently large then R is unsatisfiable if S is. If R is much smaller than S, a refutation from R may be obtainable in much less time than a refutation from S. R must be efficiently computable to achieve an overall efficiency improvement. Different relevance metrics are defined, characterized and related. The tradeoffs between the amount of effort invested in computing a relevance set and the resulting gains in finding a refutation are addressed. Relevance sets may be utilized with arbitrary complete theorem proving strategies in a completeness-preserving manner. The potential of the advanced relevance techniques for various applications of theorem proving is discussed

Keywords: Relevance; Relevance metrics; Theorem proving; Sorted inference

* Corresponding author.
E-mail addresses: plaisted@cs.unc.edu (D.A. Plaisted), yahya@ee.birzeit.edu (A. Yahya).
URL address: http://www.birzeit.edu/eng/enee/yahya (A. Yahya).

0004-3702/02/$ – see front matter
doi:10.1016/S0004-3702(02)00368-5

1. Introduction

Many applications can be presented as theorem proving tasks. Among those are proving mathematical theorems, query answering in deductive databases under different semantics, hardware verification tasks, nonmonotonic reasoning tasks and many others. The standard automated deduction technique to prove a theorem is to show that the set of clauses representing the given axioms and theorem negation is unsatisfiable. In many theorem proving applications the vast majority of data does not participate in the refutation. This is a result of a small subset of the theory being unsatisfiable. Irrelevant data may be dragged into the computation resulting in inefficient proof search. It is of interest to determine the set of clauses that are necessary for a refutation, a fact that can result in substantial computational savings. Clauses needed for the refutation are *relevant* to the theorem proving process. The selection of nonrelevant clauses in the theorem proving process was recognized as a significant source of inefficiency for many theorem provers. This is especially so in the presence of large amounts of data such as in query answering in large databases. To rectify this, relevance testing techniques were suggested for incorporation into theorem provers to improve their efficiency [6–8,14,18,20,23].

The goal of this paper is to allow for more efficient automated deduction through the use of relevance. Often it is the case that one or more clauses of the theory are distinguished: e.g., the negation of the query in deductive databases, the negation of the theorem in mathematical theorem proving and the goal in other tasks. If a refutation exists, some of the distinguished elements must contribute to this refutation. Any of these may be distinguished as a set of support (SOS) and one is generally interested in refutations that include the set of support. This is because the original theory (database, axiom set) is (assumed to be) consistent and it may become inconsistent only after adding the set of distinguished elements. Therefore one may attempt to find clauses that contribute to the refutation that involve the set of support. Relevance is defined relative to a given support set which serves as the *seed* for propagating relevance. This set of relevant clauses is referred to as the *relevance set* for the given theory and support set.

The relevance set is a subset of the original set of clauses sufficient to generate a refutation if the original theory is inconsistent. The smaller this set, the better, with the limit being the minimally unsatisfiable set with minimal cardinality.

The relevance of a clause to the support set can be parameterized by a distance function that defines how many intermediate clauses are needed to connect a given clause to the support set. The concept of related clauses can be defined in terms of links; two clauses are linked if they have two complementary literals the underlying atoms of which unify. This definition can be further refined to exclude more clauses from being declared relevant, thus decreasing the size of the relevance set. However, finding a compact relevance set can be costly. The tradeoffs between the costs of computing the relevant set and the gains achieved from working exclusively with the relevance set during the refutation search phase have to be considered.

Certain theorem proving strategies such as model elimination [12,13] and the set of support restriction of resolution [12] already restrict the search to clauses that are closely related to a single support set. A study presented in [24] gives evidence for the superiority of such *goal-sensitive* strategies. However, methods presented here permit the search to

be restricted to clauses that are closely related to multiple sets of support simultaneously, giving a more wholistic strategy. Also, goal-sensitive strategies such as model elimination and set of support are often difficult to reconcile with efficient equality strategies such as paramodulation. The methods of this paper permit one to make arbitrary complete theorem proving strategies such as resolution goal sensitive, retaining completeness, by applying them to the relevant set of clauses. Even for a single set of support, relevance strategies have a different behavior from model elimination and the set of support restriction of resolution. Relevance strategies permit one to find deep proofs involving clauses closely related to the goal, whereas model elimination and the set of support strategy cannot generate deep proofs without also selecting clauses that are only distantly related to the goal.

Relevance is appealing because it corresponds to "associations" between facts that humans use in problem solving. Facts that are associated with a problem to be solved are the ones most likely to be used in the attempted solution, as well as facts that are associated in turn with these.

The rest of the paper is organized as follows: Section 2 presents some definitions and background material. Section 3 defines relevance and relevance distance using links and paths. Section 4 presents relationships between relevance distance and proof complexity, as well as giving algorithms for finding relevant clauses. Section 5 presents other definitions of paths and discusses their impact on the concept of relevance. Section 6 discusses relationships between alternative path definitions and proof complexity. Section 7 discusses general relevance metrics and especially the incorporation of sorts into relevance measures. Section 8 discusses practical applications of relevance measures. Section 9 compares the relevance measures presented here with other approaches in the literature. Finally, Section 10 presents some conclusions.

2. Definitions and background

We adopt the standard notation as, e.g., in [5,12,17]. A first-order language $\mathcal{L}$ with function symbols is assumed. The symbols $\rightarrow$, $\wedge$, and $\vee$ denote implication, conjunction, and disjunction, respectively.

A *literal* is an atom or negated atom. If L is a literal then by $Atm(L)$ we denote the atom occurring in L. Given a set of literals $\mathcal{A}$ by $Atm(\mathcal{A})$ we denote the set of atoms occurring in $\mathcal{A}$. A literal is *negative* if it is preceded by a $\neg$, else it is *positive*. The literals L and $\neg L$ are *complementary*.

A *clause* C is a disjunction of literals, often written as a set.

A *theory* S is a set (conjunction) of clauses. We assume that no clause has two complementary literals.

A literal L is *pure* in a set of clauses S if all occurrences of L in S have the same polarity (all are positive or all are negative).

A term or formula in which no variables occur is said to be *ground.*

The *Herbrand base* of $\mathcal{L}$ is the set of ground atoms over $\mathcal{L}$.

An *interpretation* I is a (possibly infinite) set of literals in which each element of the Herbrand base occurs once either positively or negatively. All elements of I are assigned the truth value *true*.

An interpretation I *satisfies* a clause C if the intersection of the sets of literals of I and C is nonempty. Otherwise C is falsified (violated) in I.

An interpretation I is a *model* of a set of clauses S if it satisfies all clauses of S. If S has a model it is *satisfiable*, else it is *unsatisfiable*.

Occasionally, as is common in the literature, we may refer to an interpretation as the set of atoms it assigns *true* with the remaining atoms assigned *false*. Clearly, set inclusion defines a partial order on interpretations defined in this manner. A model M of S is minimal in the set inclusion sense if no proper subset of M is a model of S.

Two special atoms are used: $\top$ and $\bot$ expressing truth and falsity, respectively. $\top$ is satisfied in every interpretation, but no interpretation satisfies $\bot$.

A *substitution* θ is a finite set of the form $\{x_1/t_1, x_2/t_2, \ldots, x_n/t_n\}$ where x_i are distinct variables, and each t_i is a term with no occurrences of x_i. If S is a set of clauses containing only variables in θ and all t_i are ground then θ is a *grounding substitution* for S. If t_i are all distinct variables then θ is a *renaming* substitution. An empty substitution is called the *identity* substitution.

Given an expression (term, literal) E and a substitution θ, $E\theta$ is the result of substituting each x_i by t_i in E. $E\theta$ is called an *instance* of E.

Given a set of clauses S then by $grnd(S)$ we denote the set of (possibly infinite) ground instances of S.

Given a clause C and a set Σ of substitutions, Σ is *compatible on* C if there is an instance C' of C such that for all $\sigma \in \Sigma$, C' is an instance of $C\sigma$.

Given two substitutions θ and σ, the composite substitution $E\theta\sigma$ is the result of applying σ to $E\theta$. That is, $(E\theta)\sigma$.

These definitions can be extended to sets of expressions and formulae, in the obvious way.

Two atoms L_1 and L_2 are *unifiable* if and only if there is a substitution σ such that $L_1\sigma = L_2\sigma$. σ is a *unifying substitution* (*unifier*). Unifiers for sets of terms, atoms or literals are defined similarly. A unifier θ is a *most general unifier* (*mgu*) for a set S if θ is a unifier for S and for every unifier σ for S there is a substitution λ such that $S\theta\lambda = S\sigma$.

Two clauses C_1 and C_2 not sharing variables *resolve* if and only if there are two literals $L_1 \in C_1$ and $L_2 \in C_2$ and substitution σ such that $L_1\sigma = \neg L_2\sigma$. If σ is the mgu of L_1 and $\neg L_2$ then the *resolvent* of these clauses on L_1 and L_2, $Res(C_1, C_2, L_1, L_2)$, is $(C_1 \setminus \{L_1\})\sigma \cup (C_2 \setminus \{L_2\})\sigma$. The literals L_1 and L_2 are called the *literals of* the resolution, σ is called the *substitution* of the resolution, and (σ, σ) is called the *pair of substitutions* of the resolution. If C_1 and C_2 share variables, then $Res(C_1, C_2, L_1, L_2)$ is defined as $Res(C_1\theta, C_2, L_1\theta, L_2)$ where θ renames variables of C_1 so that no variables are shared with C_2. In this case, if (σ, σ) is the pair of substitutions of the resolution $Res(C_1\theta, C_2, L_1\theta, L_2)$ then $(\theta\sigma, \sigma)$ is the pair of substitutions for the resolution $Res(C_1, C_2, L_1, L_2)$. If L_1 and L_2 are obvious then $Res(C_1, C_2, L_1, L_2)$ may be written as $Res(C_1, C_2)$.

Definition 2.1. A *resolution proof* of clause C from a set of clauses S is a sequence of clauses $C_1, C_2, \ldots, C_n$ where C_n is C and each C_i is either in S or a resolvent of two clauses C_j and C_k, and $j, k < i$. A *refutation* (of S) is a proof of the empty clause $\Box$. n is the *length* of the proof (refutation if $C = \Box$).

Definition 2.2. Given a set of clauses S then $T \subset S$ is a set of support (SOS) for S if $S \setminus T$ is consistent.

$\mathcal{T}$ is a support class for S if $\mathcal{T} = \{T_1, \ldots, T_n\}$ and T_i is a SOS for S for $i \in \{1, \ldots, n\}$.

3. Relevance basics

Relevance is meant to filter out clauses that do not contribute to the refutation. Generally relevance is defined relative to a set of clauses S and a distinguished set of support T. The role of T may be played by the (negation of the) query or the theorem to be proved.

The idea of relevance is to find a subset R of S, or a subset R of the instances of elements in S, such that R preserves the refutational properties of S with respect to T. If such an R exists then a refutation may be sought in R rather than in S. Minimality of R is important as one would like to search for a refutation in a set with the least number of clauses. It is also important that the effort in computing R should be small relative to the effort in finding a refutation directly from S.

The concept of *related* clauses can be defined through unification (for resolution). Two clauses C_1 and C_2 are linked (related) if they resolve with each other. Clauses may be related indirectly through links to intermediate clauses. Therefore one may talk about relevance distance between two clauses C_1 and C_2. To emphasize that the relevance distance between C_1 and C_2 depends on the underlying set S, we denote such a distance by $d_S(C_1, C_2)$. We drop the S subscript and use $d(C_1, C_2)$ when the context is clear.

Two clauses are directly *linked* if they resolve. The literals on which they resolve are called *link literals*. Two clauses may resolve on more than one pair of literals and therefore may have more than one direct link. Examples are $C_1 = a \vee \neg b$, $C_2 = \neg a \vee b$ linking on both a and b; and $C_3 = P(a) \vee Q(b)$, $C_4 = \neg P(x) \vee \neg Q(x)$ linking on both $P(a)$ and $Q(b)$. The following definition formalizes these concepts.

Definition 3.1. Given a set of clauses S:

- Let C_1 and C_2 be two clauses of S and suppose that there is a resolvent $Res(C_1, C_2, L_1, L_2)$. Then (C_1, C_2, L_1, L_2) is called a *link* between C_1 and C_2. We say that C_1 and C_2 are *directly connected (linked)* through L_1 and L_2. There may be more than one link between C_1 and C_2; for example, there are two links between $\{p, q\}$ and $\{\neg p, \neg q\}$.
- A *path*[1] between clause C and clause D in S is a sequence $C_1, C_2, \ldots, C_m$ of clauses in S, such that $C_1 = C$, $C_m = D$ and there are links $(C_i, C_{i+1}, L_i, M_{i+1})$ between C_i and C_{i+1} for all i, $1 \leqslant i < m$. $m - 1$ is the *length* of the path.
- The (relevance) *distance* between clauses C and D in S, $d_S(C, D)$, is the length of the shortest path between C and D in S. Thus $d_S(C, C) = 0$ for any clause C in S. $d_S(C, D) = \infty$ if no path exists between C and D.
- If T is a set of clauses in S then the distance between T and clause C is the minimal distance between C and a clause in T. $d_S(C, T) = \min(\{d_S(C, D) \mid D \in$

[1] In [20] the path is defined through the connection graph of S. The correspondence between the two definitions is evident. Note that multiple paths may exist between two clauses.

