International Journal of Computational Intelligence Systems
Vol. **13(1)**, 2020, *pp.* 672–680
DOI: https://doi.org/10.2991/ijcis.d.200521.001; ISSN: 1875-6891; eISSN: 1875-6883
https://www.atlantis-press.com/journals/ijcis/

Research Article

A Novel Combinational ATP Based on Contradiction Separation for First-Order Logic

Jian Zhong[1,2,*], Yang Xu[2], Feng Cao[1,2]

[1]*School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China*
[2]*National-Local Joint Engineering Laboratory of System Credibility Automatic Verification, Southwest Jiaotong University, Chengdu 610031, China*

ARTICLE INFO

Article History
Received 07 Apr 2020
Accepted 15 May 2020

Keywords
First-order logic
Theorem proving
Prover9
Contradiction separation rule
Dynamic multi-clause synergized deduction

ABSTRACT

At present, most of the first-order logic theorem provers use a binary-resolution method, which can effectively solve the general first-order logic problems to a certain extent. However, the cooperative processing ability of this method for multiple clauses is insufficient, and it is easy to cause rapid expansion of clause set in the deductive process. In this paper, we propose a novel first-order logic theorem prover based on the standard contradiction separation (*S-CS*) rule. This prover can realize the dynamic cooperative deduction of multiple clauses in each deduction process, as well as it can effectively learn and control the deduction process. This paper incorporates the *S-CS* rule with the well-known prover *Prover9*, to build a combined system, which effectively integrates the advantages of the two methods, not only improves the binary-resolution prover's ability, but also solves more than 100 problems that cannot be solved by other provers.

1. INTRODUCTION

As an essential branch of artificial intelligence, automated reasoning has always been a research hotspot, especially the development of formal verification puts forward higher requirements for the automated theorem prover (ATP) [1–3]. At present, most of first-order theorem (FOF) provers (in the sequel simply *provers*) are based on the resolution method proposed by Robinson [4]. Mainly adopt the *saturation* algorithm framework for proof search [5]. That is, given two sets of the *P* (processed set) and the *U* (unprocessed set), initializing all clauses into the *U* and selecting one clause from U each time. Then making a binary resolution with the clauses in process, and putting the resolvents into U. This method inevitably produces a large number of resolvents, and hence bring a rapid expansion of the proof-search space.

Furthermore, only two clauses are resolved at a time. Not only the processing efficiency is low, but the correlation between multiple clauses is ignored. Modern *provers* introduces hyper-resolution [6] into the *saturation* algorithm framework, and achieves the resolution of one clause and multiple unit-clauses. To some extent, the collaborative processing ability between multiple clauses is considered, and excellent results have been achieved in practice. However, multiple nonunit clauses cannot be processed cooperatively. In 2018, Xu *et al.* [7] proposed a standard-contradiction-separation rule (in the sequel simply *S-CS*, also see Section 2)-based dynamic multi-clause synergized automated deduction. The standard contradiction separation (*S-CS*) rule is a novel inference rule. Unlike binary resolution, the *S-CS* rule allows multi-clause to deduction together (also call synergy). And the rule implies the thought of dynamic control of the deduction process, which has higher deductive efficiency. In the paper [8] shown this point.

In this paper, our contribution is, firstly, we propose an architecture of *prover* based on *S-CS* Rule. We were discussing some problems of applying the *S-CS* rule to proof-search. Then analyzed the characteristics of heuristic strategy based on *S-CS* rule. Finally, we proposed and implement a novel combinational system, *CoProver*, which integrates the advantages of the *S-CS* rule and traditional binary resolution. Experimental results show that the *S-CS* rule has better poof-search ability than traditional binary resolution. Furthermore, the *CoProver* solves more than 100 problems with rating = 1 that cannot be solved by other existing *provers* in the TPTP library.

The structure of this paper is as follows: Firstly, we review the current development of ATP and analyze the problems of the traditional binary resolution methods.

Section 2 introduces the basic concepts of the Contradiction Separation rule and notations and conventions of ATP.

In Section 3, we describe the specific design strategy of our proposed prover based on *S-CS* rule and describe the key technologies for implementing this prover.

* *Corresponding author. Email: 77367632@qq.com*

Section 4 introduces the construction of a hybrid system based on our proposed *provers*.

In Section 5, a series of experiments are conducted, and the experimental results are evaluated.

Finally, in Section 6, we give conclusions and our future research directions.

2. PRELIMINARIES

In this paper, we are primarily interested in the first-order formula in *conjunctive normal forms* (CNF). We assume the following notations and conventions.

Definition 2.1. [9] (***Term***). A term is defined inductively as follows:

i. Each free variable and each constant is a term.

ii. If t_1, .., t_n are terms and f is an n-ary function symbol, the $f(t_1, .., t_n)$ is also a term.

The set of all terms is called T. We write the $T = (F, V)$, F is a finite set of function symbols with an n-ary function (denoted *f1, f2*), if $n = 0$ the T is a *constant term*, is written "a_1, a_2, ..., a_n." V is a finite set of variable symbols (denoted x_1, x_2).

A term is written as lower-case with a number, e.g., "t_1, s_1," especially, term t ($t \in T$) is *ground term* if it contains no variables, the symbol G noted the set of *ground term*, and the symbol "g" is a *ground term* $g \in G$.

Definition 2.2. [10] **(*literal*).** A literal is an expression A (a positive literal) or $\sim A$ (a negative literal), where A is an atom.

An atom (or atomic formula) is an expression $P(t_1, ..., t_n)$ where P is a predicate symbol of arity n and $t_1, ..., t_n$ are terms. Two literals A and $\sim A$ *are* said to be *complementary*.

