Near-Horn Prolog and the Ancestry Family of Procedures*

David W. Reed
Department of Computer Science
Duke University
Durham, NC 27708 USA
dwr@cs.duke.edu

Donald W. Loveland
Department of Computer Science
Duke University
Durham, NC 27708 USA
dwl@cs.duke.edu

December 1992
Revised: August 1994

Abstract

The Near-Horn Prolog procedures have been proposed as effective procedures in the area of disjunctive logic programming, an extension of logic programming to the (first-order) non-Horn clause domain. In this paper, we show that these case-analysis based procedures may be viewed as members of a class of procedures called the "ancestry family," which also includes Model Elimination (and its variants), the Positive Refinement of Model Elimination, and SLWV-resolution. The common feature which binds these procedures is the extension of SLD-resolution to full first-order logic with the addition of an ancestor cancellation rule. Procedures in the ancestry family are all sound and complete first-order procedures that can be seen to vary along three parameters: (1) the number of clause contrapositives required, (2) the amount of ancestor checking that must occur, and (3) the use (if any) of a Restart rule. Using a sequent-style presentation format for all procedures, we highlight the close relationships among these procedures and compare their relative advantages.

1 Introduction

The near-Horn Prolog project at Duke University has focused on extending the logic programming paradigm to disjunctive programs, where implication clauses may have disjunctions of atoms as consequent. (See [RLS92] for a project overview.) In [Lov87, Lov91], Loveland introduced the procedure now known as Unit near-Horn Prolog (UnH-Prolog), originally referred to as simply near-Horn Prolog or Progressive near-Horn Prolog. Subsequently, Loveland and Reed [LR91, Ree92] developed the Inheritance near-Horn Prolog (InH-Prolog) variant that is simpler and sometimes permits shorter refutations, but may have a lower lips rate than UnH-Prolog. The major focus in the design of the near-Horn Prologs has been to extend SLD-resolution [Kow74, Hil74] (the underlying procedure of Prolog)

*This research was partially supported by NSF Grants CCR-8900383 and CCR-9116203.

while retaining as many of its structural properties as possible. The near-Horn Prologs have several features which make them desirable as Prolog extensions when compared to existing procedures such as Model Elimination [Lov68, Lov69, Lov78] and SLI-resolution [MZ82]. These include a positive implication form for program clauses, limited contrapositive use, a high lips rate, and a graceful performance degradation as programs become more non-Horn. See [RL92, RLS92, Ree92] for detailed discussions of these and other features.

The near-Horn Prolog procedures were proposed as effective procedures in the area of disjunctive logic programming, an extension of logic programming to the (first-order) non-Horn clause domain. The near-Horn Prologs are what we refer to as "case-analysis" based procedures. The basic idea behind these procedures is the splitting rule [Lov78] which allows one to prove the unsatisfiability of a non-Horn program by proving the unsatisfiability of a collection of Horn set cases. Previous work has shown that this case-analysis approach is also taken by procedures with different forms and design goals. In [RL92], InH-Prolog was shown to be closely related to a variant of Plaisted's Simplified Problem Reduction Format [Pla82] and a subset of Gabbay and Reyle's N-Prolog [GR84, Gab85]. There is also a close relationship between the near-Horn Prolog procedures and the reduction of a non-Horn clause set to Horn sets via matrix reduction as presented in [CEFB].

In this paper, we show that these case-analysis based procedures may also be viewed as members of a larger class of procedures, which we call the "ancestry family." The feature which characterizes the ancestry family is the extension of SLD-resolution to full first-order logic with the addition of an ancestor cancellation rule. Procedures in the ancestry family are all sound and complete first-order procedures that can be seen to vary along three parameters: (1) the number of clause contrapositives required, (2) the amount of ancestor checking that must occur, and (3) the use (if any) of a Restart rule. The base procedure in the ancestry family, Model Elimination or ME (Loveland [Lov68, Lov69, Lov78]), requires the presence of all contrapositives of the clauses of the program. In addition, all ancestors in a deduction must be checked for possible cancellation. Reduction in the use of contrapositives would reduce the branching factor at nodes of the search tree, with an expected decrease in the size of the search tree. Reducing the check of ancestors simplifies the inner-loop operation and permits possible speedup. By "inner-loop" we mean the processing of a single occurrence of an inference rule, the basic code that is the atomic step in the proof procedure. The Positive Refinement of ME (Plaisted [Pla90]) requires all contrapositives as in ME, but decreases the amount of ancestor checking necessary by only considering negative ancestors. SLWV-resolution (Pereira, Caires and Alferes [PCA90, PCA93]) requires no contrapositives due to its introduction of a restart rule. However, the restart rule is difficult to control and may be more costly than having the contrapositives. The near-Horn Prologs, of which we consider the two main variants here, may be viewed as compromise approaches which restrict ancestor checking (as in the Positive Refinement) and limit the number of contrapositives by utilizing a restricted restart rule. Using a sequent-style presentation format for all procedures (occasionally taking liberties with the sequent format), we highlight the close relationships between these procedures and compare their relative advantages. For clarity and conciseness, we present all procedures as refutation procedures at the propositional level, although each lifts to first-order logic as expected.

2 Near-Horn Prolog

The major focus in the design of the near-Horn Prolog procedures has been to extend SLD-resolution (the underlying procedure of Prolog) while retaining as much of its form and flavor as possible. The approach taken by the near-Horn Prolog procedures is to combine SLD-resolution with case-analysis to perform non-Horn reasoning. Recall that SLD-resolution is a procedure for reasoning from Horn clauses, i.e. clauses containing at most one positive literal. (In this paper a clause, unless qualified, is viewed as a disjunction of literals.) An SLD refutation (versus an SLD deduction) of Horn program P is a sequence of lines, with each line containing goals to be solved and the initial line containing the single goal `FALSE`, where `FALSE` is intended to have truth value *false*. Note that a negative clause $\neg B_1 \vee \cdots \vee \neg B_n$, where each B_i is an atom, is equivalent to $\texttt{FALSE} \leftarrow B_1 \wedge \cdots \wedge B_n$ in implication form, an alternate form we use heavily. Subsequent lines are obtained via the operation of goal reduction. Given a line containing goal H and a program clause $H \vee \neg B_1 \vee \cdots \vee \neg B_n$ (usually written in implication form as $H \leftarrow B_1 \wedge \cdots \wedge B_n$), goal reduction replaces the goal H with the goals $B_1, \ldots, B_n$. An SLD refutation is terminated by a line with no goals.

Example 2.1 Consider the Horn program P, rewritten to the right in implication form.

$$\begin{array}{lll} \neg\mathtt{x} \vee \neg\mathtt{y} & \Rightarrow & \mathtt{FALSE} \leftarrow \mathtt{x} \wedge \mathtt{y} \\ \mathtt{x} \vee \neg\mathtt{h}_1 & \Rightarrow & \mathtt{x} \leftarrow \mathtt{h}_1 \\ \mathtt{y} & \Rightarrow & \mathtt{y} \\ \mathtt{h}_1 \vee \neg\mathtt{y} & \Rightarrow & \mathtt{h}_1 \leftarrow \mathtt{y} \end{array}$$

The following is an SLD refutation of P.

```
?- FALSE
:- x, y
:- h1, y
:- y, y
:- y
:-
```

The near-Horn Prolog procedures extend SLD-resolution to handle non-Horn clauses as well. In expressing such clauses in implication form, we have a choice as to their representation. Following the conventions of the field of disjunctive logic programming, the clause $H_1 \vee \cdots \vee H_m \vee \neg B_1 \vee \cdots \neg B_n$ may be represented in disjunctive implication form as $H_1 \vee \cdots \vee H_m \leftarrow B_1 \wedge \cdots \wedge B_n$. This positive implication form is commonly used in descriptions of the near-Horn Prologs. Later in this paper, we will also make use of standard implication form, where a non-Horn clause is represented by an implication with only one consequent (head) literal. For example, $H_1 \vee \cdots \vee H_m \vee \neg B_1 \vee \cdots \neg B_n$ may be represented as $H_1 \leftarrow B_1 \wedge \cdots \wedge B_n \wedge \neg H_2 \wedge \cdots \wedge \neg H_m$ or $\neg B_1 \leftarrow B_2 \wedge \cdots \wedge B_n \wedge \neg H_1 \wedge \cdots \wedge \neg H_m$ (among others).