$T\}$). Additionally, because $d_S(C, C) = 0$ for any clause C we have $d_S(C, T) = 0$ if $C \in T$. If $\{T_1, \ldots, T_n\}$ is a set of subsets of S, then $d_S(C, \{T_1, \ldots, T_n\}) = \max(\{d_S(C, T_1), \ldots, d_S(C, T_n)\})$.

Example 1. Consider $S = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9\}$ where $C_1 = a$, $C_2 = \neg a \vee b \vee c$, $C_3 = \neg b$, $C_4 = \neg c \vee e$, $C_5 = \neg a \vee e \vee f$, $C_6 = \neg e \vee r$, $C_7 = \neg r$, $C_8 = \neg s \vee f$, $C_9 = s \vee f$.

- Clearly C_1 is directly connected to C_2 through $(a, \neg a)$.
  Note that C_9 is not connected to C_5. f has the same polarity in both.
- $d(C_1, C_2) = 1$. $d(C_2, C_3) = 1$. $d(C_2, C_4) = 1$.
  C_3 is not directly connected to any clause besides C_2.
- The following are the paths from C_1 to C_6:
  C_1, C_2, C_4, C_6 and the path length is 3.
  C_1, C_5, C_6 and the path length is 2.
  No path exists between C_1 and C_9 and therefore $d(C_1, C_9) = \infty$.
- In view of the previous item the distance from C_1 to C_6 is 2: $d(C_1, C_6) = 2$.
- Let $T_1 = \{C_1, C_9\}$.
  $d(C_6, T_1) = \min(\{d(C_6, C_1), d(C_6, C_9)\}) = \min(\{2, \infty\}) = 2$.
- Let $T_2 = \{C_2, C_5\}$. $d(C_6, T_2) = \min(\{d(C_6, C_2), d(C_6, C_5)\}) = \min(\{2, 1\}) = 1$.
  $d(C_6, \{T_1, T_2\}) = \max(\{2, 1\}) = 2$.

Convention. Given a set of clauses $S = \{C_1, C_2, \ldots, C_N\}$ and a support class $\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$ we will write $S = \bigcup_{i=1}^{N} \{C_i(d^{i,1}, d^{i,2}, \ldots, d^{i,K})\}$, where $d^{i,j} = d_S(C_i, T_j)$ for all $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$. That is, in parentheses after clause C we list the distances of C from each element of the support class $T_1, T_2, \ldots, T_K$ in that order.

4. Relevance distance and refutations

Given a large set of clauses S we would like to utilize the clause distance from the support set to compute a subset of S, say R, in which to perform the search for a refutation. The idea is that the sum of the cost of a refutation search in the *relevance set* R and the cost of computing R may be less than the cost of the search for the refutation in S.

Definition 4.1. Given an unsatisfiable set of clauses S, the *Herbrand complexity* of S is the minimum integer n such that there is an unsatisfiable set of n ground instances of the clauses in S.

Actually, for purposes of relevance it is generally not the size of the unsatisfiable set that matters but its *diameter*, that is, the smallest bound d such that for some unsatisfiable set G of ground instances, any two elements of G are at distance d from one another. This diameter may be much smaller than the number of elements in G.

Definition 4.2. A function $f_n(S)$ from nonnegative integers n and sets S of clauses to sets R of clauses is a *relevance (bounding) function* if

(1) every clause in $f_n(S)$ is an instance of some clause in S and
(2) if S is unsatisfiable and n is the Herbrand complexity of S then $f_n(S)$ is unsatisfiable.

Definition 4.3. A set R of clauses is a *relevance (bounding) set* if R is $f_n(S)$ for some relevance function f.

The idea is that any clause not needed for a refutation is not relevant. If n is the Herbrand complexity of S, clauses not in $f_n(S)$ are not needed for the refutation. The function f is called a relevance *bounding* function because clauses that are included need not be relevant, but clauses that are excluded are irrelevant. A number of relevance functions will be presented.

Relevance functions may be used together with arbitrary complete theorem proving strategies. However, relevance functions require a value n for the estimate of Herbrand complexity. Since the computation of the Herbrand complexity is as hard as testing satisfiability, it is better not to attempt to compute the Herbrand complexity in advance but rather to use an iterative procedure in which the value of n is gradually increased.

Let proof_search(R, b) be a procedure that applies some complete theorem proving strategy to a set R of clauses with a bound b on the effort performed. The quantity b may bound the time spent, the number of inferences performed, or the depth of the search, for example. Assume that this procedure returns "unsat" to indicate that R is unsatisfiable, and that unsatisfiability was detected within the effort bound b. Assume that if R is unsatisfiable and b is sufficiently large, then proof_search(R, b) returns "unsat". For efficiency purposes, assume also that if proof_search(R_1, b_1) is called and then proof_search(R_2, b_2) is called, and $R_1 \subseteq R_2$ and $b_1 \leqslant b_2$, then the total search performed is the same as if only proof_search(R_2, b_2) had been called, but the inferences performed during the call to proof_search(R_1, b_1) are not repeated.

A relevance function f and a complete theorem proving strategy "proof_search" implicitly define a complete theorem proving strategy, which we call the *relevance theorem proving strategy* for f and "proof_search," as follows:

```
procedure relevance_strat(S, δ, γ);
  for j = 1, 2, 3, ... do
    for k = 1, 2, 3, ..., j do
      if proof_search(f_kδ(S), jγ) returns "unsat"
      then return "unsatisfiable" fi;
    od;
  od;
end relevance_strat;
```

Theorem 1. *If $\delta > 0$ and $\gamma > 0$ then the procedure relevance_strat(S, δ, γ) returns "unsatisfiable" iff S is unsatisfiable.*

Proof. If the procedure returns "unsatisfiable", then S is unsatisfiable because every clause in $f_n(S)$ is an instance of some clause in S. If S is unsatisfiable, then for some integer n, $f_n(S)$ is unsatisfiable, so in some finite effort bound a complete theorem proving strategy will detect this. Since $\delta > 0$ and $\gamma > 0$, there are j and k such that unsatisfiability of $f_{k\delta}(S)$ can be detected within an effort bound of $j\gamma$. Therefore the procedure relevance_strat will eventually return "unsatisfiable". □

A relevance filter may also be used to delete clauses that are generated during the calls to proof_search, while maintaining the completeness of the procedure relevance_strat. However, a consideration of this topic is beyond the scope of this paper.

Path length may be used to define a relevance function, as follows:

Definition 4.4. A set R of clauses is *minimally unsatisfiable* if R is unsatisfiable but any proper subset of R is satisfiable.

It is necessary to show that minimally unsatisfiable sets of clauses are connected by paths.

Theorem 2. *Suppose S is an unsatisfiable set of clauses and R is a minimally unsatisfiable subset of S. Then for every pair C, D of clauses in R, $d_R(C, D) < \infty$.*

Proof. Pick $C \in R$ and let R_C be the set of D in R such that $d_R(C, D) < \infty$. If $R_C = R$ we are done. Suppose R_C is a proper subset of R. Since R is minimally unsatisfiable, R_C is satisfiable. Also, $R \setminus R_C$ is satisfiable. Let M_1 be a model of R_C and M_2 be a model of $R \setminus R_C$. Since there are no links between R_C and $R \setminus R_C$, we can construct a model M that satisfies both R_C and $R \setminus R_C$ by letting M agree with M_1 on ground instances of atoms in R_C and letting M agree with M_2 elsewhere. This implies that $M \models R$, which contradicts our assumption that R is unsatisfiable. □

Theorem 3. *Let S be a set of clauses. If S has Herbrand complexity n and T is a set of support for S then there exists an unsatisfiable set of clauses R with the following relevance properties*:

For every clause C in R, $d_S(C, T) \leqslant n - 1$.

Proof. Since S has Herbrand complexity n, there is an unsatisfiable set G of n ground instances of clauses in S. Let G be a minimally unsatisfiable set. Let R be a minimal subset of S such that every clause in G is an instance of some clause in R. Then R is also minimally unsatisfiable. Therefore R is connected, by Theorem 2. Also, R has at most n clauses in it, and R must contain at least one element D of T because R is unsatisfiable. Therefore for all C in R, $d_R(C, D) < \infty$. Since R has at most n clauses, $d_R(C, D) < n$ for all $C \in R$. □

Corollary 1. *The function $f_n(S) = \{C \in S\colon\ d_S(C, T) \leqslant n - 1\}$ is a relevance function for S if T is a set of support for S. Also, $f_n(S)$ is computable in time polynomial in S.*

Proof. $f_n(S)$ is computable in polynomial time using well-known shortest path algorithms on graphs. □

Example 2. Given the set of clauses, $S =$

$$\begin{array}{ll}
\{P(a) \vee Q(b) & \neg P(b) \\
\neg P(x) \vee P(f(x)) \vee Q(f(x)) & \neg P(f(x)) \\
\neg Q(x) \vee P(x) \vee R(x) & \neg P(x) \vee \neg Q(f(x)) \\
\neg R(x) \vee \neg Q(x) & Q(c)\}
\end{array}$$

Let $T = \{\neg P(b)\}$, $n = 7$.

The minimally unsatisfiable set G is given below, where the number in parentheses after a clause defines the distance of that clause from T. $G = \{P(a) \vee Q(b)(2), \neg Q(b) \vee P(b) \vee R(b)(1), \neg P(b)(0), \neg R(b) \vee \neg Q(b)(2), \neg P(a) \vee P(f(a)) \vee Q(f(a))(3), \neg P(f(a))(4), \neg P(a) \vee \neg Q(f(a))(3)\}$.

G is a minimally unsatisfiable set. It has 7 clauses. Each two clauses in this set are connected and the maximum distance of an element in G from T is 4 confirming the results of the last two theorems. Note that $d_S(Q(c), T) = 2$ but $Q(c)$ is not in the minimally unsatisfiable set.

$R = f_7(S) =$

$$\begin{array}{ll}
\{P(a) \vee Q(b)(2) & \neg P(b)(0) \\
\neg P(x) \vee P(f(x)) \vee Q(f(x))(3) & \neg P(f(x))(4) \\
\neg Q(x) \vee P(x) \vee R(x)(1) & \neg P(x) \vee \neg Q(f(x))(3) \\
\neg R(x) \vee \neg Q(x)(2) & Q(c)(2)\}
\end{array}$$

4.1. Fully matched sets and refutations

Definition 4.5. A set of clauses S is *fully matched* if $\forall C_1 \in S\ \forall L_1 \in C_1\ \exists C_2 \in S\ \exists L_2 \in C_2$ [L_1 and $\neg L_2$ are unifiable].

Clearly, a clause $C \in S$ is not fully matched in S if and only if a literal L of C is pure in S.

Theorem 4. *If S is a set of clauses then there is a maximal* (*possibly empty*) *subset S' of S such that S' is fully matched.*

Proof. The union of an arbitrary collection of fully matched sets is fully matched. Therefore the union of all fully matched subsets of S is the maximal fully matched subset of S. □

Lemma 1. *Let R be a minimally unsatisfiable set of clauses. Then R is fully matched.*

Proof. Suppose R is not fully matched. Then there is a clause $C \in R$ and a literal $L_1 \in C$ such that there is no link (C, D, L_1, L_2) for any clause D in R. Now, $R \setminus \{C\}$ is satisfiable

because R is a minimally unsatisfiable subset of S. Let M be a model of $R \setminus \{C\}$. Since there is no link (C, D, L_1, L_2) in R, we can modify M to satisfy all the ground instances of L_1 without contradicting any new clauses of R. Let M' be M modified in this way. Then $M' \models L_1$ hence $M' \models C$. Since $M \models R \setminus \{C\}$, $M' \models R \setminus \{C\}$. Thus $M' \models R$. This contradicts the assumption that R was unsatisfiable. □

Lemma 2. *Let S be a set of clauses. Let S' be the maximal fully matched set of S. S is unsatisfiable if and only S' is unsatisfiable.*

Proof. One direction is straightforward.

Assume that S is unsatisfiable.

