TR-683

Linear Resolution for Consequence-Finding

by
K. Inoue

September, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Linear Resolution for Consequence-Finding *

Katsumi Inoue

ICOT Research Center
Mita Kokusai Bldg. 21F
1-4-28 Mita, Minato-ku, Tokyo 108, Japan
inoue@icot.or.jp

July 24, 1991

Abstract

Since linear resolution with clause ordering is not complete for consequence-finding, it has been used mainly for proof-finding. In this paper, we re-evaluate consequence-finding based on ordered-linear resolution. Firstly, consequence-finding is generalized to the problem in which only interesting clauses having a certain property (called characteristic clauses) should be found. Then, we show how adding a skip rule to ordered linear resolution makes it complete for consequence-finding in this general sense. The important feature of the proposed method is that it constructs such a subset of consequences directly without testing each generated clause for the required property. This feature is very effective for computing abduction and nonmonotonic reasoning.

Keywords: consequence-finding, linear resolution, prime implicates, abduction

*This is a revised and extended version of a paper [14] that is to appear in *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (IJCAI-91).

Contents

1 Introduction

It is well known in automated deduction that while resolution [38] is complete for proof-finding (called *refutation* complete), that is, it can deduce *false* from every unsatisfiable set of formulas, it is not deductively complete for finding every logical consequence of a given satisfiable set of formulas. For example, resolution cannot derive the formula $q \vee r$ from a set of formulas $\Sigma = \{p, \neg p \vee q\}$ although $\Sigma \models q \vee r$. Lee [21] addresses himself to this problem and defines the *consequence-finding* problem, which is expressed in the following form:

> Given a set of formulas Σ and a resolution procedure P, for any logical consequence T of Σ, can P derive a logical consequence S of Σ such that S subsumes T?

If a resolution procedure is complete for consequence-finding, then it is useful in spite of lacking deductive completeness because in general the logical consequences not deducible from the theory are neither interesting nor useful. Namely, such a formula is subsumed by some formula deducible from the theory and thus it is weak and redundant.

Historically, consequence-finding had been investigated intensively since Robinson invented the resolution principle [38] for proof-finding. Lee's completeness theorem [21] was proved for the original resolution principle. Slagle, Chang and Lee [43] extended the result to various kinds of semantic resolution (including I-semantic resolution, hyperresolution, PI-resolution). However, after Minicozzi and Reiter [26] extended these results to some *linear resolution* strategies in the early 70s, consequence-finding was once abandoned in research of automated theorem proving and attention has been directed towards only proof-finding [1]. It appears that there are three reasons for this discouragement:

1. The results presented by [26] are in some sense negative.

   - Linear resolution involving A-ordering (a total ordering of all the ground atomic formulas of the theory [19, 36]) is complete for consequence-finding only if all possible A-ordering must be tried

[1] One can see that textbooks of resolution-based theorem proving, such as [2, 24], have no sections for consequence-finding.

to generate all possible target theorems. This result is far from obtaining a practical strategy.

- Linear resolution involving *C-ordering* (literals are ordered in each clause in the theory [24, 36]), such as Model Elimination [24] and its variants [20, 2, 41, 27], which is the most familiar and efficient class of resolution procedures because it contains many restriction strategies, is unfortunately incomplete for consequence-finding. Thus, the completeness result that we would most like to obtain does not hold.

2. Even if a resolution procedure is complete for consequence-finding, it is neither practical nor useful to find all of the theorems in general. Previous methods generate all theorems first by using consequence-finding procedures, and then filter them by some given criteria. There has not been an intellectual method which directly searches for only interesting consequences without "generate-and-test" manners.

3. As opposed to proof-finding which can be used, for instance, in planning and synthesis problems where answer extraction techniques are available to obtain useful information, consequence-finding has lacked useful applications in AI.

In this paper, we re-evaluate consequence-finding and give new perspectives. The proposals are motivated by the following recent investigations which appear to relate to the above three problems:

1. Finger gives complete procedures based on set-of-support resolution for generating formulas (called *ramification*) derivable from a theory and a newly added formula as an initial set of support [8, the RGC procedure] as well as for finding residues [8, resolution residue]. However, these procedures do not utilize C-ordering. Finger also gives another procedure based on C-ordered set-of-support resolution [8, ordered residue], but it can be used only for Horn clause theories.

2. Bossu and Siegel [1] propose a complete algorithm for finding the set of positive clauses derivable from a groundable theory (called *characteristic clauses*) by using their saturation algorithm based on A-ordering. Recently, Siegel [42] redefined the notion of characteristic clauses much

more generally for propositional theories, and proposed a complete algorithm for finding them by adding one rule called *skip* operation to C-ordered linear resolution. This algorithm is, although propositional, much more efficient than Bossu and Siegel's saturation algorithm, and achieves the demand on direct search for characteristic clauses.

3. Resolution-based procedures for *abduction* [32, 3, 8, 31, 45] generate explanations of queries, and they actually produces formulas other than the empty clause. In other words, such abductive procedures can utilize consequent-finding procedures [13]. In particular, for the propositional case, Reiter and de Kleer [37] show that an algorithm to compute *prime implicants/implicates* [35, 46] can be utilized for the *clause management system* (CMS) that is a generalization of the ATMS [4]. On the other hand, Przymusinski [34] defines MILO-resolution, which outputs a kind of characteristic clauses to be used in a query answering procedure for *circumscription* of ground theories. MILO-resolution can also be characterized as an instance of C-ordered linear resolution with skip operation [15].

These recent progress will be further expanded in this paper. We will thus give satisfactory solutions to the above three problems:

1. We provide *SOL-resolution*, C-ordered linear resolution with the skip rule, which lift Siegel's procedure [42] for the general case, and its completeness result for consequence-finding. Compared with set-of-support resolution, on which Finger's procedures [8] are based, SOL-resolution contains more restriction strategies and generates fewer clauses to find characteristic clauses.

2. We show that easy modifications of SOL-resolution, where one of operations is preferred than others so as to reduce the search space, can be shown to be applied to broad, more efficient variations of consequence-finding.

3. We show how SOL-resolution can be well applied to generate interesting formulas for abductive and nonmonotonic reasoning. Consequently, the importance of the results presented lies in their applicability to a wide class of AI problems, including diagnosis, design, and planning. In

other words, the methods shed some light on better understanding and implementation of many AI techniques.

Among other possibilities, abduction can be regarded as one of the most important applications of SOL-resolution. Although most of first-order abductive procedures [32, 3, 31, 45, 30] are based on C-ordered linear resolution, no completeness results have been obtained for abductive explanations generated by them. We can get the completeness result for an abductive procedure, by characterizing abduction as characteristic-clause-finding for which SOL-resolution is complete. The abductive completeness is very important and crucial for some applications of abduction. For example, to compute circumscription we can use an abductive procedure, but we require that the procedure has to generate every desired explanation. More recently, Demolombe and Farinãs del Cerro [7] independently provided a complete procedure for a special kind of abduction, where every literal can be hypothesized. However, it uses more primitive style of resolution calculus so that it is hard to connect it with those previous abductive procedures.

An earlier version of this paper has been announced without proof in [14]. Applications of earlier versions of SOL-resolution to computing circumscription and to the CMS/ATMS have been demonstrated in [15, 13].

The present paper is organized as follows. The next section characterizes consequence-finding in a general way, and shows how various AI problems can be well defined by using this notion of characteristic clauses. Section 3 presents the basic procedure which is sound and complete for characteristic-clause-finding, based on C-ordered linear resolution. Efficient but incomplete variations of the basic procedure and their properties are provided in Section 4, where computational complexity is also taken into account.

2 Characterizing Consequence-Finding

We define a *theory* as a set of clauses, which can be identified with a *conjunctive normal form (CNF) formula*. A *clause* is a disjunction (possibly written as a set) of *literals*, each of which is a possibly negated atomic formula. Each variable in a clause is assumed to be universally quantified. For a method converting a formula to this form of theory, see [24]. If Σ is a set of clauses, we mean by $\overline{\Sigma}$ the set formed by taking the negation of each clause in Σ. For example, when $\Sigma = \{ p, \neg q \vee r, \neg s \vee \neg t \}$, $\overline{\Sigma} = \{ \neg p, q \wedge \neg r, s \wedge t \}$. The

empty clause is denoted by $\square$. If C and D are two clauses, $C - D$ denotes a clause whose literals are those in the difference of C and D. For example, when $C = a \vee b \vee c$ and $D = b \vee d$, $C - D = a \vee c$. A clause C is said to *subsume* a clause D if there is a substitution θ such that $C\theta \subseteq D$ and C has no more literals than D [2]. For a set of clauses Σ, by $\mu\Sigma$ or $\mu[\Sigma]$ we mean the set of clauses of Σ not subsumed by any other clause of Σ. Notice that $\mu[\mu\Sigma] = \mu\Sigma$. Σ is *closed under subsumption* if it satisfies $\Sigma = \mu\Sigma$. A clause C is a *theorem*, or a *(logical) consequence* of Σ if $\Sigma \models C$. The set of theorems of Σ is denoted by $Th(\Sigma)$.

2.1 Characteristic Clauses

We use the notion of *characteristic clauses*, which is a generalized notion of logical consequences and helps to analyze computational aspects of many of AI problems. The idea of characteristic clauses was introduced by Bossu and Siegel [1] for evaluating a kind of closed-world reasoning and was later redefined by Siegel [42] for propositional logic. Inoue [13] investigated the properties extensively. The description below is more general than [1, 42, 13] in the sense that the notion is not limited to some special purposes and that it deals with general cases instead of just propositional cases. Also, these notions are independent of implementation; we do not assume any particular resolution procedure in this section. Informally speaking, characteristic clauses are intended to represent "interesting" clauses to solve a certain problem, and are constructed over a sub-vocabulary of the representation language called a *production field*.

Definition 2.1 (1) We denote by $\mathcal{R}$ the set of all predicate symbols in the language. For $R \subseteq \mathcal{R}$, we denote by R^+ (R^-) the positive (negative) occurrences of predicates from R in the language. The set of all atomic formulas is denoted as $\mathcal{A}$ ($= \mathcal{R}^+$), and the set of literals is denoted $\mathcal{L}$ ($= \mathcal{A} \cup \overline{\mathcal{A}} = \mathcal{R}^+ \cup \mathcal{R}^-$).
(2) A *production field* $\mathcal{P}$ is represented by a pair, $\langle L_{\mathcal{P}}, Cond \rangle$, where $L_{\mathcal{P}}$

[2] This definition of subsumption is called *θ-subsumption* in [24]. Unlike in the propositional case, the second condition is necessary if the deletion strategy of subsumption is incorporated in the definition of linear deductions (like Rule 4 in Definition 3.1 in Section 3.1) because a clause implies its factor. For example, $p(y) \vee p(f(z)) \supset p(f(x))$ is valid but $p(f(x))$ should not be deleted in deduction sequences.

(called the *characteristic literals*) is a subset of $\mathcal{L}$, and $Cond$ is a certain condition to be satisfied. When $Cond$ is not specified, $\mathcal{P}$ is just denoted as $\langle L_{\mathcal{P}} \rangle$. The production field $\langle \mathcal{L} \rangle$ is denoted $\mathcal{P}_{\mathcal{L}}$.
(3) A clause C is said to *belong to a production field* $\mathcal{P} = \langle L_{\mathcal{P}}, Cond \rangle$ if every literal in C belongs to $L_{\mathcal{P}}$ and C satisfies $Cond$. The set of theorems of Σ belonging to $\mathcal{P}$ is denoted by $Th_{\mathcal{P}}(\Sigma)$.
(4) A production field $\mathcal{P}$ is *stable* if for any two clauses C and D such that C subsumes D, it holds that if D belongs to $\mathcal{P}$, then C also belongs to $\mathcal{P}$.

Example 2.2 The following are examples of stable production fields.
(1) $\mathcal{P}_1 = \mathcal{P}_{\mathcal{L}}$: $Th_{\mathcal{P}_1}(\Sigma)$ is equivalent to $Th(\Sigma)$.
(2) $\mathcal{P}_2 = \langle \mathcal{A} \rangle$: $Th_{\mathcal{P}_2}(\Sigma)$ is the set of positive clauses implied by Σ.
(3) $\mathcal{P}_3 = \langle \overline{A},$ *size is less than* $k \rangle$ where $A \subseteq \mathcal{A}$: $Th_{\mathcal{P}_3}(\Sigma)$ is the set of negative clauses implied by Σ containing less than k literals all of which belong to $\overline{A}$.

Example 2.3 $\mathcal{P}_4 = \langle \mathcal{A},$ *size is more than* $k \rangle$ is not a stable production field. For example, if $k = 2$ and $p(a), q(b), r(c) \in \mathcal{A}$, then $D_1 = p(a) \vee q(b)$ subsumes $D_2 = p(a) \vee q(b) \vee r(c)$, and D_2 belongs to $\mathcal{P}_4$ while D_1 does not.

Definition 2.4 (Characteristic Clauses) Let Σ be a set of clauses, and $\mathcal{P}$ a production field. The *characteristic clauses of* Σ *with respect to* $\mathcal{P}$ are:

$$Carc(\Sigma, \mathcal{P}) = \mu Th_{\mathcal{P}}(\Sigma) .$$

$Carc(\Sigma, \mathcal{P})$ contains all the unsubsumed theorems of Σ belonging to a production field $\mathcal{P}$ and is closed under subsumption. To see why this notion is a generalization of consequence-finding, let $\mathcal{P}$ be $\mathcal{P}_{\mathcal{L}}$. From the definition of consequence-finding, for any clause $D \in Th(\Sigma)$, a complete procedure P can derive a clause $C \in Th(\Sigma)$ such that C subsumes D. Therefore, P can derive every clause $C' \in \mu Th(\Sigma)$ because C' is not subsumed by any other theorem of Σ. Hence, $Carc(\Sigma, \mathcal{P}_{\mathcal{L}}) = \mu Th(\Sigma)$ have to be contained in the theorems derivable from Σ by using P. Note also that the empty clause $\square$ belongs to every stable production field $\mathcal{P}$, and that if Σ is unsatisfiable, then $Carc(\Sigma, \mathcal{P})$ contains only $\square$. This means that proof-finding is a special case of consequence-finding. Next is a summarizing proposition.

