A Contradiction Separation Dynamic Deduction Algorithm Based on Optimized Proof Search

Feng Cao1,2,*, Yang Xu2,3, Shuwei Chen2,3,*, Jian Zhong1,2, Guanfeng Wu2,3

1 School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, Sichuan, P.R. China
2 National-Local Joint Engineering Laboratory of System Credibility Automatic Verification, Southwest Jiaotong University, Chengdu 610031, Sichuan, P.R. China
3 School of Mathematics, Southwest Jiaotong University, Chengdu 610031, Sichuan, P.R. China

ABSTRACT

Most of the advanced first-order logic automated theorem proving (ATP) systems adopt binary resolution methods as the core inference mechanism, where only two clauses are involved and a complementary pair of literals are eliminated during each deduction step. Recently, a novel multi-clause inference rule is introduced along with its soundness and completeness, which is called standard contradiction separation rule (in short, S-CS rule) and allows multiple (two or more) clauses to be involved in each deduction step. This paper introduces and evaluates the application of S-CS rule in first-order logic ATP. Firstly, it analyzes several deduction methods of S-CS rule. It is then focused on how this multi-clause deduction theory can be achieved through forming a specific and effective algorithm, and finally how it can be applied in the top ATP systems in order to improve their performances. Concretely, two novel multi-clause S-CS dynamic deduction algorithms are proposed based on optimized proof search, including related heuristic strategy; then the application method applied in the state of the art ATP system Eprover (the version of Eprover 2.3) is introduced. Eprover with the proposed multi-clause deduction algorithms are evaluated through the FOF division of the CASC-19 (in 2018) ATP system competition. Experimental results show that Eprover with the proposed multi-clause deduction algorithms outperform the plain Eprover itself to a certain extent.

1. INTRODUCTION

Automated reasoning is an important research branch in the field of artificial intelligence, and it has been successfully applied in many areas [1], e.g., mathematics [2], formal verification [3]. Compared with program testing, formal verification is an important research work in the field of current information security, and automated reasoning plays an important role in formal verification. The automated theorem proving (ATP) in first-order logic is an important research work in the field of automated reasoning. ATP systems mainly adopt saturation [4,5], inference rule [6–12], superposition calculus [13] and heuristic strategies [14–18], and have strong capability after several decades of development, e.g., Eprover [19], Vampire [20].

Saturation is a deduction algorithm for most famous first-order logic ATP systems [19–21]. It is represented by given-clause algorithm, that can be divided into discount [4] and otter [5]. The basic idea is that the algorithm divides the whole clause set into passive and active. In each deduction step, a clause (called as current clause) is selected from passive by heuristic strategy and put into active. The inference rules are applied only in active, the generated new clauses are finally put into passive. Repeat this operation until an empty clause is generated or timeout. The given-clause algorithm can effectively plan proof search for theorem proving and is easy to implement. On the other hand, since the generated clauses are put into passive in each deduction step, the number of clauses in passive increases rapidly. As a result, it makes the effective selection of an optimal current clause in such a large number of clauses very difficult. In other words, current ATP systems have a great dependence on clause selection that is experimental to some extent. It is therefore of great significance to research new inference methods that can reduce the dependence on heuristic strategy.

As a major breakthrough in the field of automated reasoning, the resolution principle [6] and its refinements have become the main inference methods for current ATP systems. Assumed $C_1, C_2$ are two clauses in first-order logic without common variable, where $C_1 = P_1 \lor P_2 \lor \ldots \lor P_m$, $C_2 = Q_1 \lor Q_2 \lor \ldots \lor Q_n$. If two literals of $P_1$ and $Q_1$ exist a unified complementary and the most general unifier $\sigma$, then $R(C_1, C_2) = (C_1^{\sigma} - P_1^{\sigma}) \lor (C_2^{\sigma} - Q_1^{\sigma})$ is called binary resolvent of $C_1$ and $C_2$. Binary resolution [6] is the core of resolution principle, and other resolution methods [22–25] presented later are almost developed based on binary resolution. The state of the art resolution-based ATP systems mainly adopt binary resolution [6], unit-resulting resolution [9] (UR resolution), hyper-resolution [10], equality resolution [26], and conflict resolution [12]. From the literature review, we noticed that these resolution rules only have 2 clauses involved and eliminated only one complementary pair of literals in each deduction step, so the essence of these inference rules...
are still binary resolution. Binary resolution is a static method, only one complementary pair of literals eliminated in each deduction step, so the binary resolvent often has many literals, which will affect deduction efficiency. Due to the restriction of binary resolution, it is of great significance to research whether it can be improved, such as multi-clause resolution method.

Aiming at handling multiple input clauses synergistically in each deduction step, the S-CS (standard contradiction separation) rule [27] was proposed as a multi-clause dynamic deduction method, which can fully exert the synergized effect between more than two input clauses. It can eliminate a literal set from the input clauses by separating a standard contradiction, and the generated clauses usually have few literals by dynamic control. In addition, the constructed standard contradiction has a guidance effect on the selection of subsequent input clauses, which reduces the dependence on clause selection by heuristic strategy. It is an important development of the traditional binary resolution method and the detailed introduction of S-CS rule in the proposition logic and the first-order logic can be found in the published paper [27]. On the other hand, the specific deduction algorithms, strategies to support this theory, and its application have not yet been covered. How to design an effective S-CS deduction algorithm and apply it effectively to ATP is the main motivation of the present work, which is necessary and important.

In this paper, we focus on designing and applying an effective multi-clause dynamic deduction algorithm according to S-CS rule. By analyzing the characteristics of S-CS dynamic deduction, and in order to make the generated clauses contain few literals, two S-CS dynamic deduction algorithms based on optimized proof search are proposed. This proposed multi-clause dynamic deduction algorithms are applied to the state of the art ATP system Eprover, and the experimental results evaluation are introduced.