A near-Horn Prolog refutation of program P is a sequence $B_0, \ldots, B_n$ of blocks, with each block resembling an SLD refutation. The start block B_0 is an SLD refutation with the alteration that if a goal H_i calls a non-Horn (disjunctive) clause with head literals $H_1, \ldots, H_m$, then the heads $H_1, \ldots, H_{i-1}, H_{i+1}, \ldots, H_m$ are simply ignored (deferred) in this block. That is, the called clause is treated as if it were a Horn clause with head H_i. As in SLD-resolution, the block terminates when there are no goals remaining. However, here the computation must continue in order to remove the deferred heads. For each deferred head, there is a restart block B_j $(j > 0)$ whose task is to remove that head. The deferred head is promoted to distinguished active head and the new operation of cancellation permits it to satisfy (cancel) any matching goal occurrence in the restart block. Except for the cancellation operation, a restart block behaves the same as the start block (it is an SLD-like refutation) so goal reduction with a non-Horn clause can introduce other deferred heads. Finally, a near-Horn Prolog refutation terminates when some block removes the last deferred head.

A standard feature of the near-Horn Prologs is the cancellation pruning rule, which states that each restart block of a successful refutation must contain a cancellation by the distinguished active head. This required cancellation provides a powerful pruning rule that removes blocks from further consideration when needless non-Horn clauses are called that lead the deduction astray. Another reason for demanding cancellation is that this provides an extra test of relevance of the accepted block.

Example 2.2 Consider the following non-Horn program P (a slight generalization of the program in Example 2.1), rewritten in disjunctive implication form on the right.

$$
\begin{array}{lcl}
\neg \mathtt{x} \vee \neg \mathtt{y} & \Rightarrow & \mathtt{FALSE} \leftarrow \mathtt{x} \wedge \mathtt{y} \\
\mathtt{x} \vee \neg \mathtt{h}_1 & \Rightarrow & \mathtt{x} \leftarrow \mathtt{h}_1 \\
\mathtt{x} \vee \neg \mathtt{h}_2 & \Rightarrow & \mathtt{x} \leftarrow \mathtt{h}_2 \\
\mathtt{y} & \Rightarrow & \mathtt{y} \\
\mathtt{h}_1 \vee \mathtt{h}_2 \vee \neg \mathtt{y} & \Rightarrow & \mathtt{h}_1 \vee \mathtt{h}_2 \leftarrow \mathtt{y}
\end{array}
$$

The following is a near-Horn Prolog refutation of P.

```
?- FALSE
:- x, y
:- h1, y
:- y, y          #    {h2}
:- y             #    {h2}
:-               #    {h2}

restart:
?- FALSE         # h2
:- x, y          # h2
:- h2, y         # h2
:- y             # h2              % cancellation
:-               # h2
```

In the start block of the refutation, the initial goal FALSE reduces with the first clause, introducing goals x and y (as in SLD-resolution). Similarly, the goal x reduces with the second clause, introducing goal h_1. Here, the goal h_1 matches a head in the non-Horn (disjunctive) clause, and so goal reduction introduces goal y and deferred head h_2 (which is delineated by brackets). The start block is completed by reducing goals y with the fact y. In the restart block, the deferred head h_2 becomes the distinguished active head (without brackets) and is used to cancel goal h_2 when that goal is introduced via reduction.

The near-Horn Prolog variants differ from each other in their treatment of active heads and restarts. InH-Prolog differs from the original UnH-Prolog variant by allowing more than one active head in blocks, all with cancellation capability. In addition to the distinguished active head promoted from the deferred head list, the restart block "inherits" all of the active heads from the block in which the active head was deferred. For example, if head d is deferred in a block with active heads $a_1, \ldots, a_k$, then the resulting restart block will have active heads $d, a_1, \ldots, a_k$, with d being the distinguished active head. The removal of the distinguished active head is still considered the task of the restart block (and required by the cancellation pruning rule). The inheritance of active heads in InH-Prolog tends to produce shorter refutations since extra active heads can be used to cancel goals. However, the presence of more than one active head implies that the inner-loop speed degrades in proportion to block depth.

Example 2.3 Consider the program from Example 2.2, altered by replacing the clauses $x \leftarrow h_1$ and $x \leftarrow h_2$ with clauses $x \leftarrow h_1 \wedge y$, $x \vee h_1 \leftarrow h_2$ and $x \leftarrow h_1 \wedge h_2$. The following is an InH-Prolog refutation of this new program.

```
        ?- FALSE
        :- x, y
        :- h1, y, y
        :- y, y, y      #     {h2}
        :- y, y         #     {h2}
        :- y            #     {h2}
        :-              #     {h2}
restart:
        ?- FALSE        # h2
        :- x, y         # h2
        :- h2, y        # h2  {h1}
        :- y            # h2  {h1}        % cancellation
        :-              # h2  {h1}
restart:
        ?- FALSE        # h1, h2
        :- x, y         # h1, h2
        :- h1, h2, y    # h1, h2
        :- h2, y        # h1, h2          % cancellation
        :- y            # h1, h2          % cancellation
        :-              # h1, h2
```

As before, the deferred head $\mathtt{h}_2$ is introduced in the start block via reduction with the clause $\mathtt{h}_1 \vee \mathtt{h}_2 \leftarrow \mathtt{y}$. In the restart block, the goal $\mathtt{x}$ reduces with the new clause $\mathtt{x} \vee \mathtt{h}_1 \leftarrow \mathtt{h}_2$, introducing goal $\mathtt{h}_2$ and deferring $\mathtt{h}_1$. The resulting restart block has $\mathtt{h}_1$ as distinguished active head and $\mathtt{h}_2$ as inherited active head. Both of these heads are used to cancel goals in the restart block.

The other feature which differentiates the near-Horn Prolog variants is the selection of initial goals in restart blocks (which we refer to as restart goals). In InH-Prolog, the restart goal for a block is always `FALSE`. In UnH-Prolog, the restart goal can be `FALSE` or any other ancestor goal of the distinguished active head, i.e. any goal which led up to the deferral of that head. This mechanism for restarting is more complex than that of InH-Prolog and can increase the branching factor, but can lead to shorter refutations. Using a search order called Progressive Search, no additional search is introduced by these added restart goals in the propositional case (see [Lov91]).

Example 2.4 Consider the program and refutation from Example 2.2. The following is a shorter UnH-Prolog refutation of the program which takes advantage of the more complex restart mechanism.

```
        ?- FALSE
        :- x, y
        :- h1, y
        :- y, y          #     {h2}
        :- y             #     {h2}
        :-               #     {h2}
restart:
        ?- x             # h2
        :- h2            # h2
        :-               # h2              % cancellation
```

Note here that a shorter refutation is obtained by selecting the ancestor goal `x` for restarting. By doing so, goal reduction with the clause $\mathtt{FALSE} \leftarrow \mathtt{x} \wedge \mathtt{y}$ is avoided and so the additional goal `y` is not reintroduced.

