A New Method for Incremental Consequence-Finding

Eric Kao and Michael Genesereth

Stanford University
353 Serra Mall
Stanford, CA 94305
United States of America

Abstract: Consequence-finding is important to various reasoning problems, including knowledge compilation, abduction, nonmonotonic reasoning, and clause maintenance systems. The incremental version of the problem is also important because it is expensive to recompute a consequence set in response to the addition of new premises. In this paper, we present the first incremental consequence-finding method for first-order logic to exploit incrementality at the literal-level. We use a novel restriction called New-Literal Filtering to restrict resolution while preserving completeness for incremental consequence-finding. We show that the same restriction also preserves the completeness of linear resolution.

1 Introduction

Consequence-finding [14] is the problem of finding all non-tautological clauses (up to subsumption) that follow logically from a given set of first-order clauses (it is essentially a first-order version of the prime implicates problem).

As Inoue argued extensively [6, 5], consequence-finding is useful as a fundamental operation in common sense reasoning [17], diagnostics and abduction [19, 4], nonmonotonic reasoning, and truth maintenance systems [12, 20]. Consequence-finding is also useful in knowledge compilation [24, 22], program synthesis, intelligent user interfaces, data integration [15], query optimization [2], data-inconsistency management [8], and logical spreadsheets [9, 10].

Data-centric applications derive particularly great benefit from consequence-finding because these applications tend to deal with a large set of fast-changing data (represented as a set of ground literals (or atoms)) governed by a relatively small and stable set of constraints (represented as a set of logic sentences). Consequence-finding can be performed once on the constraints, and then the resulting consequence set can be used many times on changing data.

Here we consider one specific application in detail. In an interactive course planning system, there are three predicates:

1. $a(x, y)$, meaning that a course x is an “anti-requisite” to course y,
2. $p(x, y)$, meaning that course x is an “prerequisite” to course y, and
3. $t(z, x)$, meaning that student z plans to take course x.

The school defines the a relation and the p relation by specifying ground atoms. The students define the t relation in making their degree plans.

We have the following constraints that govern conforming plans (not intended to be complete):

1. $\forall xyz((a(x,y) \wedge t(z,x)) \rightarrow \neg t(z,y))$
2. $\forall xyz((p(x,y) \wedge t(z,y)) \rightarrow t(z,x))$

Since the constraint set is small and stable, it is cost effective to precompute its consequence set[1]:

1. $\{\neg a(x,y), \neg t(z,x)), \neg t(z,y)\}$
2. $\{\neg p(x,y), \neg t(z,y), t(z,x)\}$
3. $\{\neg p(x,y), \neg t(z,y), \neg a(x',x), \neg t(z,x')\}$
4. $\{\neg p(x,y), \neg t(z,y), \neg a(x,y'), \neg t(z,y')\}$

Now consider the following set of data:
$\{a(\texttt{cs1x}, \texttt{cs1a}), p(\texttt{cs1a}, \texttt{cs1b}), p(\texttt{math1}, \texttt{math2}), p(\texttt{math2}, \texttt{math3}), \ldots$
$, t(\texttt{s1}, \texttt{cs1x}), t(\texttt{s1}, \texttt{cs1b}), t(\texttt{math1}), t(\texttt{math2}), \ldots\}$.

To help students troubleshoot non-conforming plans, the system finds minimally unsatisfiable subsets of the data w.r.t the constraints (pinpointing). One such subset is
$\{a(\texttt{cs1x}, \texttt{cs1a}), p(\texttt{cs1a}, \texttt{cs1b}), t(\texttt{s1}, \texttt{cs1x}), t(\texttt{s1}, \texttt{cs1b})\}$. `cs1b` requires `cs1a`, but `cs1a` conflicts with `cs1x` which the student plans to take.

With the consequence set precomputed, it is straightforward to find minimally unsatisfiable subsets of the data. The system attempts to find, for each clause in the consequence set, each ground instance whose complement is a subset of the data[2]. The operation can be accomplished, for a given clause, by taking its complement as a conjunctive query and evaluating it on the data. In this example, the complement of clause 3 is $p(x,y) \wedge t(z,y) \wedge a(x',x) \wedge t(z,x')$. Evaluating it on the data finds the minimally unsatisfiable subset
$\{a(\texttt{cs1x}, \texttt{cs1a}), p(\texttt{cs1a}, \texttt{cs1b}), t(\texttt{s1}, \texttt{cs1x}), t(\texttt{s1}, \texttt{cs1b})\}$ as minimal support for the query.

The incremental version of the problem is important because in all of the uses of consequence-finding given above, new clauses may be added and it is expensive to recompute from scratch the closure under consequences. In reasoning tasks, it is natural to add new hypotheses as more information becomes available. In the course planning example, new requirements may be added. In truth maintenance systems (and clause maintenance systems), new dependencies are added as the real world dependencies evolve. In data integration, new view definitions are added as new sources are added.

[1] In the case that there is a finite consequence set.

[2] In general, a ground subsumption condition also needs to be checked to ensure minimality: that the particular instance of the clause contradicted is not subsumed by another instance of a clause that is also contradicted by the data. With subsumption testing on the consequence set, we can eliminate subsumed clauses and also identify for each clause all other clauses that may have ground instances which subsume a ground instance of the first clause. With this information precomputed, the ground subsumption condition can be checked cheaply.

In this paper, we consider the incremental consequence-finding problem. That is, given a set of clauses S_{old} that is already closed under consequences and a set of new clauses S_{new}, find all the consequences of $S_{\text{old}} \cup S_{\text{new}}$ up to subsumption.