Let R be a minimally unsatisfiable subset of S. By Lemma 1, R is fully matched. Since S' is the maximal fully matched set of S, $R \subseteq S'$. Therefore S' is unsatisfiable. □

Note that the maximal fully matched set may contain multiple refutations as well as elements that are not part of any refutation.

Example 3. Let S be $\{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$ where $C_1 = a \vee b$, $C_2 = \neg a \vee c \vee d$, $C_3 = \neg b \vee c \vee d$, $C_4 = \neg b \vee \neg d$, $C_5 = \neg a \vee \neg c$, $C_6 = \neg a \vee e$, $C_7 = \neg b \vee e$, $C_8 = \neg e$.

S is the maximal fully matched set. $\{C_1, C_6, C_7, C_8\}$ is the minimally unsatisfiable subset and all other clauses have no contribution to the refutation despite being fully matched.

Clearly the union of S and any other fully matched (and unsatisfiable, maybe with no links to S) set is fully matched.

Theorem 5. *Suppose S is unsatisfiable and has Herbrand complexity n. Let T be a support set for S. Let R be the maximal subset of S such that R is fully matched and such that for all $C \in R$, $d_R(C, T \cap R) \leqslant n - 1$. Then R is unsatisfiable.*

Proof. Let G be a minimally unsatisfiable set of n ground instances of S. Then G is fully matched by Lemma 1. Also, for all C, D in G, $d_G(C, D) < n$ by Theorem 3. Let R' be the set of clauses in S having a ground instance in G. Then $R' \cap T \neq \phi$ because T is a support set and R' is unsatisfiable. Thus $R' \subseteq S$ is fully matched and for all $C \in R'$, $d_{R'}(C, T) \leqslant n - 1$. Therefore the maximal such subset R of S is also unsatisfiable. □

Corollary 2. *If T is a support set for S then $f_n(S)$ is a relevance function for S, where $f_n(S)$ is the maximal subset R of S such that R is fully matched and such that for all $C \in R$, $d_R(C, T \cap R) \leqslant n - 1$. Also, $f_n(S)$ is computable in time polynomial in S.*

Proof. An algorithm to compute f in polynomial time is given below. This algorithm requires a number of iterations bounded by the number of clauses in S, and each iteration is also polynomial by straightforward arguments. □

It is desirable to compute a set R as in the theorem and search for refutations from R instead of S. However, computing R is not trivial and will be discussed in the next section.

4.2. Computing a fully matched relevance set

The relevance theorem proving strategy is generally expected to find a refutation, if one exists, that involves less clauses than the entire set S and will favor shallower refutations (shorter proofs) within a smaller distance from the support set, though it may not be able to find the shortest proof.

The number of clauses that get selected in the computation can vary depending on the relevance distance metric used. Better performance generally results from computing the relevance set based on refined metrics that tend to maximize the distance between clauses because this may render more clauses outside of the given distance bound and therefore prevent them from participating in the computation. We will investigate some possible refinements later in this paper.

Another approach to reducing the size of the relevance set is to have a possibly larger than needed estimate of n and to successively refine the set of clauses in R by imposing additional restrictions on clauses that may be included in the refutation process. The removal of clauses not obeying these restrictions may make certain clauses "out of reach" (at distance more than $n-1$) from T and therefore will allow them to be removed as well. We can iterate the process until a fixed point is reached. That is, the process halts when an iteration produces no changes to the current relevance set.

As an example[2] we consider using full-matching to refine the set R. This is based on the observation that only fully matched clauses can contribute to a refutation (Lemma 2). The following is a simple way to compute R, where T is a support set for S:

Let $R_1 = \{C \in S\colon \exists D \in T \text{ s.t. } d_S(C, D) \leqslant n-1\}$.

Let R_2 be the maximal fully matched set of R_1.

Let $R_3 = \{C \in R_2\colon \exists D \in T \cap R_2 \text{ s.t. } d_{R_2}(C, D) \leqslant n-1\}$.

Let R_4 be the maximal fully matched set of R_3 and so on until $R_{N+2} = R_N$ for some integer N.

The complete formal algorithm for the computation is as follows:

Algorithm 1 (*Computing relevance set refined through full matching*). Given a set of clauses[3] S, an estimate n on the Herbrand complexity of S, and a set of support T for S.

```
Let i = 0, R_0 = S.
Repeat
    {
    R_{i+1} = {C ∈ R_i: ∃D ∈ T ∩ R_i s.t. d_{R_i}(C, D) ⩽ n − 1}.
    Let R_{i+2} be the maximal fully matched subset of R_{i+1}.
    i = i + 2.
    }
Until R_{i−2} = R_i.
```

[2] Other possible refinements include the removal of tautological clauses and subsumed clauses.

[3] Or the maximal fully matched subset of S.

$R_{i+1} \subseteq R_i$ for all i. The sequence of sets R_i is monotone decreasing relative to the partial order induced by set inclusion. The process always has a fixpoint[8,11]. It terminates in finite time for finite n because in this case R_1 is a finite set.

Clearly, if n is chosen properly (say at least as large as the Herbrand complexity of S) then S is unsatisfiable if and only if R_∞ is. The convergence (speed) of the fixpoint computation depends on the set S and the support set T, and the value of n.

Example 4. Consider the set of clauses S and $T_1 = \{C_1\}$. (The number in parentheses after the clause number is its distance from the set of support T_1 relative to the set in which the clause is listed).

$S = \{C_1(0), C_2(1), C_3(2), C_4(2), C_5(1), C_6(2), C_7(3), C_8(4), C_9(\infty)\}$ where $C_1 = a$, $C_2 = \neg a \vee b \vee c$, $C_3 = \neg b$, $C_4 = \neg c$, $C_5 = \neg a \vee e \vee f$, $C_6 = \neg e \vee r \vee c$, $C_7 = \neg r \vee e$, $C_8 = \neg s \vee r \vee t$, $C_9 = \neg s \vee v \vee t$.

Since the first 4 clauses constitute an unsatisfiable set, we see that $n = 4$.

$R_1 = \{C_1(0), C_2(1), C_3(2), C_4(2), C_5(1), C_6(2), C_7(3)\}$.

No links on f exist. Clause C_5 is not fully matched and can be deleted.

$R_2 = R_1 \setminus \{C_5\} = \{C_1(0), C_2(1), C_3(2), C_4(2), C_6(3), C_7(4)\}$.

Note the change to the distances of C_6 and C_7 from T in R_2. In particular, C_7 is at distance 4 and therefore will be deleted in computing R_3.

$R_3 = \{C_1(0), C_2(1), C_3(2), C_4(2), C_6(3)\}$.

No links on e exist in R_3. Clause C_6 is not fully matched and can be deleted.

$R_4 = \{C_1(0), C_2(1), C_3(2), C_4(2)\}$.

R_4 is the fixpoint ($R_4 = R_\infty$).

If $T_2 = \{C_3\} = \{\neg b\}$ then:

$S = \{C_1(2), C_2(1), C_3(0), C_4(2), C_5(3), C_6(3), C_7(4), C_8(5), C_9(\infty)\}$.

$R_1 = \{C_1(2), C_2(1), C_3(0), C_4(2), C_5(3), C_6(3)\}$.

$R_2 = \{C_1(2), C_2(1), C_3(0), C_4(2)\}$.

The convergence is faster than in the previous case.

Given T and a distance bound n, the process of computing the relevance set consists of iterating between finding a fully matched set of clauses R_i and computing the distance between elements of T in R_i and other clauses of R_i and removing any clause at distance more than $n-1$ to get R_{i+1}. The process stops after the iteration in which no clauses are removed.

4.3. Multiple sets of support

So far we considered cases when the refutation was sought relative to a single set of support. The choice of the set of support can influence the computation of the relevance set.

It is possible that more than one support set is available and one may want to utilize that for more efficient computations.

By its nature, elements of each set of support have to be included in all possible refutations. We have the following analogue of Theorem 5.

Theorem 6. *Suppose S is unsatisfiable and has Herbrand complexity n. Let $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ be a support class for S. Let R be the maximal subset of S such that R is fully matched and such that for all $C \in R$, for $1 \leqslant i \leqslant k$, $d_R(C, T_i \cap R) \leqslant n-1$. Then R is unsatisfiable.*

Proof. Let G be a minimally unsatisfiable set of n ground instances of S. Then G is fully matched by Lemma 1. Also, for all C, D in G, $d_G(C, D) < n$ by Theorem 3. Let R' be the set of clauses in S having a ground instance in G. Then for all i, $1 \leqslant i \leqslant k$, $R' \cap T_i \neq \phi$ because R' is unsatisfiable and the T_i are support sets for S. Thus R' is fully matched and for all $C \in R'$, for $1 \leqslant i \leqslant k$, $d_{R'}(C, T_i) \leqslant n-1$. Therefore the maximal such subset R of S is also unsatisfiable. □

This theorem may give a smaller set R than Theorem 5 because clauses at distance n or more from *any* of the sets of support in $\mathcal{T}$ are removed. The resulting relevance set is at most as large as the smallest relevance set for the individual components of $\mathcal{T}$. Actually it is in the intersection of all relevance sets of the individual components of $\mathcal{T}$.

Corollary 3. *Suppose S is a set of clauses and $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ is a support class for S. Then f is a relevance function, where $f_n(S)$ is defined as the largest subset R of S such that R is fully matched and for all $C \in R$, for $1 \leqslant i \leqslant k$, $d_R(C, T_i \cap R) \leqslant n-1$. Also, $f_n(S)$ is computable in time polynomial in S and k.*

Another possible way to define the relevance function is the largest subset R of S such that R is fully matched and for all $C \in R$, $(1/k)\sum_{1 \leqslant i \leqslant k} d_R(C, T_i) \leqslant n-1$. This *average* relevance function is more sensitive to each distance and may give better results.

The complete formal algorithm for the computation of the relevance set using multiple sets of support is as follows:

Algorithm 2 (*Computing relevance set refined through full matching for $\mathcal{T}$*). Given a set of clauses S, an estimate n of the Herbrand complexity of S, and a support class $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ for S.

Let $i = 0$, $R_0 = S$.
Repeat
{
$R_{i+1} = \{C \in R_i \mid \forall\, T_j \in \mathcal{T}\ d_{R_i}(C, T_j \cap R_i) \leqslant n-1\}$.
Let R_{i+2} be the maximal fully matched set of R_{i+1}.
$i = i + 2$.
}
Until $R_{i-2} = R_i$.

The computation of R_{i+1} can be done more efficiently as follows:

$R_{i+1} \leftarrow R_i$; **for** $j = 1, 2, \ldots, k$
do $R_{i+1} \leftarrow \{C \in R_{i+1}\colon d_{R_{i+1}}(C, T_j \cap R_{i+1}) \leqslant n-1\}$.

This can be further sped up by choosing T_1 so that a small number of clauses are at small distances from it.

Consider the following example which is an extension of Example 4 to work with two support sets.

Example 5. Consider the set of clauses S and $\mathcal{T} = \{T_1, T_2\}$ where $T_1 = \{a\}$ and $T_2 = \{\neg b\}$.

$S = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9\}$ where $C_1 = a, C_2 = \neg a \vee b \vee c, C_3 = \neg b, C_4 = \neg c, C_5 = \neg a \vee e \vee f, C_6 = \neg e \vee r \vee c, C_7 = \neg r \vee e, C_8 = \neg s \vee r \vee t$, and $C_9 = \neg s \vee v \vee t$.

Using $n = 4$ we get (the pair of numbers in parentheses gives clause distance from T_1 and T_2 in that order).

$R_1 = \{C_1(0, 2), C_2(1, 1), C_3(2, 0), C_4(2, 2), C_5(2, 3), C_6(2, 3)\}$.

Note that $C_7(3, 4)$ is not included in R_1 here because the distance from T_2 is greater than 3.

Removing clauses that are not fully matched we get $R_2 = \{C_1(0, 2), C_2(1, 1), C_3(2, 0), C_4(2, 2)\}$. R_2 is the fixpoint.

The result coincides with the case for T_2 alone.

The following example demonstrates that the result of multiple support sets may not be achievable by a single support set. It also demonstrates that the algorithm need not compute a minimally unsatisfiable set as the fixpoint.

Example 6. Consider the set of clauses S and $\mathcal{T} = \{T_1, T_2\}$ where $S = \{C_1, C_2, \ldots, C_{11}\}$ and $T_1 = \{C_1\}$ and $T_2 = \{C_3\}$. and $C_1 = a, C_2 = \neg a \vee b \vee c, C_3 = \neg b, C_4 = \neg c, C_5 = \neg b \vee \neg c \vee a, C_6 = \neg e \vee b, C_7 = \neg e \vee f, C_8 = \neg f \vee e, C_9 = \neg a \vee g, C_{10} = \neg g \vee h, C_{11} = \neg h \vee g$.

