A Multi-Clause Dynamic Deduction Algorithm Based on Standard Contradiction Separation Rule

Feng Cao[1, 4, *], Yang Xu[2, 4], Jun Liu[3, 4, *], Shuwei Chen[2, 4], and Jianbing Yi[1]

1. School of Information Engineering, Jiangxi University of Science and Technology Ganzhou 341000, China

2. School of Mathematics, Southwest Jiaotong University, Chengdu 610031, China

3. School of Computing, Ulster University, Northern Ireland, UK

4. National-Local Joint Engineering Laboratory of System Credibility Automatic Verification, Southwest Jiaotong University, Chengdu 610031, China

Abstract: in the past decades, automated theorem proving (ATP) for first-order logic has made good progress, in which binary resolution inference rule plays a crucial role. However, as shown in the latest benchmark library of the ATP system, there are still many practical problems that have not been resolved or cannot be effectively resolved. Recently, in order to overcome the limitations of ATP based on binary resolution inference rules, a novel multi-clause dynamic standard contradiction separation (S-CS) inference rule and its automated deduction theory have been proposed. Based on this theory, this paper first clarifies the generality of this S-CS rule by comparing it with some well-known variants of the binary resolution rule, and then focuses on how to design a specific and effective algorithm along with search strategies to realize the S-CS based deductive theory with its implementation. Specifically, the present work proposes a novel S-CS dynamic deduction algorithm (in short SDDA) based on different strategies and summarizes its implementation procedures. In addition, we focus on evaluating whether SDDA, as a novel perspective multi-clause dynamic automatic deduction algorithm, can be applied on top of the current leading ATP system architectures to further improve their performances. Therefore, SDDA is applied to the current leading first-order ATP systems, i.e., Vampire and E, respectively forming two integrated APT systems, denoted as SDDA_V and SDDA_E. Then the capabilities of SDDA_V and SDDA_E are evaluated on the latest benchmark database TPTP, such as the CASC-J9 problems (FOF division) as well as the hard problems with a rating of 1 in the TPTP benchmark database. The experimental results show the effectiveness of SDDA: SDDA_V outperforms Vampire itself, and SDDA_E, outperforms E itself, and the two improved ATP systems have solved a number of hard problems with the rating of 1 in TPTP, that is, some problems in the latest benchmark database TPTP which have not yet been solved by other current first-order ATP systems.

Keywords: automated theorem proving, first-order logic, binary resolution, standard contradiction separation, S-CS dynamic deduction, ATP systems

---

* The corresponding authors.

E-mail addresses: j.liu@ulster.ac.uk (J. Liu), caofeng@my.swjtu.edu.cn (F. Cao).

1. Introduction

Automated theorem proving (ATP) for first-order logic has made good progress during the past decades and has been successfully applied in many application areas [1, 2], including those state-of-the-art ATPs, such as Vampire [3] and E [4], etc. However, there are still a lot of real problems unsolved or not solved efficiently as illustrated in the latest released version (TPTP-v7.4.0) of the TPTP (Thousands of Problems for Theorem Provers) benchmark library of ATP systems [5]. In addition, from the development trend of ATPs in the past few years [6, 7], the provers with resolution method as the inference core is the current mainstream of ATPs. Therefore, improving the capability of ATPs from the perspective of the resolution method is still an important research direction, and there is much space for further development.

The resolution inference [8] has been greatly developed and has promoted the rapid development of automated reasoning for over five decades [9-11]. Many variants of resolution, e.g., [12-16], redundancy elimination techniques, e.g., [17, 18], and heuristic strategies for controlling proof search, e.g., [19, 20], have been studied from both analytical and empirical perspectives. An important feature of those methods is that only two clauses involved in each deduction step, and only one complementary pair from parent clauses is eliminated. Although the simple and elegant resolution inference scheme has been successful to a great extent, the question naturally raises of whether it can be improved, can we extend the resolution scheme into a contradiction consists of more than two clauses? Accordingly, can we replace the separate set of complementary pairs of literals in binary resolution by a larger set of literals from more than two parent clauses which is called a contradiction?

The recent multi-clause standard contradiction separation (S-CS) calculus for first-order logic [21] can be regarded as a critical initial step to answer this question. S-CS rule takes multiple (two or more) clauses as parent clauses, and selects multiple literals (one or more) from each parent to build a contradictory set of sub-clauses, then infers the clause formed from the disjunction of the non-selected literals of the parent clauses. Especially, the constructed contradiction is oriented to the choice of subsequent clauses, and it is synergistic to eliminate the literals in the selected clause. The S-CS rule combines multiple resolution (full resolution, not just binary) and factoring steps. Full and binary resolution are special cases of S-CS. Details of the S-CS inference rule are provided in Section 2.

S-CS calculus for first-order logic in [21] was introduced to address the above question from a purely proof-theoretical perspective and established a sound and complete multi-clause dynamic automated deduction theory. The use of the S-CS inference rule allows automated deduction to take “large steps" in the search space, which is normally desirable in a search for a refutation. However, the necessary proof search algorithms and strategies to support this theory and achieve the implementation for automation with detailed experiments and case studies have not yet been discussed in [21]. The present work therefore serves for the purpose on how this theory can be achieved through the development of specific and effective algorithms and how it can be implemented on top of the existing architectures of first-order theorem provers in order to improve further their performances and solve some hard problems. Specifically, a multi-clause S-CS dynamic deduction algorithm (called SDDA) is proposed based on different strategies and is then applied to the current leading first-order theorem provers, i.e., Vampire and E, respectively forming two integrated provers, denoted as SDDA_V and SDDA_E. SDDA_V (SDDA_E) runs SDDA and Vampire (E) first sequentially, and then in a cooperative way. Both SDDA_V and SDDA_E are then evaluated on the latest TPTP benchmark database in terms of its generality and capability, e.g., CASC-J9 problems (FOF division) as well as some hard problems with the rating of 1 in the TPTP benchmark database. The experimental results show that SDDA_V outperforms Vampire itself, and SDDA_E, outperforms E itself, and both of the improved provers solve a number of hard problems with rating of 1 in TPTP, i.e., the problems in the latest TPTP benchmark database which have not yet been solved by any of other current provers.

The remainder of this paper is structured as follows. In Section 2, we briefly review some preliminaries about the notations and terminologies, as well as provide an overview of the contradiction separation-based deduction with some clarification of its relationship with some popular variations of the binary resolution. Section 3 discusses the challenges to establish the S-CS deduction algorithm along with the possible ways to address those challenges. In Section 4, the heuristic strategies are introduced which are related to the specific algorithm implementation. Accordingly, multi-clause dynamic deduction framework for proof search is proposed in Section 5 first, then a novel multi-clause dynamic deduction algorithm, called SDDA, is introduced and detailed in Section 6. How can this algorithm be applied and incorporated into the leading first-order theorem provers is discussed in Section 7. Section 8 provides the detailed experimental studies using the TPTP benchmark database to comparatively illustrate the performance of leading provers and their improved ones using the SDDA. The paper is concluded in Section 9 along with indication of planned future work.

2. Overview of Standard Contradiction Separation Rule in First-order Logic

Multi-clause dynamic deduction theory for first-order logic based on standard contradiction separation rule (in short, S-CS deduction) was introduced in [21], this section firstly provides a review of basic concepts and results, then clarifies its relationship with binary resolution, finally summarizes the key features of S-CS deduction.

2.1 Preliminaries

In order to better introduce the multi-clause dynamic deduction algorithm, some definitions and terminologies are reviewed in this section, the readers are referred to [21] for more details.

Definition 2.1 [21] Suppose a clause set $S = \{C_1, C_2, \dots, C_m\}$ in first-order logic, where the following conditions hold:

(1) There does not exist the same variables among $C_1, C_2, \dots, C_m$ (if there exist the same variables, a rename substitution can be applied to make them different);

(2) For each C_i $(i = 1,2,\dots,m)$, a substitution σ_i can be applied to C_i (σ_i could be an empty substitution) and the same literals merged after substitution, the resultant clause is denoted $C_i^{\sigma_i}$. Each $C_i^{\sigma_i}$ is partitioned into two sub-clauses $C_i^{\sigma_i -}$ 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 $C_i^{\sigma_i +}$ do not share the same literal, $C_i^{\sigma_i -}$ is not empty, $C_i^{\sigma_i +}$ might be empty; moreover,

ii) For any $(x_1, \dots, x_m) \in \prod_{i=1}^{m} C_i^{\sigma_i -}$ (*m*-fold Cartesian product, where each element is an *m*-dimensional array, i.e., *m*-tuple), there exists at least one complementary pair among $\{x_1, \dots, x_m\}$ (where “complementary” has the usual meaning of “the negation of”). The conjunction $\bigwedge_{i=1}^{m} C_i^{\sigma_i -}$ is called a *Separated Standard Contradiction* (S-SC);

iii) The resulting clause $\vee_{i=1}^{m} C_i^{\sigma_i +}$, denoted as $\mathcal{C}_m^{s\sigma}$ (here "s" means "standard", $\sigma = \cup_{i=1}^{m} \sigma_i$), is called a *Standard Contradiction Separation Clause* (S-CSC) of *S*.

The inference rule that produces a new clause $\mathcal{C}_m^{s\sigma}$ is called a *Standard Contradiction Separation* rule in first-order logic, in short, an S-CS rule.

In this definition, every input clause is assumed to be split into two parts (denoted as positive and negative, although the notation has nothing to do with the sign of the involved literals) in such a form that the conjunction of the "negative" parts is a contradiction; then, the disjunction of the "positive" parts of the clauses, i.e., S-CSC, is the intended output of the rule.

Example 2.1 Let $S = \{C_1, C_2, C_3, C_4\}$ be a clause set in first-order logic, where $C_1 = P_1(f(a)) \vee P_2(b,b)$, $C_2 = P_2(f(b),b) \vee P_2(b,b)$, $C_3 = \sim P_1(f(a)) \vee P_1(b) \vee \sim P_2(x_1,x_2) \vee P_3(x_1,x_3)$, $C_4 = P_1(b) \vee \sim P_1(x_4) \vee \sim P_2(f(x_5),b) \vee \sim P_3(f(b),x_6)$. Here a and b are constants, f is a function symbol, x_1,x_2,x_3,x_4,x_5,x_6 are variables, P_1,P_2,P_3 are predicate symbols.

Applying the S-CS rule to the 4 clauses C_1, C_2, C_3, C_4, let a variable substitution σ_i and the resultant clause $C_i^{\sigma_i}$ be:

i	σ_i	$C_i^{\sigma_i}$
1	$\varnothing$	$P_1(f(a)) \vee P_2(b,b)$
2	$\varnothing$	$P_2(f(b),b) \vee P_2(b,b)$
3	$f(b)/x_1, b/x_2$	$\sim P_1(f(a)) \vee P_1(b) \vee \sim P_2(f(b),b) \vee P_3(f(b),x_3)$
4	$f(a)/x_4, b/x_5, x_3/x_6$	$P_1(b) \vee \sim P_1(f(a)) \vee \sim P_2(f(b),b) \vee \sim P_3(f(b),x_3)$

Partition the $C_i^{\sigma_i}$ as follows:

i	$C_i^{\sigma_i +}$	$C_i^{\sigma_i -}$
1	$P_2(b,b)$	$P_1(f(a))$
2	$P_2(b,b)$	$P_2(f(b),b)$
3	$P_1(b)$	$\sim P_1(f(a)) \vee \sim P_2(f(b),b) \vee P_3(f(b),x_3)$
4	$P_1(b)$	$\sim P_1(f(a)) \vee \sim P_2(f(b),b) \vee \sim P_3(f(b),x_3)$

So, the S-SC is:

$(P_1(f(a))) \wedge (P_2(f(b),b)) \wedge (\sim P_1(f(a)) \vee \sim P_2(f(b),b) \vee P_3(f(b),x_3)) \wedge (\sim P_1(f(a)) \vee \sim P_2(f(b),b) \vee \sim P_3(f(b),x_3))$.