Most prior work on incremental consequence-finding have focused on propositional logic [11, 7, 13]. The algorithms proposed do not directly generalize to first-order logic without losing completeness. (See section 6, related work, for further discussion.)

Previous incremental algorithms for first-order consequence-finding add only clause-level restrictions based on the incremental nature of the problem. These restrictions result in only modest improvement over recomputation from scratch. Once a new clause is used to resolve with an old clause, the resolvent is treated as a completely new clause even if most of the literals are from the old clause.

In this paper, we describe a refinement of resolution that exploits incrementality at the literal-level to prune many more redundant branches than a clause-level restriction could.

2 Preliminaries

We assume that the reader has a basic knowledge of the resolution method for first-order logic [21] (for a reference with modern notations, see [3]).

In this paper, a clause is a set of literals, written as comma-separated literals enclosed by (and). () denotes the empty clause. A clause is interpreted as a disjunction of the literals it contains. For easy reading, we sometimes use the disjunction symbol $\vee$ in place of a comma.

Definition 1 (Clause subsumption). *Let a and b be clauses. a subsumes b if there exists a substitution σ such that $a\sigma \subseteq b$.*

Definition 2 (Consequence Set). *Let Σ be a set of clauses. We say a set of clauses Π is a consequence set of Σ if:*

1. *$\Sigma \models \Pi$ and*
2. *for any clause c such that $\Sigma \models c$ and c is not a tautology, there exists a clause $d \in \Pi$ such that d subsumes c.*

We say simply that Π is a consequence set if Π is a consequence set of itself.

The consequence-finding problem is to find, given a set of clauses S, a consequence set of S. The incremental consequence-finding problem is to find, given two sets of clauses $S_{\text{new}}, S_{\text{old}}$ where S_{old} is a consequence set, a consequence set of $S_{\text{new}} \cup S_{\text{old}}$.

The method in this paper works on first-order clauses, but for the sake of simplicity, we now look at examples with propositional clauses.

Example 1.
$S_{\text{new}} = \{(p, s)\}$
$S_{\text{old}} = \{(\neg p, \neg r), (\neg q, r), (\neg p, \neg q)\}$

S_{old} is a consequence set. A consequence set of $S_{\text{new}} \cup S_{\text{old}}$ is $\{(\neg p, \neg r), (\neg q, r), (\neg p, \neg q), (p, s), (s, \neg r), (s, \neg q)\}$.

It is useful to note many refinements of resolution which are complete for refutation (deriving the empty clause from an unsatisfiable set of clauses) are not complete for consequence-finding. For example, semi-ordered resolution is complete for refutation but incomplete for consequence-finding. For a more detailed discussion, see [18].

Example 2. $S = \{(p, \neg r), (q, r)\}$. Clearly, a consequence set must include (p, q), but semi-ordered resolution can make no further progress because it requires one of the two resolved upon literals to be the left-most literal in the containing clause.

3 Intuition and Examples

Given two sets of clauses S_{new} and S_{old}, where S_{old} is already closed under consequences (up to subsumption), the basic idea is that we need to resolve only on literals which descend from an S_{new} clause. So we underline all the literals in each S_{new} clause used and carry that underlining throughout the resolution process. We require that whenever we resolve two clauses, one of the resolved upon literals is underlined.

Example 3.
$S_{\text{new}} = \{(\underline{p}, \underline{s})\}$
$S_{\text{old}} = \{(\neg p, \neg r), (\neg q, r), (\neg p, \neg q)\}$

First we consider in figure 1 a resolution deduction which *disobeys* the New-Literal Filtering. The deduction finds a new, unsubsumed consequence $(\underline{s}, \neg q)$.

$$\begin{tikzcd}
(\underline{p}, \underline{s}) \arrow[d] & (\neg p, \neg r) \arrow[dl] \\
(\underline{s}, \neg r) \arrow[d] & (\neg q, r) \arrow[dl, "\text{(Resolves on non-underlined literals } \neg r, r\text{!)}"'] \\
(\underline{s}, \neg q) &
\end{tikzcd}$$

Fig. 1. A resolution example where the New-Literal Filtering is disobeyed.

But it is not necessary to explore such deductions. We see in figure 2 that a resolution deduction which does obey the New-Literal Filtering and produces the same consequence.

$$\begin{tikzcd}
(\underline{p}, \underline{s}) \arrow[d] & (\neg p, \neg q) \arrow[dl] \\
(\underline{s}, \neg q) &
\end{tikzcd}$$

Fig. 2. A resolution example where the New-Literal Filtering is obeyed.

The New-Literal Filtering greatly increases efficiency by preventing the exploration of many unnecessary branches.

We consider a comparison to the clause-level restriction that requires each resolution step to involve at least one clause that is new. In the process, generated resolvents are considered new (otherwise completeness is lost). We call this restriction the New-Clause Filtering.

Using a level-saturation strategy, New-Literal Filtering and New-Clause Filtering behave exactly the same in generating the first level. In each case, each literal in each clause in S_{new} attempts unification with each literal in each clause in $S_{\text{new}} \cup S_{\text{old}}$. Let S_1 denote the newly generated clauses at this first level. In generating the second level, under New-Clause Filtering, each literal in each clause in S_1 attempts unification with each literal in each clause in $S_1 \cup S_{\text{new}} \cup S_{\text{old}}$. Under New-Literal Filtering, on the other hand, the method attempts unification only on underlined literal in S_1. The number of unification attempts under New-Literal Filtering is n/m of the number of unification attempts under New-Clause Filtering, where n is the total number of literal occurrences in S_1 that descend from S_{new} and m is the total number of literal occurrences in S_1.[3] A similar savings ratio holds for generating each additional saturation level beyond S_2. Because the restrictions causes fewer clauses to be generated at each level beyond S_1, the overall work saved can be exponential in the number of levels required.