Proposition 2.5 Let Σ be a theory, $\mathcal{P}$ a stable production field. A clause D is a theorem of Σ belonging to $\mathcal{P}$ if and only if there is a clause C in $Carc(\Sigma, \mathcal{P})$ such that C subsumes D. In particular, Σ is unsatisfiable if and only if $Carc(\Sigma, \mathcal{P}) = \{\Box\}$. $\Box$

As we will see later, when new information is added to the theory, it is often necessary to compute newly derivable consequences caused by this new information. For this purpose, consequence-finding is extended to look for such *ramification* of new information.

Definition 2.6 (New Characteristic Clauses) Let Σ be a set of clauses, $\mathcal{P}$ a production field, and F a formula. The *new characteristic clauses of F with respect to Σ and $\mathcal{P}$* are:

$$Newcarc(\Sigma, F, \mathcal{P}) = \mu\,[\,Th_{\mathcal{P}}(\Sigma \cup \{F\}) - Th(\Sigma)\,]\,.$$

In other words, $C \in Newcarc(\Sigma, F, \mathcal{P})$ if:

1. (i) $\Sigma \cup \{F\} \models C$, (ii) C belongs to $\mathcal{P}$, (iii) $\Sigma \not\models C$, and
2. No other clause subsuming C satisfies the above three.

The next three propositions show the connections between the characteristic clauses and the new characteristic clauses. Firstly, $Newcarc(\Sigma, F, \mathcal{P})$ can be represented by the set difference of two sets of characteristic clauses.

Proposition 2.7 A clause is a new characteristic clause of F with respect to Σ and $\mathcal{P}$ if and only if it is a characteristic clause of $\Sigma \cup \{F\}$ but is not a characteristic clause of Σ. Namely,

$$Newcarc(\Sigma, F, \mathcal{P}) = Carc(\,\Sigma \cup \{F\},\, \mathcal{P}) - Carc(\Sigma,\, \mathcal{P})\,.$$

Proof: By Definition 2.6, $Newcarc(\Sigma, F, \mathcal{P}) = \mu\,[\,Th_{\mathcal{P}}(\Sigma \cup \{F\}) - Th(\Sigma)\,]$. Since $Th_{\mathcal{P}}(\Sigma \cup \{F\})$ contains only clauses belonging to $\mathcal{P}$, the definition can be rewritten as:

$$Newcarc(\Sigma, F, \mathcal{P}) = \mu\,[\,Th_{\mathcal{P}}(\Sigma \cup \{F\}) - Th_{\mathcal{P}}(\Sigma)\,]\,.$$

Now, let $X = Th_{\mathcal{P}}(\Sigma \cup \{F\})$ and $Y = Th_{\mathcal{P}}(\Sigma)$. Notice that $Y \subseteq X$. We will prove that $\mu[X - Y] = \mu X - \mu Y$.

Let C be a clause belonging to $\mathcal{P}$ such that $C \in \mu[X-Y]$. Then obviously $C \in X-Y$ and thus $C \in X$. Now assume that $C \notin \mu X$. Then there exists a clause D in μX other than C such that D subsumes C. Since C is not subsumed by any other clause in $X-Y$, it holds that $D \in Y$. By the fact that D subsumes C, $C \in Y$, contradiction. Therefore $C \in \mu X$. Clearly, by $C \notin Y$, $C \notin \mu Y$. Hence, $C \in \mu X - \mu Y$.

Conversely, assume that $C \in \mu X - \mu Y$. Firstly we must prove that $C \in X - Y$. Suppose to the contrary that $C \in Y$. Since $C \notin \mu Y$, there exists in μY a clause D other than C such that D subsumes C. However, as $Y \subseteq X$, $D \in X$, contradicting the fact that $C \in \mu X$. Therefore, $C \in X - Y$. Now assume that $C \notin \mu[X-Y]$. Then, there exists in $X-Y$ a clause D' other than C such that D' subsumes C, again contradicting the fact that $C \in \mu X$. Hence, $C \in \mu[X-Y]$. □

When F is a CNF formula, $Newcarc(\Sigma, F, \mathcal{P})$ can be decomposed into a series of *primitive Newcarc operations* each of whose added new formula is just a single clause.

Proposition 2.8 Let $F = C_1 \wedge \cdots \wedge C_m$ be a CNF formula. Then,

$$Newcarc(\Sigma, F, \mathcal{P}) = \mu\,[\,\bigcup_{i=1}^{m} Newcarc(\Sigma_i, C_i, \mathcal{P})\,],$$

where $\Sigma_1 = \Sigma$, and $\Sigma_{i+1} = \Sigma_i \cup \{C_i\}$, for $i = 1, \ldots, m-1$.

Proof: Notice that in the following proof, for sets, X, Y, and Z, such that $Z \subseteq Y \subseteq X$, $X - Z = (X - Y) \cup (Y - Z)$ holds.

$$\begin{aligned}
& Newcarc(\Sigma, F, \mathcal{P}) \\
= \;& \mu\,[\,Th_{\mathcal{P}}(\Sigma \cup \{C_1, \ldots, C_m\}) - Th(\Sigma)\,] \text{ (by Definition 2.6)} \\
= \;& \mu\,[\,Th_{\mathcal{P}}(\Sigma \cup \{C_1, \ldots, C_m\}) - Th_{\mathcal{P}}(\Sigma)\,] \\
= \;& \mu\,[\,(Th_{\mathcal{P}}(\Sigma \cup \{C_1, \ldots, C_m\}) - Th_{\mathcal{P}}(\Sigma \cup \{C_1, \ldots, C_{m-1}\}) \\
& \cup \cdots \cup (Th_{\mathcal{P}}(\Sigma \cup \{C_1\}) - Th_{\mathcal{P}}(\Sigma))\,] \\
= \;& \mu\,[(Th_{\mathcal{P}}(\Sigma_{m+1}) - Th_{\mathcal{P}}(\Sigma_m)) \cup \cdots \\
& \cdots \cup (Th_{\mathcal{P}}(\Sigma_2) - Th_{\mathcal{P}}(\Sigma_1))\,]
\end{aligned}$$

$$
\begin{aligned}
&= \mu\,[\,\mu[\,Th_{\mathcal{P}}(\Sigma_{m+1}) - Th_{\mathcal{P}}(\Sigma_m)\,] \cup \cdots \\
&\quad \cdots \cup \mu[\,Th_{\mathcal{P}}(\Sigma_2) - Th_{\mathcal{P}}(\Sigma_1)\,]\,] \\
&= \mu\,[\,Newcarc(\Sigma_m, C_m, \mathcal{P}) \cup \cdots \cup Newcarc(\Sigma_1, C_1, \mathcal{P})\,] \\
&= \mu\,[\,\bigcup_{i=1}^{m} Newcarc(\Sigma_i, C_i, \mathcal{P})\,].
\end{aligned}
$$

□

Finally, the characteristic clauses $Carc(\Sigma, \mathcal{P})$ can be expressed by constructively using primitive $Newcarc$ operations. Notice that for any atomic formula p, if $\Sigma \not\models p$, $\Sigma \not\models \neg p$, and $p \vee \neg p$ belongs to some stable production field $\mathcal{P}$, then $p \vee \neg p$ belongs to $Carc(\Sigma, \mathcal{P})$.

Proposition 2.9 (Incremental Construction of the Characteristic Clauses) Let Σ be a set of clauses, C a clause.

$$
\begin{aligned}
Carc(\emptyset, \mathcal{P}) &= \{\, p \vee \neg p \mid p \in \mathcal{A} \text{ and } p \vee \neg p \text{ belongs to } \mathcal{P} \,\}, \\
Carc(\Sigma \cup \{C\}, \mathcal{P}) &= \mu\,[\,Carc(\Sigma, \mathcal{P}) \cup Newcarc(\Sigma, C, \mathcal{P})\,].
\end{aligned}
$$

Proof: The first equation follows immediately from Definition 2.4. Now,

$$
\begin{aligned}
& Carc(\Sigma \cup \{C\}, \mathcal{P}) \\
&= \mu Th_{\mathcal{P}}(\Sigma \cup \{C\}) \\
&= \mu\,[\,Th_{\mathcal{P}}(\Sigma \cup \{C\}) \cup Th_{\mathcal{P}}(\Sigma)\,] \\
&= \mu\,[\,\mu Th_{\mathcal{P}}(\Sigma \cup \{C\}) \cup \mu Th_{\mathcal{P}}(\Sigma)\,] \quad (*) \\
&= \mu\,[\,Carc(\Sigma, \mathcal{P}) \cup Carc(\Sigma \cup \{C\}, \mathcal{P})\,] \\
&= \mu\,[\,Carc(\Sigma, \mathcal{P}) \cup (\,Carc(\Sigma \cup \{C\}, \mathcal{P}) - Carc(\Sigma, \mathcal{P})\,)\,] \\
&= \mu\,[\,Carc(\Sigma, \mathcal{P}) \cup Newcarc(\Sigma, C, \mathcal{P})\,] \text{ (by Proposition 2.7)}
\end{aligned}
$$

Notice that at $(*)$, for two sets, X and Y, $\mu[\,X \cup Y\,] = \mu[\,\mu X \cup \mu Y\,]$ holds. □

Implementation of computation of these consequences depends heavily on which operation between $Carc$ and $Newcarc$ is chosen as the basis: $Carc$ can be taken up as the basic operation in Proposition 2.7, while primitive

Newcarc can be used for Propositions 2.8 and 2.9. The former approach can be seen in Bossu and Siegel's [1] saturation procedure. To derive the new positive theorems, their method should first deduce all the $Carc(\Sigma, \mathcal{P})$ prior to giving $Carc(\Sigma \cup \{F\}, \mathcal{P})$, where $L_{\mathcal{P}}$ are fixed to the ground atoms (see Example 2.2 (2)). On the other hand, Siegel [42] demonstrates that the latter approach outperforms the former to compute the new characteristic clauses. We will discuss this issue later in Section 3.2.

2.2 Applications

We illustrate how the use of the (new) characteristic clauses enables elegant definition and precise understanding of many AI problems.

2.2.1 Propositional Case

In the propositional case, $\mathcal{A}$ is reduced to the set of propositional symbols in the language. The subsumption relation is now very simple: a clause C subsumes D if $C \subseteq D$. A theorem of Σ is called an *implicate* of Σ, and the *prime implicates* [35, 46, 18] of Σ can be defined as:

$$PI(\Sigma) = \mu Th(\Sigma).$$

The characteristic clauses of Σ with respect to $\mathcal{P}$ are the prime implicates of Σ belonging to $\mathcal{P}$. When $\mathcal{P} = \mathcal{P}_{\mathcal{L}}$, it holds that $Carc(\Sigma, \mathcal{P}) = PI(\Sigma)$ [3].

The notion of prime implicants/implicates was originally investigated for uses in minimizing Boolean functions on switching circuits [35]. Computing prime implicates is also an essential task in the ATMS [4] and in its generalization called the *clause management system* (CMS) [37]. The CMS is responsible for finding minimal supports for the queries:

Definition 2.10 [37] Let Σ be a set of clauses and C a clause. A clause S is a *support for C with respect to* Σ if: (i) $\Sigma \models S \cup C$, and (ii) $\Sigma \not\models S$.
A support S for C with respect to Σ is *minimal* if there is no other support S' for C which subsumes S. The set of minimal supports for C with respect to Σ is written $MS(\Sigma, C)$.

[3] The *prime implicants* of a disjunctive normal form formula can be defined in the same manner if the duality of $\wedge$ and $\vee$ is taken into account.

The above definition can be easily extended to handle any formula instead of a clause as a query. Setting the production field to $\mathcal{P}_{\mathcal{L}}$ we see that:

Proposition 2.11 [13] Let F be any formula.

$$MS(\Sigma, F) = Newcarc(\Sigma, \neg F, \mathcal{P}_{\mathcal{L}}).$$

□

When we choose the primitive $Newcarc$ operation as a basic computational task, the above proposition does not require computation of $PI(\Sigma)$. On the other hand, the *compiled* approach [37] takes $PI(\Sigma)$ as input to find $MS(\Sigma, C)$ for any clause C easily as:

Proposition 2.12 [37] Let C be a clause.

$$MS(\Sigma, C) = \mu\{\ P - C \mid P \in PI(\Sigma) \text{ and } P \cap C \neq \emptyset\}.$$

□

In de Kleer's versions of ATMSs [4, 5], there is a distinguished set of *assumptions* $A \subseteq \mathcal{L}$. An ATMS can be defined as a system responsible for finding the negations of all minimal supports for the queries consisting of only literals from $\overline{A}$ [37, 13]. Therefore, the ATMS *label* of a formula F relative to a given theory Σ and A is characterized as

$$L(F, A, \Sigma) = \overline{Newcarc(\Sigma, \neg F, \mathcal{P})}, \quad \text{where } \mathcal{P} = \langle \overline{A} \rangle.$$

In the ATMS, a set of assumptions inconsistent with Σ is called *nogood*. Since $\Sigma \models \neg E$ holds for nogood E, the set of *minimal* nogoods is characterized as

$$NG(A, \Sigma) = \overline{Carc(\Sigma, \mathcal{P})}, \quad \text{where } \mathcal{P} = \langle \overline{A} \rangle.$$

Inoue [13] gives various sound and complete methods for both generating and updating the labels of queries relative to a non-Horn theory and literal assumptions.

2.2.2 Abductive and Nonmonotonic Reasoning

As Reiter and de Kleer [37] pointed out, the task of the CMS/ATMS can be viewed as propositional *abduction*. The abductive characterization of the CMS/ATMS can also be seen in [22, 40, 13]. For general cases, there are many proposals for a logical account of abduction [32, 3, 8, 31, 29, 45, 7], whose task is defined as generation of explanations of a query.

Definition 2.13 Let Σ be a theory, $H \subseteq \mathcal{L}$ (called the *hypotheses*), and G a closed formula. A conjunction E of ground instances of H is an *explanation of G from (Σ, H)* if [4]

1. $\Sigma \cup \{E\} \models G$, and
2. $\Sigma \cup \{E\}$ is consistent.

An explanation E of G is *minimal* if no proper sub-conjunction E' of E satisfies $\Sigma \cup \{E'\} \models G$.
An *extension of* (Σ, H) is the set of logical consequences of $\Sigma \cup \{M\}$ where M is a maximal conjunction of ground instances of H such that $\Sigma \cup \{M\}$ is consistent.

The computation of explanations is based on the observation that Σ, G and E verify

1. $\Sigma \cup \{\neg G\} \models \neg E$, and
2. $\Sigma \not\models \neg E$.