We write the letter "l" with a number for literal, e.g., l_1, l_2.

Definition 2.3. [9] (***clause***). A clause C is a finite set of literals and their disjunction, written as $C = \{l_1, ..., l_n\}$. The empty clause is denoted by. Especially if C has only on literal, the clause is called *a **Unit clause***.

Let S is a clause set, usually, is written $S = \{C_1, C_2, \dots, C_m\}$.

Definition 2.4. [9] (***substitution***). A (variable) substitution is a mapping $\sigma : V \to T$. Such that $\sigma(x) \neq x$ for only finitely many $x \in V$. We *write* $\sigma = \{t_1/x_1, L, t_n/x_n\}$ $x_n \in V$ and $t_i \neq x_i$ if $t_i \in G$ then σ is *ground substitution*.

Xu [7] proposed the new reference rules standard contradiction separation rule (*S-CS*).

Definition 2.5. [7] (***Contradiction***) Suppose a clause set $S = \{C_1, C_2, \dots, C_m\}$ is a standard contradiction (SC). If $\forall(l_1, \dots, l_m) \in \prod_{i=1}^{m} C_i$ there exists at least one complementary pair among$\{l_1, \cdots, l_m\}$. The symbol $\prod_{i=1}^{m} C_i$ denotes the cartesian product of $C_1, C_2, \dots, C_m$. If $S = \Lambda_{i=1}^{m} C_i$ is unsatisfiable, then $S = \Lambda_{i=1}^{m} C_i$ is called a *quasi contradiction* (QC).

Definition 2.6. [7] **(S-CS Rule in First-Order Logic)** Suppose a clause set $S = \{C_1, C_2, \dots, C_m\}$ in first-order logic. Without loss of generality, assume that there does not exist the same variables among $C_1, C_2, \cdots, C_m$. The following inference rule that produces a new clause from S is called *a standard contradiction separation rule*, in short, an ***S-CS* rule**.

For each C_i ($i = 1, 2, \dots, m$), firstly apply a substitution σ_i to C_i (σ_i could be an empty substitution but not necessarily the most general unifier), denoted as $C_i^{\sigma_i}$; then separate $C_i^{\sigma_i}$ into two subclauses $C_i^{\sigma_i -}$ and $C_i^{\sigma_i +}$ such that

i. $C_i^{\sigma_i} = C_i^{\sigma_i -} \vee C_i^{\sigma_i +}$, where $C_i^{\sigma_i -}$ and $C_i^{\sigma_i +}$ have no common literals.

ii. $C_i^{\sigma_i +}$ can be an empty clause, but $C_i^{\sigma_i -}$ cannot be an empty clause.

iii. $\Lambda_{i=1}^{m} C_i^{\sigma_i}$ is a SC. That is, $\forall(x_1, \dots, x_m) \in \prod_{i=1}^{m} C_i^{\sigma_i -}$, there exists at least one complementary pair among $\{x_1, \cdots, x_m\}$.

3. *PROVER* BASED ON *S-CS* RULE AND KEY TECHNOLOGIES

This section will discuss the design strategy of a novel prover based on the CS rule and the solutions to critical technologies. From Section 2, we can see that the core of CS rule is constructing the SC and obtaining the *contradiction separation clause* (S), which is the C^+ part. When the C^+ is empty, i.e., the original formula set S is proved.

3.1. The Architecture of Provers Based on the S-CS Rule

We build two sets of U (unprocessed clause set) and P (processed clause set) by using the current mainstream given-clause framework. We initialize the clauses and put all of them into U to present an improved architecture, as shown in Figure 1.

The workflow is as follows:

Step 1. Initializing and preprocessing all clauses in S and put them into ***U***.

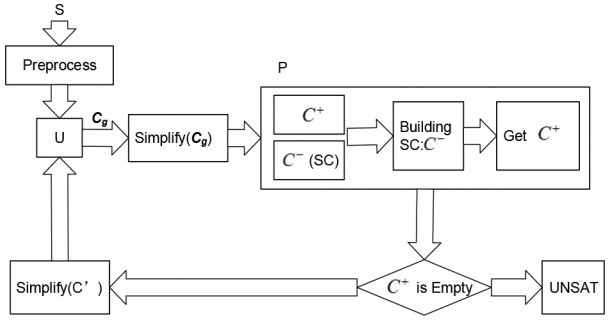

Figure 1 | The architecture of provers based on standard contradiction separation (S-CS) rules.

Step 2. Obtaining the given-clause C_g according to heuristic strategy.

Step 3. Simplifying C_g, according to some simplified rules and puts the result into set P.

Step 4. Applying the *S-CS* Rule to C_g to divide it into C_i^- and C_i^+, and building a new contradiction C^- with the other clauses in P.

Step 5. Applying CS Rule to obtain the *contradiction separation clause* C^+.

Step 6. If C^+ is empty, then S is UNSAT and the process end, else switching to Step 7.

Step 7. Obtaining a new clause C' from C^+. and simplifying C'. If C' is not trivial, then adding C' to U.

Step 8. Repeating Step 1.

As can be seen from the above framework process, implementing the ATP based on the *S-CS* Rule needs to solve the following key points:

1. Constructing the SC C^- and get the *contradiction separation clause* C^+ efficiently.
2. Obtaining the given-clause C_g.
3. Employing dynamic learning heuristics strategy in deduction

3.2. Dynamic Construction of the SC

3.2.1. The process of separating SC

Unlike a traditional binary resolution, the CS rule is essentially a dynamic, multi-clauses deduction rule. It should be noted that the contradiction separation clause C^+ is not only the result of two clause resolutions but also the cooperative deduction result of multi-clauses in the *P*.