And the S-CSC involving the synergized deduction of 4 clauses is:

$C_5 = \mathcal{C}_4^{s}(C_1, C_2, C_3, C_4) = P_1(b) \vee P_2(b,b)$.

Definition 2.2 [21] Suppose a clause set $S = \{C_1, C_2, \dots, C_m\}$ in first-order logic. $\Phi_1, \Phi_2, \dots, \Phi_t$ is called a *standard contradiction separation based dynamic deduction sequence* (S-CS deduction) from *S* to a clause Φ_t, denoted as $\mathfrak{D}^s$, if

(1) $\Phi_i \in S$; or

(2) there exist $r_1, r_2, , \dots, r_{k_i} < i$, $\Phi_i = \mathfrak{C}_{k_i}^{s\theta_i}(\Phi_{r_1}, \Phi_{r_2}, \dots, \Phi_{r_{k_i}})$.

where $\theta_i = \bigcup_{j=1}^{k_i} \sigma_j$, σ_j is a substitution to Φ_{r_j}, $j = 1,2,\dots,k_i$, $i \in \{1,2,\dots,t\}$.

The k_i in (2) varies with the deduction process, which means that the number of clauses involved in the contradiction separation in each deduction process could be different from each other, i.e., not fixed. This reflects the meaning of "dynamic deduction". Dynamic selection of different numbers of clauses during the deduction process provides much flexibility which provides an effective way to overcome the two-clause restriction to continue the proof search using multiple paths, therefore enhances the adaptive behaviour of the automated deduction. Clearly there are many choice points in the S-CS deduction that need to be determined and controlled. This is the subject of Sections 4 and 5. The algorithm in Section 6 aims to achieve this dynamic deduction and it is a core base to implement a theorem prover.

Theorem 2.1 (**Soundness**) [21] Suppose a clause set $S = \{C_1, C_2, \dots, C_m\}$ in first-order logic. $\Phi_1, \Phi_2, \dots, \Phi_t$ is an S-CS based dynamic deduction sequence from *S* to a clause Φ_t. If Φ_t is an empty clause, then *S* is unsatisfiable.

Theorem 2.2 (**Completeness**) [21] Suppose a clause set $S = \{C_1, C_2, \dots, C_m\}$ in first-order logic. If *S* is unsatisfiable, then there exists an S-CS based dynamic deduction sequence from *S* to an empty clause.

2.2 Summary of distinct features of S-CS dynamic deduction

Some distinct features of S-CS dynamic deduction are illustrated and summarized here.

Example 2.2 Let $S = \{C_1, C_2, C_3, C_4\}$ be a clause set in first-order logic, where

$C_1 = P_1(a) \vee P_2(c, x_1)$, $C_2 = P_2(a, b) \vee P_2(c, x_2)$,

$C_3 = \sim P_1(a) \vee \sim P_2(x_3, x_4) \vee P_3(x_3, f(a)) \vee P_4(x_4)$,

$C_4 = \sim P_2(x_6, x_5) \vee \sim P_3(a, f(a)) \vee P_4(x_5)$.

Here a, b, c are constants, $x_1, x_2, x_3, x_4, x_5, x_6$ are variables, f is a function symbol, P_1, P_2, P_3, P_4 are predicate symbols.

We have the variables substitutions $(a/x_3, b/x_4, b/x_5, a/x_6)$ and obtain an S-CSC $C_5 = \mathcal{C}_4^{s\sigma}(C_1, C_2, C_3, C_4) = P_4(b) \vee P_2(c, x_1)$.

The process of S-CS rule deduction is shown in Table 2.1.

Table 2.1 S-CS deduction in Example 2.2

	C_2	C_3	C_1	C_4
$C_i^{\sigma_i +}$	$P_2(c, x_2)$	$P_4(b)$	$P_2(c, x_1)$	$P_4(b)$
$C_i^{\sigma_i -}$	$P_2(\mathrm{a}, \mathrm{b})$	$\sim P_1(a) \vee P_3(a, f(a)) \vee \sim P_2(a, b)$	$P_1(a)$	$\sim P_3(a, f(a)) \vee \sim P_2(a, b)$

Different from binary resolution method as a static resolution, where only two clauses are involved and a complementary pair of literals is eliminated in each deduction step, S-CS deduction has the following distinguish features [21-23]:

(1) *Multi-clause deduction*. The number of input clauses can be more than two in each S-CS deduction step, synergized deduction among clauses can be reflected, and generates an S-CSC based on more literals elimination. In Example 2.2, $\mathcal{C}_4^{s\sigma}(C_1, C_2, C_3, C_4)$ is a result from 4 clauses C_1, C_2, C_3, C_4. From Table 2.1, the S-CS deduction has four clauses involved, if we select a subset of the literals from C_1, C_2, C_3, C_4 to build a standard contradiction, it will infer the disjunction of the non-selected literals on the four involved clauses, it is a synergized deduction method. From the perspective of literals elimination, one literal is eliminated in C_1 and C_2 respectively, three literals are eliminated in C_3, and two literals are eliminated in C_4, therefore the S-CS deduction can achieve more literals elimination, thereby improving the deduction efficiency.

(2) *Dynamic deduction*. Which input clauses, how many input clauses and which literals from input clauses are selected to construct standard contradictions are controlled dynamically in the process of the S-CS deduction during a search space.

(3) *Flexible deduction*. The input clauses are flexible in the process of the S-CS deduction. From Table 2.1, there is no requirement about any literal relevance between C_1 and C_2, it does not contain a complementary pair of literals, nor does it contain the same literal, and even predicate symbols are allowed to be different between input clauses.

(4) *Guided deduction*. The input clauses can play a guiding role in the subsequent clause selection. In Example 2.2, the clause C_4 is selected according to the input clauses C_1, C_2, C_3.

(5) *S-CSCs usually have fewer literals*. S-CS deduction separates a standard contradiction which is a literal set from input clauses, and the S-CSCs are generated by the disjunction of the non-selected literals of the input clauses can be controlled by heuristic strategy, so S-CSCs usually have fewer literals, and a large proportion of them are unit clauses.

In order to provide a graphical and intuitive illustration on the essential features of the S-CS deduction, as well as the essential difference from binary resolution deduction, we use the funnel as an intuitive figure to show the automated deduction process from the input clause set.. Fig. 2.1 and Fig. 2.2 [21] below show a graphical funnel view comparison between the binary resolution deduction process and the S-CS based dynamic deduction process. The one coming out from the exit of the funnel is the final output. Fig. 2.1 actually also illustrates some insights why the pre-processing and simplification steps are essential in the binary resolution deduction, even take the majority of the steps and time, and also why lots of work have been focused on splitting and simplifying the clause set into the simpler ones just because the exit is too narrow. Fig. 2.2 illustrates the dynamic and flexible nature of the S-CS based dynamic deduction. This dynamic nature reflects non-determinism in terms of which clauses and how many of them involved in each deduction and have its ability to arrive at outcomes using various routes. This type of nondeterminism is especially beneficial for the problems in mathematical theorem proving when there is a single outcome with multiple paths by which the outcome may be discovered.

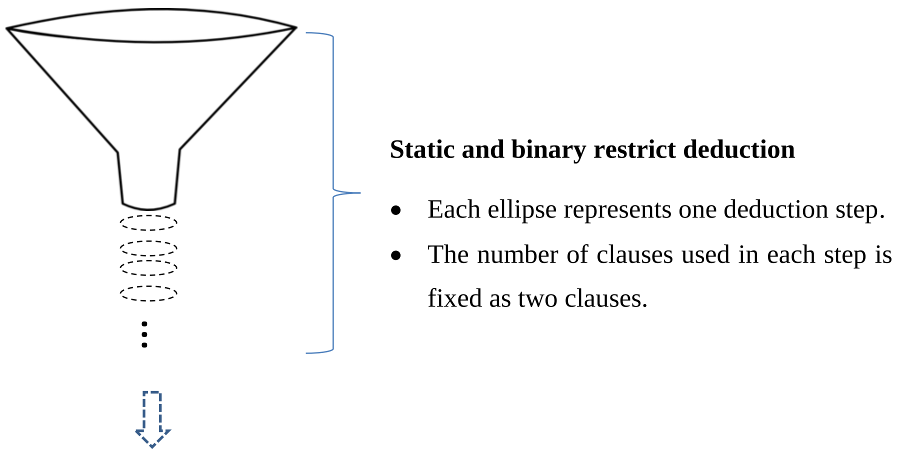
Static and binary restrict deduction
• Each ellipse represents one deduction step.
• The number of clauses used in each step is fixed as two clauses.

Fig. 2.1 The graphical funnel view of binary resolution deduction process

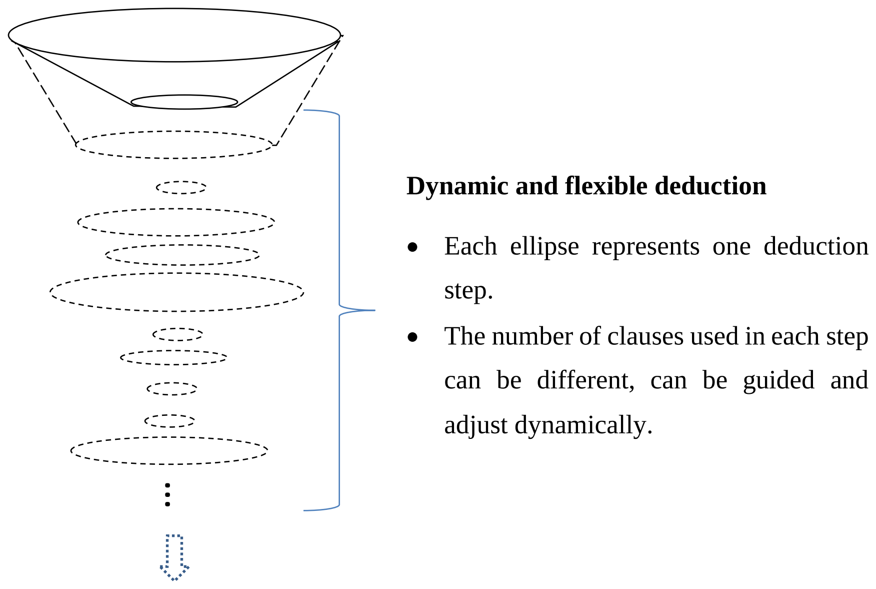
Dynamic and flexible deduction
- Each ellipse represents one deduction step.
- The number of clauses used in each step can be different, can be guided and adjust dynamically.

Fig. 2.2 The graphical funnel view of S-CS based dynamic deduction process

2.2 Generality of S-CS rule compared with binary resolution and its variations

This section clarifies the generality of the S-SC rule by comparing it with some well-known variants of the binary resolution used by some of the state-of-the-art provers.

The research of multi-clause deduction can be traced back to some resolution methods of handling several clauses, such as linear resolution [24], unit resulting resolution [25], hyper-resolution [12], which are briefly summarized below:

1) *Linear resolution* (LR) is a multi-step resolution process with a clause called a *center clause* and another one called a *side clause*; each generated clause is used as the next central clause. This method makes full use of generated clause to perform deduction and retains the intermediate new generated clause in the process of deduction.

2) *Unit resulting resolution* (UR) is a binary tree structure whose one of leaf nodes must be a unit clause (each deduction step requires a unit clause involved), and each path of the binary tree must generate a unit clause or an empty clause. The deduction process starts from a unit clause and ends to the final generation of a unit clause or an empty clause.

3) *Hyper-resolution* (HR) is also a multi-step deduction process. The clauses to be resolved are divided two types: clauses with only positive literals (called as *electrons*) and other clauses with one or more negative literals (called as *nucleus*). In the process of hyper-resolution,

the nucleus is resolved with a series of electrons, and an electron is generated as the final resultant clause where intermediate new generated clauses are discarded.