Explanations can thus be obtained by computing the set $Th(\Sigma \cup \{\neg F\}) - Th(\Sigma)$ that belong to the characteristic literals $\overline{H}$. The hypotheses H have their sign changed because we look for the negations of E's. The next proposition characterizes abduction by using the new characteristic clauses.

Proposition 2.14 [15] Let Σ, H and G be the same as Definition 2.13. Let $\mathcal{P} = (\overline{H})$. The set of all minimal explanations of G from (Σ, H) is

$$\overline{Newcarc(\Sigma, \neg G, \mathcal{P})}.$$

[4] This definition is based on Theorist [31, 29], an abductive reasoning system, and deals with *ground* explanations. To get universally quantified explanations, we need to apply the *reverse Skolemization* algorithm [3].

There is an extension of (Σ, H) in which G holds if and only if

$$Newcarc(\Sigma, \neg G, \mathcal{P}) \neq \emptyset .$$

□

Another important problem is to predict formulas that hold in all extensions. When the theory is function-free, this problem is known to be essentially equivalent to *circumscription* [25, 23] under the unique-names assumption (UNA) and the domain-closure assumption (DCA) [9, 10]. In circumscription, the predicate symbols of a theory Σ are divided into three disjoint sets: P, minimized predicates; Z, variables; and Q, fixed predicates. Let $Circum(\Sigma; P; Z)$ be the circumscription of P in Σ with Z. Then, for any formula F, $Circum(\Sigma; P; Z) \models F$ if and only if F is satisfied by every (P, Z)-minimal model M of T [23].

Proving a formula holds in a circumscriptive theory [34, 10], as well as other proof methods for nonmonotonic reasoning formalisms (including explanation-based argument systems [29] and variations of closed-world assumptions [1, 27, 9]), are based on finding explanations of the query, and showing that these explanations cannot be refuted.

Proposition 2.15 [15] Set $\mathcal{P}_{circum} = \langle P^+ \cup Q^+ \cup Q^- \rangle$.
(1) [9] Let F be a formula not containing literals from Z.
$Circum(\Sigma; P; Z) \models F$ if and only if $Newcarc(\Sigma, F, \mathcal{P}_{circum}) = \emptyset$. □
(2) [9, 10] Let F be any formula.
$Circum(\Sigma; P; Z) \models F$ if and only if there is a conjunction G of clauses from $Newcarc(\Sigma, \neg F, \mathcal{P}_{circum})$ such that $Newcarc(\Sigma, \neg G, \mathcal{P}_{circum}) = \emptyset$. □

Example 2.16 Let the theory Σ contain two clauses:

$$\neg bird(x) \vee ab(x) \vee flies(x) ,$$
$$\neg ostrich(x) \vee ab(x) .$$

In this well-known example, the predicates are divided into $P = \{ab\}$, $Z = \{flies\}$, and $Q = \{bird, ostrich\}$. Let $\mathcal{P}$ be $\langle P^+ \cup Q^+ \cup Q^- \rangle$, that is, positive occurrences of *ab*, or any occurrence of *bird* and *ostrich*. Then,

$$Carc(\Sigma, \mathcal{P}) = \{ \neg ostrich(x) \vee ab(x) \}.$$

Let us consider the first query,

$$F_1 = bird(tweety) \supset flies(tweety)\,.$$

Adding $\neg F_1 = bird(tweety) \wedge \neg flies(tweety)$ to Σ gives

$$Newcarc(\Sigma, \neg F_1, \mathcal{P}) = \{\, bird(tweety),\ ab(tweety)\,\}.$$

Since adding to Σ the negation of the conjunction of these two unit clauses, $\neg bird(tweety) \vee \neg ab(tweety)$, gives a new characteristic clause,

$$\neg bird(tweety) \vee \neg ostrich(tweety)\,,$$

it holds that

$$Circum(\Sigma; P; Z) \not\models F_1\,.$$

Now consider the second query:

$$F_2 = bird(sam) \wedge \neg ostrich(sam) \supset flies(sam)\,.$$

Adding the negation of the query to Σ gives

$$Newcarc(\Sigma, \neg F_2, \mathcal{P}) = \{\, bird(sam),\ \neg ostrich(sam),\ ab(sam)\,\}.$$

Adding the negation of the conjunction of these three clauses to Σ produces no new characteristic clauses, showing that

$$Circum(\Sigma; P; Z) \models F_2\,.$$

When a query in abduction or circumscription contains existentially quantified variables, it is sometimes desirable to know for what instances of these variables the query holds. This *answer extraction* problem is considered in [11], where the characteristic clauses of circumscriptive theories play an important role for computation.

2.2.3 Other AI theories

Since we have characterized the prime implicates, the CMS/ATMS, abduction and circumscription, any application area of these techniques can be directly characterized by using the notion of the (new) characteristic clauses:

for instance, constraint satisfaction problems [5], principles of diagnosis [6], synthesis [8, 12] (plan recognition, prediction, design), natural language understanding [45], finding first-order proofs for creating dependency networks [17], and generation of conditional answers in deductive databases [7]. Also, some advanced inference mechanisms such as inductive and analogical reasoning may also take abductive forms of representation [28].

Example 2.17 Here is an illustration of what plan synthesis looks like. To satisfy a goal G, we look for a sequence A of actions that can perform this goal. This problem is in essence the same as abduction: we can compute it by negating each clause in $Newcarc(\Sigma, \neg G, \mathcal{P})$, where Σ is the background theory and $\overline{L_{\mathcal{P}}}$ is the action vocabulary. Then, the obtained plan A should be added to the theory to check whether an unintended effect is caused. For example, to clear $block(a)$ from the table, $\forall x\ clear(x)$ would perform this goal, but this plan will cause unintended side effects. This ramification can be found from $Newcarc(\Sigma, A, \mathcal{P}')$, where $\overline{L_{\mathcal{P}'}}$ is the event vocabulary.

3 Skipping Ordered-Linear Resolution

In this section, we show the basic procedure for implementing the primitive $Newcarc$ operation by using an extension of *C-ordered linear resolution*. By the term C-ordered linear resolution, we mean the family of linear resolution using ordered clauses and the information of literals resolved upon. Examples of C-ordered linear resolution are Model Elimination [24], SL-resolution [20], OL-resolution [2], SLI-resolution [27], and the GC procedure [41]. This family is recognized to be one of the most familiar and efficient classes of resolution for non-Horn theories because it contains several restriction strategies, and has become important in theorem proving for it can be viewed as a predecessor of Prolog's SLD(NF)-resolution. PTTP [44] is a notable Model Elimination theorem prover and is extremely fast.

The important feature of the procedure presented in Section 3.1 is that it is *direct*, namely it is both sensitive to the given added clause to the theory and restricted to searching only characteristic clauses. While both $Newcarc(\Sigma, F, \mathcal{P})$ for a CNF-formula F and $Carc(\Sigma, \mathcal{P})$ can be computed by using primitive $Newcarc$ operations as seen in Propositions 2.8 and 2.9, we will show in Sections 3.2 and 3.3 that these can also be computed directly by using the basic procedure.

3.1 SOL-Resolution

Given a theory Σ, a stable production field $\mathcal{P}$ and a clause C, we show a procedure to compute $Newcarc(\Sigma, C, \mathcal{P})$, which is an extension of C-ordered linear resolution. There are two reasons why C-ordered linear resolution is, among other strategies, useful for computing the new characteristic clauses:

1. A newly added single clause C can be treated as the *top clause* of a linear deduction. This is a desirable feature for consequence-finding since the procedure can directly derive the theorems relevant to the added information.

2. It is easy to achieve the requirement that the procedure should focus on producing only those theorems that belong to $\mathcal{P}$. This is implemented by allowing the procedure to *skip* the selected literals belonging to $\mathcal{P}$ in a linear deduction. The computational superiority of the proposed technique compared to set-of-support resolution that is used by Finger's resolution residue [8], apart from the fact that C-ordered linear resolution contains more restriction strategies in natural ways, comes from this relevancy notion of directing search to $\mathcal{P}$.

Some procedures are known to perform this computation for restricted theories. For *propositional* theories, Siegel [42] proposes a complete algorithm by adding skip operation to SL-resolution [20]. For *ground* theories with a particular production field for circumscription (described in Proposition 2.15), Inoue and Helft [15] point out that Przymusinski's MILO-resolution [34], an extension of Chang and Lee's [2] OL-resolution, can be viewed as C-ordered linear resolution with skip operation.

The following proposed inference system called *SOL (Skipping Ordered Linear) resolution* is a kind of generalization of [34, 42], again in the sense that produced clauses are not limited to some special purposes and that it deals with general cases. The description below is mainly based on terminology of OL-resolution [2], but the result is not restricted to its extension [5]. An *ordered* clause is a sequence of literals possibly containing *framed literals* which represents literals that have been resolved upon: from a clause C an

[5] Historically, OL-resolution was derived from the Model Elimination procedure, most fully described in [24]. The description of OL-resolution in [2] contains an error for the non-ground case discussed later.

ordered clause $\vec{C}$ is obtained just by ordering the elements of C; conversely, from an ordered clause $\vec{C}$ a clause C is obtained by removing the framed literals and converting the remainder to the set. We assume throughout this paper that literals are ordered from left to right in each clause. A *structured* clause $\langle P, \vec{Q} \rangle$ is a pair of a clause P and an ordered clause $\vec{Q}$, whose clausal meaning is $P \cup Q$.

Definition 3.1 (SOL-Deduction) Given a theory Σ, a clause C, and a production field $\mathcal{P}$, an *SOL-deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$* consists of a sequence of structured clauses, $D_0, D_1, \ldots, D_n$, such that:

1. $D_0 = \langle \Box, \vec{C} \rangle$.
2. $D_n = \langle S, \Box \rangle$.
3. For each $D_i = \langle P_i, \vec{Q}_i \rangle$, $P_i \cup Q_i$ is not a tautology.
4. For each $D_i = \langle P_i, \vec{Q}_i \rangle$, Q_i is not subsumed by any Q_j, where $D_j = \langle P_j, \vec{Q}_j \rangle$ is a previous structured clause, $j < i$.
5. $D_{i+1} = \langle P_{i+1}, \vec{Q_{i+1}} \rangle$ is generated from $D_i = \langle P_i, \vec{Q}_i \rangle$ according to the following steps:
   (a) Let l be the *left-most* literal of $\vec{Q}_i$. P_{i+1} and $\vec{R_{i+1}}$ are obtained by applying either of the rules:
      i. **(Skip)** If $P_i \cup \{l\}$ belongs to $\mathcal{P}$, then $P_{i+1} = P_i \cup \{l\}$ and $\vec{R_{i+1}}$ is the ordered clause obtained by removing l from $\vec{Q}_i$.
      ii. **(Resolve)** If there is a clause B_i in Σ such that $\neg k \in B_i$ and l and k are unifiable with mgu θ, then $P_{i+1} = P_i\theta$ and $\vec{R_{i+1}}$ is an ordered clause obtained by concatenating $\vec{B_i}\theta$ and $\vec{Q_i}\theta$, framing $l\theta$, and removing $\neg k\theta$.
      iii. **(Reduce)** If either
         A. (*factoring*) P_i or $\vec{Q}_i$ contains an unframed literal k either different from l or another occurrence of l, or
         B. (*ancestry*) $\vec{Q}_i$ contains a framed literal $\boxed{\neg k}$,

      and l and k are unifiable with mgu θ, then $P_{i+1} = P_i\theta$ and $\vec{R_{i+1}}$ is obtained from $\vec{Q_i}\theta$ by deleting $l\theta$.

(b) $\vec{Q_{i+1}}$ is obtained from $\vec{R_{i+1}}$ by deleting every framed literal not preceded by an unframed literal in the remainder (*truncation*).

Remarks. (1) At Rule 5a, we can choose the *selected literal* l with more liberty like SL-resolution [20] or SLI-resolution [27].

(2) Rule 4 is included for efficiency. It does not affect the completeness described below. This strategy of deleting subsumed clauses can be partially implemented by the following restriction rule: when $\vec{Q_i}$ can be resolved against an ordered clause $\vec{B_i}$ from Σ with mgu θ, if $B_i\theta$ contains a literal $k\theta$ that appears in $\vec{Q_i}\theta$ as a framed literal $\boxed{k\theta}$ at the right of the literal resolved upon, then this resolution can be avoided, as **Resolve** in such a case would result in a clause subsumed by some previous clause in the deduction. The corresponding deletion rule for proof-finding is overlooked in the definition of OL-deduction [2] (and so is in MILO-resolution [34]), but is clearly present in Model Elimination [24] (and in Siegel's algorithm [42]).

(3) When the given production field $\mathcal{P}$ is in the form of $\langle L_{\mathcal{P}} \rangle$, factoring (5(a)iiiA) can be omitted in intermediate deduction steps like Weak Model Elimination [24]. In this case, Rules 3 and 4 are omitted, and factoring is performed at the final step, namely it is taken into account only for P_i in a structured clause of the form $\langle P_i, \Box \rangle$.

(4) At Rule 5a, the selection of rules 5(a)i, 5(a)ii and 5(a)iii must be nondeterministic; for $l \in L_{\mathcal{P}}$ any rule may be applied. This is not a straightforward generalization of MILO-resolution or Siegel's algorithm, because they do not deal with **Reduce** as an alternative choice of other two rules, but make $\vec{Q_{i+1}}$ as the reduced ordered clause of the ordered factor of $\vec{R_{i+1}}$ that is obtained by **Skip** or **Resolve** [6]. Both Przymusinski and Siegel claim that the lifting lemma should work for their procedures. Unfortunately, their claims are wrong: this simpler treatment violates the completeness described below. Furthermore, even if we don't consider consequence-finding, OL-resolution [2], which also handles the ancestry rule as a subsequent rule of **Resolve**, is incomplete for proof-finding. For example, when the theory is given as

$$\Sigma = \{ \begin{array}{ll} p(a) \vee p(x) \vee \neg q(x), & (1) \\ \neg p(b), & (2) \\ q(b) & (3) \ \}, \end{array}$$

[6] Furthermore, MILO-resolution prefers **Skip** to **Resolve**. See also Section 4.2.

it is easy to see that $\Sigma \models p(a)$. However, there is no OL-refutation from $\Sigma + \neg p(a)$:

(4)	$\underline{\neg p(a)}$	given top clause
(5)	$\underline{p(x)} \vee \neg q(x) \vee \boxed{\neg p(a)}$	resolution with (1)
(6)	$\underline{\neg q(a)} \vee \boxed{\neg p(a)}$	reduction

Here, each underlined literal denotes a selected literal in the next step. The clause (6) is the dead-end of the OL-deduction. Hence, unless the substitution θ is empty, i.e., the literal l and the ancestor goal $\neg k$ are exactly complementary (*merge*), the reduction rule must be an alternative choice to other rules [7]. Model Elimination and SL-resolution deal with the reduction rule as an alternative.