The remainder of the paper is structured as follows. In Section 2, the S-CS rule is reviewed and its deduction characteristics are analyzed in detail. In Section 3, the design of two multi-clause dynamic deduction algorithms according to S-CS rule are proposed. The application of the proposed algorithms in the state of the art ATP system is introduced in Section 4. Section 5 shows the performance of the improved ATP systems and gives detailed analysis. In last Section, the conclusions and future work are provided.

2. S-CS RULE

This section firstly provides a review of basic concepts of S-CS rule, then gives a detailed analysis of its deduction characteristics.

Definition 2.1. [27] Suppose a clause set \( S = \{C_1, C_2, \ldots, C_m\} \) in first-order logic, where the following hold:

1. There does not exist the same variables among \( C_1, C_2, \ldots, C_m \) (if there exist the same variables, a rename substitution can be applied to make them different);
2. For each \( C_i \) (\( i = 1, 2, \ldots, m \)), a substitution \( \sigma_i \) can be applied to \( C_i \) (\( \sigma_i \) could be an empty substitution) and the same literals merged after substitution, denoted as \( C_i^{\sigma_i} \), \( C_i \) can be partitioned into two sub-classes \( C_i^{\sigma_i^+} \) and \( C_i^{\sigma_i^-} \) such that:

\[
C_i^{\sigma_i^+} = C_i^{\sigma_i^-} \lor C_i^{\sigma_i^+}, \quad \text{where } C_i^{\sigma_i^-} \text{ and } C_i^{\sigma_i^+} \text{ do not share the same literal, } C_i^{\sigma_i^+} \text{ can be an empty clause, } C_i^{\sigma_i^-} \text{ cannot be an empty clause; moreover,}
\]

\[
(2) \text{For any } (x_1, \ldots, x_n) \in \prod_{i=1}^{m} C_i^{\sigma_i^{\pm}}, \text{there exists at least one complementary pair among } (x_1, \ldots, x_n), \text{and } C_i^{\sigma_i^{\pm}} \text{ is called a separated standard contradiction (S-SC)}.
\]

The resulting clause \( \forall_{i=1}^{m} C_i^{\sigma_i^+} \), denoted as \( \Psi \sigma \left( C_1, \ldots, C_m \right) \) (here “s” means “standard,” \( \sigma = \forall_{i=1}^{m} \sigma_i \)), is called an S-CS clause of \( C_1, C_2, \ldots, C_m \).

The inference rule that produces a new clause \( \Psi \sigma \left( C_1, \ldots, C_m \right) \) is called an S-CS rule in first-order logic.

Definition 2.2. [27] Suppose a clause set \( S = \{C_1, C_2, \ldots, C_m\} \) in first-order logic. \( \Phi_1, \Phi_2, \ldots, \Phi_r \) is an S-CS based dynamic deduction sequence from \( S \) to a clause \( \Phi_0 \), if

(1) \( \Phi_1 \in S \) or
(2) there exist \( r_1, r_2, \ldots, r_k < i, \Phi_i = \Psi \sigma \left( \Phi_{r_1}, \Phi_{r_2}, \ldots, \Phi_{r_k} \right) \).

Where \( \sigma_i = \forall_{j=1}^{k} \sigma_j \), \( \sigma_i \) is a substitution to \( \Phi_{r_j} \), \( j = 1, 2, 3, \ldots, k, i \in \{1, 2, \ldots, f\} \).

S-CS rule takes multiple (two or more) clauses as input, selects a subset of the literals from each input clause to build a standard contradiction, and infers the disjunction of the non-selected literals of the input clauses. In contrast to the chained inference rules, such as UR resolution, hyper-resolution, linear resolution, and conflict resolution, they can also be expressed in the form of handling multiple input clauses and generated a new clause. However, the essential difference is that only two clauses are involved and only a complementary pair of literals are eliminated in each deduction step for the chained inference rules, with regardless of the synergized effect between more than two clauses. S-CS rule handles multiple (two or more) input clauses by separating a standard contradiction from the input clauses in each deduction step. The standard contradiction is not simply multiple complementary pairs of literals, but a literal set as long as there are a complementary pair of literals. Therefore, in contrast to binary resolution method, S-CS rule has a stronger literals elimination ability in each deduction step.

Example 2.1. Let \( S = \{C_1, C_2, C_3, C_4\} \) be a clause set in first-order logic, where \( C_1 = P_1(a) \lor P_2 \left( b, f(b) \right) \), \( C_2 = P_2 \left( a, b \right) \lor P_3 \left( b, f(b) \right) \), \( C_3 = \sim P_1(a) \lor P_1(b) \lor \sim P_2 \left( x_1, x_2 \right) \lor P_3 \left( x_1, x_3 \right) \), \( C_4 = P_1(b) \lor \sim P_1 \left( x_1 \right) \lor \sim P_2 \left( x_2, x_3 \right) \lor \sim P_3 \left( a, f(b) \right) \). Here, \( a, b \) are constants, \( f \) is a function symbol, \( x_1, x_2, x_3 \) are variables.

Applying S-CS rule to the 4 input clause \( C_1, C_2, C_3, C_4 \), we obtain an S-CS clause involving 4 clauses: \( C_5 = P_3 \left( a, b \right) \lor P_3 \left( a, f(b) \right) \), the S-SC is \( \left( P_1(a) \land P_2(a, b) \right) \lor \left( \sim P_1(a) \land \sim P_2(a, b) \lor P_3(a, f(b)) \right) \land \left( \sim P_1(a) \land P_2(a, b) \lor P_3(a, f(b)) \right) \).

In the process of S-CS deduction for \( C_1, C_2, C_3, C_4 \) in sequence, when \( C_3 \) as an input clause is involved, \( C_3 \) is eliminated 2 literals by two complementary pairs of literals \((\sim P_1(a), P_2(a)), \sim P_2(\left(a, x_1, b, x_2\right), P_2(a, b))) \), when \( C_4 \) as an input clause is involved, \( C_4 \) is eliminated 3 literals by three complementary pairs of literals \((\sim P_1(a), P_2(a)), (\sim P_2(a, x_1, b, x_2), P_2(a, b)), (P_3(\left(a, x_1, f(b), x_3\right), P_3(a, f(b)))) \), it is quite different from binary resolution method in each deduction step. Therefore, S-CS
dynamic deduction can achieve a greater degree of literals elimination than binary resolution method, and thus the S-CS clauses usually have fewer literals. As shown in Example 2.1, the four input clauses have 12 literals, and 8 literals are eliminated by separating a standard contradiction, generates an S-CS clause only with 2 literals after the same literals are merged.