The above variants of binary resolution are used for handling multi-clauses, but each deduction step only involves two clauses and eliminates a complementary pair of literal (two literals), which are essentially a multi-step deduction of binary resolution and different from the S-CS rule.

Remark 2.1 Binary resolution and its variations are restricted forms of S-CS.

The number of clauses involved in each S-CS inference can be more than two. When the number of clauses is limited to two, the S-CS becomes a binary resolution. Variations of binary resolution were proposed by restricting or specifying the resolution path, their essence are still binary resolution in each deduction step, so the variations of binary resolution are also restricted forms of the S-CS, such as semantic resolution [25, 26] and lock-resolution [27].

In the following, some variations of the S-CS are provided which help to clarify the relationships between variants of the binary resolution and the S-CS.

Definition 2.3 In each S-CS deduction step, if it requires that: 1) the input clauses must contain at least one unit clause; 2) the S-CSC is a unit clause or an empty clause, then this deduction is called as a unit resulting resolution based on the S-CS rule (SCS-UR).

Example 2.3 Let $S = \{C_1, C_2, C_3, C_4, C_5\}$ be a clause set in first-order logic, where $C_1 = {\sim}P_1(x_1) \vee P_2(x_1) \vee {\sim}P_3(x_2)$, $C_2 = P_1(a) \vee P_1(b)$, $C_3 = {\sim}P_2(x_3)$, $C_4 = P_3(a)$, $C_5 = P_3(f(b))$. Here a, b are constants, x_1, x_2, x_3 are variables, f is a function symbol, P_1, P_2, P_3 are predicate symbols, the process of UR resolution is shown in Fig. 2.3.

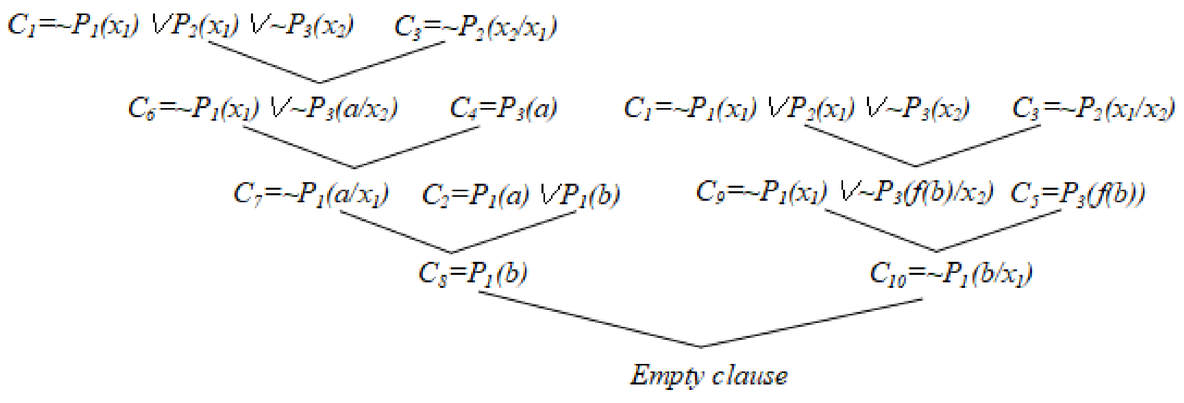

Fig. 2.3 The process of UR resolution for Example 2.3

Corresponding to the UR resolution for Example 2.3, the processes of the SCS-UR deduction are shown in Tables 2.2 and 2.3.

Table 2.2 The first SCS-UR deduction for Example 2.3

	C_2	C_4	C_1	C_3
$C_i^{\sigma_i+}(=C_6)$	$P_1(b)$			
$C_i^{\sigma_i-}$	$P_1(a)$	$P_3(a)$	$\sim P_1(a/x_1) \vee P_2(a/x_1) \vee \sim P_3(a/x_2)$	$\sim P_2(a/x_3)$

Table 2.3 The second SCS-UR deduction for Example 2.3

	C_6	C_5	C_1	C_3
$C_i^{\sigma_i+}$				
$C_i^{\sigma_i-}$	$P_1(b)$	$P_3(f(b))$	$\sim P_1(b/x_1) \vee P_2(b/x_1) \vee \sim P_3(f(b)/x_2)$	$\sim P_2(b/x_3)$

As shown in Tables 2.2 and 2.3, two unit clauses $\sim P_2(x_3)$ and $P_3(a)$ are involved in the first SCS-UR deduction, which eliminates three complementary pairs of literals in one deduction step and the S-CSC is a unit clause. The final generated S-CSC is an empty clause in the second SCS-UR deduction and also two unit-clauses are involved.

Definition 2.4 In each S-CS deduction step, if it requires that: 1) the input clauses is unrestricted and can come from the original clause set in the first deduction step; 2) in addition to the first deduction step, the input clauses must contain a clause which is an S-CSC generated by the last deduction, then this deduction is called as a LR based on the S-CS rule (SCS-LR).

Example 2.4 Let $S = \{C_1, C_2, C_3, C_4, C_5\}$ be a clause set in first-order logic, where $C_1 = P_1(x_1) \vee \sim P_2(a) \vee P_4(x_1)$, $C_2 = \sim P_1(a) \vee P_4(a)$, $C_3 = P_2(x_2)$, $C_4 = P_3(x_3) \vee \sim P_4(a)$, $C_5 = \sim P_3(f(a)) \vee \sim P_4(x_4)$. Here a is a constant, x_1, x_2, x_3, x_4 are variables, f is a function symbol, P_1, P_2, P_3, P_4 are predicate symbols, the process of LR is shown in Fig. 2.4.

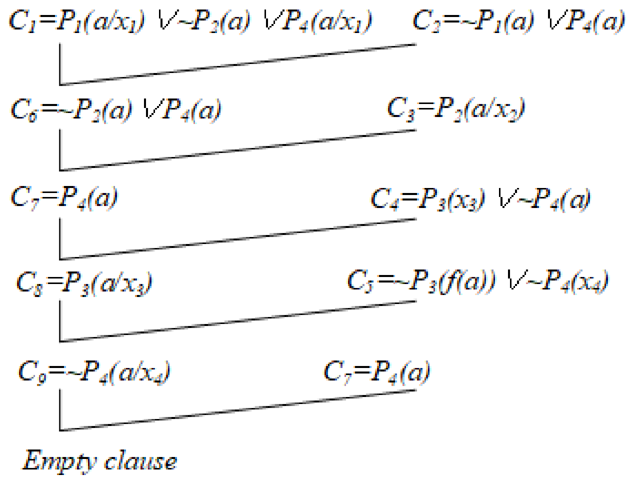
C_1=$P_1(a/x_1)$ ∨~$P_2(a)$ ∨$P_4(a/x_1)$
C_2=~$P_1(a)$ ∨$P_4(a)$
C_6=~$P_2(a)$ ∨$P_4(a)$
C_3=$P_2(a/x_2)$
C_7=$P_4(a)$
C_4=$P_3(x_3)$ ∨~$P_4(a)$
C_8=$P_3(a/x_3)$
C_5=~$P_3(f(a))$ ∨~$P_4(x_4)$
C_9=~$P_4(a/x_4)$
C_7=$P_4(a)$
Empty clause

Fig. 2.4 The process of LR for Example 2.4

Corresponding to the LR, the process of SCS-LR deduction for Example 2.4 is shown in Tables 2.4 and 2.5:

Table 2.4 The first SCS-LR deduction for Example 2.4

	C_3	C_1	C_2
$C_i^{\sigma_i +}(= C_6)$		$P_4(a/x_1)$	$P_4(a)$
$C_i^{\sigma_i -}$	$P_2(a/x_2)$	$P_1(a/x_1) \vee {\sim}P_2(a)$	${\sim}P_1(a)$

Table 2.5 The second SCS-LR deduction for Example 2.4

	C_5	C_4	C_6
$C_i^{\sigma_i +}$			
$C_i^{\sigma_i -}$	${\sim}P_3(f(a)) \vee {\sim}P_4(a/x_4)$	${\sim}P_4(a) \vee\ P_3(f(a)/x_3)$	$P_4(a)$

As shown in Tables 2.4 and 2.5, the input clauses are all from the original clause set in the first S-CS deduction, and the input clauses of the second S-CS deduction contain an S-CS clause C_6 which is generated in the first S-CS deduction.

Definition 2.5 In each S-CS deduction step, if it requires that: 1) clauses in the clause set are divided into two types: clauses with only positive literals (called as *electrons*) and other clauses with one or more negative literals (called as *nucleuses*); 2) the input clauses only contains a nucleus; 3) this S-CS deduction finally generates an electron; 4) the generated S-CS clauses are discarded in the process of deduction. This deduction is called as a hyper-resolution based on S-CS rule (SCS-HR).

Example 2.5 Let $S = \{C_1, C_2, C_3, C_4, C_5\}$ be a clause set in first-order logic, where $C_1 = {\sim}P_1(x_1) \vee {\sim}P_2(a)\ \vee P_4(x_1)$, $C_2 = P_1(a) \vee P_4(a)$, $C_3 = P_2(x_2)$, $C_4 = P_3(x_3)$, $C_5 = {\sim}P_3(f(a)) \vee {\sim}P_4(x_4)$. Here a is a constant, x_1, x_2, x_3, x_4 are variables, f is a function symbol, P_1, P_2, P_3, P_4 are predicate symbols, the process of HR is shown in Fig. 2.5.

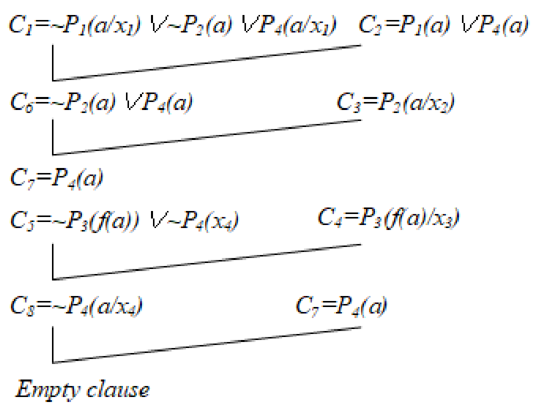
$C_1={\sim}P_1(a/x_1) \vee {\sim}P_2(a) \vee P_4(a/x_1)$
$C_2=P_1(a) \vee P_4(a)$
$C_6={\sim}P_2(a) \vee P_4(a)$
$C_3=P_2(a/x_2)$
$C_7=P_4(a)$
$C_5={\sim}P_3(f(a)) \vee {\sim}P_4(x_4)$
$C_4=P_3(f(a)/x_3)$
$C_8={\sim}P_4(a/x_4)$
$C_7=P_4(a)$
Empty clause

Fig. 2.5 The process of HR for Example 2.5

Corresponding to the HR, the process of SCS-HR deduction for Example 2.5 is shown in Tables 2.6 and 2.7:

Table 2.6 The first SCS-HR deduction for Example 2.5

	C_3	C_1	C_2
$C_i^{\sigma_i+}(= C_6)$		$P_4(a/x_1)$	$P_4(a)$
$C_i^{\sigma_i-}$	$P_2(a/x_2)$	$\sim P_1(a/x_1) \vee \sim P_2(a)$	$P_1(a)$

Table 2.7 The second SCS-HR deduction for Example 2.5

	C_5	C_4	C_6
$C_i^{\sigma_i+}$			
$C_i^{\sigma_i-}$	$\sim P_3(f(a)) \vee \sim P_4(a/x_4)$	$P_3(f(a)/x_3)$	$P_4(a)$

As shown in Tables 2.6 and 2.7, the input clauses only contains a nucleus, but contains two electrons, and finally generate an electron in the first S-CS deduction, and the input clauses of the second S-CS deduction only contains a nucleus, but contains two electrons, and finally generate an empty clause.