The **Skip** rule (5(a)i) reflects the following operational interpretation of a *stable* production field $\mathcal{P}$. By Definition 2.1 (4), assuming that a clause C subsumes a clause D, if D belongs to $\mathcal{P}$, then so does C. In other words, if C does not belong to $\mathcal{P}$, then D does not belong to $\mathcal{P}$ either. Now, consider an SOL-deduction, $D_0, \ldots, D_n$. For each $D_i = \langle P_i, \vec{Q}_i \rangle$ $(0 \leq i \leq n-1)$ and for any $D_j = \langle P_j, \vec{Q}_j \rangle$ $(i < j)$, we see that P_i subsumes P_j (because P_i has no more literal than P_i, and P_j contains an instance of P_i as its subdisjuncts). This implies that if P_i does not belong to $\mathcal{P}$, neither does P_j. That is why the condition that $P_i \cup \{l\}$ belongs to $\mathcal{P}$ is contained in the **Skip** rule. This simple condition contributes to reducing the search space as follows. If no rule can be applied to the selected literal l of a structured clause D_i, the branch is immediately pruned; if **Skip** was applied nevertheless, any resultant sequence would not succeed, thus making unnecessary computation [8].

In addition to the above advantage by using the information on the production field $\mathcal{P}$ against the selected literal at Rule 5a, C-ordering itself is very important for reducing the search space. For a very simple example, suppose that we are to deal with the structured clause $\langle P, \vec{Q} \rangle$, where $Q = a \vee b \vee \cdots \vee c$

[7] The incompleteness of MILO-resolution for first-order theories leads to the unsoundness as well as the incompleteness of the query-answering algorithm for circumscription. This observation has been reported in [16].

[8] We should also mention that the **Skip** rule can be applied to other, superior versions of C-ordered linear resolution, such as Shostak's GC procedure [41], and further improvements on these methods can be used to improve efficiency still more.

and $P \cup \{a\}$ does not belong to $\mathcal{P}$. If the selected literal a can be neither reduced nor resolved upon against clauses of the theory in such a way that the result of the deduction produces a clause belonging to $\mathcal{P}$, SOL-deduction will never try to examine the next literal b. On the other hand, the set-of-support strategy conventionally does not use C-ordering that gives priority to a over b and therefore will try all the possible operations on b as well, making unnecessary computation. This kind of case will happen in many situations.

We call the process of finding SOL-deductions *SOL-resolution*. For SOL-resolution, the following theorem can be shown to hold.

Theorem 3.2 (Soundness and Completeness of SOL-Resolution)
(1) *Soundness*: If a clause S is derived using an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$, then S belongs to $Th_{\mathcal{P}}(\Sigma \cup \{C\})$.
(2) *Completeness*: If a clause T *does not belong to* $Th_{\mathcal{P}}(\Sigma)$, *but* belongs to $Th_{\mathcal{P}}(\Sigma \cup \{C\})$, then there is an SOL-deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that S subsumes T.

Proof: See Appendix for the proof. □

The soundness theorem guarantees that every produced clause belongs to the given production field so that we can avoid "generate-and-test" manners. Recall that C-ordered linear resolution is refutation-complete as shown, for example, by [24], but is incomplete for consequence-finding [26]. Theorem 3.2 (2) says that SOL-resolution is complete for characteristic-clause-finding, and thus complete for consequence-finding when $\mathcal{P}$ is $\mathcal{P}_{\mathcal{L}}$ because it includes the additional skipping operation.

Example 3.3 Suppose that the theory Σ and the clause C are given by

$$\begin{aligned} \Sigma = \{ \quad & \neg c \vee \neg a \quad (1)\,, \\ & \neg c \vee \neg b \quad (2) \quad \}\,, \\ C = \quad & a \vee b\,. \end{aligned}$$

There is no OL-deduction of $\neg c$ from $\Sigma + C$, but $\neg c$ is derived using an SOL-deduction from $\Sigma + C$ and $\mathcal{P}_{\mathcal{L}}$ as:

(3)	$\langle\, \square\,,\ \underline{a} \vee b \,\rangle$	given top clause
(4)	$\langle\, \square\,,\ \underline{\neg c} \vee \boxed{a} \vee b \,\rangle$	resolution with (1)
(5)	$\langle\, \neg c\,,\ \boxed{\not{a}} \vee \underline{b} \,\rangle$	skip and truncation
(6)	$\langle\, \neg c\,,\ \underline{\neg c} \vee \boxed{b} \,\rangle$	resolution with (2)
(7)	$\langle\, \neg c\,,\ \boxed{\not{b}} \,\rangle$	merge and truncation

Note that an OL-deduction would stop at (4).

3.2 Computing New Characteristic Clauses

We now show how exactly $Newcarc(\Sigma, F, \mathcal{P})$ for a formula F can be computed by using SOL-resolution. In contrast to incremental saturation procedures (such as Bossu and Siegel [1]), our approach is not a naive implementation of Definition 2.6 or Proposition 2.7 that constructs the both saturated sets, $Carc(\Sigma, \mathcal{P})$ and $Carc(\Sigma \cup \{C\}, \mathcal{P})$. Firstly, we define a set of clauses derivable by SOL-resolution. The following definition is similar to definitions of MILO-derivatives [34] and supports for negation [27].

Definition 3.4 Given a set of clauses Σ, a clause C, and a stable production field $\mathcal{P}$, let us denote by $\Delta(\Sigma, C, \mathcal{P})$ the clauses deduced by using SOL-deductions from $\Sigma + C$ and $\mathcal{P}$, that is,

$$\Delta(\Sigma, C, \mathcal{P}) = \{\, S \mid S \text{ is derived using an SOL-deduction from } \Sigma + C \text{ and } \mathcal{P} \}.$$

The *production from* $\Sigma + C$ *and* $\mathcal{P}$ is:

$$Prod(\Sigma, C, \mathcal{P}) = \mu\, \Delta(\Sigma, C, \mathcal{P}).$$

Note that $Newcarc(\Sigma, C, \mathcal{P})$ may contain an infinite number of clauses. To compute them practically, we have to restrict theories to decidable subset of first-order logic and limit ourselves to finite domains.

The next proposition shows how the primitive $Newcarc(\Sigma, C, \mathcal{P})$ for a single clause C can be computed, by checking for each clause $S \in Prod(\Sigma, C, \mathcal{P})$, only whether $\Sigma \models S$ or not. Typically, $Prod(\Sigma, C, \mathcal{P})$ contains much less clauses than $Carc(\Sigma \cup \{C\}, \mathcal{P})$ so that the approach outperforms incremental saturation procedures.

Proposition 3.5 Let C be a clause.

$$Newcarc(\Sigma, C, \mathcal{P}) = Prod(\Sigma, C, \mathcal{P}) - Th_{\mathcal{P}}(\Sigma)\,.$$

Proof: By Theorem 3.2 (1), it is easy to see that

$$Prod(\Sigma, C, \mathcal{P}) \subseteq Th_{\mathcal{P}}(\Sigma \cup \{C\})\,.$$

Assume that $T \in Newcarc(\Sigma, C, \mathcal{P})$. Then, $T \in \mu[Th_{\mathcal{P}}(\Sigma \cup \{C\}) - Th_{\mathcal{P}}(\Sigma)]$. By Theorem 3.2 (2), there is a clause S in $Prod(\Sigma, C, \mathcal{P})$ such that S subsumes T. Since T is not subsumed by any other such a clause, $T = S$. Therefore, $T \in Prod(\Sigma, C, \mathcal{P})$. Hence,

$$Newcarc(\Sigma, C, \mathcal{P}) \subseteq Prod(\Sigma, C, \mathcal{P})\,.$$

It remains to show that $Prod(\Sigma, C, \mathcal{P}) - Newcarc(\Sigma, C, \mathcal{P}) \subseteq Th_{\mathcal{P}}(\Sigma)$. Suppose to the contrary, for $S \in Prod(\Sigma, C, \mathcal{P}) - Th_{\mathcal{P}}(\Sigma)$, that $S \notin Newcarc(\Sigma, C, \mathcal{P})$. As $S \notin Th_{\mathcal{P}}(\Sigma)$, $S \notin Carc(\Sigma, \mathcal{P})$. Because $S \in Prod(\Sigma, C, \mathcal{P})$, it must be that $S \in Th_{\mathcal{P}}(\Sigma \cup \{C\})$. Since $S \notin Carc(\Sigma \cup \{C\}, \mathcal{P})$ by the supposition and Proposition 2.7, there is a clause S' other than S in $Carc(\Sigma \cup \{C\}, \mathcal{P})$ such that S' subsumes S. By the stability of $\mathcal{P}$, clearly $S' \notin Th_{\mathcal{P}}(\Sigma)$ as S' subsumes S. Thus, $S' \in Th_{\mathcal{P}}(\Sigma \cup \{C\}) - Th_{\mathcal{P}}(\Sigma)$. By Theorem 3.2 (2), there must be a clause S'' in $Prod(\Sigma, C, \mathcal{P})$ such that S'' subsumes S'. However, by Definition 3.4, $Prod(\Sigma, C, \mathcal{P})$ is closed under subsumption, contradiction. Hence, $S \in Newcarc(\Sigma, C, \mathcal{P})$. □

Proposition 3.5 says the primitive $Newcarc(\Sigma, C, \mathcal{P})$ is contained in the production from $\Sigma + C$ and $\mathcal{P}$. To remove the clauses in the production derivable from Σ, we have to test whether a clause S, produced from $\Sigma + C$ and $\mathcal{P}$, belongs to $Th_{\mathcal{P}}(\Sigma)$ or not. This test is a counterpart of checking consistency of hypotheses with a theory in abduction and is undecidable in general. The question is how effectively this consistency checking can be performed in decidable cases. We already know that S belongs to $\mathcal{P}$. A direct implementation is to use proof-finding property provided by Proposition 2.5: $\Sigma \models S$ if and only if $Prod(\Sigma, \neg S, \mathcal{P}) = \{\Box\}$[9]. In this case, since the only

[9]Here, we assume that Σ is satisfiable. Each variable (universally quantified) in S is replaced by a new constant in $\neg S$ (Skolemization).

target clause produced from $\Sigma + \neg S$ is $\Box$, the production field $\mathcal{P}$ can be replaced with $\langle \emptyset \rangle$ so that **Skip** (Rule 5(a)i) will never be applied: there is a C-ordered linear refutation from $\Sigma \cup \{\neg S\}$ if and only if there is an SOL-deduction from $\Sigma + \neg S$ and $\langle \emptyset \rangle$ (see Corollary A.4).

However, there is another way for consistency checking. When the characteristic literals $L_{\mathcal{P}}$ is small compared with the whole literals $\mathcal{L}$, the computation of $Carc(\Sigma, \mathcal{P})$ can be performed better as the search focuses on $\mathcal{P}$. Having $Carc(\Sigma, \mathcal{P})$, consistency checking is much easier because the check can be reduced to subsumption tests on $Carc(\Sigma, \mathcal{P})$ by Proposition 2.5: $S \in Th_{\mathcal{P}}(\Sigma)$ if and only if there is a clause $T \in Carc(\Sigma, \mathcal{P})$ such that T subsumes S [10]. We thus propose an intermediate approach between incremental saturation procedures based on Proposition 2.7 and an interpreted computation given by Proposition 3.5. It does not compute the saturated set $Carc(\Sigma \cup \{C\}, \mathcal{P})$, but computes and keeps another smaller saturated set $Carc(\Sigma, \mathcal{P})$ for consistency checking. This checking can be embedded into an SOL-deduction by adding the following rule into Definition 3.1 [11]:

4^+. For each $D_i = \langle P_i, \vec{Q}_i \rangle$, P_i is not subsumed by any clause of $Carc(\Sigma, \mathcal{P})$.

Let us denote by $\Delta_+(\Sigma, C, \mathcal{P})$ the clauses deduced by using SOL-deductions from $\Sigma + C$ and $\mathcal{P}$ in which Rule 4^+ is incorporated. The *+-production from* $\Sigma + C$ *and* $\mathcal{P}$ is defined as:

$$Prod_+(\Sigma, C, \mathcal{P}) = \mu\, \Delta_+(\Sigma, C, \mathcal{P}).$$

Proposition 3.6 $Prod_+(\Sigma, C, \mathcal{P}) = Newcarc(\Sigma, C, \mathcal{P})$.

Proof: As we have mentioned earlier, for an SOL-deduction, $D_0, \ldots, D_n$, for any $D_i = \langle P_i, \vec{Q}_i \rangle$ $(0 \leq i \leq n-1)$ and $D_j = \langle P_j, \vec{Q}_j \rangle$ $(i < j)$, it holds that P_i subsumes P_j. Therefore, if a clause $T \in Carc(\Sigma, \mathcal{P})$ subsumes P_i, then T subsumes P_j.

[10] The role of $Carc(\Sigma, \mathcal{P})$ in this case is similar to the minimal *nogoods* in the ATMS [4].

[11] This rule enables SOL-resolution to check deducibility from Σ of each partial clause P_i consisting of literals skipped so far. Instead of using the "compiled" approach [37], i.e., checking with $Carc(\Sigma, \mathcal{P})$, the check can be carried out in a demand driven manner. For P_i such that $\Sigma \not\models P_i$, if l is skipped and added to P_i, we may check whether $\Sigma \not\models P_i \cup \{l\}$ or not by failing to find a proof of l from $\Sigma \cup \{\neg P_i\}$. For Theorist [31], Poole [30] uses a Prolog meta-interpreter for incremental consistency checking, and Satter, Goodwin and Goebel [39] utilize nogoods that have been found in previous failures of consistency checking.