The basic idea behind the near-Horn Prolog procedures is to combine SLD-resolution and case-analysis to perform non-Horn reasoning. Given a program P, an SLD refutation of the Horn clauses in P suffices as a refutation of P. If no such refutation exists, the splitting rule [Lov78] can be used to produce a refutation by cases. The variant of the splitting rule that we consider (slightly stronger than the original) states that given disjunctive program P and ground clause $H_1 \vee \cdots \vee H_m \leftarrow B_1 \wedge \cdots \wedge B_n$:

$$
\begin{aligned}
&P \cup \{H_1 \vee \cdots \vee H_m \leftarrow B_1 \wedge \cdots \wedge B_n\} \text{ is unsatisfiable} \quad \text{iff} \\
&\qquad P \cup \{H_i \leftarrow B_1 \wedge \cdots \wedge B_n\} \text{ is unsatisfiable (for some } i) \quad \text{and} \\
&\qquad P \cup \{H_j\} \text{ is unsatisfiable (for all } j \neq i).
\end{aligned}
$$

Here, $P \cup \{H_i \leftarrow B_1 \wedge \cdots \wedge B_n\}$ can be seen as the case where all head atoms except for H_i are dismissed (i.e presumed false). The other cases of the form $P \cup \{H_j\}$ represent the alternatives where each head atom is assumed true and added as a conditional fact (subsequently subsuming the original clause). The splitting rule states that showing the unsatisfiability of these cases suffices to show that the original program is unsatisfiable. Since an application of the splitting rule produces cases with one less non-Horn clause, any disjunctive clause can be repeatedly split to eventually obtain cases from which SLD refutations (i.e. nH-Prolog blocks) can be constructed. Following this reasoning, the InH-Prolog variant may be seen as more directly implementing the case-analysis approach since the inheritance of active heads corresponds to the retention of assumptions in subcases.

Example 2.5 Consider the program and refutation from Example 2.2 above. Using the splitting rule, the program can be split into two cases: $\{\mathtt{FALSE} \leftarrow \mathtt{x} \wedge \mathtt{y},\ \ \mathtt{x} \leftarrow \mathtt{h}_1,\ \ \mathtt{x} \leftarrow \mathtt{h}_2,\ \ \mathtt{y},\ \ \mathtt{h}_1 \leftarrow \mathtt{y}\}$ and $\{\mathtt{FALSE} \leftarrow \mathtt{x} \wedge \mathtt{y},\ \ \mathtt{x} \leftarrow \mathtt{h}_1,\ \ \mathtt{x} \leftarrow \mathtt{h}_2,\ \ \mathtt{y},\ \ \mathtt{h}_2\}$. The start block in the refutation corresponds to an SLD refutation of the first case (see Example 2.1), while the restart block corresponds to an SLD refutation of the second.

The case-analysis approach taken by the near-Horn Prolog procedures is a natural one for reasoning in the presence of non-Horn information. By reducing a non-Horn deduction into a sequence of Horn deductions, the form and behavior of Horn clause procedures (specifically SLD-resolution) can be retained. In [RL92], Reed and Loveland showed that this same approach is also taken by two other procedures with very different forms and design goals. In particular, InH-Prolog was shown to be closely related to a variant of Plaisted's Simplified Problem Reduction Format [Pla82] and a subset of Gabbay and Reyle's N-Prolog [GR84, Gab85]. The fact that this same form of case-analysis is present in such varied procedures speaks to the generality and significance of this approach.

3 The Ancestry Family

While the case-analysis approach taken by the near-Horn Prologs and related procedures is a natural one, it is interesting to note that these procedures also fit into a larger class of procedures, which we call the ancestry family. Members of this family are characterized by the basic goal reduction operation of SLD-resolution, with the addition of ancestor cancellation. Recall that for our discussion and comparison, we present all procedures in this paper as refutation procedures at the propositional level. To make the comparison of procedures more straightforward, we will utilize a common sequent-style format for describing all procedures. In a slight deviance from standard interpretation, we will read the sequent '$\Gamma \rightarrow \alpha$' not as '$\alpha$ follows logically from Γ', but instead as 'α follows from $P \cup \Gamma$', where P is the set of given formulas, i.e. the program. The inference rules of the systems will encode the clauses of the program rather than logical equivalences that introduce connectives and quantifiers.

SLD refutations may be expressed in this format, although in this case Γ is empty for all sequents in the proof, so the sequent arrow is omitted. The resulting trees then are also in a standard problem reduction format, a form we will not emphasize here but which does give

some of the standard terminology we adopt presently. The root of an SLD refutation tree, corresponding to the first line of a linear SLD refutation, will contain the goal `FALSE` (i.e. the refutation tree is a derivation of `FALSE`). The operation of goal reduction is described by the following inference rule schema:

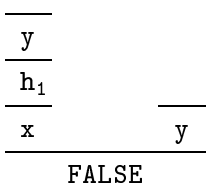
Goal Reduction Rule:
For each implication clause $H \leftarrow B_1 \wedge \cdots \wedge B_n$ in P:

$$\frac{B_1 \quad \cdots \quad B_n}{H}$$

Recall that the literals in a problem-reduction tree are called goals, with the B_i's referred to as subgoals of H. Also hereafter all clauses will be represented in standard implication form, with one consequent (head) only. Non-Horn clauses will thus be represented using negative antecedent (body) literals. The leaves of the refutation tree will then be lines with no goals (obtained via goal reduction with facts). Describing SLD-resolution in this problem-reduction format highlights the goal/subgoal nature of the goal reduction operation.

Example 3.1 Consider the SLD refutation from Example 2.1 above. To the right is the corresponding refutation written in problem-reduction form.

```
?- FALSE
:- x, y
:- h1, y
:- y, y
:- y
:-
```



Describing the goal reduction operation of SLD-resolution in this form also makes it simple to describe the operation of ancestor cancellation which is common to the procedures in the ancestry family. Ancestor cancellation allows for a goal to be solved (canceled) if it matches with the negation of some ancestor goal, i.e. some literal which appears as goal on the path between the root of the refutation tree and the goal in question. While this notion of ancestor is non-local here, we shall see that it can be made local using our sequent notation, where the list of ancestors of a goal is maintained in the sequent. Members of the ancestry family of procedures may be characterized by the basic goal reduction mechanism of SLD-resolution, with the addition of ancestor cancellation.

3.1 Model Elimination

The base procedure in the ancestry family is Model Elimination (ME), developed by Loveland [Lov68, Lov69, Lov78]. In this paper, we will use the term ME to refer to several different variants of Loveland's original procedure. The original procedure utilized chains of literals and a specific search ordering (for efficiency reasons). The MESON procedure was

described by Stickel and Loveland [SL73, Lov78] as a problem reduction version of Model Elimination, where chains of literals were replaced with "ancestor lists." SL-resolution [KK71] is a variant of Model Elimination in which factoring is allowed, while SLI-resolution [MZ82] is a generalization in which arbitrary selection is allowed. The role of ancestor resolution in ME was highlighted by Wakayama in [Wak88, Wak89], where he characterized the basic ME procedure under the name MEA (Multiple Entry with Ancestry). While the term ME will be used here to refer to all of these closely related variants, the presentation below follows most closely the description of MESON in [Pla90].

Earlier, we described the members of the ancestry family as procedures which utilize the basic SLD goal reduction operation, with the addition of ancestor cancellation. We should note that describing ME as an extension of SLD-resolution is historically misleading since in fact ME and SL-resolution came first and SLD-resolution was developed as a restriction of SL-resolution. Thus, it would be more precise to say that SLD-resolution utilizes the basic goal reduction operation of SL-resolution (ME) *without* ancestor cancellation. However, SLD-resolution (through the success of Prolog) has become the banner of logic programming, and so will remain our focus. On another historical note, we caution the reader familiar with ME terminology not to confuse the operations of "goal reduction" (as described here) and "reduction" (as described in ME literature). This overloading of the term "reduction" is regrettable, but is driven by our overriding purpose, which is to describe all of the ancestry family procedures in a common, problem-reduction style.

Like all of the procedures in the ancestry family, ME is a sound and complete procedure for reasoning from possibly non-Horn programs. In order to obtain completeness, however, ME requires the use of contrapositives. Informally, a contrapositive of an implication clause is a variant of the clause in which a different literal is chosen as head. The following formalizes this definition.