The *SC* is separated from multi-clauses in the *P*. Intuitively, the deductive based on *S-CS* Rule is the process of obtaining C^+ by separating contradictions from *P* until C^+ is empty.

According to Definition 2.1, the *contradiction* is a *cartesian set*. The core of applying the *S-CS* rule is effectively separating *CS* from multi-clauses to obtain the *contradiction separation clause* C^+.

Thus, we propose a method to obtain C^+ quickly. We assume the processed clause set $P = \{C_1, C_2, ..., C_m\}$, and the literals in each clause are divided into two parts. That is, $C_i^{\sigma_i} = \{C_i^{\sigma_i+}, C_i^{\sigma_i-}\}$, ($\sigma_i$ is a substitution to C_i). What's more, $C^+ = \{C_1^{\sigma_1+}, ..., C_m^{\sigma_m+}\}$, $C^- = \{C_1^{\sigma_1-}, ..., C_m^{\sigma_m-}\}$. The process is as follows:

Step 1. Obtaining given-clause $C_g = \{l_1, ..., l_n\}$ from U according to clause selection strategy. Putting all literal into C_g^+, then $C_g^+ = \{l_1, ..., l_n\}$ and $C_g^- = \varnothing$.

Step 2. Find a substitute σ_i make $\sigma_i l_i \{l_i \in C_g^+\}$ and $\sim \sigma_i l_j \{l_j \in C^-\}$ be a complementary pair. Then, removing $\sigma_i l_i$ from $C_g^{\sigma_i+}$.

Algorithm: Bulid Standard Constradiction

input: Clause C_g, C^-, C^+

output: new Standard Constradiction C^+

GetSC (C_g, C^-, C^+)

```
while (true)
    <l1, l2, σ >←GetComplementaryPair( C_g , C^- )
    if σ is not found then
        break
    end if
    C_g ←DelLiteral( C_g ,l1)
    if C_g is empty
        RollbackSubstitute()
        goto END
    end if
    C_g ←Substitute( C_g , σ )
    C^- ←Substitute( C^- , σ ); C^+ ←Substitute( C^+ , σ )
end while
while (true)
    <l1, l2, σ >←GetComplementaryPair( C_g , C^+ )
    if σ is not found then
        break
    end if
    C_g ←DelLiteral ( C_g ,l1); C_g ←Substitute( C_g , σ )
    C^+ ←DelLiteral ( C^+ ,l2); C^- ←AddLiteral( C^- ,l2)
    C^- ←Substitute( C^- , σ ); C^+ ←Substitute( C^+ , σ )
end while
:END
put C_g into C^+
```

Figure 2 | The pseudo-code for Bulid standard constradiction algorithm.

Step 3. If $C_g^{\sigma_i+}$ is empty, canceling all substitutions and returning to step 1.

Step 4. Finding a substitute σ_i to make $\sigma_i l_i \{l_i \in C_g^+\}$, and $\sim \sigma_i l_k \{l_k \in C_k^{\sigma_k+}$ and $C_k^{\sigma_k-} = \varnothing\}$ be a complementary pair. Then, moving the $\sim \sigma_i l_k$ from $C_k^{\sigma_k+}$ to $C_k^{\sigma_k-}$ and removing $\sigma_i l_i$ from $C_g^{\sigma_i+}$.

Step 5. Repeating Step2 until no substitute σ_i is found.

Step 6. Putting $C_g^{\sigma+}$ into C^+.

The pseudo-code in Figure 2 is correspond to the above process of build SC, when C_g added to *P*, we can obtain a new *contradiction separation clause* C^+ in the deduction. This process is dynamic. Finally, C^+ is the result of multi-clauses cooperating.

In the above process, there will be some cases of given-clause C_g:

i. All literals need to be removed from $C_g^{\sigma_i+}$
   (a) Because they are complementary to $\sim \sigma_i l_j$ $l_j \in C^-$ (Step 3). We think this inference is invalid because C_g has no direct effect on obtaining the C^+ (no increase, no decrease).

(b) Because they are complementary to $\sim \sigma_i l_k \{l_k \in C_k^{\sigma_k +}$ and $C_k^{\sigma_k -} = \varnothing\}$ (Step 4). This case will be discussed in Section 3.2.2.

ii. No literal is removed from $\boldsymbol{C_g}$.
No newly generated *CS* and $\boldsymbol{C_g}$ remain in *P*.

iii. Some literals are removed from $C_g^{\sigma_i +}$.
It is the most common case, which means that the *S-CS* rule is successfully applied with $\boldsymbol{C_g}$ so that the new $\boldsymbol{C}^+$ can be obtained.

3.2.2. Restart

The technology for finishing the current *CS* construction process and starting a new deduction process is called the restart. Employing some restart strategies can avoid falling into a local search.

i. Restarting when all literals are removed from C_g in Step 4.

Obviously, in this case, the number of words in the $\boldsymbol{C}^+$ is the least. Thus, by removing some literals from $\boldsymbol{C}^+$ without adding new ones in it, we can restore all substitution (rollback) and put the $\boldsymbol{C_g}$ back to *U*.

ii. Restarting when the clause cannot be selected from *U*.

In this case, either set *U* is empty, or each clause in *U* participating in the deduction is invalid.

Note that we say a clause is invalid has two situations: the case i) and some constraints are not met. For example, the maximum number of literals in $\boldsymbol{C}^+$ is 2, but when some given-clauses participate in deduction, this limit is exceeded.