Example 4. $S_{\text{new}} = \{(\underline{p(a)} \vee \underline{s(y)})\}$
$S_{\text{old}} = \{(\neg p(x) \vee r(x)), (t(x) \vee \neg r(x)), (\neg t(x) \vee \neg s(x)), (\neg p(x) \vee t(x)), (\neg p(x) \vee \neg s(x)), (\neg r(x) \vee \neg s(x))\}$

Computing the first level (New-Clause Filtering and New-Literal Filtering behave identically on the first level):

Attempting to resolve $(\underline{p(a)} \vee \underline{s(y)})$, on $\underline{p(a)}$ with each literal occurrence in S_{old} yields: $S_{1,1} = \{(\underline{s(y)} \vee r(a)), (\underline{s(y)} \vee \neg s(a)), (\underline{s(y)} \vee t(a))\}$.

Attempting to resolve $(\underline{p(a)} \vee \underline{s(y)})$, on $\underline{s(y)}$ with each literal occurrence in S_{old} yields: $S_{1,2} = \{(\underline{p(a)} \vee \neg t(y)), (\underline{p(a)} \vee \neg p(y)), (\underline{p(a)} \vee \neg r(y))\}$.

$S_1 = S_{1,1} \cup S_{1,2}$.

Computing the second level:

Under New-Clause Filtering, 43 clauses are generated.

Under New-Literal Filtering, only 30 clauses are generated.

For details on the clause generation, please see the appendix.

4 Resolution with New-Literal Filtering

4.1 Definition

In this section, we formalize resolution deduction with New-Literal Filtering (NL-deduction) and prove that it is complete for incremental consequence-finding.

[3] This simple analysis assumes that no indexing is used. A corresponding ratio holds when a particular indexing scheme is used.

To give a formal definition of NL-deduction, we first define binary resolution and factoring on clauses with underlining. Each is a straightforward extension of the traditional definition.

Definition 3 (Clause with underlining). *A clause with underlining is a structured clause $\langle s, u\rangle$ where s is a clause (a set of literals) and u is a subset of s that indicates the underlined literals.*

Definition 4 (Binary NL resolution). *$\langle c, u_c\rangle$ is a binary NL resolvent of $\langle a, u_a\rangle$ and $\langle b, u_b\rangle$ if there exists $l_a \in a$ and $l_b \in b$ such that:*

- *$\neg l_a$ and l_b are unifiable with a most general unifier σ,*
- *$l_a \in u_a$ or $l_b \in u_b$ (one of the resolved upon literals is underlined),*
- *$c = (a\sigma - \{l_a\sigma\}) \cup (b\sigma - \{l_b\sigma\})$ (c is a binary resolvent), and*
- *$u_c = ((u_a\sigma - \{l_a\sigma\}) \cup (u_b\sigma - \{l_b\sigma\}))$ (underlining inherited).*

Definition 5 (NL Factor). *$\langle f, u_f\rangle$ is a factor of $\langle s, u_s\rangle$ if there exists $l_1, l_2 \in s$ such that:*

- *l_1 and l_2 are unifiable by a most general unifier σ,*
- *$f = s\sigma$ (f is a factor of s), and*
- *$u_f = u_s\sigma$ (underlining inherited).*

Notation. Given a set S of unstructured clauses. Let S^u denote the set of structured clauses obtained by underlining every literal in every clause in S. Let S^n denote the set of structured clauses obtained by leaving every literal in every clause in S non-underlined.

Finally, we give a formal definition of NL-deduction.

Definition 6 (NL-deduction).
An NL-deduction of c_n from $(S_{\text{new}}, S_{\text{old}})$ is a resolution deduction:
$\langle c_1, u_{c_1}\rangle$
$\langle c_2, u_{c_2}\rangle$
$\langle c_3, u_{c_3}\rangle$
$\vdots$
$\langle c_n, u_{c_n}\rangle$
where: For each i, letting $S_i = S^u_{\text{new}} \cup S^n_{\text{old}} \cup \{\langle c_1, u_{c_1}\rangle, \ldots, \langle c_{i-1}, u_{c_{i-1}}\rangle\}$, $\langle c_i, u_{c_i}\rangle$ is a binary resolvent between (a factor of) $a \in S_i$ and (a factor of) $b \in S_i$.

4.2 Completeness for incremental CF

In this section, we show that NL resolution is complete for incremental consequence-finding. After giving some simple lemmas, we show refutation completeness for ground clauses (theorem 1). Then we show completeness for consequence-finding (theorem 2). Finally, we prove completeness for consequence-finding for first-order clauses (theorem 3).

Theorem 1 (Ground refutation completeness). *Given S, a set of ground clauses. Let $S = S_{\text{new}} \cup S_{\text{old}}$, where S_{old} is a consequence set. If S is unsatisfiable and $() \notin S_{\text{old}}$, then there is an NL-deduction of $()$ from $(S_{\text{new}}, S_{\text{old}})$.*

First we give a simple lemma.

Lemma 1. *Let N be a set of ground unit clauses and Σ a ground consequence set. Then if $N \cup \Sigma$ is unsatisfiable, then there exists a single clause $c \in \Sigma$ such that $N \cup \{c\}$ is unsatisfiable.*

Proof. Let $d = \bigvee_{l \in N} \tilde{} l$. Because $N \cup \Sigma$ is unsatisfiable, $\Sigma \models d$. There exists a $c \in \Sigma$ such that $c \subseteq d$. Then $N \cup \{c\}$ derives the empty clause and, hence, is unsatisfiable.