Definition 3.2 Let $C = L_1 \leftarrow L_2 \wedge \cdots \wedge L_k$ be a clause in implication form (where each L_i is a literal). We denote the contrapositives of C by

$$\text{CONTRA}(C) = \{L_i \leftarrow \neg L_1 \wedge \cdots \wedge \neg L_{i-1} \wedge \neg L_{i+1} \wedge \cdots \wedge \neg L_k \mid 2 \leq i \leq k\}$$

Given a program $P = \{C_1, \ldots, C_n\}$, we denote the contrapositive set for P by

$$\text{CONTRA}(P) = \text{CONTRA}(C_1) \cup \cdots \cup \text{CONTRA}(C_n)$$

Given a program P in implication form, there is an ME goal reduction inference rule for every clause in $P \cup \text{CONTRA}(P)$. In addition, ME incorporates ancestor cancellation, where a goal may be solved (canceled) if it matches the negation of an ancestor goal. In order to localize the concept of an ancestor, we utilize our sequent notation '$\Gamma \rightarrow L$', where L is a literal and Γ is a list of ancestors of L. The goal reduction operation must then collect ancestors in the ancestor list Γ. Actually, to make the cancellation operation clearer and compatible with our notation, we will have the goal reduction operation collect the

negations of ancestors. Cancellation will then occur if a goal matches any literal in the collected (negated) ancestor list.

Goal Reduction Rule:

For each implication clause $H \leftarrow B_1 \wedge \cdots \wedge B_n$ in $P \cup \text{CONTRA}(P)$:

$$\frac{\Gamma \cup \{\neg H\} \rightarrow B_1 \qquad \cdots \qquad \Gamma \cup \{\neg H\} \rightarrow B_n}{\Gamma \rightarrow H}$$

Ancestor Cancellation Axiom:

$$\Gamma \rightarrow L \quad \text{where } L \in \Gamma$$

In addition to the above rules and axiom, one of the clauses in the program is chosen as the support clause. Given the support clause $L_1 \vee \cdots \vee L_k$, we have the start rule

$$\frac{\emptyset \rightarrow \neg L_1 \qquad \cdots \qquad \emptyset \rightarrow \neg L_k}{\emptyset \rightarrow \texttt{FALSE}}$$

Often, the support clause is a negative clause, so all goals $\neg L_i$ are atoms. An ME refutation of a program P is a tree whose root is this start rule, and whose branches are built using the Goal Reduction rules and Ancestor Cancellation axiom.

Example 3.3 Consider the program P from Example 2.2, written in implication form. To the right is CONTRA(P).

$\neg\mathtt{x} \leftarrow \mathtt{y}$	+	$\neg\mathtt{y} \leftarrow \mathtt{x}$
$\mathtt{x} \leftarrow \mathtt{h}_1$	+	$\neg\mathtt{h}_1 \leftarrow \neg\mathtt{x}$
$\mathtt{x} \leftarrow \mathtt{h}_2$	+	$\neg\mathtt{h}_2 \leftarrow \neg\mathtt{x}$
$\mathtt{y}$		
$\mathtt{h}_1 \leftarrow \mathtt{y} \wedge \neg\mathtt{h}_2$	+	$\mathtt{h}_2 \leftarrow \mathtt{y} \wedge \neg\mathtt{h}_1$
	+	$\neg\mathtt{y} \leftarrow \neg\mathtt{h}_1 \wedge \neg\mathtt{h}_2$

The following is an ME refutation of P using the first clause as support clause.

$$\dfrac{\dfrac{\dfrac{}{\{\neg\mathtt{x}, \neg\mathtt{h}_1\} \rightarrow \mathtt{y}} \qquad \dfrac{\{\neg\mathtt{x}, \neg\mathtt{h}_1, \mathtt{h}_2\} \rightarrow \neg\mathtt{x}}{\{\neg\mathtt{x}, \neg\mathtt{h}_1\} \rightarrow \neg\mathtt{h}_2}}{\dfrac{\{\neg\mathtt{x}\} \rightarrow \mathtt{h}_1}{\emptyset \rightarrow \mathtt{x}} \qquad \dfrac{}{\emptyset \rightarrow \mathtt{y}}}}{\emptyset \rightarrow \texttt{FALSE}}$$

Model Elimination has several advantages which make it the subject of widespread study today [SL90, LBSB92, AL91]. It is a linear input procedure, and does not require factoring (although factoring is allowed). It is amenable to efficient implementation, as first demonstrated by Stickel's Prolog Technology Theorem Prover (PTTP) [Sti84]. Also, since it generalizes SLD-resolution, ME shares much of the behavior and simple format of SLD-resolution (and thus Prolog).

Model Elimination does have two main disadvantages. First, the need for ancestor cancellation affects the inner-loop speed. Here, the inner-loop refers to the check for possible ancestor cancellation, then either execution of the cancellation (if applicable) or else execution of an instance of goal reduction. The check for ancestor cancellation can often be optimized by sophisticated implementation techniques such as partitioning the ancestors by predicate and sign. In practice, this can keep the number of ancestors that have to be checked quite short, although it should be noted that applications exist where these techniques are ineffective (e.g., consider theorems with only one predicate letter). On balance, the ancestor check usually is not costly compared to anything that increases the branching factor. Second, the use of contrapositives greatly expands the search space. The augmented program $P \cup \text{CONTRA}(P)$ contains one implication clause for every literal occurrence in the original program P. Thus, the number of clauses available for goal reduction can be quite large. In addition, the use of contrapositives can hurt readability by destroying the procedural reading of clauses. When a user denotes a clause in implication form, say $\texttt{can\_fly} \leftarrow \texttt{is\_bird} \wedge \neg\texttt{injured}$, she often has this particular direction in mind. Utilizing a contrapositive, say $\neg\texttt{is\_bird} \leftarrow \neg\texttt{can\_fly} \wedge \neg\texttt{injured}$, may produce a deduction which is unnatural and unintended by the user.

The variants of ME such as SL-resolution [KK71], strong-ME [Lov78], and SLI-resolution [MZ82] that employ factoring introduce added tradeoffs. Checking for factors adds a cost to inner-loop processing comparable to the ancestor check. The creation of factors adds to the branching factor while sometimes realizing shorter proofs. The experiments that have been done with this (see [FLSY74]) indicate that factoring does not pay on balance, but experimentation is limited.

3.2 Positive Refinement of ME

In [Pla88], Plaisted described a variant of ME which addresses its first main disadvantage: its slower inner-loop speed due to ancestor checking. This variant, called the Positive Refinement of ME, limits ancestor cancellation to positive goals only. Thus, no ancestor checking is necessary for negative goals, and the number of ancestors which must be checked for a positive goal is the number of negative ancestors of that goal.

Using the same sequent-style notation as for ME, we can describe the Positive Refinement as follows. Note that the Goal Reduction rule still considers all contrapositives of program clauses, so the Positive Refinement does not address the second main disadvantage of ME: its expanded search space due to the full use of contrapositives. The Ancestor Cancellation rule is modified to only consider positive literals for cancellation. Correspondingly, only negative goals need be incorporated into the ancestor lists in the Goal Reduction rule.

Goal Reduction Rule:

For each implication clause $H \leftarrow B_1 \wedge \cdots \wedge B_n$ in $P \cup \text{CONTRA}(P)$:

$$\frac{\Gamma' \rightarrow B_1 \quad \cdots \quad \Gamma' \rightarrow B_n}{\Gamma \rightarrow H}$$

where $\Gamma' = \Gamma$ if H is positive, $\Gamma' = \Gamma \cup \{\neg H\}$ if H is negative.

Ancestor Cancellation Axiom:

$$\Gamma \rightarrow L \quad \text{where } L \in \Gamma$$

(Note: L is always positive)

Again, the Positive Refinement utilizes a start rule constructed using a chosen support clause. A refutation in the Positive Refinement is a tree whose root is the start rule, and whose branches are built using the new Goal Reduction rules and Ancestor Cancellation axiom.