The restart process is as follows:

i. Adding all clauses in $\boldsymbol{P}$ to $\boldsymbol{U}$ and clearing the set p.

ii. Simplifying set $\boldsymbol{U}$ and removing redundant clauses.

iii. Selecting clause $\boldsymbol{C_g}$ from $\boldsymbol{U}$ again and putting it into $\boldsymbol{P}$ according to the selection strategy to start a new round of deduction.

3.3. Clause Selection Strategy Based on S-CS Rule

A powerful ATP requires three elements [11]: 1). Powerful inference rules; 2). Good heuristic search strategy; 3). Efficient data structure.

The traditional binary resolution is due to the simple deduction rules: only two clauses for resolution. Therefore, most of the ATPs focuses on the research of heuristic search strategies. For example, vampire [12] has hundreds of different heuristic strategies. Furthermore, some articles [13,14] employ artificial intelligence or machine learning to generate different search strategies. These search strategies can improve the performance of ATPs to a certain extent. However, usually, too many strategies lead to too complicated heuristic strategies, or they are only useful for certain types of problems. These drawbacks make it difficult to find a generally good heuristic strategy.

In this section, firstly, we propose rules for formulating heuristic strategies based on the characteristics of the *S-CS* Rule, and then give the clause selection strategy based on the *S-CS* Rule.

3.3.1. Heuristic strategy based on S-CS rule

The traditional binary resolution based on saturate employs various heuristic strategies. These strategies usually come from two considerations:

i. Controlling the expansion of the proof-search space as much as possible to avoid too fast expansion of the clause set. However, such control is limited. In most cases, it is difficult to avoid the rapid expansion of the search space because the decision time increases.

ii. Based on the goal-oriented strategy, employing the conjecture clause is preferred in the resolution process. It is a rough heuristic strategy, so the conjecture-clause participation in resolution will generate a large number of lemma clauses. Therefore, most of ATPs will generally combine the goal clause with the minimum age of the hybrid strategy.

Compared with the binary resolution method, the deduction method based on the *S-CS* rule comes with an outstanding feature: If and only if the *contradiction separation clause* $\boldsymbol{C}^+$ is empty, the clause set $\boldsymbol{S}$ is UNSAT. Therefore, for the heuristic strategy based on ***S-CS*** rule, the following two rules are followed:

Rule 1: The literals in the contradiction separation clause $\boldsymbol{C}^+$ is as little as possible.

Rule 2: The effect of unification/substitution on literals in C^- is as small as possible.

For rule 1, there are two meanings. The first is to make the most use of the unit clause. The priority to use the unit clause must satisfy rule 1. Secondly, make full use of the coordination of multi-clause when there are more than two clauses in P. The more synergized effectors of all clauses are involved, the more literals will be removed from given-clause $\boldsymbol{C_g}$.

For rule 2, the literal $P(x)$ has better synergy than literal $P(a)$. So we do not want the literal instanced too early, which will be discussed in Section 3.4.

Compared with the traditional binary resolution, the above two rules make the heuristic strategy based on the *S-CS* rule pay more attention to dynamic control of the deductive process, as well as are comfortable to be implemented.

3.3.2. Clause selection

Clause selection is the most crucial choice point of resolution-based provers [5]. The VAMPIRE *prover* was the CASC (Conference on Automated Deduction ATP System Competition) winner in recent years, which uses two parameters for clause selection: the age and the weight of a clause [12]. Another powerful purely equational theorem prover E prover uses some heuristic strategiesy [5]: first-in/first-out FIFO, symbols count, and goal-directed evaluation function. GKC [15] is a fresh resolution prover, which uses a 2-layer

clause selection queues algorithm. The first layer uses the common ratio-based algorithm. Further, the second layer uses four separate queues based on the clause's age.

In the ATP based on the *S-CS* rule, we perform the selection of a given clause by employing a dynamic-static combined strategy to satisfy the above rules, as has been proposed in Section 3.3.1.

Our clause selection strategy based on *S-CS* rule as follows:

i. Static clause selection strategy
   (a) Selecting clauses from one of the queues by Least literals priority.
   (b) Selecting clauses from one of the queues by employing a stability-weight ratio. That will be discussed in Section 3.4.

ii. Dynamic clause selection strategy

   Unlike the binary resolution, the dynamic strategy is one of the significant advantages of *S-CS* rule. The core idea is to guide the entire deduction process by the control strategy of the contradiction separation clause $\boldsymbol{C}^+$.
   (a) Dynamic strategy for controlling the number of literals in $\boldsymbol{C}^+$

   Each deduction based on the *S-CS* rule will generate a $\boldsymbol{C}^+$. In order to make the literals in $\boldsymbol{C}^+$ be as little as possible, we set a threshold $\boldsymbol{N_r}$. Let $|\boldsymbol{C}^+|$denote the number of literals in $\boldsymbol{C}^+$. If $|\boldsymbol{C}^+| > \boldsymbol{N}$, contradiction separation fails, i.e., this inference is invalid. Rollback and put the $\boldsymbol{C_g}$ back to *U*, reselect given clause.

   Setting the threshold *N* is also a dynamic learning process. For example, with an initial value $\boldsymbol{N} = 1$, when all the given-clause $\boldsymbol{C_g}$ from *U* are invalid, then $\boldsymbol{N} = \boldsymbol{N} + 1$;

   (b) Least $|\boldsymbol{C}^+|$ priority strategy
   The dynamic strategy is different from the static strategy. When $\boldsymbol{N} > \boldsymbol{1}$ and there are some candidate clauses $\boldsymbol{C_1}$,..., $\boldsymbol{C_m}$ with the same number of literals. We can try to inference with them separately, so the clause with the smallest $|C_i^+|, (i \in m)$ is preferred. Due to trying each candidate clauses, the deduction efficiency is affected. In practice, the candidate clause with more complementary predicates in the C^- is preferred.

