Complete Pruning Methods and a Practical Search Strategy for SOL

Hidetomo Nabeshima
University of Yamanashi, Japan

Koji Iwanuma
University of Yamanashi, Japan

Katsumi Inoue
National Institute of Informatics, Japan

Abstract

The ability to find non-trivial consequences of an axiom set is useful in many applications of AI such as theorem proving, query answering and nonmonotonic reasoning. SOL (Skip Ordered Linear) calculus is one of the most significant calculi for consequence finding, which is complete for finding the non-subsumed consequences of a clausal theory. In this paper, we propose new complete pruning methods and a practical search strategy for SOL tableaux. These methods are indispensable for practical use in many application areas. The experimental results show that these techniques improve the performance of consequence finding process greatly.

1 Introduction

The task of *consequence finding* [8] is to discover hidden theorems of a given axiom set. These theorems to be found are not given in an explicit way, but are only characterized by some desired properties. Thus, consequence finding is clearly distinguished from proof finding or theorem proving. One of the most significant calculi for consequence finding is the SOL (Skip Ordered Linear) calculus[1], which is complete for finding the non-subsumed consequences of a clausal theory. There are many application areas of SOL calculus such as *nonmonotonic reasoning*, *abduction* [1, 2], *induction* [3], *default reasoning* [4] and *query answering* [7, 6]. The original SOL-resolution calculus can be viewed as extending Loveland's model-elimination calculus [11] with a so-called *Skip* operation. Iwanuma et al. [5] subsequently reformulated SOL-resolution within the framework of connection tableaux [9, 10] and proposed several pruning methods [5, 6].

In this paper, we propose two kinds of new complete pruning methods for SOL tableau calculus, *local failure caching for length condition* and *skip-minimality*. The former is an extension of local failure caching [9, 10, 5] which is an excellent pruning technique for connection tableau calculus. The extension ensures the completeness of the calculus in case we are only interested in consequences up to a given maximal length. A user usually prefers short and simple consequences rather than long and complicated ones. Moreover, limiting the length of consequences has the effect of reducing the search space vastly. In such a situation, our proposed method accelerates the performance of the consequence finding process significantly. The latter method, Skip-minimality, restrains the use of the Skip operation and prunes many redundant SOL derivations. In addition, we describe a new practical search strategy, called *consequence iterative lengthening* (CIL), which has the effect of finding short and simple consequences preferentially in a limited time. We demonstrate the effectiveness of these methods by the experimental evaluation.

The remainder of this paper is organized as follows: Section 2 reviews the SOL tableau calculus. Section 3 and 4 introduces new pruning methods and a search strategy for consequence finding, respectively. Section 5 shows the experimental evaluation. Section 6 concludes with a summary and discussion of future work.

Rudnicki P, Sutcliffe G., Konev B., Schmidt R., Schulz S. (eds.);
Proceedings of the Combined KEAPPA - IWIL Workshops, pp. 113- 122

2 SOL Tableaux for Consequence Finding

This section reviews the SOL tableau calculus [5]. This paper assumes a first-order language (without equality). We write $\subseteq_{ms}$ to denote the inclusion relation over multisets which is defined as usual. When C and D are clauses, C *subsumes* D if there is a substitution θ such that $C\theta \subseteq_{ms} D$. We say C *properly subsumes* D if C subsumes D but D does not subsume C. For a set Σ of clauses, $\mu\Sigma$ denotes the set of clauses in Σ not properly subsumed by any clause in Σ.

Definition 1. *A production field $\mathscr{P}$ is a pair $\langle \boldsymbol{L}, Cond \rangle$, where $\boldsymbol{L}$ is a set of literals closed under instantiation, and Cond is a certain condition to be satisfied. If Cond is not specified, $\mathscr{P}$ is just denoted as $\langle \boldsymbol{L} \rangle$. A clause C belongs to $\mathscr{P} = \langle \boldsymbol{L}, Cond \rangle$ if every literal in C belongs to $\boldsymbol{L}$ and C satisfies Cond. For a set of clauses Σ, the set of logical consequences of Σ belonging to $\mathscr{P}$ is denoted as $Th_{\mathscr{P}}(\Sigma)$. A production field $\mathscr{P}$ is stable if, for any two clauses C and D such that C subsumes D, the clause D belongs to $\mathscr{P}$ only if C belongs to $\mathscr{P}$.*

Example 1. *Let $\mathscr{L} = \mathscr{L}^+ \cup \mathscr{L}^-$ be the set of all literals in the first-order language, where $\mathscr{L}^+$ and $\mathscr{L}^-$ are the positive and negative literals in the language, respectively. The following are examples of stable production fields.*

(a) $\mathscr{P}_1 = \langle \mathscr{L} \rangle$: $Th_{\mathscr{P}_1}(\Sigma)$ is the set of logical consequences of Σ.