Proof. (of theorem 1) For a set of clauses S, let $k(S)$ be defined to be the total number of occurrences of literals in S minus the total number of clauses in S. For example, $k(\{(p(a), q(x)), (p(a))\}) = 3 - 2 = 1$.

We prove the result by induction on $k(S_{\text{new}})$.

Base case:
Let $S = S_{\text{new}} \cup S_{\text{old}}$ be an unsatisfiable set of ground clauses such that $k(S_{\text{new}}) \leq 0$ and $S = S_{\text{new}} \cup S_{\text{old}}$. Then either $() \in S_{\text{new}}$ or S_{new} consists entirely of unit clauses. If $() \in S_{\text{new}}$, then there is a one line NL-deduction of $()$. The base case holds in that subcase. So assume S_{new} consists entirely of unit clauses. By lemma 1, there is a single clause $C \in S_{\text{old}}$ such that $S_{\text{new}} \cup \{C\}$ is unsatisfiable. Clearly there is a NL-deduction of $()$ from $\langle S_{\text{new}}, \{C\}\rangle$.

Induction step:
Let $S = S_{\text{new}} \cup S_{\text{old}}$ be an unsatisfiable set of ground clauses and $k(S_{\text{new}}) = N > 0$. Induction hypothesis: assume for any unsatisfiable set of ground clauses T such that $T = T_{\text{new}} \cup T_{\text{old}}$ and T_{old} is consequence set and $k(T_{\text{new}}) < N$, there is an NL-deduction of $()$ from $\langle T_{\text{new}}, T_{\text{old}}\rangle$.

If $() \in S$, the induction step holds. So assume $() \notin S$. That is, $() \notin S_{\text{new}}$ and $() \notin S_{\text{old}}$. Since $k(S_{\text{new}}) > 0$, there is at least one clause $C \in S_{\text{new}}$ which contains more than one literal.

Let $S'_{\text{new}} = S_{\text{new}} - \{C\}$. Let $C = (A \vee L)$, where L is a literal and A is a non-empty disjunction of literals. $S'_{\text{new}} \cup \{A\} \cup S_{\text{old}} and S'_{\text{new}} \cup \{L\} \cup S_{\text{old}}$ are each unsatisfiable. Furthermore, $k(S'_{\text{new}} \cup \{A\}), k(S'_{\text{new}} \cup \{(L)\}) < N$. Apply induction hypothesis to conclude that there is an NL-deduction of $()$ from $\langle S'_{\text{new}} \cup \{A\}, S_{\text{old}}\rangle$ (call it D_1) and an NL-deduction of $()$ from $\langle S'_{\text{new}} \cup \{(L)\}, S_{\text{old}}\rangle$ (call it D_2).

We may replace A by C in D_1 and replace each resolvent by the resolvent of its (possibly updated) parents. That gives an NL-deduction of either $()$ or (L) from $(S_{\text{new}}, S_{\text{old}})$ (call it D_3). If the conclusion of D_3 is $()$, the induction step holds. Otherwise, in the last line of D_3, the literal L must have come from the L that is added back to the clause A to give $C = (A \vee L)$. Because $C \in S_{\text{new}}$, the literal L is underlined in the last line of D_3. Placing D_2 after D_3 gives an NL-deduction of $()$ from $(S_{\text{new}}, S_{\text{old}})$.

By induction, the theorem holds.

Theorem 2 (Ground completeness of for CF). *Given $S = S_{\text{new}} \cup S_{\text{old}}$ a set of ground clauses and T a ground clause. Further assume that S_{old} is a consequence set. Given that S logically implies T and T is not a tautology. Then there exists an NL-deduction of T' from $\langle S_{\text{new}}, S_{\text{old}} \rangle$ such that $T' \subseteq T$.*

First we introduce some notation and lemmas.

Notation.
For a set of clauses S and a set of literals T, $S^T = \{c - T | c \in S\}$
$\tilde{}(T)$ is the complement of T. For example, $\tilde{}(p, q, \neg r) = \{(\neg p), (\neg q), (r)\}$

Lemma 2. *(Lemma 1 of [18]) Let T be a ground clause that is not a tautology. Let S be a set of ground clauses. If $S \cup \tilde{}(T)$ is unsatisfiable, then S^T is unsatisfiable.*

Lemma 3. *Let S be a ground consequence set. Let T be a set of ground literals. Then S^T is a consequence set.*

Proof. Let c be a clause that follows from S^T. There exists a clause $d \subseteq c \cup T$ that follows from S. S is a consequence set. Choose $r \in S$ such that $r \subseteq d$. $r - T \subseteq d - T \subseteq c$. $r - T \in S^T$, from definition. Hence, there exists a clause in S^T that subsumes c.

Proof. (of theorem 2) (Using the technique from [18])

$S \cup \tilde{}(T)$ is unsatisfiable. Then by lemma 2, S^T is unsatisfiable.

Clearly, $S^T = S^T_{\text{new}} \cup S^T_{\text{old}}$.

By lemma 3, S^T_{old} is a consequence set.

So by theorem 1, there is an NL-deduction of () from $\langle S^T_{\text{new}}, S^T_{\text{old}} \rangle$ (call it D).

In D, replace each occurrence of C' in S^T by the clause C in S from which C' was produced. Also replace each resolvent in the deduction D in the obvious way. Call the new deduction D'. Let T' be the set of literals removed from the clauses of S to obtain S^T. D' is an NL-deduction of T' from $(S_{\text{new}}, S_{\text{old}})$. Furthermore, $T' \subseteq T$.