The incorporation of Rule 4^+ into SOL-resolution thus prevents SOL-deductions producing clauses subsumed by some clauses in $Carc(\Sigma, \mathcal{P})$, but it does not affect the completeness result given in Theorem 3.2 (2) (since the completeness is not concerned with clauses belonging to $Th_{\mathcal{P}}(\Sigma)$). Hence, by Proposition 3.5, the proposition follows. □

Example 3.7 Let us consider label computation for a non-Horn ATMS. Suppose that the assumptions are $H = \{ x, \neg y \}$, and the theory is

$$\Sigma = \{ \begin{array}{l} \neg a \vee \neg b \vee c\,, \\ \neg x \vee \neg b \vee a\,, \\ y \vee b \vee c \end{array} \}\,.$$

In this case, $Carc(\Sigma, \langle \overline{H} \rangle) = \emptyset$. The following finds c's label $\{ x \wedge \neg y \}$:

$\langle \Box\,,\ \underline{\neg c} \rangle$	top clause
$\langle \Box\,,\ \underline{\neg a} \vee \neg b \vee \boxed{\neg c} \rangle$	resolution
$\langle \Box\,,\ \underline{\neg x} \vee \not{\neg b} \vee \boxed{\neg a} \vee \neg b \vee \boxed{\neg c} \rangle$	resolution and merge
$\langle \neg x\,,\ \boxed{\not{\neg a}} \vee \underline{\neg b} \vee \boxed{\neg c} \rangle$	skip and truncation
$\langle \neg x\,,\ \underline{y} \vee \not{\neg c} \vee \boxed{\neg b} \vee \boxed{\neg c} \rangle$	resolution and ancestry
$\langle \neg x \vee y\,,\ \boxed{\not{\neg b}} \vee \boxed{\not{\neg c}} \rangle$	skip and truncation

For a CNF formula F, $Newcarc(\Sigma, F, \mathcal{P})$ can be computed incrementally by using a series of SOL-deductions as follows.

Proposition 3.8 Let $F = C_1 \wedge \cdots \wedge C_m$ be a CNF formula. Then,

$$Newcarc(\Sigma, F, \mathcal{P}) = \mu [\bigcup_{i=1}^{m} Prod(\Sigma_i, C_i, \mathcal{P})] - Th_{\mathcal{P}}(\Sigma)\,,$$

where $\Sigma_1 = \Sigma$, and $\Sigma_{i+1} = \Sigma_i \cup \{C_i\}$, for $i = 1, \ldots, m-1$.

Proof: By Proposition 2.8,

$$Newcarc(\Sigma, F, \mathcal{P}) = \mu [\bigcup_{i=1}^{m} Newcarc(\Sigma_i, C_i, \mathcal{P})]\,.$$

Take any union of two successive primitive *Newcarc* operations in the above equation. By applying Proposition 3.5, we get the following equation for such a union ($1 \leq k \leq m-1$).

$$
\begin{aligned}
& Newcarc(\Sigma_{k+1}, C_{k+1}, \mathcal{P}) \cup Newcarc(\Sigma_k, C_k, \mathcal{P}) \\
= \; & (\, Prod(\Sigma_k \cup \{C_k\}, C_{k+1}, \mathcal{P}) - Th_{\mathcal{P}}(\Sigma_k \cup \{C_k\})\,) \\
& \cup (\, Prod(\Sigma_k, C_k, \mathcal{P}) - Th_{\mathcal{P}}(\Sigma_k)\,) \\
= \; & (\, Prod(\Sigma_k \cup \{C_k\}, C_{k+1}, \mathcal{P}) \cup Prod(\Sigma_k, C_k, \mathcal{P})\,) \\
& - (\, Th_{\mathcal{P}}(\Sigma_k \cup \{C_k\}) - Prod(\Sigma_k, C_k, \mathcal{P})\,) \\
& - (\, Th_{\mathcal{P}}(\Sigma_k) - Prod(\Sigma_k \cup \{C_k\}, C_{k+1}, \mathcal{P})\,) \\
& - (\, Th_{\mathcal{P}}(\Sigma_k \cup \{C_k\}) \cap Th_{\mathcal{P}}(\Sigma_k)\,) \\
= \; & (\, Prod(\Sigma_{k+1}, C_{k+1}, \mathcal{P}) \cup Prod(\Sigma_k, C_k, \mathcal{P})\,) - \emptyset \\
& - (\, Th_{\mathcal{P}}(\Sigma_k) - Prod(\Sigma_k \cup \{C_k\}, C_{k+1}, \mathcal{P})\,) - Th_{\mathcal{P}}(\Sigma_k) \\
= \; & (\, Prod(\Sigma_{k+1}, C_{k+1}, \mathcal{P}) \cup Prod(\Sigma_k, C_k, \mathcal{P})\,) - Th_{\mathcal{P}}(\Sigma_k)\,.
\end{aligned}
$$

The above equation can be used successively and extended to prove the Proposition. □

As in the case of Proposition 3.6 for the primitive *Newcarc* operation, the consistency checking for the new characteristic clauses of a CNF formula in Proposition 3.8 can be embedded into SOL-deductions by adding Rule 4^+.

Corollary 3.9 Let $F = C_1 \wedge \cdots \wedge C_m$ be a CNF formula. Then,

$$Newcarc(\Sigma, F, \mathcal{P}) = \mu\,[\, \bigcup_{i=1}^{m} Prod_+(\Sigma_i, C_i, \mathcal{P})\,]\,,$$

where $\Sigma_1 = \Sigma$, and $\Sigma_{i+1} = \Sigma_i \cup \{C_i\}$, for $i = 1, \ldots, m-1$. □

3.3 Computing Characteristic Clauses

For computing the characteristic clauses $Carc(\Sigma, \mathcal{P})$, it is not necessary to check whether the clauses produced by using SOL-deduction are not theorems of Σ unlike in cases of Propositions 3.5 or 3.8.

Proposition 3.10 The characteristic clauses with respect to $\mathcal{P}$ can be generated as [12]:

$$\begin{aligned} Carc(\emptyset, \mathcal{P}) &= \{\, p \vee \neg p \mid p \in \mathcal{A} \text{ and } p \vee \neg p \text{ belongs to } \mathcal{P} \,\}, \\ Carc(\Sigma \cup \{C\}, \mathcal{P}) &= \mu[\, Carc(\Sigma, \mathcal{P}) \cup Prod(\Sigma, C, \mathcal{P}) \,]. \end{aligned}$$

Proof: The first equation is the same as Proposition 2.9. Now,

$$\begin{aligned} & Carc(\Sigma \cup \{C\}, \mathcal{P}) \\ = \; & \mu[\, Carc(\Sigma, \mathcal{P}) \cup Newcarc(\Sigma, C, \mathcal{P}) \,] \text{ (by Proposition 2.9)} \\ = \; & \mu[\, Carc(\Sigma, \mathcal{P}) \cup (Prod(\Sigma, C, \mathcal{P}) - Th_{\mathcal{P}}(\Sigma)) \,] \\ & \text{(by Proposition 3.5)} \\ = \; & \mu[\, Carc(\Sigma, \mathcal{P}) \cup Prod(\Sigma, C, \mathcal{P}) \,]. \end{aligned}$$

□

If the given theory is propositional, the prime implicates can be incrementally constructed using every clause as a top clause as follows.

Proposition 3.11 [13] Given $PI(\Sigma)$ and a clause C, $PI(\Sigma \cup \{C\})$ can be found incrementally:

$$\begin{aligned} PI(\emptyset) &= \{\, p \vee \neg p \mid p \in \mathcal{A} \,\}, \text{ and} \\ PI(\Sigma \cup \{C\}) &= \mu[\, PI(\Sigma) \cup Prod(PI(\Sigma), C, \mathcal{P}_{\mathcal{L}}) \,]. \end{aligned}$$

□

The computation of all prime implicates of Σ by Proposition 3.11 is much more efficient than the brute-force way of resolution proposed by Reiter and de Kleer [37], which makes every possible resolution until no more unsubsumed clauses are produced. The computational superiority of the proposed technique as compared with a brute-force, saturation algorithm comes from the restriction of resolution, as the key problem here is to generate as few as possible subsumed clauses together with making as few as possible subsumption tests. Also, ours uses C-ordered linear resolution, and as discussed in Section 3.1, it has naturally more restriction strategies than set-of-support

[12] In practice, no tautology will take part in any deduction; tautologies decrease monotonically (see Definition 3.1).

resolution that is used in Kean and Tsiknis's [18] extension of the consensus method [46] for generating prime implicates.

This difference becomes larger when there are some distinguished literals representing assumptions in ATMS cases. The most important difference lies in the fact that the formulations by Reiter and de Kleer [37] and by Kean and Tskinis [18] require the computation of all prime implicates whereas ours only needs characteristic clauses that are a subset of the prime implicates constructed from $\mathcal{P}$, again avoiding "generate-and-test" manners (see [13] for details).

4 Variations

In the basic procedure in Section 3.1, two rules **Skip** (Rule 5(a)i) and **Resolve** (Rule 5(a)ii) are treated as alternatives in Step 5a of an SOL-deduction (Definition 3.1). This treatment is necessary to guarantee the completeness of SOL-resolution. In this section, we violate this requirement, and show properties of efficient variations of SOL-resolution and their applications to AI problems. Note that the **Reduce** rule (Rule 5(a)iii) still remains as an alternative choice of other two rules (see Remark (4) of Definition 3.1).

4.1 Preferring Resolution

The first variation, called *SOL-R deduction*, makes **Resolve** precede **Skip**, namely **Skip** is tried to be applied only when **Resolve** cannot be applied.

In a special case of SOL-R deductions, where the literals are not distinguished, that is, the production field is fixed to $\mathcal{P}_{\mathcal{L}}$, **Skip** is always applied whenever **Resolve** cannot be applied for any selected literal in a deduction. In abduction, the resultant procedure in this case "hypothesizes whatever cannot be proven". Many resolution-based abductive systems applied to diagnosis have favored this sort of *most-specific* explanations [45]. This is also called *dead-end* abduction, which is first proposed by Pople [32] in his abductive procedure based on SL-resolution [20] [13]. The criterion is also used by Cox and Pietrzykowski [3].

[13] Pople's *synthesis* operation performs "factor-and-skip".

4.2 Preferring Skip

In the next variation, called *SOL-S deduction*, **Skip** and **Resolve** are placed in Step 5a of SOL-deductions in the reverse order of SOL-R deductions. That is, when the selected literal belongs to $L_{\mathcal{P}}$, only **Skip** is applied by ignoring the possibility of **Resolve**:

5(a)i-5(a)ii$\star$. **(Skip & Cut)**
If $P_i \cup \{l\}$ belongs to $\mathcal{P}$, then $P_{i+1} = P_i \cup \{l\}$ and $\vec{R_{i+1}}$ is the ordered clause obtained by removing l from $\vec{Q_i}$.
Otherwise, if there is a clause B_i in Σ such that $\neg k \in B_i$ and l and k are unifiable with mgu θ, then $P_{i+1} = P_i\theta$ and $\vec{R_{i+1}}$ is an ordered clause obtained by concatenating $\vec{B_i}\theta$ and $\vec{Q_i}\theta$, framing $l\theta$, and removing $\neg k\theta$.

5(a)iii$\star$. **(Reduce)** the same as Rule 5(a)ii.

This skip-preference has the following nice properties. Firstly, this enables the procedure to prune the branch of the search tree that would have resulted from the literal being resolved upon. Secondly, SOL-S deductions are correct model-theoretically. Let us divide the set of clauses Δ $(= \Delta(\Sigma, C, \mathcal{P}))$ produced by using SOL-deductions from $\Sigma + C$ and $\mathcal{P}$, not necessarily closed under subsumption, into two sets, say Δ_1 and Δ_2, such that

$$\Delta = \Delta_1 \cup \Delta_2 \text{ and } \Sigma \cup \Delta_1 \models \Delta_2 .$$

Recall that $Newcarc(\Sigma, C, \mathcal{P}) \subseteq Prod(\Sigma, C, \mathcal{P}) = \mu\Delta$ (by Theorem 3.2). Then adding Δ_2 to Δ_1 does not change the models of $\Sigma \cup \Delta_1$:

$$Mod(\,\Sigma \cup \Delta_1\,) = Mod(\,\Sigma \cup \Delta\,) = Mod(\,\Sigma \cup Prod(\Sigma, C, \mathcal{P})\,),$$

where $Mod(T)$ is the first-order models of T. Thus only Δ_1 needs to be computed model-theoretically. The next theorem shows SOL-S deductions produce precisely such a Δ_1.

Theorem 4.1 If a clause T is derived by an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$, then there is a set δ of clauses each of which is derived by an SOL-S deduction from $\Sigma + C$ and $\mathcal{P}$ such that $\Sigma \cup \delta \models T$.

Proof: See Appendix for the proof. □

Let us denote by $\Delta_{\mathbf{S}}(\Sigma, C, \mathcal{P})$ the clauses deduced by using SOL-S deductions from $\Sigma + C$ and $\mathcal{P}$. The *S-production from* $\Sigma + C$ *and* $\mathcal{P}$ is defined as:

$$Prod_{\mathbf{S}}(\Sigma, C, \mathcal{P}) = \mu\, \Delta_{\mathbf{S}}(\Sigma, C, \mathcal{P}).$$

Corollary 4.2 $\Sigma \cup Prod_{\mathbf{S}}(\Sigma, C, \mathcal{P}) \models Prod(\Sigma, C, \mathcal{P})$.

Proof: It is obvious to see from Theorem 4.1 that

$$\Sigma \cup \Delta_{\mathbf{S}}(\Sigma, C, \mathcal{P}) \models \Delta(\Sigma, C, \mathcal{P}).$$

The corollary follows from the facts that

$$\mu\Delta_{\mathbf{S}}(\Sigma, C, \mathcal{P}) \models \Delta_{\mathbf{S}}(\Sigma, C, \mathcal{P}) \text{ and } \mu\Delta(\Sigma, C, \mathcal{P}) \subseteq \Delta(\Sigma, C, \mathcal{P}).$$

□

Notice that $Prod_{\mathbf{S}}(\Sigma, C, \mathcal{P})$ is not always a subset of $Prod(\Sigma, C, \mathcal{P})$ although $\Delta_{\mathbf{S}}(\Sigma, C, \mathcal{P}) \subseteq \Delta(\Sigma, C, \mathcal{P})$ holds. Thus a clause in the S-production may not be a new characteristic clause.