Example 3.4 Consider the ME refutation from Example 3.3. The following is the corresponding refutation in the Positive Refinement.

$$\dfrac{\dfrac{\dfrac{}{\emptyset \rightarrow \mathtt{y}} \qquad \dfrac{\dfrac{\dfrac{}{\{\mathtt{h}_2\} \rightarrow \mathtt{y}}}{\{\mathtt{h}_2\} \rightarrow \neg\mathtt{x}}}{\emptyset \rightarrow \neg\mathtt{h}_2}}{\dfrac{\emptyset \rightarrow \mathtt{h}_1}{\emptyset \rightarrow \mathtt{x}}} \qquad \dfrac{}{\emptyset \rightarrow \mathtt{y}}}{\emptyset \rightarrow \mathtt{FALSE}}$$

Comparing this refutation tree with the ME refutation tree in Example 3.3, we first note that ancestor lists in the Positive Refinement do not contain negative literals (corresponding to positive ancestor goals). Subsequently, the cancellation of goal $\neg\mathtt{x}$ in the ME refutation tree is not allowed here. Instead, goal reduction must continue to complete the refutation.

As we mentioned above, for a positive goal the number of ancestor checks required is the number of negative ancestors. In the worst case, this will still be proportional to the refutation depth, as in ME. However, in domains where the amount of non-Horn information is small, the number of negative ancestors will also be small and the savings can be significant. While the Positive Refinement can improve the inner-loop speed of ME by reducing ancestor checking, it should be noted that by disallowing cancellations that are otherwise valid in ME, the Positive Refinement may produce longer refutations (as shown by Example 3.4).

3.3 SLWV-resolution

In [PCA90, PCA93], Pereira, Caires, and Alferes presented a procedure called SLWV-resolution (SL-resolution Without contrapositive clause Variants). This procedure may be viewed as a variant of ME which addresses its second main disadvantage, the use of contrapositives. Unlike ME and its Positive Refinement, SLWV-resolution does not require the contrapositives of clauses. A program clause $L_1 \vee \cdots \vee L_k$ need be represented by only one implication clause, with any literal as head (such as $L_1 \leftarrow \neg L_2 \wedge \cdots \wedge \neg L_k$). While SLWV-resolution was described both in linear and problem-reduction formats, we will consider the latter form since it more directly translates into our sequent notation. SLWV-resolution utilizes the sequent-like notation '$PN\#L$', where PN is a list of ancestor literals and L is a goal (either a literal or a list of literals). The following inference rules and axiom are defined.

Implication Rule:

$$\frac{PN \cup \{L\}\#(B_1, \ldots, B_n)}{PN\#L}\,I$$

where $H \in PN \cup \{L\}$ and $H \leftarrow B_1 \wedge \cdots \wedge B_N \in P$.

And Rule:

$$\frac{PN\#L_1 \qquad PN\#(L_2, \ldots, L_n)}{PN\#(L_1, \ldots, L_n)}\,A$$

Cancellation Axiom:

$$PN\#L \quad \text{where } \neg L \in PN$$

As with ME and the Positive Refinement, in addition to the above rules a start rule is constructed using a chosen support clause. The support clause $L_1 \vee \cdots \vee L_k$ yields the start rule

$$\frac{\emptyset\#(\neg L_1, \ldots, \neg L_k)}{\emptyset\#\texttt{FALSE}}\,S$$

An SLWV-refutation is a tree whose root is the start rule, and whose branches are built using the above rules and axiom.

Example 3.5 Consider the program P from Example 2.2, written in standard implication form. Note that more than one encoding of this program in implication form is possible.

$\neg \mathtt{x} \leftarrow \mathtt{y}$
$\mathtt{x} \leftarrow \mathtt{h}_1$
$\mathtt{x} \leftarrow \mathtt{h}_2$
$\mathtt{y}$
$\mathtt{h}_1 \leftarrow \mathtt{y} \wedge \neg \mathtt{h}_2$

The following is an SLWV refutation of P, using the first clause as support clause.

$$\dfrac{\dfrac{\dfrac{}{\{\mathtt{x},\mathtt{h}_1\}\#\mathtt{y}}I \qquad \dfrac{\dfrac{\{\mathtt{x},\mathtt{h}_1,\neg\mathtt{h}_2\}\#\mathtt{h}_2}{\{\mathtt{x},\mathtt{h}_1\}\#\neg\mathtt{h}_2}I^*}{\{\mathtt{x},\mathtt{h}_1\}\#(\neg\mathtt{h}_2)}A}{\dfrac{\dfrac{\{\mathtt{x},\mathtt{h}_1\}\#(\mathtt{y},\neg\mathtt{h}_2)}{\{\mathtt{x}\}\#\mathtt{h}_1}I \qquad \dfrac{\dfrac{}{\emptyset\#\mathtt{y}}I}{\emptyset\#(\mathtt{y})}A}{\emptyset\#\mathtt{x}}I \qquad \emptyset\#(\mathtt{y})}A^*}{\dfrac{\emptyset\#(\mathtt{x},\mathtt{y})}{\emptyset\#\mathtt{FALSE}}S}$$

Note that in this simple formulation, the Implication rule only applies to the left entry of the And rule, thus requiring extra And rule applications such as at A^*. Special attention should be paid to the step labeled I* in this refutation tree. Here, the ancestor goal $\mathtt{x}$ is used to reduce with the clause $\mathtt{x} \leftarrow \mathtt{h}_2$.

Replacing the symbols 'PN' with 'Γ' and '#' with '$\rightarrow$', we find that the rules of SLWV-resolution bear a striking resemblance to ME in the sequent notation described earlier. The Implication rule (followed by the appropriate number of And rules) is roughly equivalent to the Goal Reduction rule of ME, where (1) the goal in the lower sequent matches the head of a program clause and is replaced by the body of the clause in the upper sequent, and (2) the goal is added to an ancestor list (except in ME, the negation of the goal is added). The interesting feature of the Implication rule is that it also allows goals in the ancestor list PN to match and reduce with program clauses. This essentially allows for the deduction to restart with some ancestor goal (as in the step labeled I* in the above refutation). The And rule of SLWV-resolution is a notational convenience, permitting bodies of reduced clauses to be introduced as a single unit (by the Goal Reduction rule). Finally, the Cancellation axiom corresponds to the Ancestor Cancellation axiom of ME, noting again that the ancestor list PN here stores actual ancestors as opposed to the negations of ancestors as in ME.

Having noted the close relationship between the rules of SLWV-resolution and ME, it is not difficult to translate SLWV-resolution into a sequent notation which closely parallels ME. We will divide the work of the Implication rule into two rule schemas: one which performs the goal reduction operation of ME and one which performs the restart operation described above. Whereas the Implication rule could match an atom in the ancestor list with a program clause and reduce, this operation will now be performed by first restarting

with the ancestor goal and then performing goal reduction. We may dispose of the And rule altogether simply by allowing the Goal Reduction rule to produce multiple sequents directly (instead of producing just one, and relying on the And rule to break up the goals). In order to be consistent with the ME rules, we will store the negation of goals in the ancestor list, so the Cancellation axiom will match that of ME.

Goal Reduction Rule:

For each implication clause $H \leftarrow B_1 \wedge \cdots \wedge B_n$ in P:

$$\frac{\Gamma \cup \{\neg H\} \rightarrow B_1 \qquad \cdots \qquad \Gamma \cup \{\neg H\} \rightarrow B_n}{\Gamma \rightarrow H}\,CR$$

Restart Rule:

$$\frac{\Gamma \cup \{\neg L\} \rightarrow A}{\Gamma \rightarrow L}\,R$$

where $\neg A \in \Gamma$.