Theorem 3 (FO completeness for CF). *Given $S = S_{\text{new}} \cup S_{\text{old}}$ a set of (first-order) clauses and T a (first-order) clause. Further assume that S_{old} is a consequence set. Given that S logically implies T and T is not a tautology. Then there exists an NL-deduction of T' from $\langle S_{\text{new}}, S_{\text{old}} \rangle$ such that T' subsumes T.*

Proof. (Modeled after theorem 2 of [18])

Lemma 4. *(Theorem 1 of [23]) A ground clause C is a logical consequence of a set S of clauses if and only if there is a finite set S' of ground instances of clauses in S over the Herbrand universe of $S \cup \{C\}$ such that C is a logical consequence of S'.*

We use the technique, first used in [23], of introducing new distinct constants into the Herbrand Universe. Let $x_1, \ldots, x_n$ be all of the individual variables of T, which we now write as $T(x_1, \ldots, x_n)$. Let $b_1, \ldots, b_n$, be new distinct constants not occurring in S or T. S logically implies $T(x_1, \ldots, x_n)$, so S also logically implies $T(b_1, \ldots, b_n)$. Let $H(b_1, \ldots, b_n)$be the Herbrand Universe of $S \cup \{T(b_1, \ldots, b_n)\}$. By lemma 4, there exists a finite set of ground instances S' of S over $H(b_1, \ldots, b_n)$ such that S' logically implies $T(b_1, \ldots, b_n)$. That is, $S' \cup T(b_1, \ldots, b_n)$ is unsatisfiable.

Let $S' = S'_{\text{new}} \cup S'_{\text{rem}}$, where S'_{new} and S'_{rem} are a finite set of ground instances of S_{new} and S_{old} respectively. Let S'_{old} be a consequence set of S'_{rem} such that each clause in S'_{old} is subsumed by a clause in S_{old}. This is possible because each clause in S'_{old} also follows from S_{old} and S_{old} is a consequence set.

By theorem 2, there exists a ground NL-deduction D', from $\langle S'_{\text{new}}, S'_{\text{old}} \rangle$, of a clause $E'(b_1, \ldots, b_n) \subseteq T(b_1, \ldots, b_n)$. In D', replace each clause $C' \in S'$ by the clause $C \in S$ that subsumes C', and replace each ground resolvent of D' by the general resolvent of its two parent clauses. The steps are removed which have become inapplicable and unnecessary because a resolved upon literal no longer exists in the replacing clause. The result is a general NL-deduction, from $\langle S_{\text{new}}, S_{\text{old}} \rangle$, of a clause E which subsumes $E'(b_1, \ldots, b_n)$.

There exists a substitution σ_1, substituting $\{b_1, \ldots, b_n\}$ into the variables of E, such that $E\sigma_1 \subseteq E'(b_1, \ldots, b_n) \subseteq T(b_1, \ldots, b_n)$,

Let σ_2 be a new substitution formed by replacing each b_i in σ_1 by x_i. Since S does not contain the symbols $b_1, \ldots, b_n$, we conclude that

$E\sigma_2 \subseteq T(x_1, \ldots, x_n)$. That is, E subsumes T.

5 Linear resolution with New-Literal Filtering

Linear resolution [16] is a strong restriction on the resolution method which results in a much more efficient complete proof procedure.

In this section, we show that New-Literal Filtering also preserves the completeness of linear resolution for incremental consequence-finding. We call it linear resolution with New-Literal Filtering (NLL-resolution).

Figures 1 and 2 are both examples of linear resolution deduction. Figure 2 is also a linear resolution deduction with New-Literal Filtering.

Now we define NLL-resolution.

Definition 7 (NLL-deduction). *An NLL-deduction of c_n from $(S_{\text{new}}, S_{\text{old}})$ is an NL-deduction:*
$\langle c_1, u_{c_1} \rangle$
$\langle c_2, u_{c_2} \rangle$
$\langle c_3, u_{c_3} \rangle$
$\vdots$
$\langle c_n, u_{c_n} \rangle$
where: For each i, letting $S_i = S^u_{\text{new}} \cup S^n_{\text{old}} \cup \{\langle c_1, u_{c_1} \rangle, \ldots, \langle c_{i-1}, u_{c_{i-1}} \rangle\}$, $\langle c_i, u_{c_i} \rangle$

is a binary resolvent between (a factors of) $\langle c_{i-1}, u_{c_{i-1}} \rangle$ and (a factor of) a structured clause in S_i.

5.1 Completeness for incremental CF

As in the proof of theorem 3, it is often useful to first establish the refutation completeness in the ground case before generalizing to completeness for consequence-finding and lifting to the first-order case.

Anderson and Bledsoe proved the refutation completeness of linear resolution (with merging and subsumption) by proving a stronger result that from a minimally unsatisfiable set of clauses, there is a linear deduction of the empty clause starting from *any* of the clauses. To prove the existence of a linear deduction in the inductive step, a problem is broken into two smaller instances. Applying an induction hypothesis, they obtain a linear deduction of a unit clause in the first instance and a linear deduction of the empty clause starting from that unit clause in the second instance. The two linear deductions are then pieced together to form one linear deduction of the empty clause for the original problem instance. It is crucial here that in the second instance the induction hypothesis asserts that there is a linear deduction from any of the clauses in the minimally unsatisfiable set, otherwise, one may not piece together the two linear deductions together into one linear deduction.

Anderson and Bledsoe's technique is a powerful technique that gives a simple and easy-to-understand proof of refutation completeness. Many authors have used this technique to establish the completeness of other resolution procedures. A straightforward extension of the completeness proof of linear resolution [1] gives a completeness proof of NLL-resolution in the case of ground clauses and a single new clause. The result is not liftable to the case of first-order clauses and a single new clause because a single new first-order clause may correspond to several new ground clauses in the lifting process.