(b) $\mathscr{P}_2 = \langle \mathscr{L}^-,$ length is less than $k \rangle$: $Th_{\mathscr{P}_2}(\Sigma)$ is the set of negative clauses implied by Σ consisting of less than k literals.

In contrast, $\mathscr{P}_3 = \langle \boldsymbol{L},$ length is greater than $k \rangle$ is not a stable production field. For example, if $k = 2$ and $\boldsymbol{L} = \{p(a), q(b), \neg r(c)\}$, then $C = p(a) \vee q(b)$ subsumes $D = p(a) \vee q(b) \vee \neg r(c)$, and D belongs to $\mathscr{P}_3$ while C does not.

Note that the empty clause $\square$ is the unique clause in $\mu Th_{\mathscr{P}}(\Sigma)$ iff Σ is unsatisfiable. This means that *proof finding* is a special case of consequence finding. Stable production fields are practically important [1]. In the condition *Cond*, we can specify the maximum length of each clause, the maximum term depth of any literal in a clause, and so on.

Definition 2. *A clausal tableau T is a labeled ordered tree, where every non-root node of T is labeled with a literal. We identify a node with its label (i.e., a literal) if no confusion arises. If the immediate successors of a node are literals $L_1, \ldots, L_n$, then $L_1 \vee \cdots \vee L_n$ is called a tableau clause. The tableau clause below the root is called the start clause. T is a clausal tableau for a set Σ of clauses if every tableau clause C in T is an instance of a clause D in Σ. In this case, D is called an origin clause of C. A connection tableau is a clausal tableau such that, for every non-leaf node L except the root, there is an immediate successor labeled with $\overline{L}$. A marked tableau T is a clausal tableau such that some leaves are marked with labels* `closed` *or* `skipped`*. The unmarked leaves are called subgoals. A node N in T is said to be solved if either N itself is a marked leaf node or all leaf nodes of branches through N of T are marked. T is solved if all leaves are marked. $skip(T)$ denotes the set of literals of nodes marked with* `skipped`*.*

Notice that $skip(T)$ is a *set*, not a multiset. $skip(T)$ is also identified with a clause. We will abbreviate a marked connection tableau as a *tableau*.

Definition 3. *A tableau T is regular if no two nodes on any branch in T are labeled with the same literal. T is tautology-free if no tableau clause in T is tautology. T is complement-free if no two non-leaf nodes on any branch in T are labeled with complementary literals. A tableau T is skip-regular if there is no node L in T such that $\overline{L} \in skip(T)$. T is TCS-free (Tableau Clause Subsumption free) for a clause set Σ if no tableau clause C in T is subsumed by any clause in Σ other than the origin clauses.*

Notice that the skip-regularity applies to all over a tableau, so it is effective not only for subgoals but also for non-leaf and solved nodes.

Definition 4 (Selection Function [5]). *A selection function ϕ is a mapping assigning a subgoal to every tableau that is not solved. ϕ is said to be depth-first if ϕ selects from any tableau T a subgoal with a maximum depth. ϕ is stable under substitution if, for any tableau T and any substitution σ, $\phi(T) = \phi(T\sigma)$ holds.*

Selection functions are assumed to be stable under substitution in this paper.

Definition 5 (SOL Tableau Calculus [5]). *Let Σ be a set of clauses, C a clause, $\mathscr{P}$ a production field and ϕ a selection functions. An SOL-deduction deriving a clause S from $\Sigma + C$ and $\mathscr{P}$ via ϕ consists of a sequence of tableaux $T_0, T_1, \ldots, T_n$ satisfying that:*

(i) T_0 consists of the start clause C only. All leaf nodes of T_0 are unmarked.

(ii) T_n is a solved tableau, and $skip(T_n) = S$.

(iii) For each T_i $(i = 0, \ldots, n)$, T_i is regular, tautology-free, complement-free, skip-regular, and TCS-free for $\Sigma \cup \{C\}$.

(iv) For each T_i $(i = 0, \ldots, n)$, the clause $skip(T_i)$ belongs to $\mathscr{P}$.

(v) T_{i+1} is constructed from T_i as follows. Select a subgoal K by ϕ, then apply one of the following rules to T_i to obtain T_{i+1}:

(a) **Skip:** *If $skip(T_i) \cup \{K\}$ belongs to $\mathscr{P}$, then mark K with label* `skipped`.

