



## CSE\_E 1.0: An integrated automated theorem prover for first-order logic

Cao, F., Xu, Y., Liu, J., Chen, S., & Ning, X. (2019). CSE\_E 1.0: An integrated automated theorem prover for first-order logic. *Symmetry*, 11(9), 1. Article 1142. <https://www.mdpi.com/2073-8994/11/9/1142/pdf>

[Link to publication record in Ulster University Research Portal](#)

**Published in:**  
Symmetry

**Publication Status:**  
Published (in print/issue): 08/09/2019

**Document Version**  
Author Accepted version

### General rights

The copyright and moral rights to the output are retained by the output author(s), unless otherwise stated by the document licence.

Unless otherwise stated, users are permitted to download a copy of the output for personal study or non-commercial research and are permitted to freely distribute the URL of the output. They are not permitted to alter, reproduce, distribute or make any commercial use of the output without obtaining the permission of the author(s).

If the document is licenced under Creative Commons, the rights of users of the documents can be found at <https://creativecommons.org/share-your-work/ccllicenses/>.

### Take down policy

The Research Portal is Ulster University's institutional repository that provides access to Ulster's research outputs. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact [pure-support@ulster.ac.uk](mailto:pure-support@ulster.ac.uk)

# CSE\_E 1.0: An Integrated Automated Theorem Prover for First-Order Logic

Feng Cao<sup>1,5</sup>, Yang Xu<sup>2,5</sup>, Stephan Schulz<sup>3</sup>, Jun Liu<sup>4,5</sup>, and Shuwei Chen<sup>2,5</sup>

<sup>1</sup> School of Information Science and Technology, Southwest Jiaotong University, China

[caofeng19840301@163.com](mailto:caofeng19840301@163.com)

<sup>2</sup> School of Mathematics, Southwest Jiaotong University, Chengdu, China

[xuyang@home.swjtu.edu.cn](mailto:xuyang@home.swjtu.edu.cn) and [swchen@home.swjtu.edu.cn](mailto:swchen@home.swjtu.edu.cn)

<sup>3</sup> DHBW Stuttgart, Stuttgart, Germany

[schulz@eprover.org](mailto:schulz@eprover.org)

<sup>4</sup> School of Computing, Ulster University, Northern Ireland, UK

[j.liu@ulster.ac.uk](mailto:j.liu@ulster.ac.uk)

<sup>5</sup> National-Local Joint Engineering Lab of System Credibility Automatic Verification, China

**Abstract.** The paper presents an automated theorem prover for first-order logic, called CSE\_E 1.0, which is a combination of two provers CSE and E, where CSE is based on the recently introduced multi-clause standard contradiction separation (S-CS) calculus for first-order logic and E is the well-known equational theorem prover for first-order logic based on superposition and rewriting. The motivation of the combined prover CSE\_E 1.0 is to 1) evaluate the capability, applicability and generality of CSE\_E; 2) take advantage of novel multi-clause S-CS dynamic deduction of CSE and mature equality handling of E to solve more and harder problems. In contrast to other improvements of E, CSE\_E 1.0 optimizes E mainly from the inference mechanism aspect. The focus of the present work is given on the description of CSE\_E including its S-CS rule, heuristic strategies, and the S-CS dynamic deduction algorithm for implementation. In terms of combination, in order not to lose the capability of E and use CSE\_E to solve some hard problems which unsolved by E, CSE\_E 1.0 schedules the running of the two provers by time. It runs plain E first, if E does not find a proof it runs plain CSE, then if does not find a proof some clauses inferred in the CSE run as lemmas are added to the original clause set and the combined clause set is handed back to E for further proof search. CSE\_E 1.0 is evaluated through benchmarks, e.g., CASC-26 (2017) and CASC-J9 (2018) competition problems (FOF division). Experimental results show that CSE\_E 1.0 indeed enhances the performance of E to a certain extent.

**Keywords:** theorem prover, first-order logic, standard contradiction separation, superposition, multi-clause, inference mechanism

## 1 Introduction

Automated theorem proving (ATP) is an important branch of artificial intelligence and has been successfully applied in many application areas [1-2]. However, there are still many unsolved problems in the recently released version (TPTP-v7.2.0) of the TPTP benchmark library [3]. In addition, based on the analysis of the results of CADE ATP systems competitions in the last few years [4], there is still much room for improvement for the state-of-the-art ATP systems on the performance of solving large and hard problems.