The set $\{C_1, C_2, C_3, C_4\}$ is unsatisfiable and using $n = 4$ we get (the pair of numbers in parentheses gives clause distance from T_1 and T_2 in that order).

$R_0 = \{C_1(0, 2), C_2(1, 1), C_3(2, 0), C_4(2, 2), C_5(2, 2), C_6(3, 1), C_7(5, 3), C_8(4, 2), C_9(1, 3), C_{10}(2, 4), C_{11}(3, 5)\}$.

Removing clauses at distance 4 or more from T_1 or T_2 yields:

$R_1 = \{C_1(0, 2), C_2(1, 1), C_3(2, 0), C_4(2, 2), C_5(2, 2), C_6(3, 1), C_9(1, 3)\}$.

C_6 and C_9 are not fully matched as $\neg e$ and g, occurring in C_6 and C_9 respectively, are pure.

$R_2 = \{C_1(0, 2), C_2(1, 1), C_3(2, 0), C_4(2, 2), C_5(2, 2)\}$.

R_2 is the fixpoint. Note that clause C_5 is not in the minimally unsatisfiable subset of S.

Let superscripts denote which supports sets are used in the computation.

For T_1, $R_\infty^{T_1} = \{C_1(0), C_2(1), C_3(2), C_4(2), C_5(2), C_9(1), C_{10}(2), C_{11}(3)\}$.

For T_2, $R_\infty^{T_2} = \{C_1(2), C_2(1), C_3(0), C_4(2), C_5(2), C_6(1), C_7(3), C_8(2)\}$.

For $\{T_1, T_2\}$, $R_\infty^{T_1, T_2} = \{C_1(0, 2), C_2(1, 1), C_3(2, 0), C_4(2, 2), C_5(2, 2)\} = R_\infty^{T_1} \cap R_\infty^{T_2}$.

The following example shows how two support sets can lead to a substantial reduction in the search space.

Example 7. Let S be the following set of clauses:

(1) $at(c_i, s) \rightarrow at(c_j, g(c_j, s))$ for various i, j. (One can go directly from c_i to c_j in any situation s. $g(c_j, s)$ is the action taken.)
(2) $at(c_0, s_0)$. (The starting situation.)
(3) $\neg at(c_m, s)$. (The goal is to get to city c_m.)

The variable s represents a "situation". Define the *action distance* between two cities as the number of actions needed to get from one city to the other. Thus if $at(c_i, s) \rightarrow at(c_j, g(c_j, s))$ then the action distance between c_i and c_j is at most one. Now, the last two clauses are each a support set because both are needed for the proof. If these clauses are each used as a separate support set in the relevance computation, and n is the estimate of Herbrand complexity, then the relevance set essentially contains clauses mentioning only the cities at action distance $n-1$ or less from c_0 *and* c_m. (The full matching requirement may delete even some of these clauses.) If we put both of these clauses in one support set, then the relevance set will contain clauses mentioning only cities at action distance $n-1$ or less from c_0 *or* c_m, which can potentially be a much larger set, especially if there are thousands or hundreds of thousands of assertions. If we put these last two clauses in separate support sets but use the *average* relevance function defined earlier in this section, then the relevance set is the set of clauses mentioning only cities whose *average* action distance from c_0 and c_m is less than $n-1$. Of course, if one knows in advance that the problem has this structure, one can use efficient graph-based algorithms, but one wants a general mechanism because one may not understand the structure of the problem in advance.

For an example where many support sets may be useful, consider the following problem:

Example 8.

(1) $at(a, c_i, s) \rightarrow at(a, c_j, g(a, c_j, s))$ for various i, j. (Any individual a can go directly from c_i to c_j in situation s. $g(a, c_j, s)$ is the action taken.)
(2) $at(a_i, c_i, s_0), 1 \leqslant i \leqslant k$. (The starting situation. Individuals $a_1, \ldots, a_k$ are at cities $c_1, \ldots, c_k$, respectively.)
(3) $\neg at(a_1, c, x_1) \vee \neg at(a_2, c, x_2) \vee \ldots \neg at(a_k, c, x_k)$. (The goal is to get all individuals to some city c.)

The problem is to find a city reachable from k specified cities. Each of the k cities can be made into a support set, as above. Then if n is the estimate of Herbrand complexity, the relevance set will essentially contain clauses mentioning only cities at action distance $n-1$ or less from *all* k cities (again, some of these may be removed by the full matching requirement). No smaller number of support sets can achieve the same reduction in search space. If there are thousands or hundreds of thousands of assertions, this could result in a significant reduction in search space. In general, if one is proving a theorem and several assertions are all known to be essential for the proof, then each of these assertions can be made into a support set to make the relevance computation more effective than for a single support set.

One may question whether the computation of relevance sets can be efficient for large knowledge bases S containing hundreds of thousands of clauses. In some cases, the size of

the knowledge base does not have much influence. If n is small, then a relevance set can be computed using breadth-first search for each support set without even looking at most of the clauses in S. Only clauses at a small distance from the support sets need to be examined. This assumes that the set of links involving a clause C in S may be computed efficiently. If S is stored in a discrimination net, then the links involving C can often be obtained with a small amount of effort. In addition, the effort to update the discrimination net when a new clause is added to S is often likewise small. This permits some large knowledge bases to be used repeatedly for various queries at a small cost per query in updating the discrimination net and computing the relevance set. Of course, the efficiency of computing relevance sets will depend on the structure and size of the knowledge base. Not all large knowledge bases will be suitable for this approach.

5. Various path-based definitions of relevance

So far the relevance metric considered was based on the lengths of paths, possibly refined by the restriction to fully matched clause sets. One can clearly see other possible refinements such as using sort information to disallow certain potential links for having no contribution to the refutation. There may be many *relevance metrics*:

Definition 5.1.

(1) A *relevance metric* r on a set S of clauses is a function from pairs of clauses in S to nonnegative integers.
(2) A relevance metric r for S, is *complete* if for all support sets T for S, $\{C \in S$: $\exists D \in T$ s.t. $r_S(C, D) \leqslant \infty$ for some D in $T\}$ is unsatisfiable if S is unsatisfiable.
(3) A relevance metric r' is *stricter than* r if for any two clauses C_1 and C_2 of S, $r'_S(C_1, C_2) \geqslant r_S(C_1, C_2)$.

Additional relevance metrics may be defined by modifying the definition of a path given in Definition 3.1. Recall especially the definition of the length of a path given there.

Definition 5.2. Given a set of clauses S:

- If the literals L_1 and L_2 are complementary then the link (C_1, C_2, L_1, L_2) is a *basic* link.
- If (C_i, C_{i+1}, L_1, L_2) is a link in a path $P = C_1, \ldots, C_n$ then L_1 is called an *exiting literal* of the pair (C_i, C_{i+1}) in P and L_2 is called an *entering literal* of the pair (C_i, C_{i+1}) in P. (Think of an arrow from C_1 to C_2 with L_1 at the start and L_2 at the end.) These literals may not be unique.
- If $C_1, C_2, \ldots, C_m$ is a path and $\Theta_1, \ldots, \Theta_m$ is a sequence of substitutions, then the sequence $C_1\Theta_1, C_2\Theta_2, \ldots, C_m\Theta_m$ of clauses is called an *instance* of the path $C_1, C_2, \ldots, C_m$. Note that instances of paths need not be paths.
- If $C_1, C_2, \ldots, C_m$ is a path and for all $i, 1 \leqslant i < m$, there is a basic link between C_i and C_{i+1}, then this path is called a *basic* path. If this path has an instance

$C_1\Theta_1, C_2\Theta_2, \ldots, C_m\Theta_m$ that is a basic path, then $C_1, C_2, \ldots, C_m$ is called a *compatible* path. If for all $i < m$ there are links $(C_i, C_{i+1}, L_i, M_{i+1})$ such that for all i, $1 < i < m$, L_i and M_i are distinct literals of C_i (that is, an entering literal of (C_{i-1}, C_i) and an exiting literal of (C_i, C_{i+1}) are distinct), then this path is called an *alternating* path.

- The *basic distance* $d_S^b(C, D)$ between C and D in S is the length of the shortest basic path between C and D in S. The *compatible distance* $d_S^c(C, D)$ between C and D in S is the length of the shortest compatible path between C and D in S. The *alternating distance* $d_S^a(C, D)$ between C and D in S is the length of the shortest alternating path between C and D in S. These notations can be combined, as in $d_S^{a,c}(C, D)$, indicating that only alternating compatible paths are to be considered.
- If C is a clause and T is a set of clauses then $d^a(C, T) = \text{Min}(\{d^a(C, D) \mid D \in T\})$ and similarly for basic and compatible paths.

For example, $\{\neg p, q\}, \{\neg q, r\}, \{\neg r, s\}$ is a basic alternating path. The path $\{\neg p, q\}$, $\{\neg q, r\}, \{q, s\}$ is basic but not alternating because $\neg q$ in the second clause is used for links in both directions. The path $\{p(a)\}, \{\neg p(x), q(x)\}, \{\neg q(b)\}$ is alternating but not basic or compatible. The path $\{p(a)\}, \{\neg p(x), q(x)\}, \{\neg q(a)\}$ is alternating and compatible but not basic.

Intuitively, the smaller the relevance distance between clauses the more closely related they are. It is possible to prove the following obvious relations between these distance measures: $d_S^b(C, D) \geqslant d_S(C, D)$, $d_S^c(C, D) \geqslant d_S(C, D)$, $d_S^a(C, D) \geqslant d_S(C, D)$, $d_S^{b,c}(C, D) = d_S^b(C, D)$, $d_S^{b,a}(C, D) \geqslant d_S^b(C, D)$, $d_S^{b,a}(C, D) \geqslant d_S^a(C, D)$, $d_S^{a,c}(C, D) \geqslant d_S^a(C, D)$, $d_S^{a,c}(C, D) \geqslant d_S^c(C, D)$, and $d_S^{b,a,c}(C, D) = d_S^{b,a}(C, D)$. Also, if S is a set of ground clauses, then $d_S^c(C, D) = d_S^b(C, D) = d_S(C, D)$, and $d_S^{c,a}(C, D) = d_S^{b,a}(C, D) = d_S^a(C, D)$.

5.1. Relevance based on resolution proofs

In this section we define a path and relevance distance based on resolution and show the relationship between the newly introduced metric and the ones discussed so far.

Definition 5.3. A *linear resolution proof* involving $C_1, C_2, \ldots, C_n$ is a sequence $D_1, D_2, \ldots, D_n$ of clauses, where D_1 is C_1 and D_i is a resolvent of C_i and D_{i-1} for $1 < i \leqslant n$. The *length* of such a proof is $n - 1$. Such a proof is said to *connect* C_1 and C_n in S if all C_i are in S.

For example, a linear resolution proof involving $\{q\}, \{\neg q, r\}, \{\neg r, s\}$ is the proof $\{q\}, \{r\}, \{s\}$. This proof connects the clauses $\{q\}$ and $\{\neg r, s\}$. However, there is no linear resolution proof involving the clauses $\{q(a)\}, \{\neg q(x), r(x)\}, \{\neg r(b), s(b)\}$ because the substitutions involved are incompatible.

Definition 5.4. The *resolution distance* $d_S^r(C, C')$ between C and C' is the minimal length of a linear resolution proof that connects C and C' in S.

For example, if $S = \{\{q\}, \{\neg q, r\}, \{\neg r, s\}\}$ then the resolution distance $d_S^r(\{q\}, \{\neg r, s\})$ is 2, which is also the value of $d_S^{a,c}(\{q\}, \{\neg r, s\})$.

In Definition 5.4 we require the compatibility of substitutions for the sequence of links. The latter definition is stricter and generally results in greater distances between clauses than for ordinary links.

We now show how resolution distance relates to the other distance measures.