(b) **Skip-factoring:** *If $skip(T_i)$ contains a literal L, and K and L are unifiable with mgu θ, then mark K with* `skipped`, *and apply θ to T_i.*

(c) **Extension:** *Select a clause B from $\Sigma \cup \{C\}$ and obtain a variant $B' = L_1 \vee \cdots \vee L_m$ by renaming. If there is a literal L_j such that $\overline{K}$ and L_j are unifiable with mgu θ, then attach new nodes $L_1, \ldots, L_m$ to K as the immediate successors. Next, mark L_j with* `closed` *and apply θ to the extended tableau.*

(d) **Reduction:** *If K has an ancestor node L, and $\overline{K}$ and L are unifiable with mgu θ, then mark K with* `closed`, *and apply θ to T_i.*

Theorem 1 (Soundness and Completeness of SOL [5]). *(1) If there is an SOL-deduction of a clause S from $\Sigma + C$ and $\mathscr{P}$ via ϕ, then S belongs to $Th_{\mathscr{P}}(\Sigma \cup \{C\})$.*

(2) If a clause F does not belong to $Th_{\mathscr{P}}(\Sigma)$ but belongs to $Th_{\mathscr{P}}(\Sigma \cup \{C\})$, then there is an SOL-deduction of a clause S from $\Sigma + C$ and $\mathscr{P}$ via ϕ such that S subsumes F.

Example 2. *We define a start clause C, a set Σ of clauses and a production field $\mathscr{P}$ as follows. Let ϕ be a selection function.*

$$
\begin{aligned}
C &= p(X) \vee s(X), \\
\Sigma &= \{\, q(X) \vee \neg p(X),\ \neg s(Y),\ \neg p(Z) \vee \neg q(Z) \vee r(Z) \,\}, \\
\mathscr{P} &= \langle \mathscr{L}^{+},\ \textit{length is less than } 2 \rangle.
\end{aligned}
$$

Figure 1 shows three solved tableaux that are derived by SOL-deductions from $\Sigma + C$ and $\mathscr{P}$ via ϕ. In the tableau T_a, the node $p(X)$ is skipped since the positive literal $p(X)$ belongs to $\mathscr{P}$, and $s(X)$

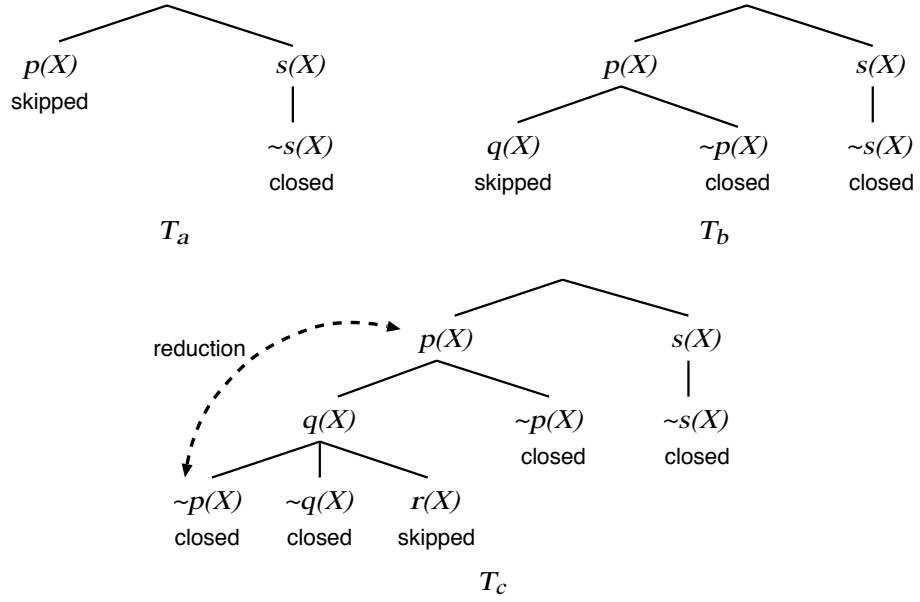

Figure 1: Solved tableaux of Example 2

is extended by using the unit clause $\neg s(Y)$. *Note that* $s(X)$ *cannot be skipped since the production field* $\mathscr{P}$ *limits the maximum length of consequences to one. The derived consequence is* $skip(T_a) = \{p(X)\}$. T_b *shows the other derived tableau whose node* $p(X)$ *is extended by* $q(X) \vee \neg p(X)$, *and the node* $q(X)$ *is skipped. The consequence of* T_b *is* $skip(T_b) = \{q(X)\}$. *In* T_c, *the nodes* $p(X)$ *and* $q(X)$ *are extended by* $q(X) \vee \neg p(X)$ *and* $\neg p(Z) \vee \neg q(Z) \vee r(Z)$ *respectively. The bottom node* $\neg p(X)$ *is closed by reduction with the ancestor* $p(X)$, *and* $r(X)$ *is skipped. The consequence is* $skip(T_c) = \{r(X)\}$. *There are some redundant solved tableaux other than* T_a, T_b *and* T_c. *The consequences of those are subsumed by* $p(X)$, $q(X)$ *or* $r(X)$[1]. *As the results, we can get three new characteristic clauses in this example:* $Newcarc(\Sigma, C, \mathscr{P}) = \{p(X), q(X), r(X)\}$.