Example 4.3 Suppose that the theory is

$$\Sigma = \{\, s \supset g,\ t \supset g,\ s \vee t \,\},$$

and that $\mathcal{P} = \langle\, \{\neg s, \neg t\} \,\rangle$. In this case,

$$Prod_{\mathbf{S}}(\Sigma, \neg g, \mathcal{P}) = \{\, \neg s,\ \neg t \,\}.$$

However, if SOL-resolution is used, we see that

$$Prod(\Sigma, \neg g, \mathcal{P}) = \{\Box\}.$$

The empty clause cannot be produced by SOL-S deductions, but Theorem 4.1 is verified:

$$\Sigma \cup \{\, \neg s,\ \neg t \,\} \models \Box.$$

When SOL-S deductions are applied to abduction especially for natural language understanding, it is sometimes called *least-specific* abduction [45]. In abduction, recall that for a clause $S \in \Delta$ and $H = \overline{L_{\mathcal{P}}}, \neg S$ is an explanation of $\neg C$ from (Σ, H) if $\Sigma \not\models S$. Thus, the explanations $\overline{\Delta_1}$ are the weakest in the sense that for any clause $S_2 \in \Delta_2$, there exists a conjunction of clauses $\delta_1 \subseteq \Delta_1$ such that $\Sigma \cup \{\neg S_2\} \models \neg\delta_1$ holds.

In circumscription, this is particularly desirable since we want to answer whether a query holds in every minimal model or not; the purpose of using explanation-based procedures is purely model-theoretic. One of advantages of Przymusinski's procedure [34] lies in the fact that MILO-resolution performs a kind of SOL-S deductions [15]. Similarly, SLINF-resolution [27] prefers **Skip** tp **Resolve** in deriving ground clauses from the axioms and the top clause.

Example 4.4 [15] Consider circumscription of P in Σ with Z, where the theory is

$$\Sigma = \{\, p_1 \vee \neg p_2, \quad p_2 \vee \neg p_3, \quad p_3 \vee z \,\},$$

the minimized predicates are $P = \{p_1, p_2, p_3\}$, and the variables are $Z = \{z\}$. The characteristic literals are $L_{\mathcal{P}} = P^+ = \{p_1, p_2, p_3\}^+$. The query is z.

Now, by adding $\neg z$ to Σ, an SOL-S deduction provides p_3. Since this literal belongs to $\mathcal{P}$, the procedure skips it and stops. Then adding $\neg p_3$ to Σ generates no new characteristic clause. According to Proposition 2.15, z is a theorem of the circumscriptive theory.

If the procedure would examine the remaining choice, resolving p_3 with the clause $p_2 \vee \neg p_3$, it would produce p_2, and a further step would produce p_1. This is exactly what SOL-resolution may do, and the production from $\Sigma + \neg z$ and $\mathcal{P}$ can provide:

$$Newcarc(\Sigma, \neg z, \mathcal{P}) = \{p_3,\, p_2,\, p_1\}.$$

The negation of the CNF formula is $\neg p_3 \vee \neg p_2 \vee \neg p_1$, which provides no new characteristic clauses with respect to Σ, verifying the result of the above computation. The SOL-S deduction did not need to generate two extra clauses because they are consequences of Σ with p_3:

$$\Sigma \cup \{p_3\} \models p_1 \wedge p_2\,.$$

4.3 Between Skipping and Resolving

We can consider another interesting procedure that offers an intermediate alternative to the SOL-R and SOL-S deductions. The set of literals $\mathcal{L}$ can be divided into four sets: those never skipped, those skipped only if cannot be resolved, those which can be non-deterministically chosen either skipped or resolved, and those immediately skipped. This is useful for a sort of abduction where we would like to get explanations in appropriate detail.

One further generalization of this idea would be *best-first* abduction. This notion was originally introduced by Lee [21] in consequence-finding. Stickel [45] also uses the minimal-cost proof for Horn theories where we can choose an operation whose expected computational cost is minimum, but it is difficult to apply the idea to non-Horn theories.

4.4 Approximation

We can consider more drastic variations of the basic procedure. To do so, let us remember the complexity issues of consequence-finding in the propositional case [14], which have been recently examined for the CMS/ATMS by [33, 40]. Provan [33] shows that the ATMS complexity inherits from enumeration of prime implicates which is NP-hard. Thus any complete algorithm for computing ATMS labels is intractable. Selman and Levesque [40] show that finding *one* explanation of an atom from a Horn theory and a set of atomic hypotheses is NP-hard. Therefore, even if we *abandon the completeness* of deductions with respect to the primitive *Newcarc* operation, for instance, by limiting the production to only those clauses having some small number of literals [5] belonging to $\mathcal{P} = \langle \overline{A},$ *size is less than* $k \rangle$[15], it is still intractable.

Are there any rescues from the computational difficulty? We can consider approximation of abduction; either *discard the consistency* or *dispense with the soundness*. In the former case, we may only run an SOL-deduction and believe the result, omitting consistency checking described in Section 3.2.

[14] The complexity of consequence-finding in the general case is bounded by the limitation that the set of all theorems is recursively enumerable. Notice, however, that whether a given formula is *not* a logical consequence of such a theory cannot be determined in a finite amount of time.

[15] Notice that this $\mathcal{P}$ is stable. In practice, this size-restriction is very useful for minimizing the search effort, because it causes earlier pruning in SOL-deduction sequences.

This is a sort of optimistic reasoner without taking care of ramification. On the other hand, the latter case happens if we skip literals not belonging to the characteristic literals: the soundness is violated in the sense that there is a clause $S \in Prod_{\mathbf{X}}(\Sigma, C, \mathcal{P})$ such that $S \notin Th_{\mathcal{P}}(\Sigma \cup \{C\})$ [16]. This is an extreme of an SOL-S deduction in Section 4.2. We can stop deductions in accordance with computational resources; the unresolved literals in a leaf of the deduction are then immediately skipped. These skipped literals are dealt with as *defaults* and will be reconsidered later. Levesque [22] also gives a hint for this kind of computation in terms of *explicit* abduction.

The fact that the procedure is sound and complete is valuable although the computational complexity is exponential. This is because we can improve the quality of solutions as time goes by; we can expect to get the correct answer if we can spend enough time to solve it. This property of "anytime algorithm" is a desirable feature for any future AI system.

5 Conclusion

We have revealed the importance of consequence-finding in AI techniques. Most advanced reasoning mechanisms such as abduction and default reasoning require global search in their proof procedures. This global character is strongly dependent on consequence-finding, in particular those theorems of the theory belonging to production fields. That is why we need some complete procedure for consequence-finding.

For this purpose, we have proposed SOL-resolution, an extension of C-ordered linear resolution augmented by the skip rule. The procedure is sound and complete for finding the (new) characteristic clauses. The significant innovation of the results presented is that the procedure is direct relative to the given production field. We have also presented incomplete, but efficient variations of the basic procedures with different properties of consequence-finding.

[16] However, since for any structured clause $D_i = (P_i, \vec{Q_i})$ in every deduction from $\Sigma + C$ and $\mathcal{P}$, it holds that $\Sigma \cup \{C\} \models P_i \cup Q_i$, we can always guarantee that $S \in Th(\Sigma \cup \{C\})$.

A Appendix: Proofs of Theorems

In this Appendix, we show the proofs of Theorems 3.2 and 4.1.

Firstly, we prove the soundness of SOL-resolution.

Lemma A.1 Let $D_0, \ldots, D_n$ be an SOL-deduction of S from $\Sigma + C$ and $\mathcal{P}$. For each $D_i = \langle P_i, \vec{Q_i} \rangle$ $(0 \leq i \leq n-1)$, it hold that

$$\Sigma \cup \{ P_0 \cup Q_0, \ldots, P_i \cup Q_i \} \models P_{i+1} \cup Q_{i+1} .$$

Notice that $P_0 \cup Q_0 = C$ and $P_n \cup Q_n = S$.

> **Proof:** Since truncation (Rule 5b) does not change the clausal meaning of each structured clause, we can consider only Rule 5a. Let $D_{i+1} = \langle P_{i+1}, \vec{Q_{i+1}} \rangle$ $(0 \leq i \leq n-1)$ be the structured clause obtained from $D_i = \langle P_i, \vec{Q_i} \rangle$ by applying either of the three choices, **Skip** (Rule 5(a)i), **Resolve** (Rule 5(a)ii) or **Reduce** (Rule 5(a)iii), and truncation.
>
> 1. **Skip** is applied. In this case, $P_i \cup Q_i = P_{i+1} \cup Q_{i+1}$.
> 2. **Resolve** is applied. In this case, $\Sigma \cup \{P_i \cup Q_i\} \models P_{i+1} \cup Q_{i+1}$.
> 3. **Reduce** is applied. If factoring (Rule 5(a)iiiA) is applied, then $\{P_i \cup Q_i\} \models P_{i+1} \cup Q_{i+1}$. If ancestry (Rule 5(a)iiiB) is applied, then there exists a structured clause $D_j = \langle P_j, \vec{Q_j} \rangle$ $(j < i)$ such that $\{P_i \cup Q_i, P_j \cup Q_j\} \models P_{i+1} \cup Q_{i+1}$ (see [2, Lemma 7.2, page 141]).
>
> By combining the above three cases, the lemma holds. □

Theorem 3.2 (1) (Soundness of SOL-Resolution)
If a clause S is derived using an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$, then S belongs to $Th_{\mathcal{P}}(\Sigma \cup \{C\})$.

> **Proof:** Let $D_0, \ldots, D_n$ be an SOL-deduction of S from $\Sigma + C$ and $\mathcal{P}$. Since **Skip** is applied to only those literals belonging to $\mathcal{P}$, it is easy to see that S belongs to $\mathcal{P}$.
>
> We prove that $\Sigma \cup \{C\} \models S$. Take any model M of $\Sigma \cup \{C\}$. Since $\Sigma \cup \{C\} \models P_1 \cup Q_1$ by Lemma A.1, M satisfies $P_1 \cup Q_1$ and

is a model of $\Sigma \cup \{C,\ P_1 \cup Q_1\}$. Again by Lemma A.1, M satisfies $P_2 \cup Q_2$, and further applications of Lemma A.1 lead to the fact that M satisfies $P_n = S$. Hence, $S \in Th_{\mathcal{P}}(\Sigma \cup \{C\})$. □

Before giving the completeness proof of SOL-resolution for consequence-finding, we prove the completeness of SOL-resolution for proof-finding. In proof-finding, since the only target clause produced from $\Sigma + C$ is □, we can consider the production field $\mathcal{P}_\emptyset = \langle\, \emptyset \,\rangle$ so that **Skip** (Rule 5(a)i) will never be applied. Thus for an SOL-deduction D of □ from $\Sigma + C$ and $\mathcal{P}_\emptyset$,

$$\langle\, \square,\ \vec{C} \,\rangle, \ldots, \langle\, \square,\ \vec{Q_i} \,\rangle, \ldots, \langle\, \square,\ \square \,\rangle\,,$$

we identify D with a sequence of ordered clauses,

$$\vec{C}, \ldots, \vec{Q_i}, \ldots, \square\,.$$

We call SOL-resolution with the production field $\mathcal{P}_\emptyset$ *OL*-resolution.*

Definition A.2 (OL*-Refutation) Given a theory Σ, and a clause C, an *OL*-refutation from* $\Sigma + C$ consists of a sequence of ordered clauses, $\vec{Q_0}, \vec{Q_1}, \ldots, \vec{Q_n}$, such that:

1. $Q_0 = C$.
2. $Q_n = \square$.
3. For each $\vec{Q_i}$, Q_i is not a tautology.
4. For each $\vec{Q_i}$, Q_i is not subsumed by any Q_j, where $\vec{Q_j}$ is a previous ordered clause, $j < i$.
5. $\vec{Q_{i+1}}$ is generated from $\vec{Q_i}$ according to the following steps:
   (a) Let l be the left-most literal of $\vec{Q_i}$. $\vec{R_{i+1}}$ is obtained by applying either of the rules:
      i. **(Resolve0)** If there is a clause B_i in Σ such that $\neg k \in B_i$ and l and k are unifiable with mgu θ, then $\vec{R_{i+1}}$ is an ordered clause obtained by concatenating $\vec{B_i}\theta$ and $\vec{Q_i}\theta$, framing $l\theta$, and removing $\neg k\theta$.
      ii. **(Reduce0)** If either

A. (*factoring*) $\vec{Q}_i$ contains an unframed literal k either different from l or another occurrence of l, or

B. (*ancestry*) $\vec{Q}_i$ contains a framed literal $\boxed{\neg k}$,

and l and k are unifiable with mgu θ, then $\vec{R}_{i+1}$ is obtained from $\vec{Q}_i\theta$ by deleting $l\theta$.

(b) $\vec{Q}_{i+1}$ is obtained from $\vec{R}_{i+1}$ by deleting every framed literal not preceded by an unframed literal in the remainder (*truncation*).

We can see that the above definition of OL*-refutation is a variant of SL-refutation [20] or ME refutation [24]. However, as noted in Remark (4) of Definition 3.1, OL*-refutation is different from OL-refutation [2] because ancestry (Rule 5(a)iiB) is an alternative choice to **Resolve** (Rule 5(a)i) or factoring (Rule 5(a)iiA). According to the completeness results for SL-refutation or ME refutation, it is not difficult to verify the completeness of OL*-refutation.

Theorem A.3 If Σ is satisfiable and $\Sigma \cup \{C\}$ is unsatisfiable, then there exists an OL*-refutation from $\Sigma + C$. □

In SOL-resolution, since the **Skip** rule is defined as an alternative rule to others, a refutation can be obtained without using **Skip** as if the rule does not exist like OL*-refutation. Therefore, for any stable production field $\mathcal{P}$, we can identify an SOL-deduction of □ from $\Sigma + C$ and $\mathcal{P}$ with an OL*-refutation from $\Sigma + C$ as follows.

Corollary A.4 Let $\mathcal{P}$ be any stable production field. If Σ is satisfiable and $\Sigma \cup \{C\}$ is unsatisfiable, then there exists an SOL-deduction of □ from $\Sigma + C$ and $\mathcal{P}$. □

The proof of the completeness of SOL-resolution for consequence-finding can be seen as an extension of the results for m.s.l. resolution by Minicozzi and Reiter [26, Theorems 1 and 2, pages 176–177]. In addition to these techniques, we have to take C-ordering and the skip operation into account.

If Σ is a set of clauses and T is a clause, we write

$$\Sigma_T = \{\, C \mid C' \in \Sigma \text{ and } C = C' - T \,\}.$$

We first show the proof for the ground cases.

Lemma A.5 [26, Lemma 1] If Σ is an unsatisfiable set of ground clauses and T is a ground clause, then Σ_T is unsatisfiable. □

Lemma A.6 Let Σ be a set of ground clauses and T a ground clause. If Σ_T is unsatisfiable, then $\Sigma \models T$.