Lemma 3. *Suppose G is a set of ground clauses. Then for all clauses C, D in G, $d_G^r(C, D) \leqslant d_G^a(C, D)$.*

Proof. Suppose $d_G^a(C, D) = m - 1$. Let $C_1, C_2, \ldots, C_m$ be an alternating path from C to D. Then for $1 < i < m$ we can write C_i as $E_i \cup \{L_i\} \cup \{\neg L_{i+1}\}$ where L_i is the entering literal of the pair (C_{i-1}, C_i) and $\neg L_i$ is the exiting literal of the pair (C_i, C_{i+1}). Also, $C_1 = E_1 \cup \{\neg L_2\}$ and $C_m = E_m \cup \{L_m\}$. Then we can obtain the resolvents D_i defined by $D_i = E_1 \cup E_2 \cup \cdots \cup E_i \cup \{\neg L_{i+1}\}$ as required by Definition 5.4. Thus $d_G^r(C, D) \leqslant m - 1$. □

Theorem 7. *Suppose S is a set of clauses. Then for all clauses C, D in S, $d_S^r(C, D) \leqslant d_S^{a,c}(C, D)$.*

Proof. Suppose $d_S^{a,c}(C, D) = m - 1$. Let $C_1, C_2, \ldots, C_m$ be a compatible alternating path from C to D. Then there must exist ground clauses $C_1', C_2', \ldots, C_m'$ that are instances of $C_1, C_2, \ldots, C_m$, respectively, such that $C_1', C_2', \ldots, C_m'$ is an alternating path, because the path is compatible. By Lemma 3, $d_G^r(C_1', C_m') \leqslant m - 1$ where $G = \{C_1', C_2', \ldots, C_m'\}$. Therefore there is a linear resolution proof involving $C_1', C_2', \ldots, C_m'$. This can be lifted to a linear resolution proof involving $C_1, C_2, \ldots, C_m$. Therefore $d_S^r(C, D) \leqslant m - 1$. □

We now show the other direction of these inequalities.

Definition 5.5. The set of clauses *used* in a resolution proof $C_1, C_2, \ldots, C_n$ is the smallest set R such that (a) $C_n \in R$ and (b) if $C_i \in R$ and C_i is a resolvent of C_j and C_k for $j, k < i$, then $C_j \in R$ and $C_k \in R$.

Lemma 4. *Suppose $C_1, C_2, \ldots, C_n$ is a resolution proof from a set S of clauses. Suppose R is the set of clauses in S that are used in this proof. Then there is a set R' of instances of R and a set $\mathcal{L}$ of links involving clauses in R' such that*

(1) *every literal L of C_n appears in some clause C of R' such that L is not a link literal of C,*
(2) *every pair of clauses of R' are connected by a basic alternating path in which all links appear in $\mathcal{L}$.*

Proof. By induction on the length of the resolution proof. Suppose $C_n \in S$. Then we can let $R = R' = \{C_n\}$. Otherwise, suppose C_n is a resolvent of C_i and C_j for $i, j < n$. Then C_i and C_j are proved by resolution from S by shorter proofs. We can therefore assume

by induction that the theorem is true for C_i and C_j. Let R'_i be the set of instances as in the theorem for C_i, and let R'_j be the set for R_j. Suppose C_n is $Res(C_i, C_j, L, M)$ and suppose the pair of substitutions for this resolution is (θ, σ). Let R' be $R'_i\theta \cup R'_j\sigma$. Then by the definition of resolution, every literal of C_n appears in some clause of R'. Also, $(C_i\theta, C_j\sigma, L\theta, M\sigma)$ is a basic link. Now, by induction, L appears in some clause in R'_i, but not in one of the links, and M appears in some clause of R'_j, but not in one of the links. Therefore $L\theta$ and $M\sigma$ both appear in some clauses of R'. Suppose $L\theta \in D_L$ and $M\sigma \in D_M$. Then $(D_L, D_M, L\theta, M\sigma)$ is a basic link connecting two clauses in R'. The clauses in R'_i (hence in $R'_i\theta$) are already connected by basic alternating paths, as are the clauses in $R'_j\sigma$. Since $D_L \in R_i\theta$ and $D_M \in R_j\theta$, all clauses in R' are connected by basic paths. These paths are alternating because no link literals appear in C_n. □

We also observe that if the resolution proof is linear, then the construction of the lemma yields a set R' having at most n clauses.

Theorem 8. *Suppose S is a set of clauses. Then for all clauses C, D in S, $d^r_S(C, D) \geqslant d^{a,c}_S(C, D)$.*

Proof. Suppose $d^r_S(C, D) = n - 1$. Then there is a linear resolution proof from $C_1, C_2, \ldots, C_n$ where $C = C_1$ and $D = C_n$. By Lemma 4 and the immediately following observation about linear proofs, there is a set R of instances of $\{C_1, C_2, \ldots, C_n\}$ that is connected by basic alternating paths. Thus there is a basic alternating path connecting an instance of C and an instance of D. The length of this path is at most $n - 1$ because $\{C_1, C_2, \ldots, C_n\}$ contains only n clauses. Each clause on this path is an instance of some clause in $\{C_1, C_2, \ldots, C_n\}$, and this is a subset of S. Therefore, by the definition of a compatible path, there is a compatible alternating path from C to D of length at most $n - 1$. Thus $d^{a,c}_S(C, D) \leqslant n - 1$. □

Corollary 4. *Suppose S is a set of clauses. Then for all clauses C, D in S, $d^r_S(C, D) = d^{a,c}_S(C, D)$.*

6. Various path-based distance definitions and refutations

There are some relationships between the path-based definitions of relevance from Section 5 and the Herbrand complexity of a set of clauses.

Theorem 9. *Let S be a set of clauses. If S has Herbrand complexity n and T is a set of support for S then there exists an unsatisfiable set of clauses R with the following relevance properties: For every clause C in R, $d^c_R(C, T) \leqslant n - 1$.*

Proof. Since S has Herbrand complexity n, there is an unsatisfiable set G of n ground instances of clauses in S. Let G be a minimally unsatisfiable set. Let R be a minimal set of clauses in S such that each element of G is an instance of a clause in R. Then R is also minimally unsatisfiable. Therefore R is connected, by Theorem 2. Also, R has at most n

clauses in it, and R must contain at least one element D of T because R is unsatisfiable. Therefore for all C in R, $d_R(C, D) < \infty$. Since R has at most n clauses, $d_R(C, D) < n$ for all $C \in R$. To obtain the bound on d^c, we can find a basic path in G of length $n-1$ or less between an instance of C and an instance of D because G is an unsatisfiable set of ground clauses. By lifting this path to a path in R and choosing the link substitutions to yield the corresponding clauses of G, we obtain links with compatible substitutions. □

Corollary 5. *Let S be a set of clauses. If S has Herbrand complexity n and T is a set of support for S then there exists an unsatisfiable set G of ground instances of clauses in S with the following relevance properties*: *For every clause C in G, there is a clause D in G such that D is an instance of some clause in T and $d^b_G(C, D) \leqslant n-1$.*

Proof. The proof is obtained by applying the theorem to the unsatisfiable set G of ground instances of clauses in S, noting that because G is ground, $d^b_G(C, D) = d_G(C, D)$. □

Consider the following example:

Example 9. Given the set $S =$

$$\begin{array}{ll}
\{P(a) \vee Q(b) & \neg S(b) \\
\neg P(x) \vee P(f(x)) \vee Q(f(x)) & \neg P(f(x)) \\
\neg Q(x) \vee P(x) \vee R(x) & \neg P(x) \vee \neg Q(f(x)) \\
\neg R(x) \vee Q(x) & \neg R(x) \vee S(x)\}
\end{array}$$

Let $T = \{\neg P(x) \vee S(x)\}$.

The minimally unsatisfiable set G is given below, where the number in parentheses after a clause defines the distance of that clause from T, where $D = \neg P(b) \vee S(b)$.

$G = \{\neg P(b) \vee S(b)(0), P(a) \vee Q(b)(3), \neg Q(b) \vee P(b) \vee R(b)(2), \neg S(b)(1), \neg R(b) \vee \neg Q(b)(3), \neg P(a) \vee P(f(a)) \vee Q(f(a))(4), \neg P(f(a))(5), \neg P(a) \vee \neg Q(f(a))(4)\}$.

G is a minimally unsatisfiable set. It has 8 clauses. Thus $n = 8$. Each two clauses in this set are connected and the maximal distance of an element in G from T is 4 confirming the results of Theorem 9 and Corollary 5.

$R =$

$$\begin{array}{ll}
\{P(a) \vee Q(b)(3) & \neg S(b)(1) \\
\neg P(x) \vee P(f(x)) \vee Q(f(x))(4) & \neg P(f(x))(5) \\
\neg Q(x) \vee P(x) \vee R(x)(2) & \neg P(x) \vee \neg Q(f(x))(4) \\
\neg R(x) \vee \neg Q(x)(3) & \neg P(x) \vee S(x)(0)\}
\end{array}$$

We now develop bounds for alternating paths.

Lemma 5. *Suppose $C_1, C_2, \ldots, C_n$ is an alternating path. Suppose $1 < i < j < n$ and C_i and C_j are identical and an entering literal of the pair (C_{i-1}, C_i) is distinct from an exiting literal of the pair (C_j, C_{j+1}). Then $C_1, C_2, \ldots, C_i, C_{j+1}, C_{j+2}, \ldots, C_n$ is also an alternating path.*

Proof. The shorter path still satisfies the definition of an alternating path. □

Lemma 6. *Suppose $C_1, C_2, \ldots, C_n$ is an alternating path. Suppose $1 < i < j < k < n$ and C_i, C_j, and C_k are identical. Then either*

- *An entering literal of (C_{i-1}, C_i) is distinct from an exiting literal of (C_j, C_{j+1}) or*
- *An entering literal of (C_{j-1}, C_j) is distinct from an exiting literal of (C_k, C_{k+1}) or*
- *An entering literal of (C_{i-1}, C_i) is distinct from an exiting literal of (C_k, C_{k+1}).*

Proof. Suppose the first two conditions fail. Then all entering literals of the pair (C_{i-1}, C_i) are identical to all exiting literals of the pair (C_j, C_{j+1}) and all entering literals of the pair (C_{j-1}, C_j) are identical to all exiting literals of the pair (C_k, C_{k+1}). Since the path is alternating, an entering literal of the pair (C_{j-1}, C_j) is distinct from an exiting literal of the pair (C_j, C_{j+1}). Thus an exiting literal of the pair (C_k, C_{k+1}) is distinct from an entering literal of the pair (C_{i-1}, C_i). □

Example 10. Consider the alternating path $P = \{C_1, C_2, \ldots, C_9\}$ where $C_1 = a, C_2 = \neg a \vee b, C_3 = \neg b \vee c, C_4 = \neg c \vee a, C_5 = \neg a \vee b, C_6 = \neg b \vee f, C_7 = \neg f \vee \neg b, C_8 = \neg a \vee b, C_9 = a \vee g$.

Let $i = 2, j = 5, k = 8$. $C_i = C_j = C_k = \neg a \vee b$.

We have the resolvent $(C_1, C_2, a, \neg a) = b$ with the entering literal $\neg a$ and $(C_5, C_6, b, \neg b) = \neg a \vee f$ with the exiting literal b. $\neg a \neq b$ and the shortened path $P' = C_1 = a, C_2 = \neg a \vee b, C_6 = \neg b \vee f, C_7 = \neg f \vee \neg b, C_8 = \neg a \vee b, C_9 = a \vee g$ is alternating as is suggested by Lemma 5.

This observation also confirms the result of Lemma 6.

Note that we have the resolvent $(C_4, C_5, a, \neg a) = \neg c \vee b$ with the entering literal $\neg a$ and $(C_8, C_9, \neg a, a) = b \vee g$ with the exiting literal $\neg a$.

Theorem 10. *Let S be a set of clauses. If S has Herbrand complexity n and T is a set of support for S then there exists an unsatisfiable set of clauses $R \subseteq S$ with the following relevance properties:*

For every clause C in R, $d_R^{a,c}(C, T) \leqslant 2n - 1$.

Proof. In [19] and above in Lemma 4, we showed that if G is a minimally unsatisfiable set of ground clauses, then there is an alternating path between any two clauses in G. This shows that a path exists, but it may be very long. The problem is to show how to obtain a shorter path without violating the alternation condition. If an alternating path has length larger than $2n - 1$, then some clause C of G must appear at least 3 times in the path. If one of these occurrences is at the beginning of the path then the path can be shortened by deleting everything before the second occurrence. If one of these occurrences is at the end of the path then a similar shortening can be done. Otherwise, by Lemma 6, there will be at least two of these three occurrences of C such that an entering literal for the first occurrence differs from an exiting literal of the second occurrence. Therefore everything in between these two occurrences of C can be deleted, yielding a shorter alternating path. By repeating this operation we obtain a path with at most two occurrences of any clause,

and the length of the path is at most $2n-1$. Applying this argument to G yields that $d_G^{a,b}(C,T) \leqslant 2n-1$ because all links in G are basic links. Lifting this argument to R as before and using compatible substitutions, $d_R^{a,c}(C,T) \leqslant 2n-1$. □

It is an open question whether the $2n-1$ bound can be reduced to $n-1$.

Corollary 6. *Let S be a set of clauses. If S has Herbrand complexity n and T is a set of support for S then there exists an unsatisfiable set G of ground instances of clauses in S with the following relevance properties:*

For every clause C in G, there is a clause D in G such that D is an instance of some clause in T and $d_G^{b,a}(C,D) \leqslant 2n-1$.