3 Pruning Methods for SOL Tableau Calculus

3.1 Local Failure Caching for Length Condition

Local failure caching [9, 10] is an excellent pruning method for refutation finding which can avoid solving repetitiously a subgoal with the same or a more specific substitution. Iwanuma *et al*. [5] reformulated the local failure caching for consequence finding. This procedure is complete if a production field does not imply a maximum length condition. We review the procedure and then show a counter example. We first define the *SOL search tree* to explicitly express all possible SOL-deductions.

Definition 6. *The SOL search tree from* $\Sigma + C$ *and* $\mathscr{P}$ *via* ϕ *is a tree* $\mathscr{T}$ *labeled with tableaux as follows. We identify a node with its label (i.e., a tableau) if no confusion arises. The root of* $\mathscr{T}$ *is a tableau which consists of the start clause C only. Every non-leaf node T in* $\mathscr{T}$ *has as many successor nodes as there are successful applications of a single inference step applied to the selected subgoal in T by* ϕ, *and the successor nodes of T are the respective resulting tableaux. A segment of* $\mathscr{T}$ *is a subtree that contains the nodes explored by* ϕ *from the root to some node.*

[1]The new pruning method *skip-minimality* proposed in Section 3.2 can prune such redundant tableaux.

Definition 7 (Solution and Failure Substitution). *Given a SOL search tree $\mathscr{T}$ and a depth-first selection function. Let T be a tableau in $\mathscr{T}$ and K the selected subgoal in T.*

1. *If T' in $\mathscr{T}$ is a descendant tableau of T such that all branches through K in T' are solved, then the composition $\sigma = \sigma_1 \cdots \sigma_k$ of substitutions applied to the tableau T on the way from T to T' is called a solution of K at T via T'.*

2. *If $\mathscr{T}'$ is an initial segment of $\mathscr{T}$ containing no proof at T' or below it, then the solution σ is named a failure substitution for K at T in $\mathscr{T}'$.*

Definition 8 (Local Failure Caching Procedure for Consequence Finding [5]). *Let $\mathscr{T}'$ be a finite initial segment of a tableau search tree $\mathscr{T}$.*

Step 1: Whenever a subgoal K in a tableau T in $\mathscr{T}'$ has been solved via a tableau T', then the computed solution σ is stored at the node K.

   a. If T' cannot be completed to a solved tableau in $\mathscr{T}'$ and the proof procedure backtracks over T', then σ is turned into a failure substitution.

   b. If T' once has been completed in $\mathscr{T}'$ at a previous stage and the proof procedure backtracks over T' for searching alternative consequences, then continue the backtracking, without adding σ to the failure substitutions.

Step 2: In any alternative solution process of the subgoal K below the search node T, if a substitution $\tau = \tau_1 \cdots \tau_m$ is computed such that one of the failure substitutions stored at the node K is more general than τ, then the proof procedure immediately backtracks.

Step 3: When the search node T (at which K was selected) is backtracked, then all failure substitutions at K are deleted.

We show a counter example for the completeness of the above procedure.

Example 3. *We define a start clause $C = p(X) \vee q(X) \vee r(X)$, a set of clauses $\Sigma = \{ \neg q(X) \vee s(X),\ \neg s(Y) \}$, and a production field $\mathscr{P} = \langle \mathscr{L}^+,$ length is less than $3 \rangle$. We consider finding consequences from this problem. We can apply the Skip operation to $p(X)$ and $q(X)$ in the start clause since they are positive. But we cannot close $r(X)$, because the maximum length of consequences is limited to two (see T_1 in Figure 2). In this case, the local failure caching procedure stores an empty failure substitution $\sigma = \emptyset$ at the node $q(X)$. Since $\emptyset$ is the most general substitution, all the other applicable operations to $q(X)$ are pruned immediately. As the result, we cannot solve this problem and get any consequences. However, if we do not use the local failure caching, we can obtain the consequence $p(X) \vee r(X)$ from the solved tableau T_2 in Figure 2. Hence, the local failure caching is incomplete if there exists a maximum length condition.*

The cause of incompleteness is that the procedure does not consider skipped nodes. We extend the definitions of solution and failure substitution, and propose a complete procedure, called *local failure caching for length condition*.