> **Proof:** Assume to the contrary that $\Sigma \not\models T$. Since $\Sigma \cup \{\neg T\}$ is satisfiable, there is a model M of $\Sigma \cup \{\neg T\}$. Let C be any clause in Σ. If $C \cap T = \{t_1, \ldots, t_k\}$ (t_i is a literal), then M satisfies $C - \{t_1, \ldots, t_k\}$ because M satisfies C and every $\neg t_i$ for $1 \leq i \leq k$. Therefore, M satisfies $C - T$ and is a model of Σ_T. Hence, Σ_T is satisfiable, contradiction. □

Lemma A.7 Let Σ be a set of ground clauses, T and C ground clauses. If $\Sigma \not\models T$ and $\Sigma \cup \{C\} \models T$, then there is an OL*-refutation from $\Sigma_T + (C - T)$.

> **Proof:** Suppose that $\Sigma \not\models T$. By Lemma A.6, Σ_T is satisfiable.
>
> Suppose further that $\Sigma \cup \{C\} \models T$. This implies that $\Sigma \cup \{C\} \cup \{\neg T\}$ is unsatisfiable. Since $\{\neg T\}_T = \{\neg T\}$, $\Sigma_T \cup \{C - T\} \cup \{\neg T\}$ is unsatisfiable by Lemma A.5. However, no literal of the CNF formula $\neg T$ has the literal complementary to a literal in the clauses of $\Sigma_T \cup \{C - T\}$. Hence, $\Sigma_T \cup \{C - T\}$ is unsatisfiable.
>
> By Theorem A.3, there is an OL*-refutation from $\Sigma_T + (C - T)$. □

Theorem A.8 Let Σ be a set of ground clauses, T and C ground clauses, and $\mathcal{P}$ a stable production field. If T does not belong to $Th_{\mathcal{P}}(\Sigma)$, but belongs to $Th_{\mathcal{P}}(\Sigma \cup \{C\})$, then there is an SOL-deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that $S \subseteq T$.

> **Proof:** Suppose that $T \in Th_{\mathcal{P}}(\Sigma \cup \{C\}) - Th_{\mathcal{P}}(\Sigma)$. Obviously, $\Sigma \not\models T$ and $\Sigma \cup \{C\} \models T$. By Lemma A.7, there is an OL*-refutation,
>
> $$\vec{Q_0}, \ldots, \vec{Q_i}, \ldots, \vec{Q_m},$$
>
> from $\Sigma_T + (C - T)$. Note that $Q_0 = C - T$ and $Q_m = \Box$.
>
> In this refutation, replace each occurrence of a clause $B_i \in \Sigma_T$ used in an application of **Resolve0** (Rule 5(a)i) to $\vec{Q_i}$ by the

clause B'_i of Σ from which it was derived. Now, construct a sequence D of structured clauses

$$D_{0,0}, \ldots, D_{0,k(0)}, \ldots, D_{i,0}, \ldots, D_{i,k(i)}, \ldots, D_{m,0}, \ldots, D_{m,k(m)},$$

from the refutation according to the following steps:

1. Let $D_{0,0} = \langle \Box, \vec{C} \rangle$.
2. $D_{i,j+1} = \langle P_{i,j+1}, \vec{Q}_{i,j+1} \rangle$ $(0 \leq i \leq m,\ 0 \leq j \leq k(i)-1)$ is obtained from $D_{i,j} = \langle P_{i,j}, \vec{Q}_{i,j} \rangle$ in the following way:
   (a) If the left-most literal of $\vec{Q}_{i,j}$ is the same as the left-most literal of $\vec{Q}_i$ in the OL*-refutation, do not construct $D_{i,j+1}$. Set $k(i) = j$.
   (b) Otherwise, skip the left-most literal l of $\vec{Q}_{i,j}$, i.e., let $P_{i,j+1} = P_{i,j} \cup \{l\}$. $\vec{Q}_{i,j+1}$ is obtained from $\vec{Q}_{i,j}$ by removing the given occurrence of l, and truncating the remainder.
3. $D_{i+1,0} = \langle P_{i+1,0}, \vec{Q}_{i+1,0} \rangle$ $(0 \leq i \leq m-1)$ is obtained from $D_{i,k(i)} = \langle P_{i,k(i)}, \vec{Q}_{i,k(i)} \rangle$ in the following way:
   (a) Let $P_{i+1,0} = P_{i,k(i)}$.
   (b) Let l be the left-most literal of $\vec{Q}_{i,k(i)}$. $\vec{Q}_{i+1,0}$ is obtained as follows.
      i. If **Resolve0** (Rule 5(a)i) was applied to $\vec{Q}_i$ in the OL*-refutation, deducing $\vec{Q}_{i+1}$ through truncation, then $\vec{Q}_{i+1,0}$ is the ordered clause obtained by concatenating $\vec{B}'_i$ and $\vec{Q}_{i,k(i)}$, framing l, removing $\neg l$, and truncating the remainder.
      ii. If **Reduce0** (Rule 5(a)ii) was applied to $\vec{Q}_i$ in the OL*-refutation, deducing $\vec{Q}_{i+1}$ through truncation, then $\vec{Q}_{i+1,0}$ is obtained from $\vec{Q}_{i,k(i)}$ by deleting the given occurrence of l, and truncating the remainder.

Note that the left-most literal of $\vec{Q}_{i,k(i)}$ $(1 \leq i \leq m)$ is always the same as the left-most literal of $\vec{Q}_i$ in the refutation. Notice also

that $k(i)$ is the number of literals skipped in the sequence from $D_{i,0}$ to $D_{i+1,0}$, i.e.,

$$k(i) = |P_{i+1,0} - P_{i,0}|\,.$$

In other words, $P_{i+1,0} = P_{i,0} \cup S_i$, where S_i is the set of non-framed literals in $\vec{Q_{i,0}}$ not preceded by the left-most literal of $\vec{Q_{i+1}}$.

Now, let $S = P_{m,k(m)}$. We see that $D_{m,k(m)} = \langle S,\ \Box \rangle$. In D, since every literal in any $P_{i,j}$ is a literal of T which comes from $B_i' \cap T$ $(= B_i' - B_i)$ or $C \cap T$ $(= C - (C - T))$, we verify that $S \subseteq T$.

It remains to verify that D is actually an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$.

1. In each construction of $D_{i+1,0}$ from $D_{i,k(i)}$ $(0 \leq i \leq m-1)$, the rules **Resolve0** and **Reduce0** have been changed to just the rules **Resolve** and **Reduce**.
2. In each construction of $D_{i,j+1}$ from $D_{i,j}$ $(0 \leq i \leq m,\ 0 \leq j \leq k(i)-1)$, **Skip** has been used. By the supposition, since T belongs to $\mathcal{P}$, every $P_{i,j}$ belongs to $\mathcal{P}$ and the condition for the application of **Skip** is satisfied.
3. No clause of $B_i \in \Sigma_T$ contains a literal of $\neg T$ because each B_i is used in the OL*-refutation from $\Sigma_T + (C - T)$. This implies that no clause of $B_i' \in \Sigma$ contains the complement of a literal of T so that for any structured clause $D_{i,j} = \langle P_{i,j},\ \vec{Q_{i,j}} \rangle$, $P_{i,j} \cup Q_{i,j}$ is not a tautology.
4. Each $\vec{Q_{i,j}}$ in D contains all the literals of $\vec{Q_i}$ in the OL*-refutation. Furthermore, every literal in $Q_{i,j} - Q_i$ is a literal of T and thus does not appear in any previous Q_k $(k < i)$ in the refutation. Therefore, as Q_i is not subsumed by any Q_k $(k < i)$, $Q_{i,j}$ is not subsumed by any other $Q_{k,l}$ $(k \leq i, l \leq j)$.

□

We now lift the result of Theorem A.8 to the general level. We use the technique, first used by Slagle, Chang and Lee [43], of introducing new distinct constants into the Herbrand Universe.

Lemma A.9 [43, Theorem 1] Let Σ be a set of clauses and T a ground clause. $\Sigma \models T$ if and only if there is a finite set Σ' of ground instances of clauses in Σ over the Herbrand Universe of $\Sigma \cup \{T\}$ such that $\Sigma' \models T$. □

Theorem 3.2 (2) (Completeness of SOL-Resolution)
If a clause T *does not belong to* $Th_{\mathcal{P}}(\Sigma)$, *but* belongs to $Th_{\mathcal{P}}(\Sigma \cup \{C\})$, then there is an SOL-deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that S subsumes T.

Proof: Let $x_1, \ldots, x_m$ be all of the individual variables of T. Let $b_1, \ldots, b_m$ be new distinct constants not appearing anywhere in Σ, C or T. Since $\Sigma \cup \{C\} \models T$, it holds that $\Sigma \cup \{C\} \models T\theta$ for a substitution $\theta = \{x_1/b_1, \ldots, x_m/b_m\}$. Similarly, as $\Sigma \not\models T$, it must be that $\Sigma \not\models T\theta$. Let $H(b_1, \ldots, b_m)$ be the Herbrand Universe of $\Sigma \cup \{C, T\theta\}$. By Lemma A.9, there is a finite set of ground instances Σ' of Σ over $H(b_1, \ldots, b_m)$ such that

1. $\Sigma' \not\models T\theta$, and
2. $\Sigma' \cup \{C'\} \models T\theta$ where C' is a ground instance of C over $H(b_1, \ldots, b_m)$.

By Theorem A.8, there is a ground SOL-deduction D' of S' from $\Sigma' + C'$ and $\mathcal{P}$ such that $S' \subseteq T\theta$. In D', replace each occurrence of a clause $B_i' \in \Sigma'$ used in an application of **Resolve** by the clause B_i of Σ of which it is a ground instance. Also, replace each structured clause D_i' in D' by the general structured clause D_i which is constructed from its parent and the corresponding B_{i-1} in the obvious way. The resultant sequence D is a general SOL-deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that S has, as an instance, S'. So, there is a substitution ρ such that

$$S\rho \subseteq S' \subseteq T\theta\,,$$

but since Σ does not contain the symbols $b_1, \ldots, b_m$, there is a substitution σ such that

$$S\sigma \subseteq T, \text{ and } \sigma = \rho\theta^{-1}\,.$$

Thus, S subsumes T. □

Before proving Theorem 4.1, we show a property of SOL-S deductions in a special case, where a stronger result can be obtained.

If a sequence, D, of structured clauses, $D_i, \ldots, D_n$, is a subsequence of an SOL-deduction (or SOL-S deduction), $D_0, \ldots, D_i, \ldots, D_n$, of S from $\Sigma + C$ and $\mathcal{P}$, we say D is an *SOL-deduction* (or *SOL-S deduction*) *of S from $\Sigma + D_i$ and $\mathcal{P}$.*

Lemma A.10 If a clause T is derived by an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$ without applying the **Reduce** rule to any structured clause in the deduction, then there is an SOL-S deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that $\Sigma \cup \{S\} \models T$.

> **Proof:** Let $D_0, D_1, \ldots, D_n$ be an SOL-deduction of T from $\Sigma + C$ and $\mathcal{P}$. Let l_i be the selected literal of $\vec{Q}_i$, where $D_i = \langle P_i, \vec{Q}_i \rangle$ and $0 \leq i \leq n-1$. Assume that **Reduce** is not used in the deduction.
>
> Firstly, if **Skip** is applied upon every l_j $(0 \leq j \leq n-1)$ such that $P_j \cup \{l_j\}$ belongs to $\mathcal{P}$, then T is actually SOL-S deduced from $\Sigma + C$ and $\mathcal{P}$, and of course $\Sigma \cup \{T\} \models T$ holds.
>
> Next, suppose that there exists a structured clause D_j in the SOL-deduction such that $P_j \cup \{l_j\}$ belongs to $\mathcal{P}$, but that **Resolve** is applied upon l_j in $\vec{Q}_j$ with a clause $B_j \in \Sigma$. We now call such D_j a *resister*. We prove the lemma in this case by induction on the number m $(1 \leq m \leq n)$ of resisters in the deduction.
>
> If D_k $(0 \leq k \leq n-1)$ is the only resister in the SOL-deduction, i.e., $m = 1$, then $D_{k+1} = \langle P_{k+1}, \vec{Q_{k+1}} \rangle$ satisfies
>
> $$\begin{aligned} P_{k+1} &= P_k\theta, \quad \text{and} \\ Q_{k+1} &= (B_k\theta - \{\neg l_k\theta\}) \cup (Q_k\theta - \{l_k\theta\}), \end{aligned}$$
>
> where θ is a substitution.
>
> Obviously, T is SOL-deduced from $\Sigma + D_{k+1}$ and $\mathcal{P}$. Now, let U be a clause SOL-S deduced from $\Sigma + (B_k\theta - \{\neg l_k\theta\})$ and $\mathcal{P}$, V a clause SOL-S deduced from $\Sigma + (Q_k\theta - \{l_k\theta\})$ and $\mathcal{P}$. Since no more **Resolve** is applied upon any subsequent selected literal l_j

$(j > k)$ such that $P_j \cup \{l_j\}$ belongs to $\mathcal{P}$, we can choose such U and V to satisfy

$$T = (P_k \cup U \cup V)\rho\,,$$

for some substitution ρ.

Now assume that instead of applying **Resolve, Skip & Cut** is applied to D_k, deducing $D'_{k+1} = \langle\, P'_{k+1},\ Q'_{k+1}\,\rangle$, where

$$\begin{aligned} P'_{k+1} &= P_k \cup \{l_k\}\,, \quad \text{and} \\ Q'_{k+1} &= Q_k - \{l_k\}\,. \end{aligned}$$

Then, $(P_k \cup \{l_k\} \cup V)\sigma$ for some substitution σ is SOL-S deduced from $\Sigma + D'_{k+1}$ and $\mathcal{P}$, and thus SOL-S deduced from $\Sigma + C$ and $\mathcal{P}$. Since $\Sigma \cup \{l_k\} \models B_k - \{\neg l_k\}$, $\Sigma \cup \{l_k\} \models U$ holds, and therefore it holds that

$$\Sigma \cup \{(P_k \cup \{l_k\} \cup V)\} \models T\,.$$

Thus, the lemma holds for the base case.

Assume that the lemma holds for cases that $m-1$ or less resisters are in number. We consider the case that there are m resisters in the SOL-deduction. Let D_k be the last resister in the SOL-deduction, namely k is the biggest number in such m resisters. In the same way as the base case, we can prove that there is an SOL-S deduction of a clause S' from $\Sigma + D_k$ and $\mathcal{P}$ such that

$$\Sigma \cup \{S'\} \models T\,,$$

where $S' = (P_k \cup \{l_k\} \cup V)$.