During the past decade, a lot of variants of resolution (e.g., [5-11]) or heuristic strategies [12-16] (for clause/literal selection, axioms selection, and proof search) have been studied from both the analytical and empirical perspectives. These methods have effectively improved the capability of ATP systems. However, we noticed that the inference mechanisms applied in the resolution-based ATP systems are binary resolution and its refinements, e.g., linear resolution [7], unit-resulting resolution [8], hyper-resolution [9]. Those refinements are all focused on binary resolution. In each deduction step, binary resolution limits only two input clauses and eliminates a complementary pair of literals. Although the simple and elegant binary resolution inference scheme has been successful in great extent, it naturally raises the question of whether it can be improved. Instead of treating a contradiction as a complementary pair, can we extend it into a contradiction come from two or more than two clauses? Can the inference rule and techniques go beyond binary resolution to enhance the efficiency and versatility of contemporary ATP systems?

The recent multi-clause standard contradiction separation (S-CS) calculus for first-order logic [17] can be regarded as a crucial initial step to answer this question. The S-CS inference rule extends from the existing static binary resolution into an S-CS based dynamic multi-clause (two or more clauses) synergized inference rule. S-CS rule takes multiple clauses as input, selects a subset of the literals from each input clause to build a contradictory set of sub-clauses, and infers the disjunction of the non-selected literals of the input clauses. Especially, the number of clauses taken in each S-CS inference can be guided and adjusted dynamically. This reflects the advantage of S-CS rule, because the synergized deduction among more than two clauses usually cannot be directly reflected by deduction of pairs of clauses many times. S-CS based deduction is a multi-clause, dynamic, synergized, guidable, and controllable deduction method [18]. Because different numbers of clauses can be selected for constructing the contradiction, different clause/ literal selection methods and deduction control methods can be applied during the implementation. Furthermore, during the process of dynamic deduction, the constructed contradiction can guide the subsequent clause selection and literal selection, and the deduction process can be controlled according to the features of generated clause to optimize proof search. When the constructed contradiction is limited to only two clauses, S-CS rule degenerates to a general chains of binary resolution method, therefore binary resolution rule is a special case of S-CS rule.

E is a well-known equational automated theorem prover for first-order logic with powerful equality processing capability [19] based on superposition and rewriting. The prover has successfully participated in many ATP systems competitions [20-22]. Over the past decade, many improvements and extensions of E have been developed from different perspectives, where some successful ones are E.T. [23] and E-Males [24-25].

Their improvements are mainly from the perspectives of the effective selection of axioms and the effective use of strategies for the improvement of E. It is an open question whether E can be improved from the inference mechanism point of view and S-CS rule can be applied effectively to E? The present work introduces a novel ATP system for first-order logic which is one-step further to answer these questions. The prover is called CSE\_E 1.0, which is a combination of CSE and E, where CSE (namely, Contradiction Separation Extension) is based on the above-mentioned S-CS rule for first-order logic. Its motivation is to solve hard problems by using multi-clause dynamic deduction of S-CS rule and mature equality handling of E. In contrast to other improvements of E, CSE\_E 1.0 optimizes E mainly from the inference mechanism aspect. In terms of combination, CSE\_E 1.0 feeds some inferred unit clauses generated by S-CS rule as lemmas along with the original clauses to E for further proof search. Most encouragingly CSE\_E 1.0 (stands on the shoulders of the plain E) achieved second place in the FOF division and awarded the “best newcomer” prize in CASC-J9 [4]. Meanwhile, CSE outperformed some mature provers, such as Prover9 [4]. CSE\_E 1.0 is evaluated through benchmarks in terms of its applicability and capability.

The remaining part of the paper is organized as follows. In Section 2, we briefly review the novel S-CS calculus, and compared with binary resolution based on saturation algorithm, the characteristics of S-CS dynamic deduction are analyzed. Section 3 introduces the heuristic strategies of S-CS dynamic deduction which are used to generate promising lemmas. A multi-clause dynamic deduction algorithm based on S-CS and lemmas filtration method are proposed in Section 4 and Section 5 respectively. Section 6 shows the performance of CSE\_E 1.0. The conclusion and follow-up research work are given in last Section.

## 2 Overview of Multi-Clause S-CS rule in The CSE\_E

Multi-clause S-CS based dynamic deduction theory for first-order logic was introduced in [17], which places the theoretical foundation for the CSE prover. This section provides a review of basic concepts of S-CS rule.