Following the similar way as Theorem 2.1, we can prove the following theorems:

Theorem 2.3 (**Soundness**) Suppose a clause set $S = \{C_1, C_2, \ldots, C_m\}$ in first-order logic. $\Phi_1, \Phi_2, \ldots, \Phi_t$ is an SCS-UR based dynamic deduction sequence from *S* to a clause Φ_t. If Φ_t is an empty clause, then *S* is unsatisfiable.

Theorem 2.4 (**Soundness**) Suppose a clause set $S = \{C_1, C_2, \ldots, C_m\}$ in first-order logic. $\Phi_1, \Phi_2, \ldots, \Phi_t$ is an SCS-LR based dynamic deduction sequence from *S* to a clause Φ_t. If Φ_t is an empty clause, then *S* is unsatisfiable.

Theorem 2.5 (**Soundness**) Suppose a clause set $S = \{C_1, C_2, \ldots, C_m\}$ in first-order logic. $\Phi_1, \Phi_2, \ldots, \Phi_t$ is an SCS-HR based dynamic deduction sequence from *S* to a clause Φ_t. If Φ_t is an empty clause, then *S* is unsatisfiable.

In summary, the S-CS is a generic inference method and a further development of binary resolution principle. Its good characteristics place the important theoretical guarantee for the deduction and proof search design and implementation, which are the key contributions of the present work, and will be detailed in Sections 4, 5 and 6.

3. Multi-clause Dynamic Deduction Algorithm: Challenges and Solutions

This established S-CS based automated deduction theory in [21, 23, 28-32] is just a first step towards the development of a proof search procedure that could be implemented as an effective S-CS based theorem prover. The ability and efficiency of proof search is the most critical part of the performance of the first-order logic automated theorem provers [3, 4, 33-36]. Practical implementation of the S-CS based automated deduction further hinges on specific algorithms and strategies making the “right” single S-CS step including the “suitable selection” or “full use” of input clauses to be involved in each deduction process useful for proof search.

The static binary resolution adopts the saturation algorithm [37-39] to achieve the proof search, which is clear and easy to implement. Compared with the binary resolution, the proposed S-CS deduction offers more chances or new windows of algorithms and implementations development for proof search, this multi-clause synergized S-CS deduction however also increases the difficulty of proof searching due to multiple clauses involvement in the search process, the strict saturation algorithm is apparently not feasible anymore. To establish this kind of S-CS based proof search algorithms or strategies and implementation is challenging, but it does not mean it is impossible. That is the motivation and major focus of the present work. The main challenges are summarized below along with the possible solutions proposed respectively. One specific algorithm is then proposed and implemented (as detailed in Sections 5 and 6) as one potential solution and evaluated through the benchmark in Sections 7 and 8.

Challenge 1: how to effectively control the number of clauses involved in each multi-clause dynamic deduction? In principle, due to the multi-clause and dynamic nature, there is no restriction given on how many clauses can be selected for each deduction. It may become rather difficult and time consuming for the large-scale clause set. It may lead to another question, that is, how to effectively control when each deduction will stop based on the number of clauses involved?

Possible solutions: 1) set a threshold for the number of input clauses for each multi-clause dynamic deduction, i.e., specifying how many clauses can be involved in each deduction process. For example, assume the threshold is 200, if 200 clauses are involved in the S-CS dynamic deduction, then this multi-clause dynamic deduction will stop and get the S-CSCs. It is a static control method; 2) control the input clauses by setting the constraints of S-CSC, including the threshold of the maximum number of literals or the maximum term depth. The feasible method is to make the generation of S-CSC a slowly increasing process in terms of the number of literals and the maximal term depth. When an S-CSC has reached either of the thresholds, this multi-clause dynamic deduction will stop and get the S-CSCs. It is a dynamic control method.

Our further theoretical analysis and practice show that the more input clauses involved in each S-CS deduction step the better the inference efficiency. The reason behind is that it can separate a more flexible and large standard contradiction from a large number of input clauses.

Challenge: 2: how to avoid getting stuck in certain proof search when the input clauses and their selected order are the same after some S-CS deductions?

The S-CS dynamic deduction generates an S-CSC by selecting input clauses and then separating a standard contradiction with literals from the input clauses. The deduction algorithm generally selects the input clause through clause selecting strategy, and the literal is selected by literal selecting strategy. Different from the binary resolution using the saturation algorithm, the S-CS deduction is controlled dynamically, and it is possible to undergo the repetitive proof search, which leads to the S-CS clause which has been generated before.

Possible solutions: This is a difficult part of the S-CS dynamic deduction algorithm implementation, which can be handled in the following ways: 1) in order to avoid getting stuck in certain proof search effectively, it is necessary to portray some information about each S-CS deduction. In each S-CS deduction step, some information about whether an input clause and the involved literals participate in deduction is an acceptable proof search or an unacceptable proof search. The S-CS deduction history are recorded by deduction weight of clause and literal. Because the deduction weight is changed dynamically in each deduction step, the deduction algorithm can select clause and literal according the changed deduction weight; 2) change clause selection strategy and literal selection strategy dynamically after a period of the S-CS deduction. Feasible practice is that multiple strategies are used one by one in equal time slices for the total set time during the problem proving, the previously generated S-CSC used the previous strategies can be allowed to use for next dynamic deduction with the next strategy.

Using different strategies provide different clause selection strategy and literal selection methods, which can avoid getting stuck in certain proof search.

Challenge 3: Intuitively, when there are *m* numbers of input clauses (*m*>2), it may takes more time to generate an S-CSC than binary resolution which just only two clauses involved, this can result in slower proof search speed during a problem proving. In addition, *m* numbers of input clauses participating in deduction only generate one S-CSC, some proof search may be lost. Therefore, how to handle this challenge to improve the efficiency of the S-CS dynamic deduction?

Possible solutions: Actually, according to the definition of S-CS rule (Definition 2.1), it can be seen that *m* input clauses participated in deduction, and the previous deduction of *m-1* input clauses also satisfies the requirements of the S-CS rule. In order to improve S-CS deduction's efficiency and prevent possible proof search lost, feasible practice is that the generated S-CSCs can be reserved in the process of S-CS dynamic deduction, that is to say, the S-CS deduction of *m* input clauses, which can generate a S-CSC set by separating different contradictions, including the S-CSCs such as ($\mathcal{C}_2^{s\sigma}(C_1, C_2)$), ($\mathcal{C}_3^{s\sigma}(C_1, C_2, C_3)$),…, ($\mathcal{C}_{m-1}^{s\sigma}(C_1, C_2, \dots, C_{m-1})$).

Challenge 4: The state of the art first-order logic automated theorem provers adopts the saturation method, and the proof search is very clear. The S-CS rule is a multi-clause deduction method, so how to plan the proof search according to certain rules based on the S-CS dynamic deduction?

Possible solutions: This is another major challenge for the S-CS dynamic deduction. In general, it is difficult to find a certain deduction scheme for multi-clause. The difficulty is reflected in how to effectively perform the next deduction after the completion of an S-CS dynamic deduction. Feasible practice is that that S-CS deduction can be an iteration deduction method based on the first input clause (called as *iteration deduction*), and the input clauses and literals used to construct standard contradictions can be selected by deduction weight. As the clause set continues to increase, when all the selectable clauses are completed the iteration deduction as the first input clause, the completed clauses can be restarted iteration deduction as the first input clause under a new clause set (called as *sheave deduction*).

The above summary of challenges and possible solutions place a good motivation and basis for the following proposed strategies and algorithms.

4. Heuristic Strategies

In the process of theorem proving, heuristic strategies are used frequently to perform optimal proof search, including some machine learning methods [40-43], which are useful for finding refutation, thus improve the performance of a prover. In the last decade, most state-of-the art provers have developed and implemented mature heuristic strategies [19, 20, 39, 44-46], and achieved good results.

In general, different inference mechanisms have large differences in the way of proof search, so their supporting heuristic strategies are also different. As the first multi-clause dynamic deduction method, the heuristic strategies for the S-CS dynamic deduction are quite different from the methods utilized in those famous provers. In order to implement a multi-clause dynamic deduction algorithm effectively, some basic heuristic strategies according to the different selection angle of the input clauses and literals are utilized to construct a standard contradiction. This section introduces these basic clause/literal selection strategies along with the strategies for setting some global thresholds, which are closely related to the multi-clause dynamic deduction algorithm introduced in Section 6.

4.1 Clause selection strategy

The heuristic control of S-CS dynamic deduction is based on various measures of each literal and clause. Clause selection strategy refers to the method of selecting an input clause to perform S-CS dynamic deduction, and it is implemented based on the combined uses of several clause attribute measures. Those clause attribute measures mainly include:

(1) *Acceptable deduction (AD) weight of a clause* [22, 32]. This measure is based on two concepts: deduction weight of a clause and acceptable deduction. The former represents the number of times a clause has been used in an S-CS dynamic deduction during a theorem proving. The latter means an S-CS dynamic deduction that generates an S-CSC which is either not redundant or satisfies the set global threshold (e.g., the global threshold of the number of literals or the maximum term depth etc.) (in other words, is acceptable). Correspondingly, this generated S-CSC is called an acceptable clause. Therefore, AD weight of a clause is the number

of times that a clause has participated in an acceptable S-CS dynamic deduction during a theorem proving. A clause used in an acceptable deduction once, its AD weight is increased by 1.

Accordingly, a clause with the smaller AD weight is selected preferentially to perform the S-CS dynamic deduction, because this clause may be seldom selected or most of the generated S-CSCs due to this clause's involvement are rarely acceptable, therefore, excessive frequent use of certain clauses is avoided. To avoid falling into the local optimum, the AD weight of a clause is updated dynamically when the set global threshold is reached. The AD weight of a clause C_i is updated as follows:

$$ADW(C_i)_{K+1}= ADW(C_i)_K/Avg(ADW(C_1)_K,\ldots, ADW(C_m)_K),\ i=1, 2,\ldots, m;\ K=0, 1, \ldots. \qquad (1)$$

In Eq. (1), *ADW* stands for the AD weight, *Avg* means average, and *K* counts the times of the AD weight of a clause being updated.

(2) *Unacceptable deduction (UD) weight of a clause* [22, 32]. UD weight is the number of times that a clause has participated in an unacceptable S-CS dynamic deduction during a theorem proving. An unacceptable S-CS dynamic deduction means an S-CS dynamic deduction that generates an S-CSC which is either redundant or does not satisfy the set global threshold (i.e., unacceptable). Correspondingly, this generated S-CSC is called an unacceptable clause. A clause participated in an unacceptable deduction once, its UD weight is increased by 1.

Accordingly, a clause with the smaller UD weight is selected preferentially to perform the S-CS dynamic deduction, because this clause may be seldom selected or most of the generated S-CSCs due to this clause's involvement are acceptable, therefore excessive frequent use of certain clauses is avoided. To avoid falling into the local optimum, the UD weight of a clause is updated dynamically when the set global threshold is reached. The UD weight of a clause C_i is updated as follows:

$$UDW(C_i)_{K+1}= UDW(C_i)_K/Avg(UDW(C_1)_K,\ldots, UDW(C_m)_K),\ i=1, 2,\ldots, m;\ K=0, 1, \ldots; \qquad (2)$$

In Eq. (2), *UDW* stands for unacceptable deduction weight, *Avg* means average, and *K* counts the times of the UD weight of a clause being updated.

(3) *Clause complexity* (CC). The unified substitution may cause the structure of a clause more complicated, especially for the multi-clause deduction because the variables in an input

clause can be substituted many times in the deduction process. This may lead to rather complicated structure for the generated clause and affect the deduction efficiency. Because only the function symbols in a clause have different depths, so the complexity of a clause is measured by the complexity of function symbols included, which is measured by the depth of function symbols as follows:

$$CC(C_i)=\Sigma\,(FO(T_f)),\ i=1, 2,\ldots, m; \tag{3}$$

$$FO(T_f)=Dep(T_f)+\Sigma\,(FO(SubT_f)). \tag{4}$$

In Eqs. (3) and (4), CC stands for clause complexity, T_f means the terms included in the clause C_i, FO stands for the function symbol complexity, Dep stands for the depth of function symbol, $SubT_f$ is a sub-function symbol of T_f.