According to the definition of S-CS rule and the illustration of Example 2.1, the S-CS deduction has multi-clause, dynamic, synergized, controllable, guided characteristics [28], so it is a challenge to design an effective S-CS dynamic deduction algorithm. For theorem proving by S-CS rule, the first input clause and the subsequent input clauses are optionally selected, and even the same input clause can be repeatedly participated in the process of S-CS deduction. The process of input clauses participating in S-CS deduction is dynamic, and the deduction can be halted through the number of input clauses or the measures of the S-CS clause generated in each deduction step. Because the S-CS dynamic deduction is performed by continuously selecting input clauses and generating S-CS clauses by separating different standard contradictions, the constructed standard contradiction has synergized deduction effect, and has a guidance effect on the selection of subsequent input clauses. Therefore, the dependency of clause selection strategy is reduced. At the same time, the clause selection of the S-CS deduction is controllable, that is, when an input clause participates in S-CS deduction and generates an S-CS clause, it can be evaluated whether the input clause satisfies the user-defined deduction requirements by the measures of the S-CS clause. When the requirements are not satisfied, the input clause can be reselected until the deduction meets the requirements. In addition, a selected literal from the input clause used for constructing a standard contradiction is also controllable, that is because different construction methods based on the literals from input clauses can separate different standard contradictions.

Example 2.2. Let \( S = \{ C_1, C_2, C_3, C_4, C_5, C_6 \} \) be a clause set in first-order logic, where \( C_1 = P_1(a_1, a_2), C_2 = \sim P_1(x_22, x_31) \lor P_1(x_21, x_22), C_3 = \sim P_2(x_31, a_2) \lor \sim P_2(x_31, a_1), C_4 = \sim P_2(x_43, x_41) \lor P_1(x_41, x_42) \lor \sim P_2(x_43, x_42), C_5 = \sim P_1(x_51, x_52) \lor P_1(f_1(x_51, x_52), x_52), C_6 = \sim P_1(x_61, x_62) \lor P_2(f_2(x_61, x_62), x_61). \) Here, \( a_1, a_2 \) are constants, \( f_1 \) is a function symbol, \( x_22, x_23, x_31, x_41, x_42, x_43, x_51, x_52, x_61, x_62 \) are variables.

In the process of S-CS dynamic deduction, the input clause allows for the absence of unification with the literals in the constructed standard contradiction, so the efficiency of S-CS deduction with the irregular selection of input clauses is very low. Generally, for the resolution calculus, clauses with a smaller number of literals usually have higher deduction efficiency, because their resolvent often contains fewer literals. Follow this principle and according to the characteristics of S-CS deduction, we use S-CS rule for \( C_1, C_2, C_3, C_5, C_6, C_3 \), and obtain the S-CS clause:

\[
C_7 = \{ (P_1(a_1, a_2)) \wedge (\sim P_1(a_1/x_22, a_2/x_22) \lor P_1(a_2/x_21, a_1/x_22)) \wedge (\sim P_1(a_1/x_51, a_2/x_52) \lor P_2(f_1(a_1/x_51, a_2/x_52), x_52)) \wedge (\sim P_1(a_1/x_61, a_2/x_62) \lor P_2(f_1(a_1/x_61, a_2/x_62), x_61)) \wedge (\sim P_2(f_1(a_1, a_2)/x_31, a_1) \lor \sim P_2(f_1(a_1, a_2)/x_31, a_1)) \}
\]

The design of the two algorithms are described in detail in this paper, and the comparative analysis of experimental results are given.

3. DESIGN OF S-CS DYNAMIC DEDUCTION ALGORITHM BASED ON OPTIMIZED PROOF SEARCH

The S-CS dynamic deduction selects input clauses in clause set, selects literals of the input clauses to construct standard contradictions, and generates S-CS clauses. According to the evaluation of the generated S-CS clause in each deduction step, it is decided whether the measures of the S-CS clause satisfies the user-defined requirement, if the requirements are not satisfied, it need to go back to the last deduction step and reselect a new input clause to participate in the deduction. Repeat this operation and generate S-CS clauses until the exit conditions are met. Therefore, before designing the S-CS dynamic deduction algorithm, it is necessary to determine how to select the input clause, how to select a literal in the input clause to construct a standard contradiction, how to evaluate a deduction step based on the S-CS clause, how to perform the retreat of the deduction step if current step does not satisfy the user-defined requirements, and how to exit S-CS deduction.

3.1. Clause Selection Strategy

Clause selection strategy is used to select an input clause for S-CS dynamic deduction, it is implemented mainly based on the following clause attributes:
1. **Acceptable deduction weight.** Acceptable deduction weight of an input clause is the number of times that the clause has participated in an acceptable S-CS dynamic deduction. The acceptable deduction means the deduction generates an S-CS clause that meets the user-defined requirements (e.g., maximum number of literals, maximum term depth).

2. **Unacceptable deduction weight.** Unacceptable deduction weight of an input clause is the number of times that the clause has participated in an unacceptable S-CS dynamic deduction. An unacceptable deduction means the deduction generates an S-CS clause that does not meet the user-defined requirements, and it need to go back to the last deduction step.

3. **The number of literals.** The number of literals is a most important attribute of an input clause. When an input clause with few literals participates in S-CS dynamic deduction, it is likely that the generated S-CS clause also has few literals by separating a standard contradiction from the input clauses.

4. **Clause weight.** Clause weight is used to characterize the statistics of symbols in an input clause. Set the initial weight of each symbol (including constant, variable, function) equal to 1, clause weight is equal to the sum of each symbol weight in the input clause.