**Definition 2.1** [17] 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  $\sigma_i$  can be applied to  $C_i$  ( $\sigma_i$  could be an empty substitution) and the same literals merged after substitution, denoted as  $C_i^{\sigma_i}$ ;  $C_i^{\sigma_i}$  can be partitioned into two sub-clauses  $C_i^{\sigma_i^-}$  and  $C_i^{\sigma_i^+}$  such that:

1)  $C_i^{\sigma_i} = C_i^{\sigma_i^-} \vee C_i^{\sigma_i^+}$ , where  $C_i^{\sigma_i^-}$  and  $C_i^{\sigma_i^+}$  do not share the same literal,  $C_i^{\sigma_i^+}$  can be an empty clause,  $C_i^{\sigma_i^-}$  cannot be an empty clause; moreover,

2) For any  $(x_1, \dots, x_m) \in \prod_{i=1}^m C_i^{\sigma_i^-}$ , there exists at least one complementary pair among  $\{x_1, \dots, x_m\}$ ,  $\bigwedge_{i=1}^m C_i^{\sigma_i^-}$  is called a *separated standard contradiction* (S-SC);

3) The resulting clause  $\bigvee_{i=1}^m C_i^{\sigma_i^+}$ , denoted as  $\mathcal{E}_m^{s\sigma}(C_1, C_2, \dots, C_m)$  (here “s” means “standard”,  $\sigma = \bigcup_{i=1}^m \sigma_i$ ), is called a *standard contradiction separation clause*

(S-CS Clause) of  $C_1, C_2, \dots, C_m$ .

The inference rule that produces a new clause  $\mathcal{C}_m^{s\sigma}(C_1, C_2, \dots, C_m)$  is called a *standard contradiction separation rule* in first-order logic, in short, an S-CS rule.

**Definition 2.2** [17] 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* from  $S$  to a clause  $\Phi_t$ , denoted as  $\mathcal{D}^S$ , if

(1)  $\Phi_i \in S$ ; or (2) there exist  $r_1, r_2, \dots, r_{k_i} < i$ ,  $\Phi_i = \mathcal{C}_{k_i}^{s\theta_i}(\Phi_{r_1}, \Phi_{r_2}, \dots, \Phi_{r_{k_i}})$ .

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

**Example 2.1** Let  $S = \{C_1, C_2, C_3, C_4\}$  be a clause set in first-order logic, where  $C_1 = P_1(a) \vee P_2(b, f(a))$ ,  $C_2 = P_2(a, b) \vee P_2(b, f(a))$ ,  $C_3 = \sim P_1(a) \vee P_1(b) \vee \sim P_2(x_1, x_2) \vee P_3(x_1, x_3)$ ,  $C_4 = P_1(b) \vee \sim P_1(x_1) \vee \sim P_2(x_2, x_3) \vee \sim P_3(a, c)$ . Here  $a, b, c$  are constants,  $f$  is a function symbol,  $x_1, x_2, x_3$  are variables. Applying S-CS rule to the 4 clauses  $C_1, C_2, C_3, C_4$ , we obtain an S-CS clause involving 4 clauses:  $C_5 = C_4^s(C_1, C_2, C_3, C_4) = P_1(b) \vee P_2(b, f(a))$ , the S-SC is:  $(P_1(a)) \wedge (P_2(a, b)) \wedge (\sim P_1(a) \vee \sim P_2(a, b) \vee P_3(a, c)) \wedge (\sim P_1(a) \vee \sim P_2(a, b) \vee \sim P_3(a, c))$ .

The state of the art first-order logic ATP systems usually use saturation algorithm [26-27] as the deduction framework, such as Vampire [28-30], Eprover [19, 31], iProver [32]. The saturation algorithm runs in an iterative manner. In each iteration, it selects a current clause in the *passive* according to heuristic strategy, and uses the current clause and the clause in *active* to apply inference rules by the way of saturation, and the inference rules adopt binary resolution methods. S-CS rule is a multi-clause deduction method, so it is quite different from binary inference method under the saturation algorithm.

It can be obtained from the analysis of Example 2.1 that the deduction is very complicated by binary resolution method under saturation algorithm, and the new generated clauses usually have many literals when any two clauses are involved. The S-CS dynamic deduction can generate S-CS clauses with fewer literals by separating a contradiction, which derived from the input clauses under the control of reasonable strategy. In Example 2.1, four input clauses with a total of 12 literals generate an S-CS clause just contains 2 literals by S-CS rule.