Example 2.6 Let $C_1=P(f(f(f(a))))+P(b)$. Here a and b are constants, f is a function symbol. According to Eqs. (3) and (4), we have:

$$Dep(f(f(f(a))))=3,\ Dep(f(f(a)))=2,\ Dep(f(a))=1,$$

$$\Sigma\,(FO(f(f(a))))=Dep(f(f(a)))+\Sigma(FO(f(a)))=2+Dep(f(a))=3,$$

$$CC(C_1)=Dep(f(f(f(a))))+\Sigma(FO(f(f(a))))=6.$$

(4) *The number of literals*. The number of literals included in a clause is an important indicator of clause evaluation. The resolvent of binary resolution usually contains many literals, but the S-CS rule can eliminate multiple literals by separating a standard contradiction (a literal set), so the generated S-CSC usually contains less literals. The number of literals in an input clause can reflect the number of literals in the generated S-CSC to certain extent [22], which is used to control the number of literals in an S-CSC, thus improving the deduction efficiency.

4.2 Literal selection strategy

Literal selection strategy refers to the method of selecting a literal from the input clause to construct a standard contradiction, and it is implemented based on the combined uses of several literal attribute measures. Those literal attribute measures mainly include:

(1) *Deduction weight of a literal*. Deduction weight of a literal is the number of times that a literal has been used in the S-CS dynamic deduction for constructing standard contradiction, and the generated S-CSC is unacceptable. It effectively controls the order in which a literal is selected in an involved clause for deduction. In the process of the S-CS dynamic deduction, if

a literal is used to construct a standard contradiction once, and the generated S-CSC is unacceptable, then the deduction weight of this literal is increased by 1.

A literal with the smaller deduction weight is selected preferentially to construct standard contradiction, because this literal may be seldom selected or most of the generated S-CSCs due to this literal's involvement are acceptable, and excessive frequent use of certain literals is avoided effectively. To avoid falling into the local optimum, deduction weight of a literal is updated dynamically when the set global threshold is reached. The deduction weight of a literal P_i from clause C is updated as follows:

$$DW(P_i)_{K+1}= DW(P_i)_K/Avg(DW(P_1)_K,\ldots, DW(P_m)_K),\ i=1, 2,\ldots, m;\ K=0, 1, \ldots. \tag{5}$$

In Eq. (5), *DW* stands for deduction weight, *Avg* means average, and *K* counts the times of the deduction weight of a literal being updated.

(2) *Literal complexity*. We describe literal complexity by its function symbols included, and it can be measured as follows:

$$LC(P_i)= \Sigma\,(FO(T_f)),\ i=1, 2,\ldots, m. \tag{6}$$

In Eq. (6), *LC* stands for literal complexity, *FO* stands for function symbol complexity (its computing method is the same as Eq. (3)), *i* is the serial literal number of a clause, *m* is the number of literals in the clause.

(3) *The number of variables*. Because the S-CS rule is a multi-clause deduction, and is a process of constructing standard contradictions dynamically, the literals that have participated in the construction of a standard contradiction can continue to construct the subsequent new standard contradictions. In general, in the process of the S-CS dynamic deduction, literals with more variables are more flexible than the literals which only contains constants or fewer variables.

(4) *Literal weight*. Literal weight is used to characterize the statistics of symbols in a literal, it is the number of symbols (predicates, functions, constants, and variable occurrences) in a literal (the initial value of literal weight is 1).

4.3 Global threshold setting strategy

Global thresholds are the parameters required in the design of the S-CS dynamic deduction algorithm; their values are set in advance through the global thresholds set strategy. The global thresholds used are listed as follows:

(1) *Global threshold of AD weight of a clause*. According to our empirical experience via experiments, the default setting is 500.

(2) *Global threshold of UD weight of a clause*. According to our empirical experience, the default setting is 1000.

(3) *Global threshold of deduction weight of a literal*. According to our empirical experience, the default setting is 1000.

(4) *Global threshold of the maximum number of literals in a generated S-CSC*. According to our empirical experience, the default setting is 6. If the number of literals in the generated S-CSC exceeds this threshold, this S-CS dynamic deduction is regarded as an unacceptable deduction.

(5) *Global threshold of the maximum literal term depth in a generated S-CSC* (literal term depth is the maximum depth of function nesting in a term in a literal). According to our empirical experience, the default setting is 15. If the maximum term depth of the generated S-CSC exceeds this threshold, this S-CS dynamic deduction is regarded as an unacceptable deduction.

(6) *Global threshold of iteration deduction times*. According to our empirical experience, the default setting is 500.

(7) *Global threshold of sheave deduction times*. According to our empirical experience, the default setting is 1000.

(8) *Global threshold of the maximum number of input clauses in an S-CS deduction*. According to our empirical experience, the default setting is 2000.

(9) *Global threshold of the runtime for each problem proving*. According to the standard runtime from the TSTP (Thousands of Solutions from Theorem Provers), the default setting is 300 seconds.

(10) *Global threshold of the remaining memory for each problem proving*. According to our empirical experience, the default setting is 200 megabytes.

In order to solve different problems effectively, an alternative method is used to set some relevant thresholds according to the attribute values of the original clause set parsed in the pre-processing stage, such as setting the S-CSC thresholds of the maximum number of literals and the maximum literal term depth.

5. S-CS Dynamic Deduction Framework for Proof Search

In general, the first-order logic theorem proving is considered to be an infinite search space problem, and it is impossible in general to perform all possible proof search within a limited runtime. In order to improve the ability of the first-order logic automated theorem prover, it is important to perform proof search effectively. Saturation method can plan proof search effectively for the saturation-based provers, which use binary resolution, but it does not suit for multi-clause dynamic deduction because it is rather complicated to handle multi-clause synergize deduction. S-CS dynamic deduction framework for proof search is proposed in this section which is used to describe the scheme and guideline on how to select different input clauses to apply the S-CS rule iteratively, and generate different S-CSCs during the process of theorem proving. The S-CS dynamic deduction framework for proof search considers the comprehensiveness and efficiency, and performs proof search in the following ways:

(1) *Iteration deduction*. It refers to the deduction process in which an appointed input clause (denoted as C) repeats M (a global threshold) times as the first input clause of the S-CS dynamic deduction algorithm (i.e., SDDA introduced in Section 6). This is called M-iteration deduction based on the appointed input clause C, so that the input clause C can be used to perform the S-CS dynamic deduction and proof search as preferred. The S-CS dynamic deduction plans proof search of iteration deduction based on an appointed input clause in this way: in order to give full play to the diversity of an S-CS dynamic deduction and generate more S-CSCs, each iteration deduction based on the first input clause also includes the proof search based on another appointed input clause in the selected input clauses, such as selecting the second input clause to continue to perform iteration deduction. Therefore, an S-CS dynamic deduction can perform more proof search and improve deduction efficiency effectively.

In addition, in order to reduce repetitive proof search, when iteration deduction based on an appointed input clause are performed, if an acceptable S-CSC cannot be obtained, this iteration deduction will stop, and a new iteration deduction base on a next appointed input clause starts.

(2) *Sheave deduction*. After an S-CS iteration deduction, the whole clause set is changed, and the proof search is also changed. In order to make the proof search more comprehensive, when all input clauses in the clause set have completed *M* times of iteration deduction, it is necessary to restart a new S-CS iteration deduction based on those completed and appointed input clauses and continue to find refutation under the new clause set (called *a sheave deduction*). The restarting time is set as *N* (a global threshold). By this way, the iteration deduction based on an appointed input clause restarts time after time under the new clause set to ensure the comprehensiveness of proof search.

Next, we demonstrate the S-CS dynamic deduction framework including the above two deduction schemes through an example below.

Example 2.7 Let $S = \{C_1, C_2, C_3, C_4, C_5\}$ be a clause set in first-order logic, where $C_1 = {\sim}P_1(x_1) \vee {\sim}P_2(a) \vee P_4(x_1)$, $C_2 = P_1(f(a)) \vee P_4(a)$, $C_3 = P_2(x_2)$, $C_4 = P_3(x_3)$, $C_5 = {\sim}P_3(f(b)) \vee {\sim}P_4(x_4)$, $C_6 = P_1(x_1) \vee {\sim}P_2(a) \vee P_3(c)$. Here a, b, c are constants, x_1, x_2, x_3, x_4 are variables, f is a function symbol, P_1, P_2, P_3, P_4 are predicate symbols.

Assume the times of *Iteration deduction* is 3. Now using C_3 as the first input clause, we have the variables substitutions $(a/x_1, a/x_2, f(b)/x_3, a/x_4)$ and obtain an S-CSC $C_7 = \mathrm{C}_5^{s\sigma}(C_3, C_1, C_2, C_4, C_5) = {\sim}P_1(a)$.

Continue to use C_3 as the first input clause, we have the variables substitutions $(f(a)/x_1, a/x_2, f(b)/x_3, a/x_4)$ and obtain an S-CSC $C_8 = \mathrm{C}_5^{s\sigma}(C_3, C_1, C_2, C_4, C_5) = P_4(f(a))$.

Continue to use C_3 as the first input clause, we have the variables substitutions $(a/x_1, a/x_2)$ and obtain an S-CSC $C_9 = \mathrm{C}_4^{s\sigma}(C_3, C_6, C_7, C_8) = P_3(c)$.

The *Iteration deduction* using C_3 now ended.

Next using C_4 as the first input clause, we have the variables substitutions $(f(b)/x_3, f(a)/x_4)$ and obtain an S-CSC $C_{10} = \mathrm{C}_4^{s\sigma}(C_4, C_5, C_7, C_8) = \emptyset$.

If the deduction does not generate empty clause, then continue to use C_4 as the first input clause to apply the S-CS rule and generate a new S-CSC. After 3 times of *Iteration deduction* using C_4, another clause in clause set is selected as the first input clause for new *Iteration deduction*. When all clauses in clause set are selected for *Iteration deduction*, it is called a *Sheave deduction*. Repeat the above steps to start a new *Iteration deduction*.

In contrast to saturation method for planning proof search by binary resolution, S-CS dynamic deduction uses iteration deduction and sheave deduction to plan its deduction proof search. In order to balance the inference ability and efficiency, the deduction selects an optimal input clause by clause selection strategy to perform iteration deduction and selects an optimal literal by literal selection strategy to be used in each S-CS deduction step, which is able to improve proof search efficiency.

6. A Novel S-CS Dynamic Deduction Algorithm

Based on the definition of the S-CS rule, the established S-CS dynamic deduction theory, and heuristic strategies proposed in Section 4, a novel and effective S-CS Dynamic Deduction Algorithm, in short SDDA, is designed and detailed in this section. This algorithm can make full use of input clauses, the S-CSCs are generated with a slowly increasing process in terms of the number of literals and the maximum literal term depth for generating unit S-CSCs effectively.

The basic flow of the SDDA firstly selects an input clause through clause selection strategy and fixes this first input clause for iteration deduction. Literal selection strategy selects a literal in the input clause and searches for a clause list in the whole clause set, where the clauses in the list should contain some literals that can construct standard contradictions along with the selected literal. Iteratively traverse this clause list as the new input clause, perform the S-CS rule and generate an S-CSC. When the generated S-CSC is an unacceptable clause, try to construct another standard contradiction with the selected input clauses, until an acceptable S-CSC is generated. Make full use of the current input clause (if it is completed, select another input clause), the deduction is repeated until an exit condition is satisfied, so the S-CS dynamic deduction is performed one time, and finally a S-CSC set is obtained.

In more specific details, SDDA is described in the following steps:

Step 1: If it is an iteration deduction, select the clause which is the first input clause in last S-CS deduction; otherwise, select a clause according to clause selection strategy, and mark it as the first labeled input clause *C* for iteration deduction.