5. **Function symbol weight.** S-CS rule is a multi-clause deduction method, which can handle multiple input clauses (two or more) in each deduction step. Furthermore, one or more literals in an input clause can be used for constructing a standard contradiction, which causes the variable to be substituted for unification multiple times, thereby greatly increasing the generated S-CS clause weight, wherein the main reason is the variable substitution in a function symbol. Function symbol weight is used to characterize the statistics of symbols in a function symbol. Set the initial weight of each symbol (including constant, variable, function) equal to 1, function symbol weight is equal to the sum of each symbol weight in the function symbol.

Clause selection strategy selects an input clause to participate in S-CS dynamic deduction based on the priority order with multiple attributes of the clause. For example, the priority order is: **The number of literals > Acceptable deduction weight > Unacceptable deduction weight.** That means the clause with the smallest number of literals is preferentially selected. When the clauses have the same number of literals, then the clause with the smallest Acceptable deduction weight is preferentially selected. When the clauses have the same number of literals and the same Acceptable deduction weight, then the clause with the smallest Unacceptable deduction weight is preferentially selected. When the clauses have all the same of the three attributes, the clause is selected by the storage order.

**3.2. Literal Selection Strategy**

Literal selection strategy is used to select a literal in the input clause to construct a standard contradiction. The selected literal is different, and the constructed standard contradiction is also different. It is implemented mainly based on the following literal attributes:

1. **Acceptable deduction weight.** Acceptable deduction weight of a literal is the number of times that the literal used for constructing standard contradictions has participated in an acceptable S-CS dynamic deduction.

2. **Unacceptable deduction weight.** Unacceptable deduction weight of a literal is the number of times that the literal used for constructing standard contradictions has participated in an unacceptable S-CS dynamic deduction.

3. **The number of complementary predicates.** In the process of S-CS multi-clause deduction, it is used to evaluate the potential ability of a literal used for separating standard contradictions. That is, if the predicate in a literal has a large number of complementary predicates in clause set, there will be many input clauses can be selected to participate in S-CS dynamic deduction.

4. **The number of variables.** In the process of S-CS multi-clause deduction, the literal in an input clause used for constructing standard contradictions can be reused, therefore, if a literal contains more variables, the stronger ability of separating standard contradictions has.

5. **Literal type.** If the same variable comes from different literals in a clause, then the variable is called a shared variable. The literal which contains shared variable is called a shared literal. Otherwise, the literal is called an independent literal. The literals are marked as independent or shared. In the process of S-CS dynamic deduction, an independent literal is used for separating standard contradictions, it will not affect the left literals form the same input clause which has added to S-CS clause.

Literal selection strategy selects a literal to separate standard contradictions based on the priority order with multiple attributes of the literal, the priority order is the same as that of clause selection strategy.

**3.3. S-CS Deduction Step Evaluation Strategy**

S-CS deduction step evaluation strategy is used to evaluate each S-CS deduction step, whether it is acceptable or unacceptable according to the attributes of the generated S-CS clause. This evaluation method is based on the assumption that the attribute of an S-CS clause exceeds the set requirement, it will not contribute to find refutation. If the attribute of an S-CS clause does not satisfy the set requirement, it needs to go back to the last deduction step, and reselects a new input clause to participate in the deduction. There are two attributes of the S-CS clause which is used to evaluate each deduction step.

1. **The maximum number of literals.** It is used to control the maximum number of literals in S-CS clause. When the number of literals exceeds the set threshold, the S-CS dynamic deduction will reselect a new input clause to participate in the deduction, until the number of literals in the generated S-CS clause meets the threshold requirement.

2. **Maximum term depth.** It is used to control the maximum term depth in S-CS clause. When the maximum term depth exceeds the set threshold, the S-CS dynamic deduction will reselect a new input clause to participate in the deduction, until the
maximum term depth in the generated S-CS clause meets the threshold requirement.

### 3.4. S-CS Dynamic Deduction Algorithm Based on Optimized Proof Search

The S-CS deduction is dynamic and controllable, it is a process of multi-clause deduction which is combined with a lot of deduction steps. In each deduction step, the clause selection strategy and the literal selection strategy are used to select an input clause and select which literal from the input clause to construct a standard contradiction, and evaluate the deduction step whether it is acceptable or unacceptable by the attributes of the generated S-CS clause. In order to effectively optimize the proof search, an S-CS dynamic deduction algorithm for making the generation of S-CS clause a slowly increasing process in terms of the number of literals is proposed (SCSAN). At the same time, based on optimized proof search, taking care of the literal in an input clauses exists many ways to construct different standard contradictions, an S-CS dynamic deduction algorithm of the repeated use of the input clauses is also proposed (SCSA).

The pseudo-code for the algorithm is described in Figure 1 below. The corresponding algorithm is described in detail as follows:

**Step 1:** According to clause selection strategy, select an input clause in clause set (marked as initial input clause \( C_0 \), which is the first selected clause), and a literal from the input clause \( C_0 \) is selected according to literal selection strategy, and the literal is prepared for constructing a standard contradiction.

**Step 2:** Select next input clause in clause set according to clause selection strategy, and select a literal from the selected input clause to construct a standard contradiction. If all clauses in clause set have been searched or selected, go to Step 4.

**Step 3:** This step is used to search optimal an input clause. Apply S-CS rule with the selected clauses, separate a standard contradiction, and generate an S-CS clause, called as \( R \). If \( R \) is an empty clause, go to Step 7. The number of literals in the generated S-CS clause is calculated (marked as \( K \)). If \( K \) value calculated for the first time, then it is temporarily recognized as the minimum value and the current input clause is recorded as \( \text{optimal clause} \). Otherwise \( K \) is decided whether it is a minimum value (the minimum value was updated during the search process). If \( K \) is less than the minimum value, record the current input clause as \( \text{optimal clause} \). Perform search backtracking. The input clause search backtracking algorithm (see the description below) which is used to go back to the last search step, go to Step 2.