In addition, since the current first-order logic ATP systems are generally impossible to cover all proof search in a fixed runtime. Under the framework of saturation algorithm, the current clause is deduced with the clauses in *active* by binary resolution method, and the generated clauses will not be used in current iteration. Therefore, it is difficult to the proof search by binary resolution method under saturation algorithm with a given strategy, completely cover the proof search by S-CS dynamic deduction. On the other hand, S-CS rule is a multi-clause dynamic deduction process and has good deduction characteristics, which can make full use of the synergized ability between clauses in the deduction process and the S-CS clauses usually contain fewer literals, so it is a good complement to binary inference. Compared to binary inference based on saturation, the S-CS dynamic deduction has the following advantages in individual iteration deduction:

(1) Ahead deduction. The input clause used in S-CS dynamic deduction can be the newly generated clause, and thus can make up the loss problem of proof search by

binary inference based on saturation.

(2) Synergized deduction among multiple (two or more) clauses. Different from hyper-resolution, unit-resulting resolution, the S-CS dynamic deduction can make synergized deduction among multiple clauses, so it has stronger capability of literal elimination. As showed in Example 2.1, three literals are eliminated for clause C3 and C4 respectively by S-CS rule. The bigger the standard contradiction separated, the stronger the ability of literal elimination.

(3) In process of proof search, the S-CS clause generated by S-CS dynamic deduction is usually difficult to be obtained by binary inference under saturation algorithm with a given strategy. This is because the S-CS clauses usually contain fewer literals, where the separated standard contradiction by S-CS rule is not simply composed by multiple (two or more) complementary pairs of literals, it is different from that of binary resolution or the chains of binary resolution method.

(4) The S-CS clauses usually contain fewer literals. Theory and practice show that under a reasonable control of heuristic strategy, the S-CS dynamic deduction is guidable and controllable, and generates S-CS clause by separating a standard contradiction (a literal set). The S-CS clause usually has fewer literals or unit clauses. Because finding refutation for a problem of first-order logic needs to infer an empty clause, it means that it is getting closer to find refutation if the generated clauses have smaller number of literals, and thus it is able to improve the deduction efficiency.

(5) Unit clauses can flexibly participate in S-CS dynamic deduction. According to the definition of S-CS rule, a separated standard contradiction is still a standard contradiction after adding some unit clauses. At the same time, when a unit clause participates in the deduction, the literal in the S-CS clause which is unified complementary to the literal in the unit clause can be removed and added to the current separated standard contradiction to form a new standard contradiction.

### 3 Heuristic Strategies in the CSE\_E

Based on the characteristics of S-CS rule, CSE\_E uses different heuristic strategies to generate promising lemmas. The following criteria are considered: 1) make full use of the synergized effect among multiple clauses, and make up the loss of proof search of binary deduction under the saturation algorithm; 2) through the control of S-CS deduction and backtracking mechanism which is used to return to the last deduction step for generating satisfying S-CS clauses (evaluated by their attributes), the S-CS clauses with fewer literals are preferentially generated in each deduction step.

CSE\_E implements different heuristic strategies for the S-CS dynamic deduction as detailed below, where the unit clause selection strategy is specially designed in CSE\_E since unit clauses can flexibly participate in the S-CS dynamic deduction. These strategies are implemented mainly based on the attributes of related clause set, clauses and literals. Clause set attributes mainly include:

(1) *The maximum number of literals in an original clause.* The maximum number of literals is recorded, in order to guide setting thresholds of the number of literals in S-CS clause.

(2) *The maximum number of term depth in an original clause.* The maximum number of term depth is used to guide setting the term depth threshold of S-CS clause,

and evaluating substitutions in the process of S-CS dynamic deduction.

(3) *The maximum symbol count in an original literal.* The maximum symbol count is recorded and used to guide evaluating substitutions in the process of S-CS dynamic deduction and the lemmas filtration.

### 3.1 Unit Clause Selection Strategy

This strategy is implemented mainly based on the following unit clause attributes: 1) *deduction weight*: deduction weight of a unit clause is the number of times that the clause has participated in the S-CS dynamic deduction; 2) *symbol count*: it is used to characterize the statistics of symbols in a unit clause. It is based on the counting of each constant, function symbol, and variable; 3) *maximum term depth*: it describes the maximum term depth level in a unit clause; 4) *number of variables*: when the unit clause participates in the S-CS dynamic deduction, different substitutions will be tried on the variables for the deduction; 5) *total number of complementary predicates*: it is used to describe the potential S-CS ability of the predicate in a unit clause. Accordingly, the unit clause selection strategies include: original unit clause priority, unit S-CS clause priority, smaller deduction weight priority, smaller symbol count priority, smaller maximum term depth priority, larger number of variables priority, and predicate with larger total number of complementary predicates priority.