Definition 9 (Extended Solution and Failure Substitution). *Given a tableau search tree $\mathscr{T}$ and a depth-first selection function. Let T be a tableau in $\mathscr{T}$ and K the selected subgoal in T.*

1. *If T' in $\mathscr{T}$ is a descendant tableau of T such that all branches through K in T' are solved, then the pair $\langle \sigma, s \rangle$, where σ is the composition $\sigma = \sigma_1 \cdots \sigma_k$ of substitutions applied to the tableau T on the way from T to T' and $s = skip(T')$, is called an extended solution of K at T via T'.*

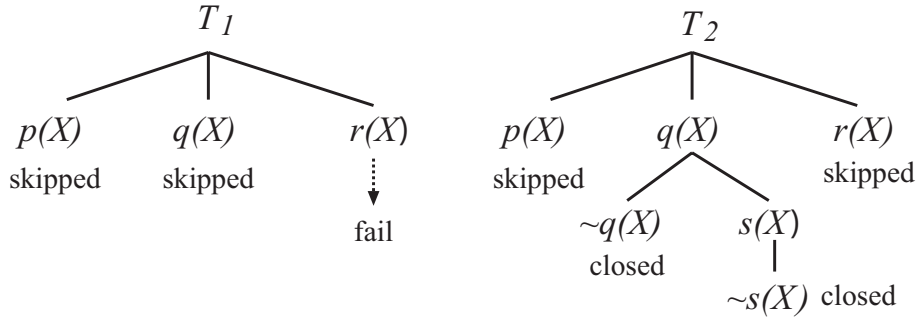

Figure 2: A counter example of local failure caching for consequence finding

2. *If $\mathscr{T}'$ is an initial segment of $\mathscr{T}$ containing no proof at T' or below it, then the extended solution $\langle\sigma, s\rangle$ is named an extended failure substitution for K at T in $\mathscr{T}'$.*

For two extended failure substitutions $\langle\sigma, s_1\rangle$ and $\langle\tau, s_2\rangle$, $\langle\sigma, s_1\rangle$ is more general than $\langle\tau, s_2\rangle$ if $\exists\theta(\sigma\theta = \tau \wedge s_1\theta \subseteq s_2)$, that is, σ is more general than τ and s_1 subsumes s_2. If $\mathscr{P}$ does not have a maximum length condition, then we simply ignore the latter condition, that is, $\langle\sigma, s_1\rangle$ is more general than $\langle\tau, s_2\rangle$ if $\exists\theta(\sigma\theta = \tau)$.

Definition 10 (Local Failure Caching Procedure for Length Condition)**.** *The definition is same as Definition 8 except for replacing terms of "solution" and "failure substitution" with "extended solution" and "extended failure substitution" respectively.*

The completeness of the local failure caching for length condition is given by the following proposition.

Proposition 1. *Let $\mathscr{T}$ be a tableau search tree, T a tableau in $\mathscr{T}$, K the selected subgoal in T, and T_a, T_b descendant tableaux of T. Suppose that $\langle\sigma, s_1\rangle$ is the extended failure substitution for K at T via T_a, and $\langle\tau, s_2\rangle$ is the extended solution of K at T via T_b generated by an alternative process. If T_b has a solved tableau T_{bs} as a descendant node in $\mathscr{T}$, then $\langle\sigma, s_1\rangle$ is not more general than $\langle\tau, s_2\rangle$.*

Proof. We prove the completeness by a reductio-ad-absurdum. We assume that $\langle\sigma, s_1\rangle$ is more general than $\langle\tau, s_2\rangle$, that is, $\exists\theta(\sigma\theta = \tau \wedge s_1\theta \subseteq s_2)$. Let S_a and S_{bs} be the subtableaux with root K in T_a and T_{bs} respectively (see Figure 3). Then, we replace S_{bs} in T_{bs} with $S_a\theta$, and denote the resulting tableau as T'_{bs}, which is solved and satisfies the maximum length condition. The reason is as follows. Let $\mathscr{N}$ be the set of selected subgoals on the way to T_{bs} after T_b. We extract the set $\mathscr{N}_f$ of nodes from $\mathscr{N}$ which are skipped by the Skip-factoring operation. There might exist the skipped nodes in $\mathscr{N}_f$ which have the factoring target in S_{bs} but not in $S_a\theta$. However, the number of such nodes is at most $|s_2 \setminus s_1\theta|$ since $s_1\theta \subseteq s_2$. Therefore, we can apply the Skip operation instead of the Skip-factoring to such nodes. As the results, $skip(T'_{bs})$ satisfies the maximum length condition.