**Step 4:** An optimal clause \( C_i (i = 1, 2, \ldots, n) \) is search out which can make the generated S-CS clause has the minimum number of literals in the process of S-CS dynamic deduction. Apply S-CS rule for \( C_i \) and the selected input clauses, separate a standard contradiction and generate an S-CS clause, called as \( R_j \). \( R_j \) is checked whether this deduction step is acceptable.

1. \( R_j \) is a tautology. 2. \( R_j \) is a redundant clause. 3. The maximum term depth exceeds the set threshold.

If one of the above three conditions is satisfied, the S-CS dynamic deduction backtracking algorithm (see the description below) is performed, and go to Step 2.

The input clause search backtracking algorithm: (1) Clear the substitutions and the deduction identifier of the input clauses, which produced by the current input clause participated in deduction. (2) Remove the literals in the constructed standard contradiction and the S-CS clause, which come from the current input clause. (3) Mark the input clause that have been searched.

The S-CS dynamic deduction backtracking algorithm: (1) Clear the substitutions and the deduction identifier of the input clauses, which produced by clause \( C_i \) participated in deduction. (2) Remove the literals in the constructed standard contradiction and the S-CS clause, which come from clause \( C_i \). (3) Unacceptable deduction weight of clause \( C_i \) is increased by 1, unacceptable deduction weight of the literals in clause \( C_i \) which are used for constructing a standard contradiction are increased by 1. (4) Clear the tags of the clauses which have been searched. (5) Mark the input clause that caused an unacceptable deduction.
Step 5: It is an acceptable S-CS deduction, the acceptable deduction weight of clause \( C_i \) is increased by 1, the acceptable deduction weight of literals in clause \( C_i \) which are used for constructing standard contradictions are increased by 1, and mark the deduction identifier of clause \( C_i \). Clear the tags of the clauses which have been searched. The S-CS clause is added to the clause set \( S_1 \) (initially empty) which is used to store the generated S-CS clause in each deduction step. Check the exit conditions (see the description below) for the S-CS dynamic deduction, if satisfied, go to Step 7. Check whether clause \( C_i \) contains substitution, if it contains and the S-CS deduction allows clause \( C_i \) to be reused, go to Step 6 (only for SCSA algorithm). Otherwise, go to Step 2.

Step 6: This is an optional step for reusing the same input clause in the process of S-CS dynamic deduction (only for SCSA algorithm). In order to avoid repeated proof search, record the proof search which caused by clause \( C_i \). Clear substitutions of clause \( C_i \) and reconstruct clause \( C_i \), go to Step 4.

Step 7: Satisfy the exit conditions, exit this S-CS dynamic deduction. The substitutions of all the input clauses are cleared.

The exit conditions are described as following:

1. The S-CS clause is an empty clause. (2) The total number of input clauses reaches the set threshold. (3) The number of literals in S-CS clause reaches the set threshold. (4) The maximum term depth in S-CS clause reaches the set threshold. (5) The runtime reaches the set threshold. (6) The remaining memory reaches the set threshold.

The involved thresholds in the above S-CS dynamic deduction algorithm are set by parameters in the heuristic strategies. There are several special cases of S-CS dynamic deduction need to be further processed, they are described as following:

1. If the clause \( C_q \) fails to perform an acceptable S-CS dynamic deduction, it is added into a special clause list in which has the lowest selection priority in the subsequent S-CS deduction. (2) If all the clauses in clause set as the first input clause cannot perform an acceptable S-CS dynamic deduction, automatically adjust the set thresholds of S-CS clause (number of literals, maximum term depth). (3) Internal simplification is performed on the S-CS clause set \( S_1 \). (4) Apply backward simplification on the whole clause set using the clauses in \( S_1 \), then each S-CS clause in clause set \( S_1 \) is added in whole clause set.

The functions in the above pseudo-code are remarked as follows:

- **selectFirstInputClause(S)**: select the first input clause for S-CS dynamic deduction according to clause selection strategy, and mark the selected clause as an initial input clause.
- **selectNextInputClause(S)**: select an input clause for S-CS dynamic deduction according to clause selection strategy, the difference between **selectFirstInputClause** and **selectNextInputClause** is that the input clause selected by **selectNextInputClause** is to be used to construct a standard contradiction, but the input clause selected by **selectFirstInputClause** has no restrictions.
- **extendedSplitContradiction(d, D)**: apply S-CS rule with the selected input clause \( d \) and the input clause set \( D \), construct a standard contradiction and generate an S-CS clause.
- **findPriorInputClause(q, m)**: compared to \( m \) (\( m \) is the minimum number of literals in an S-CS clause which is generated in the previous search steps, the corresponding input clause is assumed to be \( h \)), if the number of literals in S-CS clause \( q \) is less than \( m \), return current selected input clause. Otherwise, return clause \( h \).
- **traverseComplete(S)**: Clause set \( S \) has been traversed, it is used to exit the search for the optimal input clause.
- **searchBacktracking()**: It is The input clause search backtracking algorithm, which is used to go back to the last search step.
- **checkUnacceptable(q, S)**: check whether the S-CS clause \( q \) in clause set \( S \) is an unacceptable clause (lead to an unacceptable S-CS deduction), the conditions of an unacceptable clause are described in Step 4 above.
- ** deductionBacktracking(w)**: It is The S-CS dynamic deduction backtracking algorithm, which is used to return to the last search step.
- **checkExitConditions()**: check the exit conditions (described in Step 7) of S-CS dynamic deduction.
- **permitReuseClause()**: Check whether the deduction allows the input clauses to be reused.
- **reconstructClause(w)**: reconstruct a new clause which does not contain substitution according to clause \( w \).
- **inner_Simplification(S_q)**: apply forward simplification, factoring rule, and equality resolution in S-CS clause set \( S_q \).
- **backward_Simplification(S, c)**: check whether the clauses in clause set \( S \) are redundant clauses using the clause \( c \). If yes, delete the redundant clauses.