Step 2: According to literal selection strategy, select a literal in the input clause to search a clause list (called *Q*) from the whole clause set in which can construct a standard contradiction. Reorder the clause list *Q* according to clause combined sorting method of acceptable deduction weight from small to large, unacceptable deduction weight from small to large, and the number of literals.

Step 3: Sequentially traverse the clause in *Q* and search an optimal input clause. The search process is summarized: sequentially traverse clause C_i in *Q*, apply the S-CS rule with the input clauses, separate a standard contradiction (there are many different forms of contradictions which need to be tried one by one), then generate an S-CSC. If the S-CSC is an empty clause, exit the traversal, and go to Step 7. Record the number of literals and the maximum term depth of the S-CSC. When the traversal is complete, an optimal input clause is obtained. Otherwise, if the number of literals in S-CSC exceeds the global threshold, mark the next selected clause as the new labeled input clause *C*, perform backtracking operations and go to Step 2. Here again, the searched optimal clauses that have tried different contradiction construction will no longer be searched.

Step 4: The S-CSC is checked whether it is an acceptable clause. Considering the following two conditions:

1) If an S-CSC is a tautology, then it is an unacceptable clause.

2) If an S-CSC is a redundant clause by forward simplification, then it is an unacceptable clause.

If one of the above two conditions is satisfied, backtracking operations is performed, and go to Step 3.

Backtracking operations are described as follows:

1) Clear the record of proof search by the current input clause participating in the deduction and remove literals in the S-CSC and the constructed standard contradiction, which come from the current input clause.

2) Clear substitutions which are caused by the current input clause participating in the deduction.

3) Unacceptable deduction weight of the current input clause is increased by 1.

4) Deduction weight of the literals in the current input clause which are used for constructing a standard contradiction is increased by 1.

Step 5: If the generated S-CSC is an acceptable clause, the acceptable deduction weight of the current input clause is increased by 1. Add this generated S-CSC into a temporary clause set (the initial set is empty, called *R*).

Step 6: The algorithm exit conditions is checked: if it is satisfied, go to Step 7. Make full use of the current input clause C_i according to whether C_i contains substitutions. If yes, reconstruct clause C_i, go to Step 3; otherwise, go to Step 3.

Step 7: Exit this S-CS dynamic deduction. The exit conditions include:

1) If the S-CSC is an empty clause, then the unsatisfiable conclusion is obtained.

2) The number of input clauses has exceeded the set global threshold.

3) The clause in the whole clause set has used up.

4) The runtime has reached the set global threshold.

5) The remaining memory has reached the set global threshold.

6) If the generated S-CSC set *R* does not contain an empty clause, an internal simplification is performed on *R*, then apply forward simplification on *R* using the clause in the whole clause set, and apply backward simplification on the whole clause set using the clause in *R*, then each S-CS clause in *R* is added in the whole clause set.

This algorithm will be executed iteratively following the deduction framework for proof as introduced in Section 5 until the refutation found, or the set thresholds of runtime and the remaining memory are reached.

The pseudo-code for SDDA is described below.

SDDA: S-CS Dynamic Deduction Algorithm

Input: a given clause set (S); selected clauses are denoted as *g*, *c*; a selected literal is denoted as *p*; two empty clause sets are denoted as D, Q.

Output: a generated S-CS clause set R.

```
1: $g$ = selectFirstClause($S$)
2: $D = D \cup g$
3: $p$ = selectLiteral($g$)
4: $Q$ = selectCandidateClauseSet($p,S$)
5: $c$ = TraversalClauseSet($Q$)
6: While $c \neq$ null begin
7:     $r$ = extendedSplitContradiction($c$, $D$)
8:     if $r == \varnothing$
9:         return “Unsat”
10:    else if checkInvalid($r$, $S$)
11:        Backtracking($c$)
12:        goto 5
13:    else
14:        $R = R \cup r$
15:        $D = D \cup c$
16:        if checkExitConditions()
17:            goto 25
18:        if checkFullUse($c$)
19:            $c$ = reconstructClause($c$)
20:            goto 5
21:        else
22:            goto 5
23:     end
24: end
25: innerSimplification($R$)
26: foreach $c \in R$
27:    backwardSimplification($S$, $c$)
28: $S = S \cup R$
```

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

selectFirstClause(*S*): Select a clause as the first input clause in S-CS deduction.

selectLiteral(*g*): Select a literal in the clause *g* according to literal selection strategy, which is used to construct contradictions.

selectCandidateClauseSet(*p, S*): Select a clause set in the clause set *S*, which at least contains a unified complementary literal with the literal *p*.

TraversalClauseSet(Q): Sequentially traverse the clause set *Q* and return a clause according to the given search conditions.

extendedSplitContradiction(*c*, *D*): Apply the S-CS rule with the current input clause *c* and the input clauses *D* that has participated in the deduction.

checkInvalid(*r*, *S*): Check whether the generated S-CS clause *r* is an acceptable clause under the clause set *S*.

Backtracking(c): It is used to return to the last S-CS deduction step.

check*ExitConditions*(): Check whether one of the deduction exit conditions is satisfied, and the exit conditions are described in the algorithm description.

checkFullUse(c): Check whether the clause *C* needs to be fully used, and the condition of full use are described in the algorithm description.

reconstructClause(c): reconstruct a new clause which does not contain any substitutions according to the clause *c*.

innerSimplification(R): Apply forward simplification in the S-CS clause set *R*.

backwardSimplification(*S*, *c*): Check whether the clauses in the clause set *S* are redundant clauses using the clause *c*.

SDDA implements a sequence of steps, aiming at finding a proof of a first-order problem in the TPTP FOF syntax [47].

1) E is used to convert the problem to clause normal form (in the TPTP CNF syntax).

2) The CNF is parsed and stored in a binary-tree based structure, with sharing of nodes for variables and constants.

3) A strategy is set, with control parameter values that are either specified by the user, or automatically computed from the input clauses. The strategy parameterizes the clause selection and literal selection, as well as those global thresholds set, described in Section 4.

4) The deduction framework for proof search described in Section 5 is invoked with the strategy and the SDDA algorithm above, to search for a refutation.

5) If a refutation is found, a proof is output. Proof checking is available by verifying the S-CS inference using the (trusted) Prover9 system [48].

SDDA has an option to do strategy scheduling, using multiple strategies. In this case each strategy is given an allocated CPU time limit. The clause storage (clause set) is retained throughout, so that clauses inferred using one strategy are passed on when SDDA switches to the next strategy. SDDA is implemented mainly in C++.

7. Combination of SDDA with Leading First-Order Theorem Provers

The leading first-order automated theorem provers Vampire [3] and E [4] have won the first and the second places of the CADE ATP System Competition (CASC) respectively for several years since 2002 [49]. It is worth noting that although the proposed S-CS-based SDDA algorithm offers a new window of algorithms and implementations development for improving proof search, has not yet included some important rules or strategies as those leading provers, such as superposition (to handle equality effectively), many effective heuristic strategies, as well as many pre-processing or simplification procedures etc. So, it does not make sense or even impossible to compete with those two leading provers. Now that those two provers have been very well developed, therefore the SDDA evaluation in terms of its effectiveness, applicability and generality, is focused on applying SDDA on top of the existing architectures of ATP systems to see if it can enhance further their performances. Therefore, SDDA is applied to Vampire 4.1 [1] (released in the CASC-26) and leads to a combined prover called SDDA_V; SDDA is also applied to E 2.3 [2] and leads to a combined prover called SDDA_E.

SDDA_V (SDDA_E) runs SDDA and Vampire (E) first sequentially, and then in a cooperative way. The combination scheme is summarized below: SDDA_V (SDDA_E) first tries to solve the given problem with Vampire (E), and if unsuccessful it tries with SDDA. If neither solves the problem, then selected clauses inferred by SDDA are combined with the original problem's clauses to form a new problem for Vampire (E). Experiential experiences show that the initial run of Vampire (E) solves many problems, and the subsequent run of SDDA solves some problems that cannot be solved by Vampire (E). However, the unique strength of SDDA_V (SDDA_E) in the third step, as the inferred clauses passed from SDDA allow Vampire (E) to solve some further problems that neither SDDA nor Vampire (E) can solve alone.

8. Experimental and Performance Analysis

8.1 Experimental setup

[1] http:/tptp.cs.miami.edu/CASC/
[2] https://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html

For a comprehensive evaluation purpose, SDDA_V (the given version number is 0.1) and SDDA_E (the given version number is 0.1) are tested on the CADE ATP competition CASC-J9 FOF division problems respectively. We also test SDDA_V and SDDA_E on some hard problems with rating of 1 (no current prover can solve them according to the latest benchmark database TSTP solutions library[3]) by setting strategies interactively. The experiments are carried on a PC with 3.6GHz Inter(R) Core (TM) i7-4790 processor and 16GB memory, OS Ubuntu15.04 64-bit, with the standard CPU time limit of 300s. In order to check the correctness of proof procedure by SDDA, the well-known ATP Prover9 [48] is used to check each deduction step, which is useful for finding refutation. For experimental comparisons, Vampire 4.1 and E 2.3 are tested under the same hardware environment.

8.2 Experiment results overview and analysis

8.2.1 Performance analysis of SDDA_V 0.1 for CASC-J9 FOF division problems

In order to evaluate if SDDA is able to enhance the performance of Vampire, SDDA_V 0.1 and Vampire 4.1 are tested on the CASC-J9 FOF division problems respectively in the same hardware environment. Fig 8.1 shows the comparison on solved problems by SDDA_V 0.1 and Vampire 4.1.

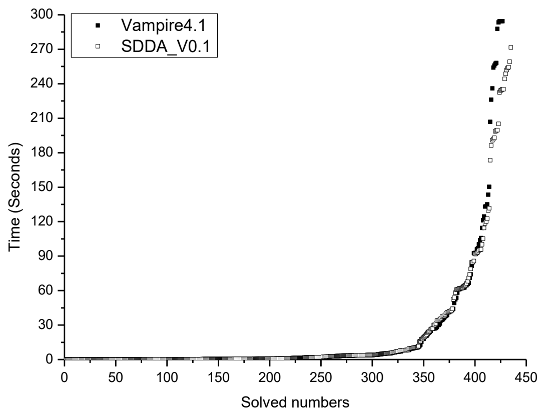

Fig. 8.1 Comparison on solved problems by SDDA_V 0.1 and Vampire 4.1

From Fig. 8.1, SDDA_V 0.1 has solved 435 problems with 8 more than Vampire 4.1 which has solved 427 problems. The average time spent for 427 problems by Vampire 4.1 is 18.9 seconds, and 21.35 seconds spent for 435 problems by SDDA_V 0.1. In particular, the average

[3] http://www.tptp.org/TSTP/

time spent for 427 problems by SDDA_V0.1 is 17.02 seconds, 1.88 seconds less than that of Vampire 4.1. Therefore, SDDA_V0.1 has better time efficiency than Vampire 4.1 when solving the same number of problems.

According to the CPU time spent, the number of solved problems by SDDA_V 0.1 and Vampire 4.1 are almost within the half of standard runtime (300 seconds). From the 150 seconds point onwards, SDDA_V 0.1 outperforms Vampire 4.1. Within 200 seconds, SDDA_V 0.1 has solved 422 problems with 8 more than Vampire 4.1. Within 250 seconds, SDDA_V 0.1 has solved 430 problems with 13 more than Vampire 4.1. Within 300 seconds, SDDA_V 0.1 has solved 435 problems with 8 more than Vampire 4.1. Experimental results show that SDDA_V 0.1 outperformed Vampire 4.1 in terms of the ability and time efficiency.

In addition, for those 73 problems unsolved by Vampire 4.1, SDDA_V 0.1 solved 14 problems accounting for 19.2% of the total, with the average CPU time 223.8 seconds. Table 8.3 lists those 14 problems solved by SDDA_V 0.1 but not by Vampire 4.1. 1 of 14 problems were solved by SDDA alone, and 13 solved by Vampire 4.1 with clauses added from the SDDA. For those 14 solved problems, there are 5 problems with rating greater than 0.9; 10 problems with rating greater than 0.8, accounting for 35.7.6% and 71.4% of the total (listed in Table 8.1) respectively. The average TPTP difficulty rating of these problems is 0.84.