Ancestor Cancellation Axiom:

$$\Gamma \rightarrow L \quad \text{where } L \in \Gamma$$

Correspondingly, the start rule will produce multiple sequents as in ME. Thus, the support clause $L_1 \vee \cdots \vee L_k$ yields the start rule

$$\frac{\emptyset \rightarrow \neg L_1 \qquad \cdots \qquad \emptyset \rightarrow \neg L_k}{\emptyset \rightarrow \texttt{FALSE}}\,S$$

Example 3.6 Consider the SLWV refutation from Example 3.5 above. The following is the corresponding refutation in sequent notation.

$$\dfrac{\dfrac{}{\{\neg \mathtt{x}, \neg \mathtt{h}_1\} \rightarrow \mathtt{y}}\,CR \qquad \dfrac{\dfrac{\{\neg \mathtt{x}, \neg \mathtt{h}_1, \mathtt{h}_2\} \rightarrow \mathtt{h}_2}{\{\neg \mathtt{x}, \neg \mathtt{h}_1, \mathtt{h}_2\} \rightarrow \mathtt{x}}\,CR}{\{\neg \mathtt{x}, \neg \mathtt{h}_1\} \rightarrow \neg \mathtt{h}_2}\,R}{\dfrac{\dfrac{\{\neg \mathtt{x}\} \rightarrow \mathtt{h}_1}{\emptyset \rightarrow \mathtt{x}}\,CR \qquad \dfrac{}{\emptyset \rightarrow \mathtt{y}}\,CR}{\emptyset \rightarrow \texttt{FALSE}}\,S}\,CR$$

Note that in this refutation, goal reduction produces an upper sequent for each goal in the clause body, whereas the corresponding goals are obtained in Example 3.5 through the Implication rule and repeated applications of the And rule. Also, the use of the Implication rule where an ancestor goal reduces with a clause (labeled I* in Example 3.5) is performed here by a combination of the Restart rule and the Goal Reduction rule.

Transforming SLWV-resolution and ME into common notation highlights the distinctions between the two procedures. SLWV-resolution does not require the contrapositives of program clauses for goal reduction, but instead has the additional Restart rule. In fact, comparing the refutation in Examples 3.3 and 3.6, it can be seen how the use of the Restart rule can make contrapositives unnecessary. Where the ME refutation utilizes the contrapositive $\neg\mathtt{h}_2 \leftarrow \neg\mathtt{x}$, the SLWV-resolution refutation instead utilizes the Restart rule to return to the ancestor $\mathtt{x}$ and then the Goal Reduction rule to reduce with the original clause $\mathtt{x} \leftarrow \mathtt{h}_2$. The trade-offs between these two procedures then reduce to the advantages/disadvantages of these features. Since SLWV-resolution does not require contrapositives it maintains the order of clauses as written and so retains their natural procedural reading. The lack of contrapositives can also reduce the size of the search space since fewer clauses will be applicable for goal reduction. Conversely, the search space is expanded by the restart mechanism of SLWV-resolution. The restart mechanism is especially costly since it can be applied at any point in the deduction. Furthermore, since one is free to restart with any ancestor goal, the number of possible restarts at a given time is proportional to the refutation depth. This seems to imply a serious degradation of the inner-loop speed as refutation depth increases.

A side issue worth mentioning here is the loss of linearity in SLWV-resolution due to the Restart rule. An appealing property of ME is that each step in a derivation is obtained via goal reduction on a goal in the previous step. This property does not follow in SLWV-resolution, where the Restart rule can essentially return to a previous point in the deduction by restoring an ancestor goal. As we shall see in the following sections, the near-Horn Prolog procedures also lose the linearity property, but are able to achieve a kind of local linearity by restricting the restart mechanism.

3.4 InH-Prolog

Whereas the Positive Refinement may be seen as addressing the first main disadvantage of ME (excessive ancestor checking), and SLWV-resolution may be seen as addressing the second (the use of contrapositives), the near-Horn Prolog procedures may be viewed as compromises which address both. Like the Positive Refinement, the near-Horn Prologs limit the amount of ancestor cancellation necessary. Like SLWV-resolution, they utilize a mechanism for restarting the deduction as an alternative to contrapositives. However, by allowing a limited number of contrapositives the near-Horn Prologs are able to implement a better controlled restart mechanism.

As we described earlier, the near-Horn Prologs were designed to reason from disjunctive logic programs, where a program clause of the form $H_1 \vee \cdots \vee H_m \vee \neg B_1 \vee \cdots \vee \neg B_n$ (where each H_i and B_i is an atom) is written in positive implication form: $H_1 \vee \cdots \vee H_m \leftarrow B_1 \wedge \cdots \wedge B_n$. Again, the atom `FALSE` is added as the head of all-negative clauses. Since a disjunctive clause

is callable by any head, the near-Horn Prologs essentially utilize a contrapositive for each extra disjunctive head (or, equivalently, for each extra positive literal in the original clause). These contrapositives, denoted CONTRA$^+(P)$, are precisely those in CONTRA(P) with positive heads. In general, CONTRA$^+(P)$ is a relatively small subset of CONTRA(P). When P is Horn, CONTRA$^+(P) = \emptyset$, and for near-Horn programs (those containing few non-Horn clauses), the size of CONTRA$^+(P)$ is small.

We will first consider the InH-Prolog variant. Recall that goal reduction in InH-Prolog is similar to SLD goal reduction, where a disjunctive clause is callable by any head and the remaining heads are deferred for later consideration. Essentially, this is goal reduction using implication clauses from $P \cup$ CONTRA$^+(P)$. The deferred heads (corresponding to negative goals in the clauses), however, are not subject to goal reduction or cancellation within a block. Instead, each deferred head results in a restart block, with that head as distinguished active head and `FALSE` as initial goal. Since active heads are promoted deferred heads, cancellation with an active head corresponds to ancestor cancellation with a negative ancestor.

Describing InH-Prolog in our sequent notation, we find that since goal reduction and cancellation are limited to positive goals only, there is no need to collect positive ancestors in the Goal Reduction rule. The restart mechanism of InH-Prolog is described by the Restart rule, which is applicable at a negative goal and introduces goal `FALSE`. Note here that the inheritance of active heads is captured by the Restart rule since the negative goal (corresponding to an active head) is added to the ancestor list and inherited by all sequents above. The operation of canceling a goal with an active head is then described by the Ancestor Cancellation axiom.

Goal Reduction Rule:

For each implication clause $H \leftarrow B_1 \wedge \cdots \wedge B_n$ in $P \cup$ CONTRA$^+(P)$:

$$\frac{\Gamma \rightarrow B_1 \qquad \cdots \qquad \Gamma \rightarrow B_n}{\Gamma \rightarrow H}\,CR$$

(Note: H is always positive)

Restart Rule:

$$\frac{\Gamma \cup \{\neg L\} \rightarrow \texttt{FALSE}}{\Gamma \rightarrow L}\,R$$

where L is negative.

Ancestor Cancellation Axiom:

$$\Gamma \rightarrow L \quad \text{where } L \in \Gamma$$

(Note: L is always positive)

Unlike the earlier systems, the nH-Prologs do not require an additional start rule. Due to the positive implication form of clauses, any unsatisfiable clause set will have a clause with FALSE as a head. The Goal Reduction rule for such a clause may be used at the root of an InH-Prolog refutation, and so the refutation will still have root sequent '$\emptyset \rightarrow$ FALSE'. If desired, a start rule can be constructed for any clause in the same manner as for the other systems, but the additional rule is not needed for completeness.