### 4. APPLICATION OF THE PROPOSED S-CS DYNAMIC DEDUCTION ALGORITHMS

In order to effectively evaluate the proposed two S-CS dynamic deduction algorithms based on optimized proof search, we apply the two multi-clause algorithms to the state of the art ATP system Eprover [19]. Eprover is a well-known equational automated theorem prover for first-order logic with powerful equality processing capability based on superposition [13] and rewriting [29,30], and has won the second place in the Conference on Automated Deduction (CADE) ATP System Competition (CASC) for many years since 2002 [31–33], but its inference core still adopts the binary resolution methods. Eprover is also an open source ATP system in first-order logic, we choose the version E 2.3 [34] to combined with the proposed S-CS dynamic deduction algorithms (a combined ATP system), and evaluate its effectiveness. The running mode of the combined ATP system is: it runs the plain Eprover first at a certain time, then if Eprover does not find a proof, the proposed algorithm is applied to find refutation or generated lemmas in a given time. If the proposed algorithm does not find a proof, the filtered lemmas along with the original clause set are handed back to Eprover for further proof search in the left time. The total time of the three stages is the set total runtime for a problem testing. The runtime of the three stages are specified as strategy parameters.

The proposed S-CS deduction algorithms can generate a large number of S-CS clauses as lemmas in a given runtime, most of them are unit clauses because of the S-CS deduction based on optimized...
proof search. Adding a large number of lemmas will greatly increase the size of the clause set and reduce the efficiency of the third stage of proof search, so it is necessary to filter the generated lemmas. Lemmas filtration strategies are mainly achieved as follows:

1. Setting a threshold on the number of literals in the lemmas, and feed to Eprover only the lemmas whose numbers of literals does not exceed the set threshold. In order to make a preliminary attempt, this threshold is fixed to 1.
2. Setting a threshold on maximum term depth in the lemmas, and feed to Eprover only the lemmas whose maximum term depth does not exceed the set threshold.
3. Setting a threshold on clause weight of the lemmas, and feed to Eprover only the lemmas whose clause weight does not exceed the set threshold.
4. Requiring the lemmas to have at least one variable symbol.
5. Requiring the lemmas to have at least one equality literal.
6. Requiring the lemmas which are generated by a negated conjecture as an input clause.

The involved thresholds used for lemmas filtration are set by parameters in the heuristic strategies. For the convenience of description, the version of Eprover (E 2.3) which adds S-CS Dynamic Deduction Algorithm Based on Optimized Proof Search but not reusing clause, in short, called as SCSAN_E 2.3, and adds S-CS Dynamic Deduction Algorithm Based on Optimized Proof Search but reusing clause, in short, called as SCSA_E 2.3.

5. EXPERIMENTAL AND PERFORMANCE ANALYSIS

5.1. Experimental Setup

The two ATP systems SCSAN_E 2.3 and SCSA_E 2.3 are tested on FOF division (500 in total) of the CASC-J9 (in 2018) ATP System competition [35]. The experiments were carried on a PC with 3.6 GHz Inter(R) Core(TM) i7-4790 processor and 16GB memory, with OS Ubuntu15.04 64-bit, standard CPU time limit being 300 seconds. In order to confirm the correctness of proof procedure by S-CS dynamic deduction, Prover9 [36] check system is used to check each deduction step which is used for finding refutation. For experimental results comparisons, the plain E 2.3 is tested on the same hardware environment, and solved 362 problems.

5.2. Experimental Results and Analysis

(1) Performance analysis of SCSAN_E 2.3 on FOF division of the CASC-J9.

In order to evaluate the effectiveness of SCSAN, SCSAN_E 2.3 is tested on FOF division of the CASC-J9. Figure 2 shows the comparison on solved problems by SCSAN_E 2.3 and E 2.3.

From Figure 2, SCSAN_E 2.3 has solved 369 problems with 7 more than E 2.3 which has solved 362 problems. The average time spent for the 369 solved problems by SCSAN_E 2.3 is 23.79 seconds, and 20.14 seconds spent for the 362 solved problems by E 2.3. In special, the average time spent for 362 problems solved by SCSAN_E 2.3 is 18.95 seconds, 1.19 seconds less than that of E 2.3. Therefore, SCSAN_E 2.3 has better time efficiency than E 2.3 when solving the same number of problems.

According to the spent time point, within 150 seconds, SCSAN_E 2.3 has solved 350 problems with 12 more than E 2.3. Within 300 seconds, SCSAN_E 2.3 has solved 369 problems with 7 more than E 2.3. Experimental results show that SCSAN_E 2.3 outperformed E 2.3 in terms of the ability and time efficiency.

In contrast to the solutions by E 2.3, Table 1 lists the problems solved by SCSAN_E 2.3 but unsolved by E 2.3. For those 138 problems unsolved by E 2.3, SCSAN_E 2.3 solved 16 problems accounting for 11.6% of the total, with the average spent time is 226.57 seconds. Because these problems are relatively hard, so the time efficiency is acceptable. For those 16 solved problems, there are 2 problems with rating greater than or equal to 0.9, 12 problems with rating greater than 0.7, accounting for 12.5% and 75% of the total (listed in Table 1) respectively. Experimental results show that SCSAN_E 2.3 can solve some hard problems which unsolved by E 2.3.

(2) Performance analysis of SCSA_E 2.3 on FOF division of the CASC-J9.