   (c) Older clause or goal clause priority strategy
   Firstly, the older clause needs to be chosen; the two clauses are of the same age, choosing the goal clause or the descendants of the goal clause.

3.4. Dynamic Learning Heuristics Strategy

According to the analysis in Section 3.3.1, there are two guiding rules for the heuristic strategy based on the *S-CS* rule, where Rule 2 primarily refers to the control over the literal instances during the inference process.

In this section, we will propose a weight evaluation method and discuss a dynamic learning heuristics strategy based on *Term Stability*.

3.4.1. Weight function based on term stability

In the unification and substitution process, the variable in first-order logic probably replaces other variables and constants. Term stability is applied to evaluate how easy it is for a *term* to substitute for another *term* during the inference process, which reflects the complexity of the term's substitution. Thus we the term $\boldsymbol{f(x)}$ is more stable than $\boldsymbol{x}$, and ground term $\boldsymbol{f(a)}$, $\boldsymbol{a}$ is the most stable.

Definition 3.1. Let t be a term in first-order logic, $t \in T$, term stability $\boldsymbol{S(t)}$ is defined as if $S : T \mapsto R^+$, which satisfies

$$S(t) = \begin{cases} W_x, & t \in V \\ W_g, & t \in G \\ W_x + \dfrac{W(t_f)}{W(t_f)+1}, & t \in F_v \end{cases}. \tag{1}$$

In (1) V is a finite set of variable symbols, G denotes the set of ground terms, $\boldsymbol{F_v}$ indicates a set of function terms with variables, and $\boldsymbol{t_f}$ is a function term involve variables. $\boldsymbol{W_x}$, $\boldsymbol{W_g}$ are the weight of variable and ground-term, respectively. They are fixed values and satisfy $\boldsymbol{W_g} > \boldsymbol{W_x} > 0$. $\boldsymbol{W(t_f)}$ is the weight of $\boldsymbol{t_f}$:

$$W(t_f) = W_g * |t|_g + W_x * \sum_{i=1}^{n} |t|_{DX_i} + \sum_{t_{f_i}} S(t_{fi}). \tag{2}$$

where $\boldsymbol{|t|_g}$ denotes the number of ground terms in the term $\boldsymbol{t}$. $\sum_{i=1}^{n} |t|_{DX_i}$ represents the sum of the nesting depths of variable x_i in term $\boldsymbol{t}$, e.g., there is a term $\boldsymbol{t_1}$

$t_1 = f5(f(x_1), f2(x_2, x_3), x_4, f1(a_1), a_2)$. Then $\boldsymbol{|t|_g = 2}$,

$$\sum_{i=1}^{n} |t|_{DX_i} = 2 + 2 + 2 + 1 = 7.$$

Example 3.1. Calculate the term stability of the following three terms:

$$t_1 = f3(a1, f1(x_1), f1(x_2)).$$

$$t_2 = f3(x_1, x_2, f1(f1(x_3))).$$

$$t_3 = f3(x_1, a2, f1(f1(x_3)), x_3).$$

Let $\boldsymbol{W_g} = 2$, $\boldsymbol{W_x} = 1$. Then $W(t_1) = 5.333$, $S(t_1) = 1.842$.

$$W(t_2) = 3.636, S(t_2) = 1.784.$$

$$W(t_3) = 5.636, S(t_3) = 1.849.$$

Definition 3.2. (Clause stability) There is a clause $C = l_1 \vee ... \vee l_n$. Clause stability of C is defined as

$$S_c = \frac{1}{n} \sum_{i=1}^{k} S(l_i). \tag{3}$$

When selecting $\boldsymbol{C_g}$, the clauses with smaller $\boldsymbol{S_c}$ are preferred to participate in the deduction.

In practice, if there are multiple literals with the same variable in a clause, then any variables instance will affect more than one literal. For example, for clause *C1*: $P2(x_1, x_2) \vee P1(f(x_1))$, if x_1 in $P1(f(x_1))$ is substituted as $\boldsymbol{a_1}$, then $P2(x_1, x_2)$ changes to $P2(a_1, x_2)$ by substitution σ:$\{\boldsymbol{a_1}/\boldsymbol{x_1}\}$.

We call this correlation between such kind of literals is ***literal-relational*** in the same clause.

Definition 3.3. Let l be literal in clause C. *literal-relational* $R(l)$ is defined as if $R : T \to R^+$, which satisfies

$$R(l) = 1 - \frac{\sum |l|_{x_i}}{|V|_c} \tag{4}$$

where $|V|_c$ denotes the total number of variables in $\boldsymbol{C}$ and $\sum |l|_{x_i}$ indicates the variable $\boldsymbol{x_i}$ in literal $\boldsymbol{l}$.