This means that if the subtableau S_a was solved with the substitution $\sigma\theta$ instead of σ, then there should exist a solved tableau in $\mathscr{T}$ below T_a. Hence, if S_a is solved with the more general substitution σ, then there must be a solved tableau in $\mathscr{T}$ below T_a. But this contradicts the assumption of $\langle\sigma, s_1\rangle$ being a failure substitution. □

We consider solving Example 3 by using local failure caching for length condition. Since we cannot close $r(X)$ in T_1, the extended failure substitution $\langle\emptyset, \{p(X), q(X)\}\rangle$ is stored to the node $q(X)$. When the alternative process solves $q(X)$ using two extension operations (see T_2 in Figure 2), the extended solution substitution $\langle\emptyset, \{p(X)\}\rangle$ is stored to $q(X)$. Now, $\langle\emptyset, \{p(X), q(X)\}\rangle$ is not more general than $\langle\emptyset, \{p(X)\}\rangle$

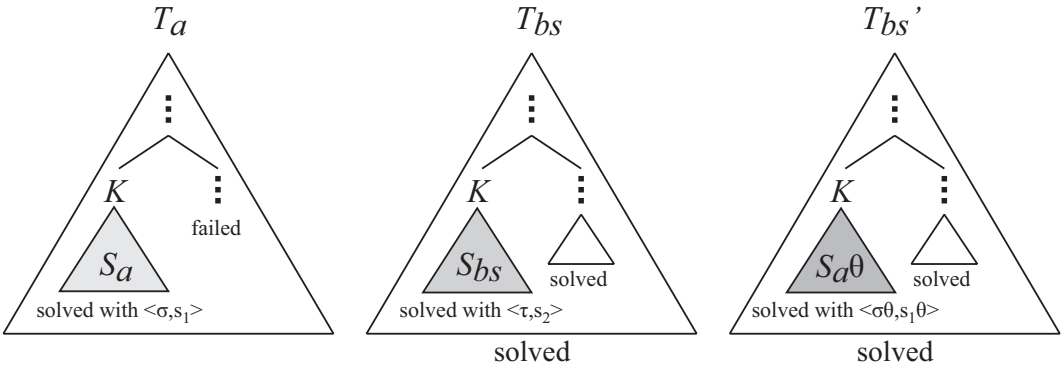

Figure 3: Completeness of local failure caching for length condition

because $\{p(X), q(X)\} \not\subseteq \{p(X)\}$. Hence, we can obtain the solved tableau $\mathcal{T}_2$ by skipping $r(X)$ and get the consequence $p(X) \vee r(X)$.

Since the Skip operation never instantiates a tableau, local failure caching is significantly effective for the SOL tableau calculus. Moreover, using the maximum length condition is important for finding simple and short consequences in practical time. In such a situation, local failure caching for length condition is an essential technique for practical use.

3.2 Skip-minimality

The task of consequence finding is to find all minimal consequences with respect to subsumption. This means that even if we find a solved tableau, we have to continue searching other solved tableaux for finding all minimal consequences. In proof finding, we can halt the computational process immediately if one refutation is found. This is an important difference between consequence finding and proof finding. Of course, if a refutation is found in a consequence finding process, then we can stop the computation immediately since the refutation is the most general consequence.

Skip-minimality improves the efficiency of such a consequence enumeration process. This method can prune unsolved tableaux which will generate only non-minimal consequences with respect to subsumption by using the consequences derived from already solved tableaux.

Definition 11 (Skip-Minimality)**.** *Let* Γ *be a set of clauses. A tableau* T *is skip-minimal for* Γ *if the clause* $skip(T)$ *is not properly subsumed by any clause in* Γ.

Suppose that Γ is a set of consequences derived by some SOL-deductions in a consequence enumeration process. If a tableau T is not skip-minimal for Γ, then T and all its descendant tableaux can be pruned immediately. Skip-minimality is complete since the four kinds of operations in Definition5 never generalize $skip(T)$. If we find general and short consequences in early stages, then skip-minimality is very effective since it prevents generating many redundant tableaux.

4 Search Strategy

Limiting the maximum length of consequences is important. When there is no limit, the branching factor of a SOL search tree increases and the size of the tree grows exponentially. However, it is difficult to know the appropriate length condition in advance. One of the solutions is to execute a search process with incrementing the maximum length limit iteratively, like DFID (*depth-first iterative deepening*) search strategy.

We propose the *consequence iterative lengthening* (CIL) strategy. Let P be a consequence enumeration process and l a length limit of consequences (initially $l = 0$). CIL strategy executes the process P with the length limit l repeatedly, incrementing l for each iteration. If l reaches a given limit or satisfies the condition $s < l$, then the iteration stops, where s is the maximum number of used Skip operations in every tableau.