Unfortunately, to prove the case of several new clauses (or even a single new clause in the first-order case), it appears that an alternative technique is needed because the stronger completeness statement that an empty clause may be derived from any clause in a minimally unsatisfiable subset does not hold for NLL-resolution. So instead, we used a technique that transforms an NL-deduction into a linear resolution deduction while preserving the obedience to the New-Literal Filtering.

Lemma 5 (Transformation to NLL deduction). *Given a ground NL-deduction D concluding s from $\langle S_{\text{new}}, S_{\text{old}} \rangle$, there exists a ground NLL-deduction D' concluding $s' \subseteq s$ from $(S_{\text{new}}, S_{\text{old}})$.*

Proof. (sketch)

Given a non-linear NL-deduction tree (DAG) G, we may assume that G does not contain tautologies. Consider a node c such that both of c's parents (a and b) are resolvents and neither is an ancestor of the other.

Figure 3 illustrates this section of the deduction tree.

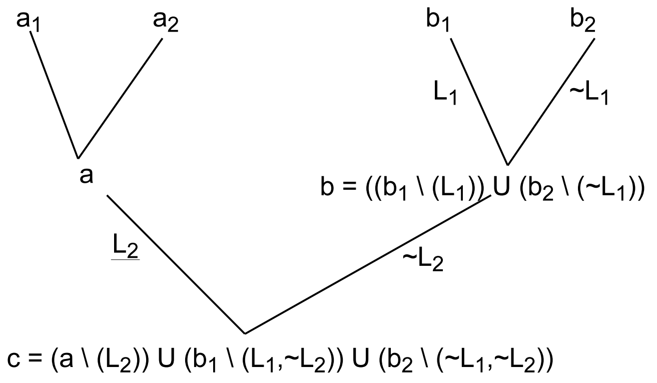

a1
a2
b1
b2
L1
~L1
a
b = ((b1 \ (L1)) U (b2 \ (~L1))
L2
~L2
c = (a \ (L2)) U (b1 \ (L1,~L2)) U (b2 \ (~L1,~L2))

Fig. 3. The original subtree. At least one of L_1 and ˜L_1 is underlined.

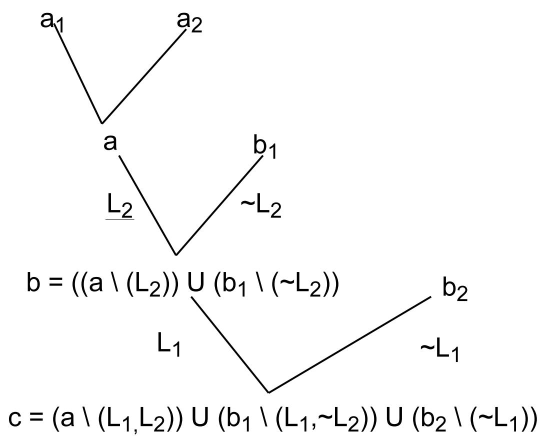

a1
a2
a
b1
L2
~L2
b = ((a \ (L2)) U (b1 \ (~L2))
b2
L1
~L1
c = (a \ (L1,L2)) U (b1 \ (L1,~L2)) U (b2 \ (~L1))

Fig. 4. A transformed subtree. At least one of L_1 and ˜L_1 is underlined.

At least one of b_1 and b_2 contains the literal ˜L_2. Assume without loss of generality that b_1 does.

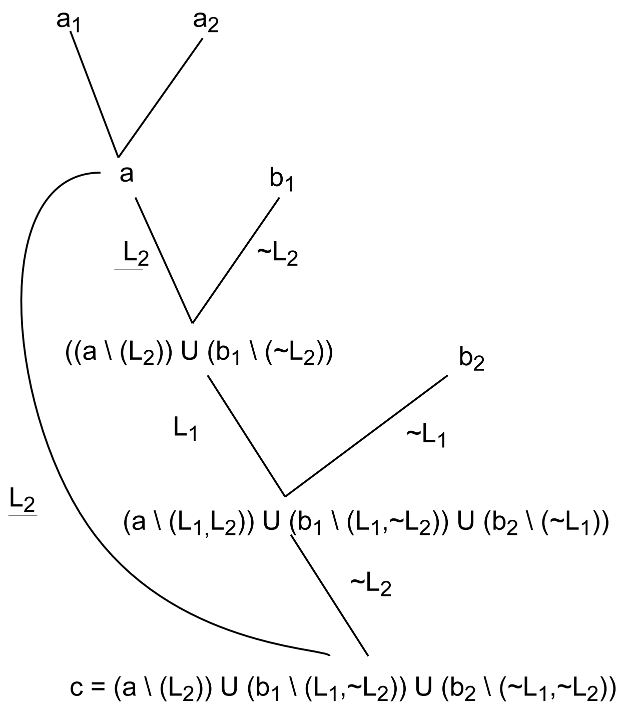
In an NL deduction, at least one of the two parents of c has the resolved upon literal underlined. Assume without loss of generality that in this case a does. Also, at least one of the two parents of b have the resolved upon literal underlined.

We transform this section of the deduction tree by choosing a as the main branch and moving b_1 and b_2 to be part of this branch.

If ˜$L_2 \notin b_2$, we replace the subtree in figure 3 with the subtree in figure 4.

Otherwise, ˜$L_2 \in b_2$, then we may replace the subtree in figure 3 with the subtree (DAG) in figure 5.