### 3.2 Non-unit Clause Selection Strategy

This strategy is implemented mainly based on the following non-unit clause attributes: 1) *acceptable deduction weight*: acceptable deduction weight of a non-unit clause is the number of times that the clause has participated in an acceptable S-CS dynamic deduction. The acceptable deduction means the deduction generates an S-CS clause that meets the set requirements (e.g., number of literals, maximum term depth); 2) *unacceptable deduction weight*: unacceptable deduction weight of a non-unit clause is the number of times that the clause has participated in an unacceptable S-CS dynamic deduction. An unacceptable deduction means the deduction generates an S-CS clause that does not meet the set requirements and will lead to backtracking; 3) *stability*: stability characterizes the structure features of the constants in a clause, and it is used to describe the activity level of the clause participating in the S-CS dynamic deduction; 4) *number of literals*: number of literals in a clause can reflect the number of literals in the S-CS clause to certain extent; 5) *deduction distance*: deduction distance is a measure of a clause and describes the possibility of the clause to be applicable for inferences with a negated conjecture clause; 6) *Flexibility*: Flexibility is a variable measure of a clause and reflects the distribution of variables. Accordingly, the non-unit clause selection strategies mainly include: smaller acceptable deduction weight priority, smaller unacceptable deduction weight priority, smaller stability priority, smaller number of literals priority, smaller deduction distance priority, and greater flexibility priority.

### 3.3 Literal Selection Strategy

This strategy is implemented mainly based on the following literal attributes: 1) *acceptable deduction weight*: acceptable deduction weight of a literal is the number of

times the literal has been used for an acceptable deduction; 2) *unacceptable deduction weight*: unacceptable deduction weight of a literal is the number of times the literal has been used for an unacceptable deduction; 3) *variable attributes*: the independent variable and shared variable are marked correspondingly, where an independent variable exists in one literal only, a shared variable appears in more than one literal in a clause; 4) *maximum term depth*: CSE\_E implements dynamic literal selection according to this attribute, so that the literals with term depth from small to large are controlled to participate in the process of deduction gradually; 5) *symbol count*: it is used to characterize the statistics of symbols for a literal based on the counting of each constant, function symbol, and variable in the literal. Accordingly, the literal selection strategies mainly include smaller acceptable deduction weight priority, smaller unacceptable deduction weight priority, literal only with independent variable priority, literal with shared variable priority, smaller maximum term depth priority, and smaller symbol count priority.

### 3.4 S-CS Clause control strategy

S-CS rule is a multi-clause and synergized deduction method that generates an S-CS clause dynamically by separating a standard contradiction, which is a subset of the literals from the input clauses, and infers the disjunction of the non-selected literals of the input clauses as a S-CS clause. The choice of input clauses and which input clauses are selected to construct standard contradictions, as well as the generated S-CS clause are controllable. S-CS clause control strategy is used to limit the attributes of S-CS clause, it first set the threshold of some attributes of S-CS clause, if the attribute of an S-CS clause exceeds the set threshold, then this S-CS clause will be discarded. We use this method to evaluate whether an S-CS clause has a role in finding refutation. The attributes of an S-CS clause mainly include:

(1) *maximum number of literals*: number of literals is an important indicator for evaluating clause, especially for the new generated clauses by deduction. Control the maximum number of literals in an S-CS clause, and improve the efficiency of the S-CS dynamic deduction. For example, when this threshold value is set to 1, the generated S-CS clauses by S-CS dynamic deduction will be all unit clauses. CSE\_E supports two methods to set the threshold of the maximum number of literals in an S-CS clause, one is user-defined by strategy, and the other is set automatically according to the maximum number of literals in different original clause set when facing different problems.

(2) *maximum term depth*: when a variable in function symbol is unified in the process of S-CS dynamic deduction, it may increase the depth of the function symbol, which will increase the complexity of subsequent deduction. Control maximum term depth of the S-CS clause, and improve the S-CS deduction efficiency. CSE also supports two methods to set the threshold of the maximum term depth in the S-CS clause, one is user-defined by strategy, and the other is automatically set according to the maximum term depth in different original clause set when facing different problems.