<table>
<thead>
<tr>
<th>Theorem Name</th>
<th>Rating</th>
<th>Number of Formulae</th>
<th>Maximum Formula Depth</th>
<th>Number of Variables</th>
<th>Solved Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGT013+2</td>
<td>0.72</td>
<td>923</td>
<td>8</td>
<td>70</td>
<td>257.64</td>
</tr>
<tr>
<td>AGT018+1</td>
<td>0.62</td>
<td>556</td>
<td>8</td>
<td>71</td>
<td>207.86</td>
</tr>
<tr>
<td>AGT019+1</td>
<td>0.76</td>
<td>556</td>
<td>8</td>
<td>72</td>
<td>210.23</td>
</tr>
<tr>
<td>AGT022+1</td>
<td>0.83</td>
<td>556</td>
<td>8</td>
<td>72</td>
<td>275.84</td>
</tr>
<tr>
<td>AGT022+2</td>
<td>0.72</td>
<td>923</td>
<td>8</td>
<td>72</td>
<td>271.62</td>
</tr>
<tr>
<td>BOO109+1</td>
<td>0.27</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>225.23</td>
</tr>
<tr>
<td>GEO506+1</td>
<td>0.83</td>
<td>143</td>
<td>22</td>
<td>564</td>
<td>268.33</td>
</tr>
<tr>
<td>GEO511+1</td>
<td>0.9</td>
<td>162</td>
<td>22</td>
<td>650</td>
<td>217.74</td>
</tr>
<tr>
<td>NUM120+1</td>
<td>0.62</td>
<td>402</td>
<td>19</td>
<td>122</td>
<td>177.64</td>
</tr>
<tr>
<td>SWB082+1</td>
<td>0.79</td>
<td>560</td>
<td>27</td>
<td>973</td>
<td>196.07</td>
</tr>
<tr>
<td>SWB094+1</td>
<td>0.79</td>
<td>560</td>
<td>27</td>
<td>973</td>
<td>194.13</td>
</tr>
<tr>
<td>SWB098+1</td>
<td>0.79</td>
<td>560</td>
<td>27</td>
<td>973</td>
<td>196.22</td>
</tr>
<tr>
<td>SWW189+1</td>
<td>0.9</td>
<td>1150</td>
<td>14</td>
<td>3058</td>
<td>278.63</td>
</tr>
</tbody>
</table>
In order to evaluate the effectiveness of SCSA, SCSA_E 2.3 is also tested on FOF division of the CASC-J9. Figure 3 shows the comparison on solved problems by SCSA_E 2.3 and E 2.3.

From Figure 3, SCSA_E 2.3 has solved 379 problems with 17 more than E 2.3 which has solved 362 problems. The average time spent for the 379 solved problems by SCSA_E 2.3 is 31.8 seconds, and 20.14 seconds spent for the 362 solved problems by E 2.3. In special, the average time spent for 362 problems solved by SCSA_E 2.3 is 21.9 seconds, 1.76 seconds more than that of E 2.3, so the time efficiency is acceptable.

According to the spent time point, within 200 seconds, SCSA_E 2.3 has solved 361 problems with 4 more than E 2.3. Within 250 seconds, SCSA_E 2.3 has solved 374 problems with 12 more than E 2.3. Within 300 seconds, SCSA_E 2.3 has solved 379 problems with 17 more than E 2.3. Experimental results show that SCSA_E 2.3 also outperformed E 2.3 in terms of the ability, and the time efficiency is slightly less than E 2.3.

In contrast to the solutions by E 2.3, Table 2 lists the problems solved by SCSA_E 2.3 but unsolved by E 2.3. For those 138 problems unsolved by E 2.3, SCSA_E 2.3 solved 21 problems accounting for 15.2% of the total, with the average spent time is 221.34 seconds. Of the 138 problems, the time efficiency are also acceptable. For those 21 solved problems, there are 3 problems with rating greater than or equal to 0.9, 16 problems with rating greater than or equal to 0.7, accounting for 14.3%, and 76.2% of the total (listed in Table 2) respectively. Experimental results show that SCSA_E 2.3 also can solve some hard problems which unsolved by E 2.3.

(3) Comprehensive analysis of the two systems SCSAN_E 2.3 and SCSA_E 2.3.

According to the solutions of SCSAN_E 2.3 and SCSA_E 2.3 on FOF division of the CASC-J9, we can see that both SCSAN_E 2.3 and SCSA_E 2.3 outperform E 2.3, and SCSA_E 2.3 outperforms SCSAN_E 2.3. More interesting is that SCSAN_E 2.3 solves 16 unsolved problems by E 2.3, and SCSA_E 2.3 solves 21 unsolved problems by E 2.3, 14 problems of them are the same. The comprehensive analysis shows that SCSAN_E 2.3 and SCSA_E 2.3 have different main points in the proof search, which SCSAN_E 2.3 uses more input clauses in clause set to participate in S-CS deduction, but the full use of each input clause is insufficient, and SCSA_E 2.3 uses a relatively small number of input clauses in clause set to participate in S-CS deduction, but the full use of each input clause is relatively sufficient. Furthermore, the problems which solved by SCSA_E 2.3 but unsolved by E 2.3 basically include that of SCSAN_E 2.3. Therefore, the full use of each input clause to participate in S-CS dynamic deduction is a good deduction method for S-CS rule.

The experimental analysis shows that the proposed S-CS dynamic deduction algorithm based on optimized proof search is an effective method, which is an effective complement to the binary resolution method, and can be effectively applied to the first-order logic problem proving. The effectiveness of S-CS deduction is mainly reflected in:

1. In contrast to binary resolution methods, S-CS rule is a multi-clause dynamic deduction method, which improve binary resolution method only eliminating a complementary pairs of literals (two literals) to an S-CS (a literal set), so it has a strong literal elimination ability, and the S-CS clauses have few literals, thus improve the deduction efficiency.

2. In contrast to binary resolution methods, S-CS rule can handle multiple (two or more) clauses in each deduction step, it has advantageous for handling input clauses with large number of literals.

3. The current advanced ATP systems adopt binary resolution methods under saturation algorithm, and the proof search of S-CS dynamic deduction can effectively supplement the proof search of binary resolution methods, such as the deduction description of Example 2.1.

4. The proposed S-CS dynamic deduction algorithms preferentially generate S-CS clauses with a small number of literals, which is not equivalent to binary resolution methods using input clause with small number of literals first. That is because S-CS rule handles an input clause by separating a standard contradiction, it does not limit the number of literals in the input clause in S-CS deduction.