In each case, we see that the transformed subtree obeys the New-Literal Filtering and the linear restriction and obtain a conclusion that subsumes the orig-



a1
a2
a
b1
L2
~L2
((a \ (L2)) U (b1 \ (~L2))
b2
L1
~L1
L2
(a \ (L1,L2)) U (b1 \ (L1,~L2)) U (b2 \ (~L1))
~L2
c = (a \ (L2)) U (b1 \ (L1,~L2)) U (b2 \ (~L1,~L2))

Fig. 5. A transformed subtree. At least one of L_1 and $\tilde{}L_1$ is underlined.

inal conclusion. Furthermore, each underlined literal in the original conclusion either remains underlined or is removed in the new conclusion. More precisely, letting $\langle s_o, u_{s_o}\rangle$ be the original conclusion and $\langle s_n, u_{s_n}\rangle$ be the new conclusion, there exists a substitution σ such that $s_n\sigma \subseteq s_o$ and $u_{s_o} \cap s_n\sigma \subseteq u_{s_n}\sigma$. These conditions ensure that transforming one section of the tree does not cause further deduction steps that depend on the conclusion of the transformed section to disobey the New-Literal Filtering. By applying these transformations bottom-up (transform a later deduction step before an earlier deduction step) to each section of the deduction tree D that does not satisfy the linear restriction[4], one obtains an NLL deduction D' concluding $s' \subseteq s$.

Theorem 4 (Completeness of NLL-resolution). *Given $S = S_{\text{new}} \cup S_{\text{old}}$ a set of (first-order) clauses and T a (first-order) clause. Further assume that S_{old} is a consequence set. Given that $S \models T$ but $S_{\text{old}} \not\models T$, then there exists an NLL-deduction of T' from $(S_{\text{new}}, S_{\text{old}})$ such that T' subsumes T.*

Proof. By theorem 2 and lemma 5, NLL-deduction is complete for ground consequence-finding.

[4] Discarding each branch that is no longer applicable because the a literal resolved upon at the beginning of the branch no longer exists.

By the lifting procedure in theorem 3, NLL-deduction is complete for first-order consequence-finding.

6 Related work

We have searched extensively but failed to find any work on first-order incremental consequence-finding that exploits incrementality at the literal-level. The closest piece of related work is the work on SOL resolution [6, 5]. SOL resolution places a weak, clause-level restriction on resolution to exploit the incremental nature of the problem. The work on SFK-resolution [24] also considers the consequence-finding problem in first-order logic, but it does not consider the incremental version of the problem.

There is some prior work on resolution-based approaches to incremental consequence-finding in the setting of propositional logic (often called the prime implicates problem) [11, 13, 7]. The incremental algorithms proposed in [11] and [7] do not generalize to the setting of first-order clauses without losing completeness. That is because the central restriction used to exploit incrementality in both cases (not resolving two new consequence clauses with each other) leads to incompleteness in the first-order case. De Kleer is concerned with optimizing the efficiency of the method in [11] (through clever algorithms for the basic operations) without changing the general method.

7 Conclusions and future work

In this paper, we presented New-Literal Filtering, a novel resolution restriction that aggressively prunes the resolution tree to increase efficiency while preserving completeness for incremental consequence-finding. New-Literal Filtering significantly reduces the number of unification attempts as well as the number of clauses generated at each level of resolution search. The overall savings can be exponential in the height of the search tree. Some of the savings may be mitigated by the potential for some of the clauses New-Literal Filtering avoids generating to be equivalent to or subsumed by previously generated clauses. If equivalent clause elimination and subsumed clause elimination are used, these clauses would be eliminated anyway. In this case the avoidance of these clauses saves computation at each search level but does not reduce the branching factor. The avoidance of clauses not subsumed by previously generated clauses would still reduce the branching factor. Furthermore, New-Literal Filtering may allow us to turn off subsumed clause elimination (an expensive check) and still maintain a manageable branching factor.

We are currently applying the results to improve several data management systems, including Gates Information Network, an online services system that handles event scheduling, degree planning, and people information in the computer science department at Stanford University.

To achieve even greater efficiency, we are also investigating an ordered resolution method that combines New-Literal Filtering with c-order restriction and linear restriction, to incrementally find consequences in a target production field.

8 Acknowledgements

We would like to acknowledge Carl Hewitt, Mary-Anne Williams, Charles Petrie, Alyssa Glass, Michael Kassoff and Euijong Whang for their helpful comments.

References

1. Anderson, R., Bledsoe, W.W.: A linear format for resolution with merging and a new technique for establishing completeness. J. ACM 17(3), 525–534 (1970)
2. Chakravarthy, U.S., Grant, J., Minker, J.: Logic-based approach to semantic query optimization. ACM Trans. Database Syst. 15(2), 162–207 (1990)
3. Chang, C.L., Lee, R.C.T.: Symbolic Logic and Mechanical Theorem Proving. Academic Press, Inc., Orlando, FL, USA, 1st edn. (1997)
4. Cox, P.T., Pietrzykowski, T.: Causes for events: their computation and applications. In: Proc. of the 8th international conference on Automated deduction. pp. 608–621. Springer-Verlag New York, Inc., New York, NY, USA (1986)
5. Inoue, K.: Consequence-finding based on ordered linear resolution. In: Proc. of IJCAI. pp. 158–164 (1991)
6. Inoue, K.: Linear resolution for consequence finding. Artif. Intell. 56(2-3), 301–353 (1992)
7. Jackson, P.: Computing prime implicates incrementally. In: Automated Deduction CADE-11, pp. 253–267. Springer (1992)
8. Kao, E.J.Y., Genesereth, M.: Answers with consistent support – complexity and query rewriting. Tech. Rep. LG-2010-01, Stanford University, Stanford, CA (2010), http://logic.stanford.edu/reports/LG-2010-01.pdf
9. Kassoff, M., Valente, A.: An introduction to logical spreadsheets. Knowl. Eng. Rev. 22, 213–219 (September 2007)
10. Kassoff, M., Zen, L.M., Garg, A., Genesereth, M.: Predicalc: a logical spreadsheet management system. In: Proceedings of the 31st international conference on Very large data bases. pp. 1247–1250. VLDB '05, VLDB Endowment (2005)
11. Kean, A., Tsiknis, G.: An incremental method for generating prime implicants/implicates. J. Symb. Comput. 9(2), 185–206 (1990)
12. de Kleer, J.: An assumption-based tms. Artif. Intell. 28(2), 127–162 (1986)
13. de Kleer, J.: An improved incremental algorithm for generating prime implicates. In: AAAI'92: Proceedings of the 10th National Conference on Artifical Intelligence. pp. 780–785 (1992)
14. Lee, C.T.: A completeness theorem and a computer program for finding theorems derivable from given axioms. Ph.D. thesis, University of California, Berkeley (1967)
15. Levy, A.Y.: Logic-based techniques in data integration pp. 575–595 (2000)
16. Loveland, D.W.: A linear format for resolution. In: Symposium on Automatic Demonstration. Lecture Notes in Mathematics, vol. 125, pp. 147–162. Springer (1970)
17. McCarthy, J.: Circumscription—a form of non-monotonic reasoning pp. 145–152 (1987)