Now, since S' is SOL-deduced from $\Sigma + C$ and $\mathcal{P}$ involving $(m-1)$ resisters, there is an SOL-S deduction of S from $\Sigma + C$ and $\mathcal{P}$ such that

$$\Sigma \cup \{S\} \models S'\,,$$

by induction hypothesis. Hence, $\Sigma \cup \{S\} \models T$. This completes the proof. □

We now take the **Reduce** rule into account. The result of Lemma A.10 still remains to hold even if factoring is incorporated. However, if ancestry

occurs in SOL-deduction of T upon a selected literal against a framed literal that had been resolved upon before a resister appeared, then the relation between an SOL-S deduced clause S and T that $\Sigma \cup \{S\} \models T$ no longer holds and another SOL-S deduced clause S' is required to imply T, namely $\Sigma \cup \{S, S'\} \models T$ (see Example 4.3).

Theorem 4.1 If a clause T is derived by an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$, then there is a set δ of clauses each of which is derived by an SOL-S deduction from $\Sigma + C$ and $\mathcal{P}$ such that $\Sigma \cup \delta \models T$.

Proof: Let $D_0, D_1, \ldots, D_n$ be an SOL-deduction of T from $\Sigma + C$ and $\mathcal{P}$. Let l_i be the selected literal of $\vec{Q}_i$, where $D_i = \langle P_i, \vec{Q}_i \rangle$ $(0 \leq i \leq n-1)$. We prove the theorem by induction on the number m $(0 \leq m \leq n)$ of **Reduce** operations in the SOL-deduction.

If **Reduce** is not applied upon any l_j $(0 \leq j \leq n-1)$, i.e., $m = 0$, then by Lemma A.10 there is an SOL-S deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that $\Sigma \cup \{S\} \models T$. Thus the theorem holds for the base case.

Assume that the theorem holds for cases that $m-1$ or less times of **Reduce** operations occur in SOL-deductions. We consider the case that there are m selected literals upon which **Reduce** is applied in the SOL-deduction. Let D_k be the last structured clause to which **Reduce** is applied in SOL-deductions, namely k is the biggest number in such structured clauses. In this case, $D_{k+1} = \langle P_{k+1}, \vec{Q_{k+1}} \rangle$ satisfies

$$\begin{aligned} P_{k+1} &= P_k\theta, \quad \text{and} \\ Q_{k+1} &= Q_k\theta - \{l_k\theta\}, \end{aligned}$$

where θ is a substitution. Note that T is SOL-deduced from $\Sigma + D_{k+1}$ and $\mathcal{P}$ without involving further application of **Reduce**.

1. Factoring (Rule 5(a)iiiA) is applied to D_k. In this case, since $\{P_k \cup Q_k\} \models P_{k+1} \cup Q_{k+1}$ holds, it is easy to verify that there is an SOL-deduction of S' from $\Sigma + D_k$ and $\mathcal{P}$ without involving any application of **Reduce** such that $\{S'\} \models T$.

Since D_k is a structured clause in the SOL-deduction of T, S' can be derived by an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$ in which $(m-1)$ **Reduce** operations are applied. By induction hypothesis, there is a set δ of clauses SOL-S deduced from $\Sigma + C$ and $\mathcal{P}$ such that $\Sigma \cup \delta \models S'$. Thus, $\Sigma \cup \delta \models T$.

2. Notice that in the above proof for applications of factoring, if all of m **Reduce** operations are factoring, it holds that there is an SOL-S deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that $\Sigma \cup \{S\} \models T$. Hence, Lemma A.10 holds even if factoring is taken into account. Thus, in the following, we can turn to focus on induction on the number of ancestry (Rule 5(a)iiiB) operations in the SOL-deduction of T.

3. Assume that the theorem holds for cases that $m-1$ or less ancestry operations are applied in SOL-deductions. Suppose that m ancestry operations occurs in the SOL-deduction of T. D_k is now the m-th structured clause to which ancestry is applied. In this case, there is a structured clause $D_j = \langle P_j, \vec{Q_j} \rangle$ $(j < k)$ such that

$$\{P_j \cup Q_j, P_k \cup Q_k\} \models P_{k+1} \cup Q_{k+1}$$

(see the proof of Lemma A.1). Now, apply **Resolve** to D_k against a clause $(P_j \cup Q_j)$, then apply **Skip** and factoring appropriately to the result. Then, we get an SOL-deduction of S_1 from $\Sigma \cup \{P_j \cup Q_j\} + D_k$ without involving any application of ancestry, such that

$$\{P_j \cup Q_j, S_1\} \models T\,.$$

Since D_k is a structured clause in the SOL-deduction of T from $\Sigma + C$ and $\mathcal{P}$, S_1 can be derived by an SOL-deduction from $\Sigma \cup \{P_j \cup Q_j\} + C$ and $\mathcal{P}$. But since $\Sigma \cup \{C\} \models P_j \cup Q_j$ by Lemma A.1, S_1 can be derived by an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$ in which $(m-1)$ ancestry operations are applied. By induction hypothesis, there is a set δ_1 of clauses SOL-S deduced from $\Sigma + C$ and $\mathcal{P}$ such that

$$\Sigma \cup \delta_1 \models S_1\,.$$

In the similar way, apply **Resolve** to D_j against a clause S_1, then apply **Skip** and factoring appropriately to the result. Then, we get an SOL-deduction of S_2 from $\Sigma \cup \{S_1\} + D_j$ without involving any application of ancestry, such that

$$\{S_1, S_2\} \models T .$$

Since D_j is a structured clause in the SOL-deduction of T from $\Sigma + C$ and $\mathcal{P}$, S_2 can be derived by an SOL-deduction from $\Sigma \cup \{S_1\} + C$ and $\mathcal{P}$. But since S_1 is SOL-deduced from $\Sigma + C$ and $\mathcal{P}$, it holds that $\Sigma \cup \{C\} \models S_1$ and thus S_2 can be derived by an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$ in which less than m ancestry operations are applied. Again, by induction hypothesis, there is a set δ_2 of clauses SOL-S deduced from $\Sigma + C$ and $\mathcal{P}$ such that

$$\Sigma \cup \delta_2 \models S_2 .$$

And therefore,

$$\Sigma \cup \delta_1 \cup \delta_2 \models T .$$

By letting $\delta = \delta_1 \cup \delta_2$, it holds that $\Sigma \cup \delta \models T$. This completes the proof of the theorem.

□

Acknowledgment

I especially wish to thank Mark Stickel for his valuable comments on SOL-resolution and Nicolas Helft for discussion on earlier work. I would also like to thank Donald Loveland for his comments on an earlier draft, Ray Reiter and Wolfgang Bibel for helpful suggestions on earlier work, Jun Arima and my supervisor Toshihide Ibaraki for discussions on this topic, and Kazuhiro Fuchi for giving me the opportunity to do this work.

References

[1] G. Bossu and P. Siegel, Saturation, nonmonotonic reasoning, and the closed-world assumption, *Artificial Intelligence* **25** (1985) 23–67.

[2] C.L. Chang and R.C.T. Lee, *Symbolic Logic and Mechanical Theorem Proving* (Academic Press, New York, 1973).

[3] P.T. Cox and T. Pietrzykowski, Causes for events: their computation and applications, in: *Proceedings of the Eighth International Conference on Automated Deduction*, Lecture Notes in Computer Science 230, Springer-Verlag (1986) 608–621.

[4] J. de Kleer, An assumption-based TMS, *Artificial Intelligence* **28** (1986) 127–162.

[5] J. de Kleer, A comparison of ATMS and CSP techniques, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 290–296.

[6] J. de Kleer, A.K. Mackworth and R. Reiter, Characterizing diagnoses, in: *Proceedings AAAI-90*, Boston, MA (1990) 324–329.

[7] R. Demolombe and L. Fariñas de Cerro, An inference rule for hypothesis generation, in: *Proceedings IJCAI-91*, Sydney, Australia (1991).

[8] J.J. Finger, Exploiting constraints in design synthesis, Technical Report STAN-CS-88-1204, Department of Computer Science, Stanford University, Stanford, CA, April 1987.

[9] M. Gelfond, H. Przymusinska and T. Przymusinski, On the relationship between circumscription and negation as failure, *Artificial Intelligence* **38** (1989) 75–94.

[10] M.L. Ginsberg, A circumscriptive theorem prover, *Artificial Intelligence* **39** (1989) 209–230.

[11] N. Helft, K. Inoue and D. Poole, Query answering in circumscription, in: *Proceedings IJCAI-91*, Sydney, Australia (1991).

[12] N. Helft and K. Konolige, Plan recognition as abduction and relevance, Technical Report, SRI International, Menlo Park, CA. 1990.

[13] K. Inoue, An abductive procedure for the CMS/ATMS, in: J.P. Martins and M. Reinfrank (eds.), *Proceedings of the ECAI-90 Workshop on Truth Maintenance Systems* (Stockholm, Sweden, August 1990), Lecture Notes in Artificial Intelligence, Springer-Verlag, 1991.

[14] K. Inoue, Consequence-finding based on ordered linear resolution, in: *Proceedings IJCAI-91*, Sydney, Australia (1991).

[15] K. Inoue and N. Helft, On theorem provers for circumscription, in: *Proceedings of the Eighth Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, Ottawa, Ontario (1990) 212–219.

[16] K. Iwanuma, M. Harao and S. Noguchi, A computation method for parallel circumscription based on equivalent transformation of queries, *Reports of the Group for Computation IEICE Japan*, COMP89-42 (1989) 21–30 (in Japanese).

[17] U. Junker and K. Konolige, Computing the extensions of autoepistemic and default logics with a truth maintenance system, in: *Proceedings AAAI-90*, Boston, MA (1990) 278–283.

[18] A. Kean and G. Tsiknis, An incremental method for generating prime implicants/implicates, *J. Symbolic Computation* **9** (1990) 185–206.

[19] R.A. Kowalski and P.J. Hayes, Semantic trees in automatic theorem-proving, in: B. Meltzer and D. Michie (Eds.), *Machine Intelligence* **4** (Edinburgh University Press, 1969) 87–101.

[20] R.A. Kowalski and D.G. Kuhner, Linear resolution with selection function, *Artificial Intelligence* **2** (1971) 227–260.

[21] R.C.T. Lee, A completeness theorem and computer program for finding theorems derivable from given axioms, Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, 1967.

[22] H.J. Levesque, A knowledge-level account of abduction (preliminary version), in: *Proceedings IJCAI-89*, Detroit, MI (1989) 1061–1067.

[23] V. Lifschitz, Computing circumscription, *Proceedings IJCAI-85*, Los Angeles, CA (1985) 121–127.

[24] D. Loveland, *Automated Theorem Proving: A Logical Basis*, (North-Holland, Amsterdam, 1978).

[25] J. McCarthy, Circumscription—a form of non-monotonic reasoning, *Artificial Intelligence* **13** (1980) 27–39.

[26] E. Minicozzi and R. Reiter, A note on linear resolution strategies in consequence-finding, *Artificial Intelligence* **3** (1972) 175–180.

[27] J. Minker and A. Rajasekar, A fixpoint semantics for disjunctive logic programs, *J. Logic Programming* **9** (1990) 45–74.

[28] P. O'Rorke (ed.), *Working Notes of the AAAI Spring Symposium on Automated Abduction*, Stanford, CA, March 1990.

[29] D. Poole, Explanation and prediction: an architecture for default and abductive reasoning, *Computational Intelligence* **5** (1989) 97–110.

[30] D. Poole, Compiling a default reasoning system into Prolog, *New Generation Computing* **9** (1991) 3–38.

[31] D. Poole, R. Goebel and R. Aleliunas, Theorist: a logical reasoning system for defaults and diagnosis, in: N. Cercone and G. McCalla (eds.), *The Knowledge Frontier: Essays in the Representation of Knowledge.* Springer-Verlag, New York (1987) 331–352.

[32] H.E. Pople, Jr., On the mechanization of abductive logic, in: *Proceedings IJCAI-73*, Stanford, CA (1973) 147 152.

[33] G.M. Provan, The computational complexity of multiple-context truth maintenance systems, in: *Proceedings ECAI-90*, Stockholm, Sweden (1990) 522–527.

[34] T.C. Przymusinski, An algorithm to compute circumscription, *Artificial Intelligence* **38** (1989) 49–73.

[35] W.V. Quine, The problem of simplifying truth functions, *Am. Math. Monthly*, **59** (1952) 521–531.

[36] R. Reiter, Two results on ordering for resolution with merging and linear format, *J. ACM* **18** (1971) 630–646.

[37] R. Reiter and J. de Kleer, Foundations of assumption-based truth maintenance systems: preliminary report, in: *Proceedings AAAI-87*, Seattle, WA (1987) 183–188.

[38] J.A. Robinson, A machine-oriented logic based on the resolution principle, *J. ACM* **12** (1965) 23–41.

[39] A. Sattar, S. Goodwin and R. Goebel, What's in a consistency tree: a uniform treatment of theory selection, in: *Proceedings of the Pacific Rim International Conference on Artificial Intelligence '90*, Nagoya, Japan (1990) 426–431.

[40] B. Selman and H.J. Levesque, Abductive and default reasoning: a computational core, in: *Proceedings AAAI-90*, Boston, MA (1990) 343–348.

[41] R. Shostak, Refutation graphs, *Artificial Intelligence* **7** (1976) 51–64.

[42] P. Siegel, Représentation et utilisation de la connaissance en calcul propositionnel, Thèse d'État, Université d'Aix-Marseille II, Luminy, France, 1987 (in French).

[43] J.R. Slagle, C.L. Chang and R.C.T. Lee, Completeness theorems for semantic resolution in consequence finding, in: *Proceedings IJCAI-69*, Washington, D.C. (1969) 281–285.

[44] M.E. Stickel, A Prolog technology theorem prover: implementation by an extended Prolog compiler, *J. Automated Reasoning* **4** (1988) 353–380.

[45] M.E. Stickel, Rationale and methods for abductive reasoning in natural-language interpretation. in: R. Studer (ed.), *Natural Language and Logic, Proceedings of the International Scientific Symposium* (Hamburg, Germany, May 1989), Lecture Notes in Artificial Intelligence 459, Springer-Verlag (1990) 233–252.

[46] P. Tison, Generalized consensus theory and application to the minimization of boolean functions, *IEEE transactions on electronic computers* **16** (1967) 446–456.