Proof. The proof is obtained by applying the theorem to the unsatisfiable set G of ground instances of clauses in S, noting that because G is ground, $d_G^{b,a}(C,D) = d_G(C,D)$. □

6.1. Computing relevance sets based on various distance measures

Analogous to Theorem 5, one can define relevance sets based on the various distance measures given above. These results can also easily be extended to multiple sets of support, as before. However, because the results are fairly straightforward, they are omitted.

7. Other relevance refinements

In this section we discuss ways to refine the definition of relevance and related concepts that may result in smaller relevance sets. The basic idea behind these refinements is the fact that link properties underlying the various relevance definitions are not strict enough: they allow declaring relevant too many clauses that have no contribution to the refutation. We already noticed this fact by refining the relevance set through the full-match criterion. This is based on the realization that only fully matched clauses have the potential to contribute to a refutation. One can easily see that not every fully matched clause contributes to the refutation (as, e.g., in Examples 6 and 9).

The ideal refinement would achieve a relevance set that includes only clauses in the minimally unsatisfiable set. However, computing such a set is probably as hard as testing unsatisfiability.

Therefore one must be aware of the tradeoffs involved in defining the relevance set. On the one hand the computation needs to be *efficient* so that it doesn't dominate the refutation search. On the other hand it must be *effective* in the sense that it generates a small enough relevance set to be used to generate the refutation. Ideally the relevance set should exclude enough clauses to speed up the overall test for satisfiability.

So far we defined several relevance metrics based on different types of links between clauses and investigated the relationships between distances induced by theses metrics. These metrics have been strengthened by requiring links to be between elements of the maximal fully matched subset of S, which will generally result in smaller relevance sets. Other tightenings are possible. Next we discuss other possible refinements such as

using sort information to disallow certain potential links for having no contribution to the refutation and using a reduced support set.

7.1. Sorts and relevance

In general, the smaller the number of clauses in the relevance set the better. This number generally depends on the way the distance metric is defined. Metrics that maximize the distance between clauses (stricter metrics) while still complete are preferable. Such metrics ensure that distant clauses are not included in the computations until no set of closer clauses is proved unsatisfiable. The more distant the clause, the less chance it has to be selected in the computation for a given relevance bound. One way to achieve such a distance metric is to utilize sorts.

For sorts, one assumes a finite or infinite set $\mathcal{S}$ whose elements are called *sorts*, and a *signature* $\Sigma = \langle \mathcal{S}, \mathcal{F}, \tau \rangle$ where $\mathcal{F}$ is a set of function and predicate symbols and τ is a function associating a *set of arities* with each function and predicate symbol. An arity is of the form $s_1, s_2, \ldots, s_n \to s$ or $s_1, s_2, \ldots, s_n \to \circ$; the former indicates a function $f(x_1, \ldots, x_n)$ in which the arguments $x_1, \ldots, x_n$ may have sorts $s_1, \ldots, s_n$, respectively, and in this case f returns a result of sort s. The arity $s_1, s_2, \ldots, s_n \to \circ$ indicates a predicate $P(x_1, \ldots, x_n)$ in which the arguments $x_1, \ldots, x_n$ may have sorts $s_1, \ldots, s_n$, respectively. Variables are also assigned sorts. The first-order language includes only the *well-sorted* terms; these are terms in which the arguments of functions and predicates have the correct sorts.

For example, suppose that there is a unary function symbol f, a constant symbol 0, and a unary predicate P. Then we can obtain a sort structure from an interpretation of these symbols. If we specify the domain of the interpretation to be $\{0, 1\}$, interpret f as incrementation modulo 2, and interpret 0 as 0, then we naturally obtain the arities $0 \to 1$ and $1 \to 0$ for f and the arity $\to 0$ for 0. The arities for P can be chosen as $0 \to \circ$ and $1 \to \circ$. In general, one can obtain sort structures from interpretations in this way.

Sorts permit a much improved treatment of equality, because the equality predicate $x = y$ can be given an arity restricting x and y to have the same sort. This permits sort information to propagate across equations in the relevance computation, whereas in the absence of sorts, no constraint relating x and y will be enforced.

In the following sections, we only sketch how sorts may be combined with relevance.

7.1.1. Sorts for stricter links

A factor that may generate smaller relevance distance is "unreal" links between clauses. Examples are syntactic links that can be excluded on closer examination, for example because of incompatible substitutions, as we have seen already. However, keeping track of compatibility appears to require exponential time. Sorts give some of the advantages of compatibility with less cost.

An example where sorts or compatibility is needed is the equality axioms. An equality axiom of the form $x = y \wedge y = z \to x = z$ will link to any clauses with equality in the head or the body. (The *head* of a clause C is the set of positive literals in C and the *body* is the set of atoms appearing in negative literals of C.) So any clauses with equality in the body are linked with clauses with equality in the head (directly and through other clauses). It is

very likely that this equality axiom will also be fully matched when sort incompatibilities between terms are ignored.

The following is an example where equality is a factor even in the absence of the equality axioms.

Example 11.

$$\mathit{office}(x, o_1) \wedge \mathit{sameproject}(x, y) \wedge \mathit{office}(y, o_2) \rightarrow o_1 = o_2 \vee \mathit{neighbor}(x, y).$$

(Workers on the same project have the same or adjacent offices.)

$$\mathit{studentyear}(x, y_1) \wedge \mathit{studentyear}(z, y_2) \wedge y_1 = y_2 \rightarrow \mathit{samefloor}(x, z).$$

(Same year students are located on the same floor.)

The clauses have a link resulting from unifying the equality predicates with the substitution $\{y_1/o_1, y_2/o_2\}$, though the underlying sorts (year and office, respectively) are not compatible.

Sorts can be incorporated into the relevance process to disallow the inclusion of certain clauses into the computation. The way to do that is to give sorts to terms and to prohibit connections (links) between clauses when the sorts of the terms are not compatible. The disqualified links will generally increase the distance between clauses and may disallow certain clauses from being fully matched resulting in smaller relevance sets.

Therefore, sorted unification can replace simple unification in the definition of relevance (and full matching). Only sort compatible unification can influence relevance. A relevance bound of n gives a set of sorted clauses that are unsatisfiable if there is a proof having n ground instances. Then one performs sorted inferences on this set of sorted clauses and this will generate a contradiction if they are unsatisfiable. Sorted inference prevents certain instantiations and unifications which violate sortedness, so it reduces the search space. Thus sorted relevance reduces the search both by reducing the set of starting clauses, and also by reducing the set of inferences possible among them.

Two clause instances $C_1\sigma$ and $C_2\sigma$ are linked iff there exist two complementary literals $L_1 \in C_1$ and $L_2 \in C_2$ such that $L_1\sigma =_s \neg L_2\sigma$. $=_s$ means equality with compatible sort for all terms of L_1 and L_2. In this case we say that $d^s(C_1, C_2) = 1$. $d^s(C_1, C_2) = \infty$ if no such links exists. In general, $d^s(C_1, C_2) \geqslant d(C_1, C_2)$ where d^s is the sorted distance between clauses and d is the non-sorted distance between clause.

7.1.2. Sorts and relevance distance

In the same spirit as using relevance distance to relate clauses to the support set and therefore select clauses that may contribute to a refutation one may utilize sorts and links between them for the same purpose. One may achieve different behavior by moving between the two extremes of declaring all constants to have the same sort and therefore pay no attention to sorts at all (as has been our treatment until the current subsection) and the other extreme of declaring each individual constant as a unique sort by itself. Recalling that sorts can be obtained from interpretations, an interpretation with a small domain will lead to a small number of sorts. A large domain leads to a large number of sorts, which gives a more refined relevance measure at an increased cost. Probably the optimal solution is to

deal with a limited number of sorts and to check for sort compatibility between terms ranging over these sorts. Here we only argue that different sort selections can result in different relevance/sort behavior that may contribute in different ways to the refutation search.

For a human, it is more natural to think of a specific example (interpretation) and to think of relevance as relating the objects in this interpretation rather than the clauses. It would be interesting to study relevance measures on the sorts themselves, that is, relevance distances between sorts, and see how they relate to the relevance distances on clauses presented so far. One might say, for example, that two sorts are linked if they occur in a common clause. Then the distance between two sorts could be defined in terms of the shortest path of links between them.

The following example modified from Example 7 shows how sorts can lead to better behavior even than compatible paths:

Example 12. Let S be the following set of clauses:

(1) $d(c_i, c_j)$ for various i, j, indicating that one can drive directly from city i to city j.
(2) $\neg at(x, s) \vee \neg d(x, y) \vee at(y, g(y, s))$. (One can go from city x to city y in situation s if $d(x, y)$. $g(y, s)$ is the action taken.)
(3) $at(c_0, s_0)$. (The starting situation.)
(4) $\neg at(c_m, s)$. (The goal is to get to city c_m.)

If one does not use sorts, then $d_S^{a,c}(at(c_0, s_0), \neg at(c_m, s)) = 2$ because the path $at(c_0, s_0), \neg at(x, s) \vee \neg d(x, y) \vee at(y, g(y, s)), \neg at(c_m, s)$ is alternating and compatible. The problem is that the substitution for this path binds x to c_0 and y to c_m and with this binding, the middle clause is not fully matched because there is no link involving the literal $\neg d(x, y)$ (unless $d(c_0, c_m)$ is asserted). With sorts, one can specify that each city is a separate sort, but all situations are the same sort. Then the clause $\neg at(x, s) \vee \neg d(x, y) \vee at(y, g(y, s))$ has m^2 sorted versions, in which x and y have as sort one of the m cities. All of these sorted clauses fail to be fully matched except those corresponding to the clauses $d(c_i, c_j)$. This gives a more realistic distance between the last two clauses and a better relevance function. Even adding equality axioms to this example would not hinder the effectiveness of a relevance filter, because of the use of sorts. As in Examples 7 and 8, the use of sorts with two support sets can lead to a substantial reduction in search space, especially for very large sets of assertions.

It is also of interest in this example that for a human, it is more natural to think of distances between cities than distances between clauses. Since cities are sorts in this example, it is natural to investigate how relevance measures involving distances between sorts relate to the clause-based relevance measures described in this paper.

The following example also illustrates how sorts can be helpful for very large sets of assertions:

Example 13. Let S be the following set of clauses:

(1) $p(a_i, a_j)$ for various individuals i, j, indicating that individual a_i is the parent of individual a_j.

(2) $s(a_i, a_j)$ for various individuals i, j, indicating that individual a_i is the spouse of individual a_j.
(3) $\neg p(x, y), r(x, y)$. (Parents and children are related.)
(4) $\neg s(x, y), r(x, y)$. (Spouses are related.)
(5) Reflexivity, symmetry, and transitivity of the "r" relation.
(6) $\neg r(a_0, a_m)$. (The goal is to show that individuals a_0 and a_m are related.)

In this example, sorts are needed for a good relevance function, as above. One can let each individual be a separate sort. Two support sets can be specified, one containing all p and s assertions involving a_0 and one containing all p and s assertions involving a_m. Then the relevance set consists of clauses mentioning only individuals that are closely related to *both* a_0 and a_m; some of these clauses may be removed by the full matching requirement. This use of two support sets can result in a substantial reduction in search space as compared to specifying just one set of support, especially for knowledge bases containing thousands or hundreds of thousands of facts.

A problem with this use of sorts is that the number of sorted versions of a clause can be very large. If a clause C has three variables and each one can have m sorts, then the number of sorted versions of C is at least m^3. For large knowledge bases where m is large, this large number of sorted clauses makes this approach impractical. This problem can be reduced by only generating sorted clauses as they are needed, and also by representing large sets of sorted clauses by *sort constrained* clauses (C, A) where C is an unsorted clause and A is a constraint on the sorts of the variables in C. The sort constrained clause (C, A) represents the set of sorted versions of C that satisfy the constraint A. Thus $(C[x, y, z], sort(x) \in \{c_1, c_2\})$ represents the set of sorted versions of C in which the sort of x is either c_1 or c_2, but the sorts of y and z may be arbitrary. The computation of relevance sets can be adapted to the use of sort constrained clauses, but a detailed discussion of this is beyond the scope of this paper.

7.2. Reduced support set

Reducing the size of the set of support will generally have the effect of increasing the distances of clauses in S. Assuming a constant Herbrand complexity n one may try to minimize the size of T to achieve better performance by generating smaller relevance sets.

One common choice of the support set is the all positive clauses of S or all the negative clauses of S. Both are correct choices in the sense that the remaining clauses are never unsatisfiable (for a refutation one always needs some positive and some negative clauses).