Example 3.7 Consider the program P from Example 2.2. The only clause in CONTRA$^+(P)$ is an alternate representation of the non-Horn clause.

$$\begin{array}{l} \texttt{FALSE} \leftarrow \texttt{x} \wedge \texttt{y} \\ \texttt{x} \leftarrow \texttt{h}_1 \\ \texttt{x} \leftarrow \texttt{h}_2 \\ \texttt{y} \\ \texttt{h}_1 \leftarrow \texttt{y} \wedge \neg\texttt{h}_2 \qquad + \quad \texttt{h}_2 \leftarrow \texttt{y} \wedge \neg\texttt{h}_1 \end{array}$$

The following is an InH-Prolog refutation of P in sequent notation (corresponding to the linear refutation in Example 2.2).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\{\texttt{h}_2\} \rightarrow \texttt{h}_2}{\{\texttt{h}_2\} \rightarrow \texttt{x}}CR \quad \dfrac{}{\{\texttt{h}_2\} \rightarrow \texttt{y}}CR}{\{\texttt{h}_2\} \rightarrow \texttt{FALSE}}CR}{\emptyset \rightarrow \neg\texttt{h}_2}R \quad \dfrac{}{\emptyset \rightarrow \texttt{y}}CR}{\emptyset \rightarrow \texttt{h}_1}CR \quad \dfrac{}{\emptyset \rightarrow \texttt{y}}CR}{\emptyset \rightarrow \texttt{FALSE}}CR$$

There are several advantages of the InH-Prolog procedure over previous procedures. Since InH-Prolog limits ancestor cancellation to positive goals only, it obtains the fast inner-loop speed of the Positive Refinement. Also, InH-Prolog requires only a limited number of clause contrapositives. The number of contrapositives required is precisely the number of extra heads in disjunctive (implication) clauses, and so is minimal when programs are close to being Horn. While this is more than is required by SLWV-resolution, the extra contrapositives required by InH-Prolog relative to SLWV-resolution seem to be far outweighed by the control obtained over restarts. Whereas SLWV-resolution restarts can occur anywhere in a deduction, and the number of possible restarts depends upon the refutation depth, InH-Prolog restarts occur only at negative goals and always restart with the same goal, the initial query. The restricted restart mechanism of InH-Prolog also implies a kind of local linearity, where within a block (in tree form, a block corresponds to a subtree with root goal FALSE), the desirable property of always reducing with the previous goal holds. It should also be noted that InH-Prolog does maintain the procedural reading of clauses. A disjunctive clause is treated as a multi-entry procedure callable by any head.

3.5 UnH-Prolog

Like InH-Prolog, the UnH-Prolog variant also represents a compromise approach to alleviating the disadvantages of ME. Both variants limit the amount of ancestor cancellation required and introduce a restart mechanism to avoid contrapositives. While InH-Prolog has a simple and controlled restart mechanism, UnH-Prolog utilizes a less controlled restart mechanism with the advantage of a constant inner-loop speed. Instead of always restarting with `FALSE`, UnH-Prolog restart blocks may begin with any positive ancestor of the distinguished active head. However, the fact that UnH-Prolog restart blocks do not inherit active heads implies that checking for cancellation is a constant-time operation.

As was the case for InH-Prolog, UnH-Prolog in our sequent notation requires a Goal Reduction rule for each implication clause in $P \cup \text{CONTRA}^+(P)$, where $\text{CONTRA}^+(P)$ is the set of all contrapositives of P with positive heads. In contrast, here the Goal Reduction rule must collect positive ancestors since these ancestors are needed when selecting a restart goal. We will separate the antecedent of sequents into two lists: the positive ancestor set Δ (whose members are used for restarts) followed by at most one negative ancestor in Γ (used for cancellation). Since the positive ancestors are not to be used for cancellation purposes, there is no reason to negate a positive goal when placing it in the antecedent. Negative goals, which are handled by the Restart rule, are still negated when placed in the antecedent since negative ancestors are used for cancellation. The fact that active heads are not inherited is reflected in the Restart rule where only the most recent negative ancestor (corresponding to the distinguished active head) is retained.

Goal Reduction Rule:

For each implication clause $H \leftarrow B_1 \wedge \cdots \wedge B_n$ in $P \cup \text{CONTRA}^+(P)$:

$$\frac{\Delta \cup \{H\}, \Gamma \rightarrow B_1 \qquad \cdots \qquad \Delta \cup \{H\}, \Gamma \rightarrow B_n}{\Delta, \Gamma \rightarrow H}\, CR$$

(Note: H is always positive)

Restart Rule:

$$\frac{\Delta, \{\neg L\} \rightarrow A}{\Delta, \Gamma \rightarrow L}\, R$$

where L is negative and $A \in \Delta$.

Ancestor Cancellation Axiom:

$$\Delta, \Gamma \rightarrow L \quad \text{where } L \in \Gamma$$

(Note: L is always positive)

As was the case for InH-Prolog, an additional start rule is not needed since an unsatisfiable clause set in positive implication form will have a clause with FALSE as a head. A start rule may be introduced if desired, but it is sufficient to begin the refutation with root sequent '$\emptyset \rightarrow$ FALSE' and apply only those rules listed above.

Example 3.8 Consider the UnH-Prolog refutation from Example 2.4. The following is the corresponding refutation in sequent notation.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{\texttt{FALSE}, \texttt{x}, \texttt{h}_1\}, \emptyset \rightarrow \texttt{y}}CR \quad \dfrac{\dfrac{\dfrac{}{\{\texttt{FALSE}, \texttt{x}, \texttt{h}_1\}, \{\texttt{h}_2\} \rightarrow \texttt{h}_2}CR}{\{\texttt{FALSE}, \texttt{x}, \texttt{h}_1\}, \{\texttt{h}_2\} \rightarrow \texttt{x}}R}{\{\texttt{FALSE}, \texttt{x}, \texttt{h}_1\}, \emptyset \rightarrow \neg\texttt{h}_2}CR}{\{\texttt{FALSE}, \texttt{x}\}, \emptyset \rightarrow \texttt{h}_1}CR \quad \dfrac{}{\{\texttt{FALSE}\}, \emptyset \rightarrow \texttt{y}}CR}{\{\texttt{FALSE}\}, \emptyset \rightarrow \texttt{x}}CR}{\emptyset, \emptyset \rightarrow \texttt{FALSE}}CR$$

At the step labeled R, the positive ancestor x is selected as restart goal. Again, note that positive ancestors are stored directly in the leftmost list of the antecedent, whereas the negations of negative ancestors are stored in the rightmost list.

Like InH-Prolog, UnH-Prolog utilizes only a limited number of contrapositives and maintains the procedural reading of disjunctive clauses. Goal reduction and ancestor cancellation are similarly limited to positive goals only. One (possibly minor) advantage of UnH-Prolog over the other procedures is its constant inner-loop speed due to considering only the most recent negative ancestor for cancellation. Another advantage is possibly shorter proofs due to restarts from a more favorable ancestor than FALSE. While we have witnessed the possibility of shorter proofs, any advantage here is currently hypothetical. The cost of possible shorter proofs is a more complex restart mechanism and an increase in restart possibilities, suggesting increased search. However, it turns out that the number of restart possibilities does not increase the search space swept when the Progressive search strategy is employed [Lov91]. At the first-order level there is a redundancy factor of only two. Thus, the apparent branching factor increase at restart goals is not realized in practice.

4 Summary

As described earlier, procedures in the ancestry family can be seen as extensions of SLD-resolution which utilize ancestor cancellation to perform non-Horn reasoning. The procedures vary along three parameters: (1) the number of clause contrapositives required, (2) the amount of ancestor checking that must occur, and (3) the use (if any) of a Restart rule. Thus, the relationships between these procedures can be summarized in a table with three columns, one for each of these parameters. Table 1 highlights the trade-offs between these related procedures, allowing for straightforward comparisons. For example, the near-Horn Prolog procedures are designed for and appear especially well-suited for programs which contain relatively few non-Horn clauses. In such cases, the number of contrapositives required

by the procedures is small and so the restricted ancestor checking is obtained at minimal cost. Conversely, SLWV-resolution would not appear to be well-suited for near-Horn programs since the small reduction in number of clauses would most likely be overshadowed by the cost of the unrestricted Restart rule.