<table>
<thead>
<tr>
<th>Theorem Name</th>
<th>Rating</th>
<th>Number of Formulae</th>
<th>Maximum Formula Depth</th>
<th>Number of Variables</th>
<th>Solved Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGT013+2</td>
<td>0.72</td>
<td>923</td>
<td>8</td>
<td>70</td>
<td>291.47</td>
</tr>
<tr>
<td>AGT018+1</td>
<td>0.62</td>
<td>556</td>
<td>8</td>
<td>71</td>
<td>186.88</td>
</tr>
<tr>
<td>AGT019+1</td>
<td>0.76</td>
<td>556</td>
<td>8</td>
<td>72</td>
<td>235.29</td>
</tr>
<tr>
<td>AGT022+1</td>
<td>0.83</td>
<td>162</td>
<td>8</td>
<td>72</td>
<td>248.56</td>
</tr>
<tr>
<td>AGT022+2</td>
<td>0.72</td>
<td>923</td>
<td>8</td>
<td>72</td>
<td>241.11</td>
</tr>
<tr>
<td>BOO109+1</td>
<td>0.57</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>258.08</td>
</tr>
<tr>
<td>GEO506+1</td>
<td>0.83</td>
<td>143</td>
<td>22</td>
<td>564</td>
<td>263.49</td>
</tr>
<tr>
<td>GEO511+1</td>
<td>0.9</td>
<td>162</td>
<td>22</td>
<td>650</td>
<td>233.32</td>
</tr>
<tr>
<td>GRA002+1</td>
<td>0.76</td>
<td>18</td>
<td>13</td>
<td>68</td>
<td>234.35</td>
</tr>
<tr>
<td>NUM320+1</td>
<td>0.62</td>
<td>402</td>
<td>19</td>
<td>122</td>
<td>161.75</td>
</tr>
<tr>
<td>SET222+4</td>
<td>0.52</td>
<td>29</td>
<td>19</td>
<td>138</td>
<td>217.39</td>
</tr>
<tr>
<td>SEU222+2</td>
<td>0.93</td>
<td>311</td>
<td>14</td>
<td>661</td>
<td>218.75</td>
</tr>
<tr>
<td>SWB027+1</td>
<td>0.79</td>
<td>560</td>
<td>27</td>
<td>976</td>
<td>186.34</td>
</tr>
<tr>
<td>SWB082+1</td>
<td>0.79</td>
<td>560</td>
<td>27</td>
<td>973</td>
<td>190.12</td>
</tr>
<tr>
<td>SWB088+1</td>
<td>0.83</td>
<td>560</td>
<td>27</td>
<td>973</td>
<td>174.19</td>
</tr>
<tr>
<td>SWB094+1</td>
<td>0.79</td>
<td>560</td>
<td>27</td>
<td>973</td>
<td>170.09</td>
</tr>
<tr>
<td>SWB098+1</td>
<td>0.79</td>
<td>560</td>
<td>27</td>
<td>973</td>
<td>173.14</td>
</tr>
<tr>
<td>SWB104+1</td>
<td>0.83</td>
<td>560</td>
<td>27</td>
<td>975</td>
<td>213.75</td>
</tr>
<tr>
<td>SWW189+1</td>
<td>0.9</td>
<td>1150</td>
<td>14</td>
<td>3058</td>
<td>247.04</td>
</tr>
<tr>
<td>SWW470+6</td>
<td>0.83</td>
<td>623</td>
<td>19</td>
<td>2150</td>
<td>233.65</td>
</tr>
<tr>
<td>SY0606+1</td>
<td>0.29</td>
<td>9</td>
<td>14</td>
<td>55</td>
<td>269.32</td>
</tr>
</tbody>
</table>

Figure 3 | Comparison on solved problems by SCSA_E 2.3 and E 2.3.
6. CONCLUSIONS AND FUTURE WORK

S-CS rule is a novel inference method for automated reasoning, its advantage is that it can handle multiple input clauses in each deduction step and generate S-CS clauses by separating standard contradictions, which are not the same as multiple complementary pairs of literals. Due to the dynamic characteristic of S-CS deduction and the challenges of proof search, the design of the S-CS dynamic deduction algorithm is further discussed. According to theory and practice analysis, two S-CS dynamic deduction algorithms including corresponding heuristic strategies are proposed in this paper, they can preferentially generate S-CS clauses with a small number of literals, and one of them can give full play to the ability of synergized deduction with the same input clause. The proposed two S-CS dynamic deduction algorithms are applied to the state of the art ATP system Eprover 2.3, which are called as SCSAN_E 2.3 and SCSA_E 2.3 respectively. The two ATP systems are tested on the FOF division of the CASC-J9. Both SCSAN_E 2.3 and SCSA_E 2.3 show excellent performance, which outperform the plain E 2.3 to a certain extent. Experimental results show that S-CS rule is an important complement to the binary resolution method, and the proposed S-CS dynamic deduction algorithm can effectively improve the capability of the state of the art ATP systems, and can be effectively applied to the first-order logic problem proving.

Although different kinds of heuristic strategies has proposed for SCSAN and SCSA, and deduction backtracking algorithm is used for effectively optimizing proof search, there is still room for improvement, such as how to effectively select an input clause and a literal to separate a standard contradiction, how to further exert the synergized ability of multiple input clauses, and how to effectively filter lemmas for the further proof search. Because the clause selection strategy, the literal selection strategy and lemmas filtration strategy are foundational implementation, effective use and further development of these strategies are the main future research work. The proposed S-CS dynamic deduction algorithm does not include equality handling, adding the rewriting method is also a future research work.

CONFLICT OF INTEREST

There is no conflict of interest.

AUTHORS’ CONTRIBUTIONS

Feng Cao has proposed the main idea of the paper, including the algorithm implementation, application and experimental evaluation. Yang Xu, Shuwei Chen, Jian Zhong, Guanfeng Wu have contributed with a lot of good suggestions for the algorithm implementation. In addition, Shuwei Chen has contributed with some writing, review and helpful comments to further enhance the quality of the paper.

ACKNOWLEDGMENTS

This paper is supported by the National Natural Science Foundation of China (Grant No.61673320).

REFERENCES