Rather than selecting the entire set of clauses with the given polarity (positive or negative) one may want to select T to be a subset of the positive (respectively negative) clauses of S that are still a support set. This information may be readily available by the nature of the task being performed. For example, on proving a theorem, one may take the set of support to be the negative clauses corresponding to the negation of the theorem but that includes no negative clauses of the original theory. In answering a positive query Q against a database DB, one may select $T = \{Q \rightarrow \bot\}$ and not include in T any negative clauses of DB. The large support set problem (in the form of taking T to be the set of all

negative clauses of S) was recognized as a source of inefficiency for the relevance based approach of [14] and was rectified using an approach based on restricted initial relevance in [15].

Consider the following example:

Example 14. Consider the set of clauses $S = \{C_1, C_2, \ldots, C_9\}$ and $T_1 = \{C_5\}$ and $T_2 = \{C_5, C_6, C_7, C_8, C_9\}$ where the clauses $C_1, C_2, \ldots, C_9$ are as follows: (Recall that the numbers in parentheses give the distance of a clause from T_1 and T_2 in that order.) $C_1(1,1) = Q \vee a$, $C_2(\infty,1) = e \vee f$, $C_3(\infty,1) = d \vee f$, $C_4(2,1) = \neg a \vee b \vee c$, $C_5(0,0) = Q \rightarrow \perp$, $C_6(2,0) = \neg a \vee \neg b$, $C_7(2,0) = \neg a \vee \neg c$, $C_8(\infty,0) = \neg f \vee \neg d$, $C_9(\infty,0) = \neg e \vee \neg d$. Because the clause set $\{C_1, C_4, C_5, C_6, C_7\}$ is unsatisfiable we take $n = 5$.

Let R_1 be the set of clauses at distance 4 or less from a support set. Clearly, $R_1 = S$ for T_2 while $R_1 = \{C_1, C_4, C_5, C_6, C_7\}$ for T_1. The saving resulting from selecting the smaller T_1 is substantial.

The approach can be easily extended to the case of multiple support sets.

However, one may want to consider the cost of selecting a minimal support set as opposed to selecting just a support set. The latter may be computationally efficient as it is generally based on syntactic criteria such as clause polarity, while the former may be computationally expensive as it may involve some sort of minimization. A middle ground may be the optimal choice. For example, given a set of axioms (a database *DB*) select as T the set of the negative clauses representing the theorem (query) negation. This T is known to contribute to the refutation, if one exists, as the original theory is known to be consistent. T however, may not be a minimal support set in the sense that $T' \subset T$ may also be a support set. Another possibility is to choose a collection $\{I_1, I_2, \ldots, I_k\}$ of interpretations of S and let T_i be the set of clauses of S that are not satisfied by I_i. Then $\{T_1, T_2, \ldots, T_k\}$ is a support class for S. Each I_i can be chosen as an interpretation with a finite domain, for example, so that it is decidable for a clause C of S whether $I_i \models C$.

7.3. Combined criteria

Each of the restrictions discussed above resulted in stricter metrics than the original. Further refinements can be achieved by combining the discussed approaches to achieve still stricter metrics. One possibility is to limit the resolution based definition of distance only to clauses that are members of the fully matched set. Only such clauses are allowed in defining links and therefore distances. The same can be done with sorting. Only links (resolutions) that are applied to compatible sorts need to be taken into account in defining links. Links with incompatible sorts can be ignored. Another option to use multiple sets of support each of which is minimal.

However, we have always to keep in mind that the amount of time spent on relevance testing should be offset by the gains resulting from seeking a refutation in the computed set of relevant clauses as opposed to that of performing the refutation search in the entire input set.

8. Applications

There have been a number of implementations of relevance techniques. One relevance filter was implemented [8] and obtained good results for two knowledge bases of over 100 clauses each. This approach produced small unsatisfiable subsets of the input clauses for several queries. Veroff [25] implemented another relevance filter that helped on some theorem proving problems with fairly deep proofs; this approach computes the relevance distance of each *symbol* from a single set of support using a resolution distance measure, a paramodulation distance measure, and a combined measure, and uses the distances of the symbols in a clause to assign a weight to the clause. This means that clauses containing symbols that are closely related to the theorem, are more likely to participate in the search. The weighting is performed on clauses generated by resolution and paramodulation, as well as input clauses. Veroff [26] obtained good results overall on set theory problems in this way, and sometimes was able to obtain proofs with relevance techniques that could not be obtained otherwise. Other approaches discussed in the next section also employ relevance concepts, although not using the same formalism as developed in the current paper, and achieve improved performance [7,14,15,18,23,27].

Relevance methods appear most promising for large sets of assertions and small refutations. In contrast, typical test problems for theorem provers have relatively small clause sets and moderate sized refutations, and therefore may not benefit significantly from relevance filters. However, such small clause sets often give theorem provers an unrealistic advantage. Human mathematicians generally do not know which axioms are necessary to prove a theorem until a proof is found. If a theorem prover attempts to find a proof using all potentially relevant mathematical knowledge without some kind of a relevance filter, the proof attempt would be hopeless. For such problems, a relevant set of clauses is generally chosen by the user in advance before attempting the problem on a theorem prover. For some problems involving abstract algebra, abstract logic, and equality, the relevant axioms are known in advance and are relatively few in number. Set theory axioms are somewhat more numerous but have many common predicates such as “member” and “subset” that connect the clauses closely together. This appears to make set theory problems unsuitable for relevance techniques, but Veroff [25,26] was able to obtain significant improvements on such problems using relevance nonetheless. Even for problems involving equality and large axiom sets, relevance with sorts can be effective in reducing the search space, as mentioned in Examples 11 and 12.

Relevance techniques also appear promising for other kinds of problems that are currently not suitable for automatic theorem provers. Problems of this type include many mathematical theorems without a pre-selected relevant set of input clauses. This also includes problems in expert systems, natural language understanding, and common sense reasoning, in which the deductions are often simple but the knowledge base can be very large, often including many thousands or hundreds of thousands of assertions. One example of this is the Cyc project [9] for common sense reasoning in which there are currently over a million assertions stored in predicate logic. Another example with many assertions and simple deductions is the proposed semantic web [3]. It is reasonable to believe that relevance techniques, especially with multiple sets of support, could lead to a systematic method for choosing small, relevant sets of assertions for these applications. This would

probably entail some preprocessing of the large knowledge bases to enable efficient relevance computations. The human mind also efficiently selects relevant assertions from a huge knowledge base, and may use some kind of a relevance filter for this purpose.

A number of applications naturally generate multiple sets of support for a relevance filter. A concrete example will help to illustrate the potential of a relevance filter with multiple sets of support. Suppose that a concept hierarchy is formalized by assertions $\forall x(P(x) \rightarrow Q(x))$ specifying that every instance of concept P is also an instance of concept Q. Suppose that the concept hierarchy has a tree structure, so that each concept P appears in at most one such assertion. Suppose that there are also assertions of the form $P(a)$ for concepts P and individuals a. The clause $\neg P_1(x) \vee \neg P_2(x) \vee \cdots \vee \neg P_n(x)$ expresses the query of finding an individual that is an instance of a collection $P_1, P_2, \ldots, P_n$ of concepts. To apply relevance techniques to this query, each P_i can be made into a set of support T_i including all assertions mentioning P_i. Applying relevance with the support class $\{T_1, T_2, \ldots, T_n\}$ can significantly reduce the search. Suppose $n = 3$ and P_1, P_2, and P_3 each have 1000 subconcepts, but only one individual a is a common instance of subconcepts of P_1, P_2, and P_3. Suppose for concreteness that $Q_1(a)$, $Q_2(a)$, and $Q_3(a)$ are true, where Q_i are subconcepts of P_i by a hierarchy of length 10, and a is not an instance of any other concepts. Then a relevance filter with sorts corresponding to individuals will choose about 30 relevant clauses, much reduced from the over 3000 assertions in the knowledge base.

A different concept hierarchy problem is to show that some object a is an instance of P if one only knows that a is an instance of *one* of the concepts $Q_1, Q_2, \ldots, Q_n$, that is, $Q_1(a) \vee Q_2(a) \vee \cdots \vee Q_n(a)$. Then each Q_i can be made into a set of support T_i including all clauses mentioning Q_i, and P can be made into a support set T_0, and a relevance filter with the support class $\{T_0, T_1, T_2, \ldots, T_n\}$ leads to similar reductions in search space size. In general, many description logic problems [1] with large knowledge bases may benefit from a relevance filter, as well as other problems containing a large number of predicate symbols.

Many planning problems can benefit from relevance techniques. For planning problems involving n movable objects, there can be $2n$ support sets, obtained from the starting and finishing locations of each object. Suppose one wants to push n objects to the right end of a one dimensional track, where the objects all start at the same location in the middle of the track. Then a situation can be formalized as a tuple of integer locations of each object. The effects of actions can be formalized by assertions of the form $Can(e_1) \rightarrow Can(e_2)$ where $Can(s)$ means that situation s is achievable and e_1 and e_2 are terms representing situations such that situation e_2 is reachable from situation e_1 by an action. Relevance techniques will at least show that it does not help to push any object to the left, which can substantially reduce the search space. If one can push all objects at the same time, then relevance techniques using the average relevance function of Section 4.3 can reduce the search space to the optimal plan, which is to push all objects to the right together. If the track has length t, then the relevance filter reduces the search space to size $O(t)$ for this latter problem, while an unrestricted search would have size $O(t^n)$ or more. Of course, specialized techniques can do as well, but there may be advantages to using uniform proof procedures in some cases.

Some queries to databases can benefit from relevance techniques. A disjunctive query $a_1 \vee a_2 \vee \cdots \vee a_n$, when negated, results in the clauses $\neg a_1, \neg a_2, \ldots, \neg a_n$. If no simpler disjunction is provable, then each $\neg a_i$ clause can be a set of support for a relevance filter, yielding n sets of supports in all. If a query Q fails, and some clauses T are added and the query is attempted again, then both Q and T can function together as sets of support for a relevance filter. This also applies to checking the violation of integrity constraints for deductive databases after an update. Deductive databases in general provide a class of problems with a large number of clauses and query processing and integrity constraint checking tasks solvable within a small relevance bound, and are therefore good candidates for the relevance approach.

Relevance techniques can help even when the final proofs are large. This can happen when simple connectedness criteria are sufficient to choose a small, relevant clause set. Veroff [25,26] obtained significant improvements using relevance on some problems with fairly complex proofs. For some applications, it may be desirable to let the cost of links vary, so that some links are much cheaper than others for relevance computations. This may also help to find relevant sets of clauses when the proofs are large.

9. Other approaches

Relevance as an approach to improving the efficiency of theorem provers is an early paradigm[8,20]. The dominance of irrelevant clauses was recognized as the source of inefficiency of theorem provers based on model generation [14,23]. Relevance was suggested as a possible remedy. The approach of [23] is based on partial relevance and was refined to total relevance in [14] that is in the spirit of limiting consideration to a modification of fully matched sets. Using the terminology of the current paper, both approaches are based on declaring all the negative clauses to be the support set and propagating relevance to other clauses through the (auxiliary) relevance checking component of the procedure. Only clauses passing the relevance test are employed in the main computation. Realizing that the potential large size of the support set may drag many essentially irrelevant clauses into the computation, the approach of [15] uses a reduced support set that represents the negation of the query as the seed for relevance, resulting in more compact computation for the proper class of theories. While some systems [14,23, 27] have a specific component to deal with relevance, the most common approach is to integrate the relevance checking into the deduction process by specifying clause selection functions that always pick (or at least give preference to) clauses linked to the top clause or to clauses already participating in the proof.

Regarding the specification of support set, some of the approaches in the literature avoid that altogether and allow the user to start from any clause and all clauses are eventually considered until a refutation is found. This is the approach in the original resolution and tableaux methods [5]. Other approaches give the set of support implicitly, by specifying an initial interpretation and selecting for expansion clauses that are violated in that interpretation [22]. Clearly, a careful choice of this interpretation has a major impact on the resulting gains. A common choice is the selection of the all positive interpretation [2,16] and the all negative interpretation [27]. One advantage is that such a specification will result

automatically in the starting clauses constituting a set of support and no additional testing for that is needed. The approach of [15] assumes the database to be consistent. It specifies a limited set of negative clauses, corresponding to the query negation and therefore known to constitute a set of support, as the source of relevance.

A common feature of all the mentioned approaches is that they don't explicitly utilize the concept of the relevance distance in the computation. Clauses are introduced once they are shown to be relevant and no clear relevance distance bounds are employed. However, some of these proof procedures use the fact that giving preference to clauses with certain relevance distance properties may result in specific properties of the search mechanism. For example, preferring increased distance may give the computation a depth-first flavor while exhausting clauses at a certain distance before moving to the next value has a breadth first flavor.