Example 3.2. There is a clause

$$C_1 : P2(x_1, x_2) \vee P1(f(x_1)) \vee P2(f2(x1, x3), f2(x1, x2)$$

$|V|_c = 7$, then *literal-relational* of each literal is as follows:

$$R(P2(x_1, x_2)) = 1 - (3 + 1)/7 = 3/7.$$

$$R(P1(f(x_1))) = 1 - 3/7 = 4/7.$$

$$R(P2(f2(x1, x3), f2(x1, x2)) = 1 - (2 + 1 + 0)/7 = 4/7.$$

3.4.2. Literal selection

During the *S-CS* building process, it needs to find a substitute σ_i that makes $\sigma_i l_i \{l_i \in C_g^+\}$ and $\sim \sigma_i l_j \{l_j \in C^-$ or $l_j \in C^+\}$ a complementary pair. It is known that different substitution sequences lead to different results. For example, there is a set C^- in $\boldsymbol{P}$ and a given clause.

$$C_g : P2(x_{11}, a_1) \vee Q2(x_{12}, a_1) \vee P3(a_1, a_2, a_3).$$

C^-:{$\sim P3(x_1, x_2, x_3)$, $\sim P2(x_1, x_2)$, $\sim Q2(x_1, x_3)$}. If we prefer literal $P3(a_1, a_2, a_3)$, we will obtain a substitution σ_1:{a_1/x_1, a_2/x_2, a_3/x_3}. Obviously, there is no complementary pair in C^- for the literals $P2(x_{11}, a_1)$, $Q2(x_{12}, a_1)$.They will be added into the $\boldsymbol{C}^+$. But if we find σ for literal $P2(x_{11}, a_1)$, at first. Then only literal $P3(a_1, a_2, a_3)$ is retained. In other words, literal selection strategies directly affect ATP performance.

Actually, we hope that literal selection strategies can achieve the following effects:

A. Keeping the literals as few as possible in $\boldsymbol{C_g}$

B. Avoiding literal instance in C^-, if possible, in the deduction process. To has the minimum influence on the C^- after the unification.

Therefore, we employ some *literal-selection* strategies as follows:

i. Literal-stability priority queues.
It gives higher priority to literal with smaller $\boldsymbol{S(l)}$.

ii. Literal-relational priority queues.
It gives higher priority to literal with smaller $\boldsymbol{R(l)}$.

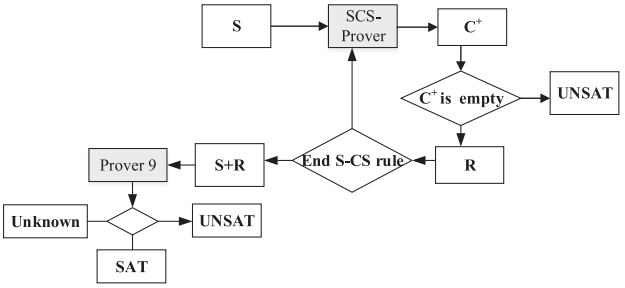

Figure 3 | The framework of *CoProver*.

iii. Dynamic information for literal priority queues.
The dynamic weight function is: $d(l) = use_l + \alpha f_i$, $\alpha > 1$ where use_l indicates the actual usage count of literal l, $\boldsymbol{f_i}$ denotes the statistics on the number of failures of the l in the inference process. It gives higher priority to literal with smaller $\boldsymbol{d(l)}$.

3.4.3. Other dynamic restriction strategies

According to the limitation of computing resources, such as CPU time, memory size, and storage space，we give the dynamic restriction strategies based on the *S-CS* rule.

The main idea is to set some thresholds, initially small values. When some property of a new clause exceeds these thresholds, discard this new clause. If the *S-CS* rule fails and the proof-search restart, these thresholds are increased.

These limits are as follows:

- Limit the maximum function layer of clause.
At first, we count the maximum function nesting layer of the original clause, let $f_{limit} = [f_{max}/2, f_{max}]$. The proof-search begin of the limit f_{limit}, if maximum funcion nesting layer of $\boldsymbol{C}^+$ exceeds f_{limit}, it is discarded. If the S-CS proof-search fails, $f_{limit} = f_{limit+1}$.
- Limit the maximum size of clause.
As same, count the maximum length of clause in S, let $|\boldsymbol{C}|_{limit} = [1, |C|_{max}]$. If $|\boldsymbol{C}^+| > |\boldsymbol{C}^+|_{limit}$, the $\boldsymbol{C}^+$ is discarded

4. THE COMBINED ATP BASED ON S-CS RULES

The *CoProver* is a hybrid inference system constructed by SCS-Prover and Prover9. Figure 3 shows the main framework of *CoProver*.

SCS-Prover is an ATP system based on *S-CS* rule, while *Prover9* is a famous resolution-based prover, its overall architecture is very similar to that of Otter-3.3 [16,17]. In the works of other scholars, *Prover9* is often used nowadays as a benchmark to measure the performances of the proposed provers.

The workflow of *CoProver* is as follows:

Step 1. *SCS-Prover* is first applied to the initial set of clauses S for deduction and generates an inference result, i.e., *contradiction separation clause* $\boldsymbol{C}^+$.

Step 2. If $\boldsymbol{C}^+$ is empty, the proof is found, output *UNSAT*. Otherwise, putting $\boldsymbol{C}^+$ into the clause set *R*.

Step 3. If the terminating condition of *S-CS* rule is reached, then we obtaining new clause set *S'* = *S* ∪ *R*. Otherwise, go *Step1*.

Step 4. Input *S'* into the *Prover9* for proof-search and obtaining a result.

In the above process, there are some points.

i. The termination condition for *SCS-Prover* mainly includes *time* limit and *restart times* limit.

ii. The clause set *R* can be regarded as a lemma set of *S*.

According to the features of *S-CS* rules, we can control the $\boldsymbol{C}^+$ in the deduction process; to generate as more *unit clauses* and *ground clauses* as possible. Experiments show that the addition of these lemmas can greatly improve the ability of *Prover9*.

5. EXPERIMENTS

C++ in the 64-bit Unbuntu operating system is used to implement the first-order logic prover *CSC-Provers* based on the *S-CS* rule, and the combined system *CoProver*.