Since CIL strategy visits the same tableaux in a SOL search tree multiple times, it may seem wasteful. However, it turns out to be not so costly, because the number of tableaux in the SOL search tree increases exponentially in proportion to incrementing the length limit of consequences. We show the property of CIL strategy in Section 5 experimentally.

5 Experimental Results

We have implemented skip-minimality, local failure caching for length condition and CIL strategy in SOLAR [12] which is a Java implementation of the SOL tableau calculus. Table 1 shows the experimental results for our proposed methods. We used some problems in the TPTP library v2.5.0 [14] as consequence finding problems, which are satisfiable instances (for example, "BOO008-3.p" is the file name of the problem). There are 1,092 satisfiable problems in the library and 24 categories contain such satisfiable instances. We selected 10 categories and chose one satisfiable problem from each category. We defined a production field in each problem as $\langle \mathscr{L},\ \text{length} \leq x \rangle$, where $\mathscr{L}$ is the set of all literals and the length condition is given as "len $\leq x$" in Table 1. "dep $\leq y$" means that we limited the maximum depth of tableaux to y. "sr", "sm" and "lfc" denote skip-regularity, skip-minimality and local failure caching for length condition respectively. "#C." and "#Steps" are the number of found consequences and the total number of derived tableaux (that is the number of nodes in a SOL search tree) respectively. The experiments were done on a Core Duo (1.66GHz) machine with 2GB memory. "t.o." means that the problem could not solve within 600 CPU seconds.

Table 1 shows that our proposed pruning methods have a great ability for reducing the search space and improving the speed. In particular, skip-minimality is very effective for solving GRP123-1.005. If we do not use skip-minimality to solve the problem, then we cannot solve the problem within 600 CPU seconds. In MSC009-1 and NLP026-1, local failure caching for length condition shows a high pruning effect. In BOO008-3 and LCL168-1, skip-minimality is not effective because the current implementation of clause-subsumption checking in SOLAR is naive. The improvement of the clause-subsumption check algorithm is one of the important future work. The rightmost row of Table 1 indicates that CIL strategy has almost no overhead. For example, in KRS005-1, CIL executes DFID five times with the maximum length limit of consequences from 0 to 4. However, the number of inferences and execution time are not so different than DFID without CIL.

6 Conclusion and Future Work

We have proposed new complete pruning methods and a search strategy for SOL tableau calculus. Local failure caching for length condition and skip-minimality can avoid producing many redundant tableaux. CIL is helpful to a user for finding short and simple consequences preferentially in a limited time. The completeness of local failure caching for length condition supports CIL strategy, and the pruning power of skip-minimality is promoted by CIL since it generates short consequences in early stages.

Currently, SOLAR does not have a special mechanism for handling equality. In order to solve a problem with equality efficiently, the development of the complete equality handling mechanism for consequence finding is one of the important future work.