One may be able to relate relevance concepts advanced here to efficiency enhancement techniques in theorem proving discussed under various titles. Here are some examples:

(1) As is common in model generation theorem proving [2,4,16], one may select the positive clauses as the set of support, say by choosing the initial interpretation to be empty. This is possible as the remaining clauses will be negative and mixed and thus satisfiable. Only a clause (instance) with matched body atoms, is allowed to participate in the computation. Such clauses constitute a superset of fully matched clauses. The body of a positive clause is trivially matched. The tree expansion can be viewed as a way of propagating relevance. Relevance is propagated to a new clause if all the body atoms are matched. The process is terminated by negative clauses with no head atoms to link to. An atom that is not linked can serve as a cause for dropping that clause (instance) from the expansion set as it is not fully matched (locally). This (level cut) is a common approach to reducing the search space in model generation theorem proving. Note that finding relevant clauses and computing the refutation are interleaved.
(2) Alternatively, one may select the negative clauses as the set of support. This is possible because the remaining clauses constitute a satisfiable set. Only clauses with matched heads are used in the computation. The tree expansion can be viewed as a way of propagating relevance to a new clause if all its head atoms are matched. Once all head atoms are matched, the body atoms are expanded and a link is sought for them. Any expanded atom that is not linked can serve as a cause for dropping that clause (instance) from any further consideration in the current branch. In the terminology of this paper, it is not (locally) fully matched.
(3) Other choices of the support set are possible and can be given interpretations along the above lines [15,22].

The relevance computation as discussed here is a general approach that can be used in conjunction with many existing theorem provers without major modifications to the systems. The computed relevance set preserves the refutational properties of the original set. To utilize relevance it is sufficient to have the computation of the relevance set R as a preprocessing step in the prover.

Additionally, many theorem provers use unification. Among these are the tableaux methods: Disconnection Tableaux, First Order Davis Putnam and others [2,4,10]. If sorted

unification is used instead certain unifications will not be applicable thus reducing the search space.

10. Conclusions, comparisons and future work

We presented an approach to introduce relevance into theorem proving. The goal is to allow a theorem proving system to operate on a restricted subset of clauses related to a given support set rather than the entire set of input clauses. Since many theorem proving tasks have high computational complexity, one would expect that the reduction in the clause set size will influence the time it takes to find a refutation. This gain can be especially large when one deals with a huge number of clauses only a small portion of which is relevant to any individual task the system is asked to perform. An example is query answering in very large (possibly distributed) databases where relevance may result in reductions of many orders of magnitude in the size of the input set for the theorem prover. Another example is the case of mathematical theorem proving in the presence of many noise clauses representing axioms that have nothing to do with the theorem being proved. A third example is the case of theorem proving based diagnostics where an error is local to a certain components that constitutes a small fraction of the otherwise correctly functioning circuit. Several examples (Examples 7, 8, 12, and 13) have been presented in which relevance functions together with multiple support sets can result in substantial reductions in search space over other known theorem proving strategies, especially for knowledge bases having thousands or hundreds of thousands of assertions. Theorem proving strategies that are not goal sensitive will probably generate huge search spaces on such large knowledge bases. Existing goal-sensitive strategies such as model elimination only restrict the search to clauses that are closely related to a single support set. Because relevance strategies can restrict the search to clauses that are closely related to multiple support sets simultaneously, they should be considerably more efficient on large knowledge bases.

Relevance testing as a way for improving performance has been a topic of research since the early days of automated deduction. While much of the work did not explicitly refer to relevance, the exclusion of irrelevant clauses can be viewed as an instance of the relevance testing approach. Several techniques such as pure literal rule applications, set of support/input resolution strategy, semantic hyper-tableaux and many others can be interpreted as relevance incorporation methods. Relevance as presented here can be viewed as an elaboration and extension of the results reported in [8,20]. In contrast to the latter, we considered several refinements to the relevance concept that included the use of sorts, manipulating the support set, and using stricter links to define relevance metrics. The resulting relevance set is generally smaller though at the expense of a higher cost of computing relevance.

The relevance distance bound which is generally based on the Herbrand complexity of the theory played an important role in the computation of the relevance set. The nature of this bound can influence the characteristics of the refutations found. The shortest proofs are always retained [7]. However, if the bound is based on the length of a proof then all proofs of lesser sizes are guaranteed to be found in the computed relevance set. Longer proofs

may be missed. This may be a drawback in cases when one is interested in all possible proofs, such as when one wants to compute all possible answers to a given query.

We considered several refinements to the link condition underlying the relevance distance discussed in the paper. Clearly, others can be devised to further refine the link concept and consequently reduce the relevance set. On the other hand one may want to relax some of the definitions to get quick though rough estimates on the relevance set. For example, rather than computing the fully matched set as a basis for relevance one may want to compute the fully matched set that results from replacing all arguments of predicates by the same variable and seeking full-matching in the resulting set. Clearly, this may declare relevant, clauses that are not really so but may exclude many irrelevant clauses at a low cost. At least this can serve as a first estimate on the relevance set. A point that one should keep in mind is the trade-offs between the cost of the relevance computation and the gains to the refutation search.

On the other hand, the relevance approach as outlined here may be considered as orthogonal to many popular search space reduction techniques employed in the literature such as tautology removal and clause subsumption processing.

The approaches in [14,23] are based on explicitly computing relevant clauses through the concept of a relevant atom. The partial relevance as defined in [23] and total relevance as defined in [14] and the related concept of nonhorn magic sets of [18] do not utilize the concept of relevance distance. The relevance there is used in a forward chaining model generation procedure and is based on the negative clause set which plays the role of the support set in the current paper. While the relevance approach discussed in the current paper defines relevant clauses globally, that is in relation to the entire set of support, the approaches of [14,15] do that in a more localized fashion: the relevant clauses are computed relative to the current state of the computation and change as the refutation search advances. This may result in recognizing certain clauses as irrelevant at the current stage while they would be declared relevant by the approach of the current paper. One can adopt that approach here by refining the relevance computation to give it a local character. However, one should take into account the computational cost of this refinement which can be substantial under unfavorable conditions.

Another difference (or way to look at things) is that the approach of this paper is static in the sense that the computation of the relevance set precedes the refutation search (in the computed relevance set) while the approach of [14] is dynamic in the sense that the relevance computation is done in the changing environment of the search. It is not difficult to see that the approaches can be combined: a relevance of [14] style can be applied to the relevance set computed using the (static) approach of this paper.

The works [6,7] offer approaches to computing relevance that are in the line of the current work. They are classified as heuristic and exact. In that spirit, the relevance discussed here is an exact one in the sense that it is based on strong results regarding the (non)contribution of irrelevant clauses to the proofs sought. Our approach deals directly with the given clause set rather than with their abstractions [21] as detailed in [7]. Additionally, the approach of [6] is integrated into a tableaux construction process and therefore has the ability to take local (branch) considerations into account when testing relevance. This can result in more refined relevance. The fact that the relevance approach as defined in this paper is static makes it generic in the sense that it is not geared towards

any particular theorem prover or theorem proving strategy [6,14]. It can be incorporated into existing theorem provers, though clearly one can further refine the approach to exploit the properties of the system into which it is integrated. However, relevance has a global nature and as such may include clauses that have no contribution to the refutation or those whose contribution has already been exploited.

One topic that merits attention is the presence of equality in S. A straightforward treatment of equality requires that the equality axioms be included in the relevance computation. Sorts help in this respect, but it might be preferable to eliminate the equality axioms altogether and define relevance concepts directly related to paramodulation. This seems to require an approach in which subterms can be linked with other subterms having the same sort. A detailed treatment of paramodulation is beyond the scope of the current paper but may be a topic of future research.

The detailed treatment of sorts and the relationship to relevance is another topic for further research.

Acknowledgements

The authors wish to thank Donald Loveland for many stimulating discussions during the course of this research. The comments of a referee also significantly improved the presentation of these results. This work was partially supported by the Arab Fund for Economic and Social Development (AFESD) and by the National Science Foundation under grant NSF CCR-9972118.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P.F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation and Applications, Cambridge University Press, Cambridge, 2002.

[2] P. Baumgartner, FDPLL—A first-order Davis–Putnam–Logemann–Loveland procedure, in: Proceedings of 17th International Conference on Automated Deduction, in: Lecture Notes in Artificial Intelligence, Vol. 1831, Springer, Berlin, 2000, pp. 200–219.

[3] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Scientific American 284 (5) (2001) 34–43.

[4] F. Bry, A. Yahya, Positive unit hyper-resolution tableaux and their application to minimal model generation, J. Automat. Reason. 25 (1) (2000) 35–82.

[5] C.L. Chang, K.C.T. Lee, Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York, 1973.

[6] M. Fuchs, Relevancy-based lemma selection for model elimination using lazy tableaux enumeration, in: Proceedings ECAI-98, Brighton, UK, 1998, pp. 346–350.

[7] M. Fuchs, D. Fuchs, Abstraction-based relevancy testing for model elimination, in: H. Ganzinger (Ed.), Proceedings of 16th International Conference on Automated Deduction, in: Lecture Notes in Artificial Intelligence, Vol. 1632, Springer, Berlin, 1999, pp. 344–358.

[8] S. Jefferson, D. Plaisted, Implementation of an improved relevance criterion, in: IEEE 1984, Proceedings of the First Conference on Artificial Intelligence Applications, IEEE Computer Society Press, 1984, pp. 476–482.

[9] D. Lenat, R. Guha, Building Large Knowledge-Based Systems, Addison-Wesley, Reading, MA, 1990.

[10] R. Letz, G. Stenz, DCTP—A disconnection calculus theorem prover—system abstract, in: R. Goré, A. Leitsch, T. Nipkow (Eds.), Automated Reasoning, First International Joint Conference (IJCAR)

Proceedings, Siena, Italy, in: Lecture Notes in Comput. Sci., Vol. 2083, Springer, Berlin, 2001, pp. 381–385.
[11] J.W. Lloyd, Foundations of Logic Programming, 2nd Edition, Springer, Heidelberg, 1987.
[12] D.W. Loveland, Automated Theorem Proving: A Logical Basis, North Holland, Amsterdam, 1978.
[13] D.W. Loveland, Mechanical theorem-proving by model elimination, J. ACM 15 (2) (1968) 236–251.
[14] D.W. Loveland, D. Reed, D. Wilson, SATCHMORE: SATCHMO with RElevancy, J. Automat. Reason. 14 (1995) 325–351.
[15] D.W. Loveland, A. Yahya, SATCHMOREBID: SATCHMO(RE) with BIDirectional Relevancy, New Generation Computing, to appear.
[16] R. Manthey, F. Bry, SATCHMO: A theorem prover implemented in Prolog, in: E. Lusk, R. Overbeek (Eds.), Proceedings of the 9th International Conference on Automated Deduction, Springer, Berlin, 1988, pp. 415–434.
[17] A. Nerode, R. Shore, Logic for Applications, Springer, Berlin, 1993.
[18] Y. Ohta, K. Inoue, R. Hasegawa, On the relationship between non-Horn magic sets and relevance testing, in: Proceedings of 15th International Conference on Automated Deduction, in: Lecture Notes in Artificial Intelligence, Vol. 1421, Springer, Berlin, 1998, pp. 333–348.
[19] D. Plaisted, Theorem proving and semantic trees, Ph.D. Thesis, Stanford University, 1976.
[20] D. Plaisted, An efficient relevance criterion for mechanical theorem proving, in: Proceedings of the First Annual National Conference on Artificial Intelligence, Stanford University, AAAI, 1980, pp. 79–83.
[21] D. Plaisted, Theorem proving with abstraction, Artificial Intelligence 16 (1) (1981) 47–108.
[22] D. Plaisted, Y. Zhu, Ordered semantic hyperlinking, J. Automat. Reason. 25 (3) (2000) 167–217.
[23] A. Ramsay, Generating relevant models, J. Automat. Reason. 7 (1991) 359–368.
[24] W. Reif, G. Schellhorn, Theorem proving in large theories, in: W. Bibel, P.H. Schmitt (Eds.), Automated Deduction—A Basis for Applications, Vol. III: Applications, Kluwer Academic, Dordrecht, 1998, pp. 225–241.
[25] R. Veroff, Reasoning at multiple levels of abstraction, Technical Report TR-CS-2000-50, Department of Computer Science, University of New Mexico, Los Alamos, NM, 2000.
[26] R. Veroff, Personal communication, July 2002.
[27] A. Yahya, Duality for goal-driven query processing in disjunctive deductive databases, J. Automat. Reason. 28 (1) (2002) 1–34.