18. Minicozzi, E., Reiter, R.: A note on linear resolution strategies in consequence-finding. Artif. Intell. 3(1-3), 175–180 (1972)
19. Pople, H.E.: On the mechanization of abductive logic. In: IJCAI'73: Proceedings of the 3rd international joint conference on Artificial intelligence. pp. 147–152. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1973)
20. Provan, G.M.: The computational complexity of multiple-context truth maintenance systems. In: ECAI. pp. 522–527 (1990)
21. Robinson, J.A.: A machine-oriented logic based on the resolution principle. J. ACM 12, 23–41 (January 1965)
22. Simon, L., Val, A.D.: Efficient consequence finding. In: Proc. of the 17 th International Joint Conference on Artificial Intelligence (IJCAI01. pp. 359–365. Morgan Kaufmann (2001)
23. Slagle, J.R., Chang, C.L., Lee, R.C.T.: Completeness theorems for semantic resolution in consequence-finding. In: IJCAI'69: Proceedings of the 1st international joint conference on Artificial intelligence. pp. 281–285. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1969)
24. del Val, A.: A new method for consequence finding and compilation in restricted languages. In: AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence. pp. 259–264. American Association for Artificial Intelligence, Menlo Park, CA, USA (1999)

Appendix

Below is the resolution process, under New-Clause Filtering, for generating S_1 and S_2 in example 4. The clauses that would NOT be generated under New-Literal Filtering are marked by an asterisk after its line number.

```
Step            Proof                         Justification
1              {~p(X),r(X)}                  S_{old}
2              {t(X),~r(X)}                   S_{old}
3              {~t(X),~s(X)}                S_{old}
4              {t(X),~p(X)}                  S_{old}
5              {~s(X),~p(X)}               S_{old}
6              {~r(X),~s(X)}                S_{old}
7              {p(a),s(Y)}                     S_{new}
8              {s(V2),r(a)}             Resolution: 7, 1
9              {p(a),~t(X)}             Resolution: 7, 3
10            {s(V8),t(a)}             Resolution: 7, 4
11            {s(V10),~s(a)}             Resolution: 7, 5
12            {p(a),~p(X)}             Resolution: 7, 5
13            {p(a),~r(X)}             Resolution: 7, 6
14            {s(V16),t(a)}             Resolution: 8, 2
15*          {r(a),~t(X)}             Resolution: 8, 3
16            {r(a),~p(X)}             Resolution: 8, 5
17            {r(a),~r(X)}             Resolution: 8, 6
18*          {s(V24),~s(a)}             Resolution: 8, 6
```

```
19            {~t(V28),r(a)}            Resolution: 9, 1
20*          {~r(X),p(a)}            Resolution: 2, 9
21*          {~p(X),p(a)}            Resolution: 4, 9
22            {~t(V34),t(a)}            Resolution: 9, 4
23            {~t(V36),~s(a)}            Resolution: 9, 5
24*          {t(a),~t(X)}            Resolution: 10, 3
25            {s(V50),~s(a)}            Resolution: 10, 3
26            {t(a),~p(X)}            Resolution: 10, 5
27            {t(a),~r(X)}            Resolution: 10, 6
28*          {s(V62),p(a)}            Resolution: 10, 9
29*          {p(a),s(V10)}            Resolution: 7, 11
30*          {r(a),s(V10)}            Resolution: 8, 11
31*          {t(a),s(V10)}            Resolution: 10, 11
32            {~s(a),~t(X)}            Resolution: 11, 3
33            {~s(a),~p(X)}            Resolution: 11, 5
34            {~s(a),~r(X)}            Resolution: 11, 6
35            {~s(a),s(V10)}            Resolution: 11, 11
36            {s(V80),p(a)}            Resolution: 7, 12
37            {~t(V82),p(a)}            Resolution: 9, 12
38*          {~p(V85),r(a)}            Resolution: 12, 1
39*          {~p(V85),t(a)}            Resolution: 12, 4
40*          {~p(V85),~s(a)}            Resolution: 12, 5
41            {~p(V85),p(a)}            Resolution: 12, 12
42*          {s(V93),p(a)}            Resolution: 8, 13
43            {~r(V98),r(a)}            Resolution: 13, 1
44            {~r(V98),t(a)}            Resolution: 13, 4
45            {~r(V98),~s(a)}            Resolution: 13, 5
46            {~r(V98),p(a)}            Resolution: 13, 12
```