Table 2 gives for each procedure an upper bound on the number of inference rule applications (either Goal Reduction rule or Restart rule) that are possible at a node in the search tree. This is the key variable for determining the branching factor at each node. As is well-known, the greater the branching factor, the bigger the search to a given depth. We note that excluded from this count is the number of possible Ancestor Cancellation applications. In the propositional case, this need only be one due to the fact that if the goal matches an ancestor, no other (identical) ancestor cancellations need be attempted. At the first-order level, however, the goal may unify with more than one ancestor, introducing further branching (worst case, a different Ancestor Cancellation application for each ancestor).

Although a big branching factor is generally a big handicap to a procedure regarding performance, there are many variables that are beyond the control of a simple analysis based on the parameters we have studied. Most obvious, and often mentioned in this paper, is that devices that increase branching factor also may shorten proofs. Also affecting performance are pruning rules (such as the Cancellation Pruning Rule for the nH-Prologs mentioned earlier) and choice of implementation architectures (such as partitioning the ancestor list). These make a direct translation of the characteristics studied here into performance predictions very dangerous. However, characteristics leading to a large branching factor, and to a lesser extent the devices that lead to slower inner-loop execution, are going to require some offsetting savings. Thus, besides the better understanding of the procedures that the analysis does provide, we think this also provides insight for implementors of any of the procedures considered here.

We remark that thought was given to including results of naive implementations of the procedures developed by the first author to aid his own understanding. Only at the end of our writing this paper did we come to understand that such an inclusion would be a mistake. For many reasons, some cited above, naive implementations can be misleading. The only fair experimental comparisons involve serious implementations that bring in the sophisticated architectures, the pruning rules, and whatever else the procedure allows. This is beyond the scope of this paper.

Procedure	Contrapositives	Cancellation	Restart
ME (MESON, SL, SLI, MEA)	all	any goal	none
Positive Refinement	all	positive goals only	none
SLWV	none	any goal	at any goal, restart with any ancestor
InH-Prolog (SPRF-D, subset of N-Prolog)	limited	positive goals only	at negative goals, restart with `FALSE`
UnH-Prolog	limited	positive goals only (check only most recent negative ancestor)	at negative goals, restart with positive ancestor

Table 1: The Ancestry Family (characterized by three parameters)

Procedure	Possible inference rules
ME (MESON, SL, SLI, MEA)	# of clauses in $P \cup \mathrm{CONTRA}(P)$
Positive Refinement	# of clauses in $P \cup \mathrm{CONTRA}(P)$
SLWV	# of clauses in P + # of ancestors
InH-Prolog (SPRF-D, subset of N-Prolog)	# of clauses in $P \cup \mathrm{CONTRA}^+(P)$ (if goal is positive) 1 (if goal is negative)
UnH-Prolog	# of clauses in $P \cup \mathrm{CONTRA}^+(P)$ (if goal is positive) # of positive ancestors (if goal is negative)

Table 2: Key Variable to Search Branching Factor

References

[AL91] O.L. Astrachan and D.W. Loveland. METEORs: High performance theorem provers for Model Elimination. In R.S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*. Kluwer Academic Publ., Dordrecht, 1991.

[CEFB] R. Caferra, E. Eder, B. Fronhöfer, and W. Bibel. Extension of Prolog through matrix reduction. *Sixth European Conference on Artificial Intelligence*, Pisa, Italy, 1984.

[FLSY74] S. Fleisig, D. Loveland, A. Smiley, and D. Yarmash. An implementation of the Model Elimination proof procedure. *J. ACM*, 21:124–139, 1974.

[Gab85] D.M. Gabbay. N-Prolog: an extension of Prolog with hypothetical implication, Part 2. *J. Logic Programming*, 4:251–283, 1985.

[GR84] D.M. Gabbay and U. Reyle. N-Prolog: an extension of Prolog with hypothetical implication, Part 1. *J. Logic Programming*, 4:319–355, 1984.

[Hil74] R. Hill. LUSH resolution and it completeness. Technical Report DCL Memo 78, Department of Artificial Intelligence, University of Edinburgh, August 1974.

[KK71] R. Kowalski and D. Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2:227–260, 1971.

[Kow74] R. Kowalski. Predicate calculus as a programming language. In *Proc. of the Sixth IFIP Congress*, pages 569–574. North-Holland Publ., 1974.

[LBSB92] R. Letz, S. Bayerl, J. Schumann, and W. Bibel. SETHEO—a high-performance theorem prover. *Journal of Automated Reasoning*, 8:183–212, 1992.

[Lov68] D.W. Loveland. Mechanical theorem proving by model elimination. *J. ACM*, 15:236–251, 1968.

[Lov69] D.W. Loveland. A simplified format for the Model Elimination procedure. *J. ACM*, 16:349–363, 1969.

[Lov78] D.W. Loveland. *Automated Theorem Proving: A Logical Basis*. North-Holland Publ., Amsterdam, 1978.

[Lov87] D.W. Loveland. Near-Horn Prolog. In J. Lassez, editor, *Logic Programming: Proc. of the Fourth Int'l Conf.*, pages 456–469. MIT Press, 1987.

[Lov91] D.W. Loveland. Near-Horn Prolog and beyond. *J. Automated Reasoning*, 7:1–26, 1991.

[LR91] D.W. Loveland and D.W. Reed. A near-Horn Prolog for compilation. In J. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT Press, 1991.

[MZ82] J. Minker and G. Zanon. An extension to linear resolution with selection function. *Information Processing Letters*, 14(3):191–194, 1982.

[PCA90] L. M. Pereira, L. Caires, and J. Alferes. Classical negation for normal logic programs. In *Proc. of the Seventh "Seminário Brasileiro de Inteligência Artificial"*, 1990.

[PCA93] L. M. Pereira, L. Caires, and J. Alferes. SLWV – a theorem prover for logic programming. In *Proc. of the Third Workshop on Extensions of Logic Programming (ELP '92). Lecture Notes in Artificial Intelligence 660*, pages 1–23. Springer-Verlag, Berlin, 1993.

[Pla82] D. Plaisted. A simplified problem reduction format. *Artificial Intelligence*, 18:227–261, 1982.

[Pla88] D. Plaisted. Non-Horn clause logic programming without contrapositives. *J. Automated Reasoning*, 4:287–325, 1988.

[Pla90] D. Plaisted. A sequent style Model Elimination strategy and a positive refinement. *J. Automated Reasoning*, 6(4):389–402, 1990.

[Ree92] D.W. Reed. *A Case-analysis Approach to Disjunctive Logic Programming*. PhD thesis, Duke University, May 1992.

[RL92] D.W. Reed and D.W. Loveland. A comparison of three Prolog extensions. *J. Logic Programming*, 12(1), 1992.

[RLS92] D.W. Reed, D.W. Loveland, and B.T. Smith. A near-Horn approach to disjunctive logic programming. In *Proc. of the Second Workshop on Extensions of Logic Programming (ELP '91). Lecture Notes in Artificial Intelligence 596*, pages 345–369. Springer-Verlag, Berlin, 1992.

[SL73] M.E. Stickel and D.W. Loveland. A hole in goal trees: Some guidance from resolution theory. In *Proc. of the Third Int'l Joint Conf. on Artificial Intelligence*, pages 153–161, 1973. Also in *IEEE Trans. on Computing*, C-25 (April, 1976).

[SL90] J. Schumann and R. Letz. PARTHEO – a high performance parallel theorem prover. In *Proc. of the Tenth Int'l Conf. on Automated Deduction*, pages 40–56, 1990.

[Sti84] M.E. Stickel. A Prolog technology theorem prover. *New Generation Computing*, 2(4):371–383, 1984.

[Wak88] T. Wakayama. Indefinite query answers in deductive knowledge bases. In *Proc. of the Third IASTED Int'l Symposium on Expert Systems Theory and Applications*, 1988.

[Wak89] T. Wakayama. *Reasoning with Indefinite Information in Resolution-Based Languages*. PhD thesis, Syracuse University, 1989.