Table 1: Experimental results

Problem	Params	Pruning Methods	#C	DFID		DFID + CIL	
				#Steps	Time [sec]	#Steps	Time [sec]
BOO008-3.p	dep ≤ 3 len ≤ 5	none	831	308,437	2.3	355,371	2.4
		sr	831	303,424	2.3	350,358	2.5
		sm	831	308,437	7.2	355,371	7.4
		lfc	831	299,195	2.4	342,711	2.5
		all	831	295,944	7.3	339,460	7.7
GRP123-1.005.p	dep ≤ 5 len ≤ 1	none	-	-	t.o.	-	t.o.
		sr	-	-	t.o.	-	t.o.
		sm	65	822,283	4.5	935,057	4.9
		lfc	-	-	t.o.	-	t.o.
		all	65	784,888	4.8	897,662	5.2
HWV034-1.p	dep ≤ 10 len ≤ 3	none	4	7,187,849	18.3	7,409,286	18.8
		sr	4	4,392,088	11.6	4,571,153	12.1
		sm	4	6,686,239	17.6	6,907,676	18.1
		lfc	4	4,733,454	14.2	4,952,443	15.1
		all	4	3,112,295	10.6	3,288,912	11.0
KRS005-1.p	dep ≤ 4 len ≤ 4	none	131	160,767,095	438.5	165,203,316	441.4
		sr	131	44,631,879	136.2	46,621,089	136.4
		sm	131	46,303,085	139.3	47,444,019	141.5
		lfc	131	106,297,270	319.2	110,027,318	333.5
		all	131	8,730,512	36.8	9,166,063	37.9
LCL168-1.p	dep ≤ 5 len ≤ 1	none	2,091	39,103	33.3	40,752	33.4
		sr	2,091	38,615	33.5	40,264	33.6
		sm	2,091	39,103	35.4	40,752	35.6
		lfc	2,091	38,629	33.4	40,278	33.8
		all	2,091	38,141	35.5	39,790	36.0
MSC009-1.p	dep ≤ 4 len ≤ 4	none	43	11,160,097	23.4	11,442,019	24.7
		sr	43	7,453,146	16.7	7,705,908	17.0
		sm	43	8,835,505	20.1	9,117,427	20.4
		lfc	43	4,786,900	12.0	4,915,009	12.2
		all	43	1,992,519	6.4	2,107,194	6.7
NLP026-1.p	dep ≤ 5 len ≤ 2	none	15	129,692,801	301.8	130,198,526	307.0
		sr	15	72,182,022	178.0	72,667,948	176.5
		sm	15	129,025,421	309.8	129,531,146	305.0
		lfc	15	8,218,827	23.7	8,716,795	25.6
		all	15	6,391,292	20.8	6,870,050	22.8
PUZ001-3.p	dep ≤ 10 len ≤ 3	none	26	34,997,018	134.7	39,946,809	145.8
		sr	26	14,845,990	59.6	17,823,280	66.4
		sm	26	731,711	3.8	1,157,985	5.0
		lfc	26	33,799,877	143.4	38,495,682	154.7
		all	26	458,832	3.2	759,780	4.3
SET777-1.p	dep ≤ 6 len ≤ 2	none	3	3,906,678	9.3	3,994,168	9.6
		sr	3	1,118,100	3.4	1,158,829	3.5
		sm	3	3,502,145	8.5	3,583,800	8.7
		lfc	3	2,251,658	7.1	2,302,211	7.3
		all	3	656,933	2.8	680,619	2.9
SYN084-1.p	dep ≤ 3 len ≤ 4	none	5	1,335,866	7.0	1,622,436	8.0
		sr	5	1,210,436	6.9	1,486,172	7.8
		sm	5	328,239	2.5	441,974	2.7
		lfc	5	1,335,866	7.3	1,621,274	8.2
		all	5	285,808	2.4	395,557	2.7

Ray and Inoue [13] have proposed a transformation method for unstable production fields, which converts these into stable ones. This method greatly helps to define a desired production field and to prune an unnecessary search space.

References

[1] K Inoue. Linear resolution for consequence finding. *Artificial Intelligence*, 56:301–353, 1992.

[2] Katsumi Inoue. Automated abduction. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, volume 2408 of *Lecture Notes in Computer Science*, pages 311–341. Springer, 2002.

[3] Katsumi Inoue. Induction as consequence finding. *Machine Learning*, 55(2):109–135, 2004.

[4] Katsumi Inoue, Koji Iwanuma, and Hidetomo Nabeshima. Consequence finding and computing answers with defaults. *Journal of Intelligent Information Systems*, 26(1):41–58, 2006.

[5] K Iwanuma, K Inoue, and K Satoh. Completeness of pruning methods for consequence finding procedure SOL. In *Proceedings of FTP-2000*, pages 89–100, 2000.

[6] Koji Iwanuma and Katsumi Inoue. Conditional answer computation in SOL as speculative computation in multi-agent environments. In *Proceedings of the Third International Workshop on Computational Logic in Multi-Agent Systems (CLIMA-02)*, pages 149–162, 2002.

[7] Koji Iwanuma and Katsumi Inoue. Minimal answer computation and SOL. In *Proceedings of the Eighth European Conference on Logics in Artificial Intelligence (JELIA 2002)*, volume 2424 of *Lecture Notes in Artificial Intelligence*, pages 245–257. Springer, 2002.

[8] Char Tung Lee. *A completeness theorem and a computer program for finding theorems derivable from given axioms*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, 1967.

[9] R. Letz, C. Goller, and K. Mayr. Controlled integration of the cut rule into connection tableau calculi. *Journal of Automated Reasoning*, 13(3):297–338, 1994.

[10] Reinhold Letz. Clausal tableaux. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction - A Basis for Applications*, volume I: Foundations, pages 39–68. Kluwer, Dordrecht, 1998.

[11] Donald W. Loveland. *Automated Theorem Proving: a logical basis*. North-Holland Publishing Company, Amsterdam, 1978.

[12] Hidetomo Nabeshima, Koji Iwanuma, and Katsumi Inoue. SOLAR: A consequence finding system for advanced reasoning. In *Proceedings of Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2003)*, pages 257–263, 2003.

[13] Oliver Ray and Katsumi Inoue. A consequence finding approach for full clausal abduction. In *Proceedings of the 10th International Conference on Discovery Science (DS 2007)*, pages 173–184, 2007.

[14] Geoff Sutcliffe and Christian Suttner. The TPTP problem library for automated theorem proving v2.5.0. http://www.tptp.org/, 2002.