Table 8.1 List of problems solved by SDDA_V 0.1 but not by Vampire 4.1

Theorem Name	Rating	Number of formulae	Maximum formula depth	Number of variables	Maximum term depth	CPU time (seconds)
COM132+1	0.75	67	23	362	5	248.61
GEO324+1	0.94	181	46	816	3	244.16
GRA009+1	0.72	18	13	71	3	199.64
GRA009+2	0.69	18	13	71	3	190.77
GRP746+1	0.62	9	6	16	4	254.33
LCL468+1	0.94	43	6	65	5	198.53
LCL474+1	0.94	43	6	65	5	199.32
LCL552+1	0.84	77	6	110	5	253.68
LCL553+1	0.84	77	6	110	5	259.15
LCL558+1	0.88	77	6	110	5	251.91
LCL680+1.005	0.86	3	104	187	1	235.25
NUN056+1	0.95	19	13	62	1	191.92
SWW264+1	0.97	1288	13	2686	12	173.50
SWW384+1	0.81	5250	22	16634	19	232.5

Specially, from Table 8.1, those solved problems with high rating mostly contain a large maximum formulae depth or a large number of literals (e.g., SWW264+1 and SWW384+1). Experimental results show that SDDA_V 0.1 indeed enhances the performance of Vampire 4.1 alone, it can solve some hard problems which cannot be solved by Vampire 4.1, and multi-clause dynamic deduction can eliminate more literals, fully use the input clauses contains lots of variables during the deduction process and has the advantage of dealing with clauses with a large number of literals.

8.2.2 Performance analysis of SDDA_E 0.1 for CASC-J9 FOF division problems

In order to further evaluate if SDDA is able to enhance the performance of E, SDDA_E 0.1 and E 2.3 are also test on the CASC-J9 FOF division problems respectively in the same hardware environment. Fig. 8.2 shows the comparison on solved problems by SDDA_E 0.1 and E 2.3.

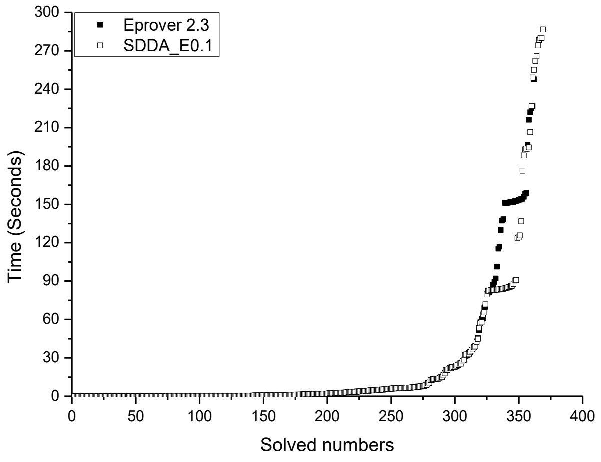

Fig. 8.2 Comparison on solved problems by SDDA_E0.1 and E 2.3

From Fig. 8.2, SDDA_E 0.1 has solved 369 problems with 7 more than E 2.3 which has solved 362 problems. The average time spent for the 362 solved problems by E 2.3 is 20.14 seconds, and 22.54 seconds spent for the 369 solved problems by SDDA_E 0.1. In particular, the average time spent for 362 problems solved by SDDA_E 0.1 is 17.65 seconds, 2.49 seconds less than that of E 2.3. Therefore, SDDA_E 0.1 also has better time efficiency than E 2.3 when solving the same number of problems.

According to the CPU spent time, the number of solved problems by SDDA_E 0.1 and E 2.3 are almost within 260 seconds. From the 260 seconds point onwards, the performance of

SDDA_E 0.1 outperforms E 2.3. Within 270 seconds, SDDA_E 0.1 has solved 364 problems with 2 more than E 2.3. Within 280 seconds, SDDA_E 0.1 has solved 367 problems with 5 more than E 2.3. Within 300 seconds, SDDA_E 0.1 has solved 369 problems with 7 more than E 2.3. Experimental results show that SDDA_E 0.1 outperformed E 2.3 in terms of the ability and time efficiency.

In addition, for those 138 problems unsolved by E 2.3, SDDA_E 0.1 solved 14 problems accounting for 10.1% of the total, with the average CPU time 228.73 seconds. Table 8.2 lists those 14 problems solved by SDDA_E 0.1 but not by E 2.3. 7 of 14 problems were solved by SDDA alone, and 7 solved by E 2.3 with clauses added from the SDDA. For those 14 solved problems, there are 2 problems with the rating greater than 0.9, 11 problems with rating greater than 0.8, accounting for 14.3% and 78.6% of the total (listed in Table 8.2) respectively. The average TPTP difficulty rating of these problems is 0.83. SDDA_E 0.1 solved one problem, GEO506+1, that was not solved by any other system in CASC-J9. The clauses in these problems usually contain many literals, which SDDA is able to deal with effectively with the S-CS inference rule, inferring new clauses with fewer literals. Experimental results show that SDDA_E 0.1 also can solve some hard problems which cannot be solved by E 2.3.

Table 8.2 List of problems solved by SDDA_E0.1 but not by E 2.3

Theorem Name	Rating	Number of formulae	Maximum formula depth	Number of variables	Maximum term depth	CPU time (seconds)
AGT018+1	0.69	556	8	71	5	206.37
AGT022+1	0.81	556	8	72	5	286.61
AGT022+2	0.72	923	8	72	5	274.12
BOO109+1	0.81	3	6	9	6	226.81
GEO506+1	0.91	143	22	564	3	280.01
GEO511+1	0.84	162	22	650	3	188.1
LCL466+1	0.91	43	6	65	5	262.03
SWB016+1	0.72	559	27	973	2	193
SWB022+1	0.84	560	27	981	2	249.15
SWB027+1	0.88	560	27	976	2	176.3
SWB082+1	0.88	560	27	973	2	193.39
SWB094+1	0.88	560	27	973	2	193.42
SWB098+1	0.84	560	27	973	2	194.50
SWW189+1	0.88	1150	14	3058	9	278.39

8.2.3 Performance analysis of SDDA_V and SDDA_E for hard problems with rating of 1

In this section, the ability of SDDA_V and SDDA_E is further evaluated on the problems with rating of 1 within FOF division from TPTP. Table 8.3 lists the total 40 problems with rating of 1 solved by SDDA_V and SDDA_E, 27 of which by SDDA_V and 13 of which by SDDA_E, and the rating of these problems are acquired from TSTP by the time of this paper submission.

From Table 8.3, the average CPU time for each problem is 218.9 seconds. It is rather promising performance considering the number of problem and the average time spent because the problems with rating of 1 are the most difficult, that means, no current state-of-the-art ATP can solve them based on the claim from the latest benchmark database TPTP. That also means the leading provers Vampire and E cannot solve them by themselves, it is indeed the SDDA plays the crucial role in the combined SDDA_V and SDDA_E in solving those problems.

Table 8.3 The list of 40 problems with rating of 1 solved by SDDA_V and SDDA_E

No	Problem	category	Time(s)	Prover	No	Problem	category	Time(s)	Prover
1	ANA005-4	Rating=1	231.7	SDDA_V	21	NUM522+1	Rating=1	292.2	SDDA_V
2	GEO508+1	Rating=1	260.5	SDDA_E	22	NUM522+3	Rating=1	295.3	SDDA_V
3	GEO516+1	Rating=1	293.6	SDDA_V	23	NUM553+3	Rating=1	288.8	SDDA_V
4	GEO326+1	Rating=1	250.2	SDDA_V	24	SET133-6	Rating=1	65	SDDA_V
5	KLE165+1	Rating=1	287.8	SDDA_E	25	SET177-6	Rating=1	192.4	SDDA_E
6	LAT077-1	Rating=1	272.5	SDDA_E	26	SET180-6	Rating=1	122	SDDA_V
7	LCL450+1	Rating=1	169.2	SDDA_V	27	SET182-6	Rating=1	274	SDDA_E
8	LCL450+2	Rating=1	209.3	SDDA_V	28	SET279-6	Rating=1	259.9	SDDA_E
9	LCL471+1	Rating=1	283.8	SDDA_V	29	SET289-6	Rating=1	299	SDDA_V
10	LCL478+1	Rating=1	265.5	SDDA_V	30	SET290-6	Rating=1	156.5	SDDA_E
11	LCL479+1	Rating=1	248.7	SDDA_V	31	SET305-6	Rating=1	166	SDDA_V
12	LCL554+1	Rating=1	263.7	SDDA_V	32	SET345-6	Rating=1	138	SDDA_E
13	LCL560+1	Rating=1	298	SDDA_V	33	SET348-6	Rating=1	128.4	SDDA_V
14	NUM054-1	Rating=1	96	SDDA_V	34	SET368-6	Rating=1	172.1	SDDA_V
15	NUM057-1	Rating=1	267.9	SDDA_V	35	SEU372+2	Rating=1	158.8	SDDA_V
16	NUM076-1	Rating=1	152.7	SDDA_E	36	SEU410+3	Rating=1	167.9	SDDA_E
17	NUM090-1	Rating=1	155.5	SDDA_V	37	SWC186+1	Rating=1	270.5	SDDA_E
18	NUM136-1	Rating=1	155.7	SDDA_V	38	SWC199-1	Rating=1	185	SDDA_V
19	NUM428+1	Rating=1	272.1	SDDA_V	39	SWC199+1	Rating=1	277	SDDA_V
20	NUM486+1	Rating=1	115.6	SDDA_E	40	SWV276-1	Rating=1	298	SDDA_E

The above comparison analysis shows that SDDA_V (SDDA_E) is able to effectively improve Vampire (E) in terms of capability and time efficiency, and the S-CS dynamic deduction can be effectively applied to first-order logic automated theorem prover. The possible reasons can be: 1) the multi-clause dynamic deduction can fully exert the synergized deduction of more than two clauses, and the ability of multi-clause dynamic deduction to proof search is not equivalent to the binary deduction by saturation under heuristic strategies; 2) the multi-clause and dynamic nature of S-CS deduction, which reflects non-determinism in terms of which clauses, how many of them involved in each deduction, and select which clauses from the input clauses to construct contradictions, so the S-CS deduction has the ability to arrive at various proof search by handling multi-clause. This type of non-determinism is maybe beneficial for the hard problems in mathematical theorem proving. Compared with binary resolution method, the S-CS deduction offers more chances or new windows of algorithms and implementations development for proof search. Experimental results show that multi-clause dynamic deduction is an effective complement to binary resolution method, which can arrive at the proof search that current state-of-the-art provers cannot reach, as illustrated in Table 8.3.

9. Conclusions and Future Work

Multi-clause standard contradiction separation (S-CS) calculus for first-order logic is a new deduction theory and method for automated reasoning based on the S-CS rule [21]. The present work focused on how this theory can be achieved through the development of specific and effective algorithms and how it can be implemented on top of the existing architectures of the leading first-order theorem provers in order to further improve their performances and solve some hard problems. For effective implementation, the proposed multi-clause dynamic deduction algorithm SDDA can make full use of input clauses and give full play to the ability of synergized deduction with more than two clauses. Under the proposed heuristic strategies, the input clauses can eliminate more literals through standard contradiction separating, make most of the S-CS clauses are unit clauses or contains fewer literals.

SDDA was applied to the leading provers Vampire 4.1 and E 2.3 and formed the combined provers SDDA_V and SDDA_E respectively. The experimental results have shown that SDDA_V 0.1 outperforms Vampire 4.1 and SDDA_E 0.1 outperforms E 2.3 to a certain extent so indeed enhance their performance. Especially, SDDA_V and SDDA_E have solved some hard problems with rating of 1 that are unsolved by the state-of-the-art ATP system, indicating that the S-CS rule is an effective method for theorem proving and an important extension of the binary resolution method, and the proposed deduction algorithm can effectively improve the ability of the state-of-the-art provers.