5.1. The Result of Competition Problems

We testify *CoProver* by the first-order logic problems taken from the category FOFs of the last three years CASC [18] competition (2017–2019). We assign the value 2 to the Max literals number of $\boldsymbol{C}^+$ ($|\boldsymbol{C}^+| = 2$). For the problems in 2017 and 2018, the *CoProver* runs with a CPU time limit of 300 s per problem; specifically, the CPU time limit of *CSC-prover* is 180s, and the CPU time limit of *Prove9* is 120s. In addition, for the competition problems in 2019, we set the total CPU time limit to 180s. first, the maximum CPU time of *CSC-prover* was 120s, and then 60s is left to Prover9

All the experiments are done in an Intel(R) CPU I7 @ 3.4 GHz and 8GB of main memory. The experimental results are presented in Table 1.

Table 1 shows the comparison result of *Prover9* and *CoProver* solving the competition problems in 2017–2019. The "Ratio" row is the radio of the number of solved problems of *CoProver* and *Prover9*. We can see that *CoProver* can solve 63% more problems than by *Prover9*.

Next, the performance of the hybrid ATP system is analyzed further. Table 2 shows the results of different phases of *CoProver*. The "*CSC-Prover*" denotes the results that only the *CSC-Prover* based on *S-CS* rule is used in the first phase, the "*Prover9+*" row denotes the results of the second phase. In this stage, the input clause set ***S'*** for *Prover9* is *S* ∪ *R*, where *S* is the original clauses set, and *R* is the obtained lemma set by *CSC-Prover*. The "*New Solved*" indicates the number of newly solved problems. That is, these problems cannot be proved by using *Prover9* itself alone. For example, for the problems in 2017, the "*Prover9+*" can only solve 55 problems of them, while the other 18 problems cannot be solved only by *Prover9*. The "*Ratio*" is the ratio of the above two values. Higher ratios indicate that the combined system has a better performance than other provers. That is, the lemma-set *R* is more helpful to improve for *Prover9*.

Table 2 also shows that the lemmas provided by *S-CS* rule-based ATP have a significant effect on improving *Prover9*'s performance. Especially for the problems in 2019, even if they are harder. Actually, the performance of *Prover9* has been improved to 47.22%.

Figure 4 shows the relationship between the solution number and CPU time. It can be seen that with some lemmas (i.e., *contradiction separation clause*) are provided by *CSC-Prover*, most problems can be solved in a short time, where many problems cannot be solved by *Prover9*.

Table 1 Comparison of the number of solved problems from Conference on Automated Deduction ATP System Competition (CASC).

	2017 (500)	2018 (500)	2019 (500)
Prover9	140	122	100
CoProver	238	199	163
Ratio (%)	70	63	63

5.2. The Result of Benchmark Problems in TPTP

We testify *CoProver* on the first-logic problems with rating = 1 from the TPTP-v6.1.0 [19] problems library. The rating is an indicator of how hard the problem is, so the problem with rating = 1 meant that it could not be solved by any ATPs system. However, we proposed a hybrid ATP system based on the *S-CS* rule. Namely, the *CoProver* can solve 103 problems with rating = 1, within the CPU time limit 300 s. See the detailed proof in Appendix.

Table 2 Analysis at the different phase of CoProver.

	2017	2018	2019
CoProver	238	199	163
CSC-Prover	183	129	136
Prover9+	55	70	36
New solved	18	21	17
Ratio (%)	32.73	30.00	47.22

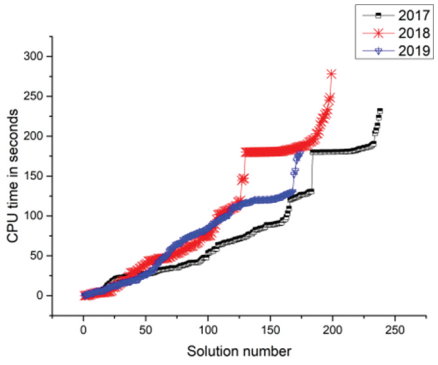

Figure 4 First-order theorem (FOF) results-proof-time by CoProver.

6. CONCLUSIONS AND FUTURE WORKS

In this paper, we firstly proposed a simplified and effective implementation scheme for the *S-CS* rule. We also presented a dynamic heuristic strategy based on the *S-CS* rule scheme. Then, we introduced the *CoProver* as a novel combined ATP based on *S-CS* rule for more effectively and efficiently solving the problems with rating = 1.

Experimental results showed that compared with the *provers* based on binary resolution, the performance of the *prover* based on *S-CS* rule is significantly improved. As a result, in the TPTP problem library, our proposed combinational ATP system CoProver successfully solved 103 problems, which could not be solved by any other existing prover.

In the future, we will further optimize the dynamic coordination strategy based on the separation rules of contradictions. Notably, the first-order logic formula decision strategy of equivalent words will be optimized and improved.

CONFLICT OF INTEREST

The authors declare they have no conflicts of interest.

AUTHORS' CONTRIBUTIONS

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

ACKNOWLEDGMENTS

This paper is supported by the National Natural Science Foundation of China (Grant No. 61673320) and the Fundamental Research Funds for the Central Universities (Grant No. 2682018CX59, 2682020CX59).

REFERENCES

[1] B. Blanchet, Modeling and verifying security protocols with the applied Pi calculus and ProVerif, Found. Trends Privacy Secur. 1 (2016), 1–135.

[2] G. Shi, Toward automated reasoning for analog IC design by symbolic computation – a survey, Integr. VLSI J. 60 (2018), 117–131.

[3] V. D'Silva, D. Kroening, G. Weissenbacher, A survey of automated techniques for formal software verification, IEEE Trans. Comput. Aided Design Integr. Circuits Syst. 27 (2008), 1165–1178.