Although SDDA has proposed different kinds of heuristic strategies, and the proposed deduction algorithm can effectively perform the optimal proof search, there is still much room for improvement, such as how to effectively perform deduction weights update, how to fully use the synergies of multi-clause to a greater extent. As the clause and literal selection strategies are foundational approach, how to efficient use and enrich these strategies for multi-clause dynamic deduction is the main task. The implementation of multi-clause dynamic deduction does not include the superposition calculus for dealing with equality, adding superposition calculus is the next major research work.

Although SDDA_V 0.1 outperforms Vampire 4.1 and SDDA_E 0.1 outperforms E 2.3 to a certain extent, there is also much room for their improvement. In the setting runtime, multi-clause dynamic deduction can generate a large number of S-CSCs as lemmas, how to effectively evaluate these new clauses and more precise choice as lemmas also is the main task. In terms of combination, the multi-clause dynamic deduction and the leading prover run sequentially according to their runtime, this unidirectional method restricts the ability of combination. Therefore, the more effectively combined multi-clause dynamic deduction with the leading provers is also one of the focus in our future work.

Acknowledgments

This paper is supported by the Natural Science Foundation of China (Grant No. 61673320); The Natural Science Foundation of Jiangxi Province (Grant No. 20181BAB202004); The Humanities and Social Sciences Research Project of the Ministry of Education of P. R. China (Grant No. 20XJCZ H016).

REFERENCES

[1] V. Pavlov, A. Schukin, T. Cherkasova. Exploring Automated Reasoning in First-Order Logic: Tools, Techniques and Application Areas, In: The 4th International Conference on Knowledge Engineering and Semantic Web, vol. 394 in Communications in Computer and Information Science, St. Petersburg, Russia, October 7-9, 2013, pp. 102-116.

[2] G. Burel, G. Bury, R. Cauderlier, D. Delahaye, P. Halmagrand, and O. Hermant. First-Order Automated Reasoning with Theories: When Deduction Modulo Theory Meets Practice. Journal of Automated Reasoning, 64(6)(2020): pp. 1001-1050.

[3] L. Kovács, A. Voronkov, First-order Theorem Proving and Vampire, In: The 25th International Conference on Computer Aided Verification, LNCS 8044, Saint Petersburg, Russia, July 13-19, 2013, pp. 1-35.

[4] S. Schulz, System Description: E 1.8, In: The 19th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LNCS 8312, Stellenbosch, South africa, December 14-19, 2013, pp. 735-743.

[5] G. Sutcliffe, TSTP Solution Domains, http://www.tptp.org/cgi-bin/SeeTPTP?Category= Solutions, Accessed 10 July 2020.

[6] G. Sutcliffe, The 9th IJCAR Automated Theorem Proving System Competition-CASC-J9, AI Communications, 31(6) (2018) 495–507.

[7] G. Sutcliffe, The CADE-27 Automated Theorem Proving System Competition-CASC-27, AI Communications, 32(5-6) (2019) 373 – 389.

[8] J.A. Robinson, A machine-oriented logic based on the resolution principle, Journal of the ACM, 12(1) (1965) 23-41.

[9] J.A. Robinson, A. Voronkov, Handbook of Automated Reasoning, Vols. 1 and 2, the MIT Press and North Holland, 2001.

[10] J. Harrison, Handbook of Practical Logic and Automated Reasoning, Cambridge University Press, 2009.

[11] D. Plaisted, History and prospects for first-order automated deduction, In: The 25th International Conference on Automated Deduction, LNCS 9195, Berlin, Germany, August 1-7, 2015, pp. 3-28.

[12] J.A. Robinson, Automatic Deduction with Hyper-resolution, International Journal of Computer Mathematics, 1(1965) 227-234.

[13] J.R. Slagle, Automatic Theorem Proving with Renamable and Semantic Resolution, Journal of the ACM, 14(4) (1967) 687-697.

[14] A. Degtyarev, R. Nieuwenhuis, and A. Voronkovc, Stratified resolution. Journal of Symbolic Computation, 36(1) (2003) 79–99.

[15] H. de Nivelle, J. Meng, Geometric Resolution: A Proof Procedure Based on Finite Model Search, In: Proc. of the 3th International Joint Conference on Automated Reasoning, LNAI 4130, Seattle, WA, United states, August 17-20, 2006, pp. 303-317.

[16] J. Slaney, B.W. Paleo, Conflict resolution: a first-order resolution calculus with decision literals and conflict-driven clause learning, Journal of Automated Reasoning, 12(4) (2016) 1-24.

[17] B. Loechner, A Redundancy Criterion Based on Ground Reducibility by Ordered Rewriting, In: Proc. of the 2nd International Joint Conference on Automated Reasoning, LNCS 3097, Cork, Ireland, July 4-8, 2004, pp. 45-59.

[18] S. Schulz, Simple and Efficient Clause Subsumption with Feature Vector Indexing, In: Automated Reasoning and Mathematics: Essays in Memory of William W. McCune, LNCS 7788, 2013, pp. 45-67.

[19] K. Hoder, G. Reger, M Suda, A, Voronkov. Selecting the selection, In: The 8th International Joint Conference on Automated Reasoning, LNCS 9706, Coimbra, Portugal, June 27 – July 2, 2016, pp. 313-329.

[20] S. Schulz, M. Mohrmann, Performance of clause selection heuristics for saturation-based theorem proving, The 8th International Joint Conference on Automated Reasoning, LNCS 9706, Coimbra, Portugal, June 27 – July 2, 2016, pp. 330-345.

[21] Y. Xu, J. Liu, S.W. Chen, et al., Contradiction separation based dynamic multi-clause synergized automated deduction, Information Sciences, 462 (2018) 93-113.

[22] F. Cao, Y. Xu, J. Liu, et al., CSE_E 1.0: An integrated automated theorem prover for first-order logic, Symmetry, 11(9) (2019) 1142.

[23] Y. Xu, S.W. Chen, J. Liu, X.M. Zhong and X.X. He, Distinctive features of the contradiction separation based dynamic automated deduction, In: The 13th International FLINS Conference on Decision Making and Soft Computing, Vol. 11, Belfast, UK, August 21-24, 2018, pp: 725-732.

[24] D.W. Loveland, A Linear Format for Resolution, In: Symposium on Automatic Demonstration, 1970, pp. 147-162.

[25] J.A. Robinson, The generalized resolution principle, Journal of Symbolic Computation, 3 (1968) 135-151.

[26] X.H. Liu, A new semantic resolution principle, Journal of Jilin University, 2 (1978) 112-117 (in Chinese).

[27] R.S. Boyer, Locking: A Restriction of Resolution. Doctoral Dissertation, University of Texas at Austin, 1971.

[28] Y. Xu, J. Liu, S.W. Chen, and X.M. Zhong, A novel generalization of resolution principle for automated deduction, In: The 12th International FLINS Conference on Uncertainty Modelling in Knowledge Engineering and Decision Making, Roubaix, France, August 24-26, 2016, pp. 483-488.

[29] S.W. Chen, Y. Xu, Y. Jiang, J. Liu, X.X. He, Some synergized clause selection strategies for contradiction separation based automated deduction, In: The 12th International Conference on Intelligent Systems and Knowledge Engineering, Nanjing, China, November 24-26, 2017, pp. 143-148.

[30] F. Cao, Y. Xu, J. Zhong, G.F. Wu, Holistic deductive framework theorem proving based on standard contradiction separation for first-order logic, In: The 12th International

Conference on Intelligent Systems and Knowledge Engineering, Nanjing, China, November 24-26, 2017, pp. 389-393.

[31] F. Cao, Y. Xu, X.R. Ning and X.C. Wang, Deductive control strategies based on contradiction separation rule, In: The 13th International FLINS Conference on Decision Making and Soft Computing, Vol. 11, Belfast, UK, August 21-24, 2018, pp. 766-773.

[32] F. Cao, Y. Xu, S.W. Chen, et al, A contradiction separation dynamic deduction algorithm based on optimized proof search, International Journal of Computational Intelligence Systems, 12(2) (2019) 1245-1254.

[33] C. Weidenbach, R.A. Schmidt, T Hillenbrand, R Rusev, D. Topic, System description: Spass version 3.0, In: The 21th International Conference on Automated Deduction, LNAI 4603, Bremen, Germany, July 17-20, 2007, pp. 514–520.

[34] A. Voronkov, AVATAR: The architecture for first-order theorem provers, In: The 26th International Conference on Computer Aided Verification, LNCS 8559, Vienna, Austria, July 18-22, 2014, pp. 696-710.

[35] K. Korovin, iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description), In: The 4th International Joint Conference on Automated Reasoning, LNAI 5195, Sydney, Australia, August 12-15, 2008, pp. 292-298.

[36] D. Itegulov, J. Slaney, B.W. Paleo, Scavenger 0.1: A Theorem Prover Based on Conflict Resolution, In: The 26th International Conference on Automated Deduction, LNAI 10395, Gothenburg, Sweden, August 6-11, 2017, pp. 344-356.

[37] J. Denzinger, M. Kronenburg, S Schulz, DISCOUNT: a distributed and learning equational prover, Journal of Automated Reasoning, 18(2) (1997) 189–198.

[38] W. McCune, L. Wos, Otter: the CADE-13 competition incarnations, Journal of Automated Reasoning, 18(2) (1997) 211–220.

[39] A. Riazanov, A. Voronkov, Limited resource strategy in resolution theorem proving, Journal of Symbolic Computation, 36(1) (2003) 101-115.

[40] S. Schulz, Learning Search Control Knowledge for Equational Theorem Proving, In: The Joint 24th German Conference on Artificial Intelligence and 9th Austrian Conference on Artificial Intelligence, LNCS 2174, Vienna, Austria, September 19-21, 2001, pp. 320-334.

[41] M. Khalifa, H. Raafat, M. Almulla, Machine Learning Approach to Enhance the Design of Automated Theorem Provers, In: The 9th International Conference on Neural Information Processing, LNCS 7664, Doha, Qatar, November 12-15, 2012, pp. 673-682.

[42] D. Kühlwein, S. Schulz, J. Urban, E-MaLeS 1.1, In: The 24th International Conference on Automated Deduction, LNAI 7898, Lake Placid, NY, United states, June 9-14, 2013, pp. 407-413.

[43] C. Kaliszyk, J. Urban, FEMaLeCoP: Fairly Efficient Machine Learning Connection Prover, In: The 20th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LNCS 9450, Suva, Fiji, November 24-28, 2015, pp. 88-96.

[44] S. Schulz, Fingerprint Indexing for Paramodulation and Rewriting, In: The 6th International Joint Conference on Automated Reasoning, LNCS 7364, Manchester, UK, June 26-29, 2012, pp. 477-483.

[45] C. Kaliszyk, S. Schulz, J. Urban, J Vyskocil, System Description: E.T. 0.1, In: The 25th international Conference on Automated Deduction, LNCS 9195, Berlin, Germany, August 1-7, 2015, pp. 389-398.

[46] G. Reger, D. Tishkovsky, A. Voronkov, Cooperating Proof Attempts, In: The 25th international Conference on Automated Deduction, LNCS 9195, Berlin, Germany, August 1-7, 2015, pp. 339-355.

[47] G. Sutclifie, J. Zimmer, S. Schulz, TSTP Data-Exchange Formats for Automated Theorem Proving Tools, In W. Zhang and V. Sorge (Ed.), Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems, number 112 in Frontiers in Artificial Intelligence and Applications, IOS Press, 2004, pp. 201-215.

[48] W.W. McCune. Prover9. http://www.cs.unm.edu/ mccune/prover9/. Accessed 2019.

[49] Sutcliffe G. The CADE ATP System Competition. http:/tptp.cs.miami.edu/CASC/. Accessed 30 May 2019.