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

[5] S. Schulz, M. Möhrmann, Performance of clause selection heuristics for saturation-based theorem proving, Automated Reasoning, Springer, Cham, Switzerland, 2016, pp. 330–345.

[6] P. Baumgartner, U. Furbach, B. Pelzer, Hyper tableaux with equality, International Conference on Automated Deduction–CADE, Springer, Berlin, Heidelberg, 2007, pp. 492–507.

[7] Y. Xu, J. Liu, S. Chen, Contradiction separation based dynamic multi-clause synergized automated deduction, Inf. Sci. 462 (2018), 93–113.

[8] F. Cao, Y. Xu, S. Chen, A contradiction separation dynamic deduction algorithm based on optimized proof search, Int. J. Comput. Intell. Syst. 12 (2019), 1245–1254.

[9] M. Baaz, U. Egly, A.Leitsch, Normal form transformations, in: Handbook of Automated Reasoning, vol. I, chap. 5, North Holland, 2001, pp. 273–333.

[10] L. Bachmair, H. Ganzinger, Resolution theorem proving, in: Handbook of Automated Reasoning, vol. I, chap. 2, Elsevier Science 2001, pp. 19–99.

[11] A. Voronkov, Automated reasoning: past story and new trends, in Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, Acapulco, Mexico, 2003, pp. 1607–1612. https://www.escholar.manchester.ac.uk/uk-ac-man-scw:2h1156

[12] L. Kovács, A. Voronkov, First-order theorem proving and vampire, in International Conference on Computer Aided Verification, Saint Petersburg, Russia, 2013, pp. 1–35.

[13] Y. Jiang, P. Papapanagiotou, J. Fleuriot, Machine learning for inductive theorem proving, in: J. Fleuriot, D. Wang, J. Calmet (Eds.), Artificial Intelligence and Symbolic Computation, Springer International Publishing, Cham, Switzerland, 2018, pp. 87–103.

[14] S. Loos, G. Irving, C. Szegedy, Deep network guided proof search, arXiv: Artificial Intelligence (2017).

[15] T. Tammet, GKC: a reasoning system for large knowledge bases, in: P. Fontaine (Ed.), Automated Deduction - CADE 27, vol. 11716, Springer International Publishing, Cham, Switzerland, 2019, pp. 538–549.

[16] W. Mccune, OTTER 3.3 reference manual, Office Sci. Tech. Inf. Tech. Rep, 11, (2007), 217-220.

[17] W. Mccune, L. Wos, Otter-the CADE-13 competition incarnations, J. Automat. Reason. 18 (1997), 211–220.

[18] G. Sutcliffe, The CADE ATP system competition, The competition problems, 2020, http://tptp.org/CASC/.

[19] G. Sutcliffe, The TPTP problem library and associated infrastructure, J. Automat. Reason. 43 (2009), 337.

APPENDIX

Table A.1 | List of problem with rating = 1 by CoProver.

Problem	Time (s)	Problem	Time (s)	Problem	Time (s)
LCL148-1	124.04	RNG010-1	224.05	LCL530+1	242.63
REL040-4	156.56	RNG027-1	105.46	LAT216-1	216.61
ROB006-3	168.78	LAT064-1	160.68	LCL554-1	217.03
GEO046-2	121.13	LAT225-1	79.84	CT065-1	106.55
RNG028-9	86.91	REL038-1	135.08	WV850-1	227.17
RNG028-7	192.77	REL040-1	293.45	ET021-3	162.16
LCL147-1	273.25	REL037-1	41.84	REL039+1	99.76
LAT191-1	224.73	LAT231-1	43.96	LCL477+1	229.48
LAT190-1	218.57	LAT206-1	84.41	NUM005+1	233.36
REL032-1	123.05	ALG238-1	58.17	ALG001-1	219.57
GRP740-1	0.94	LAT221-1	149.93	REL040+4	85.43
REL032-2	284.34	KLE149+1	175.64	LAT226-1	270.66
GRP196-1	187	KLE149+2	2.56	LAT188-1	249.16
REL017-1	1.25	LAT161-1	139.06	KLE047+1	97.77
LAT193-1	279.66	REL016+1	19.33	LAT187-1	143.5
REL016-1	7.1	REL017+1	4.40	WW456-1	44.52
LAT074-1	140	REL020+1	92.31	RNG028-1	295.03
ALG243-1	31.14	REL032+1	91.99	LAT139-1	168.89
LAT077-1	124.6	REL032+2	216.19	LCL417-2	287.98
LAT229-1	280.74	REL040+1	169.07	KLE162+1	133.51
RNG029-2	244.72	REL041+1	40.09	LAT140-1	64.18
GEO031-3	59.89	KLE033+1	58.16	ANA004-3	2.96
REL040-2	169.73	KLE155+2	205.42	GEO113+1	86.60
RNG027-2	249.82	LDA009-2	64.46	REL016+2	99.37
GRP732-1	211.75	ET970+1	11.26	REL039-1	89.98
LAT181-1	252	KLE164+1	75.77	LDA005-2	91.12
LAT078-1	12.26	KLE168+1	37.92	NUM923+1	106.25
LAT224-1	294.31	LAT075-1	276.75	KLE163+1	196.50
ALG241-1	42.4	RNG029-1	294.78	LAT215-1	193.22
LAT202-1	102.77	REL040+2	240.94	LAT189-1	178.95
LAT185-1	78.48	KLE077+1	41.7	WV953-1	77.68
LAT184-1	154.19	REL040+3	153.87	REL017+2	182.15
LAT186-1	258.24	WV719-1	84.05	REL017-2	133.86
LAT180-1	177.93	LCL511+1	110.45	ALG344-1	110.87
ALG010-1	284.51				