In summary, the selection of which preferences are used is specified as parameters, and some parameters (e.g., maximum number of literals of S-CS clause) are set by analyzing the attributes of the original clause set. The implementation supports the single-strategy mode and the multi-strategy mode, When solving a problem, the single-

strategy mode uses only one strategy, and the multi-strategy mode uses multiple heuristics in a time slice manner: Using multiple strategies one by one in equal time slices for the total set time, the previously generated S-CS clauses used the previous strategies will be allowed to use for next dynamic deduction with the next strategy, which effectively improves the capability of CSE\_E.

#### 4 Multi-clause S-CS dynamic deduction Algorithm in the CSE\_E

Based on S-CS rule and the corresponding heuristic strategies, CSE\_E implements a multi-clause S-CS dynamic deduction algorithm that can generate a large number of S-CS clauses with few literals, and most of them are unit clauses. With the heuristic strategies, the algorithm first selects unit clauses based on the unit clause selection strategy and makes them participate in the standard contradictions construction. If there is no unit clause in the clause set, the non-unit clause will be chosen directly, which is based on the non-unit clause selection strategy and applied S-CS rule along with the constructed standard contradictions by the participated input clauses. The generated S-CS clause in this way is used as a lemma. If the generated S-CS clause does not meet the threshold requirement, another non-unit clause will be selected through the backtracking mechanism, which can thereby optimize the proof search.

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

*Step 1:* According to the threshold, denoted as  $K$ , of the number of selected unit clauses, a unit clause set is selected by the strategy and added to  $S_u$ . The selected unit clause is identified as  $C_i$  according to the order of sequence number  $i$  ( $i=1, 2, \dots, K$ ).

*Step 2:* Select a clause by the non-unit clause selection strategy, and denote it as  $C_j$  ( $j=K+1, \dots$ ).

*Step 3:* Apply S-CS rule to the selected clause  $C_j$ , construct standard contradiction with the input clauses, and get  $C_k^{\sigma_k^+}$  and  $C_k^{\sigma_k^-}$  ( $k$  is the serial number of the selected clause). Then the S-CS clause is generated. If S-CS clause is an empty clause, go to Step 5. Otherwise, the S-CS clause will be checked based on the control strategies as follows: 1) if the S-CS clause is a tautology or a redundant clause, the S-CS deduction is unacceptable; 2) the number of literals in the S-CS clause exceeds the threshold; 3) the term depth exceeds the threshold. When one of the above three conditions is satisfied, the backtracking algorithm is performed, and go to Step 2.

Backtracking algorithm is used to return to the last deduction step, and needs do some cleanup tasks. It mainly includes: 1) clear the substitutions which generated in current deduction step; 2) remove literals in  $C_k^{\sigma_k^+}$  and  $C_k^{\sigma_k^-}$  which added in current deduction step; 3) set the proof search sequence number in the last deduction step.

*Step 4:* If the S-CS clause does not satisfy the three conditions in Step 3, the acceptable deduction weight of the clause  $C_j$  increases by 1, the acceptable deduction weights of literals in the clause  $C_j$  which were used for standard contradiction construction increases by 1. The generated S-CS clause is represented as a lemma and added in the S-CS clause set, with the deduction distance being calculated and the deduction path being recorded. If the exit conditions are satisfied, then go to Step 5. Otherwise, in order to make full use of the clause  $C_j$  to achieve a synergized deduction effect, do the following iterative processes: 1) if the clause  $C_j$  contains variable

substitution, then record this deduction path to avoid the repetition, clear variable substitutions of  $C_j$  and reuse  $C_j$ , and go to Step 3. When the number of times of the clause  $C_j$  reused exceeds the set threshold, the sequence number  $j$  is increased by 1, and go to Step 2; 2) when the clause  $C_j$  does not contain variable substitution, the sequence number  $j$  is increased by 1, and go to Step 2.

*Step 5:* Exit this multi-clause dynamic deduction. The exit conditions include: 1) the S-CS clause is an empty clause, so we can conclude that the original clause set is unsatisfiable; 2) the number of input clauses reaches the set threshold; 3) the runtime reaches the set threshold; 4) the remaining memory reaches the set threshold.

*Step 6:* The S-CS clause processing: internal simplification is performed on the generated S-CS clause set, apply forward simplification on the S-CS clause set using the clauses in the clause set, and apply the backward simplification on the clause set using the clauses in the S-CS clause set, then each S-CS clause is added in clause set.

<p><b>Input:</b> a given clause set (<math>S</math>)</p> <p><b>Output:</b> a generated S-CS clause set (<math>R</math>)</p> <pre> 1: <math>G = select\_unitClauses(S)</math> 2: <math>D = G</math> 3: <math>p = select\_nonUnitClause(S)</math> 4: While <math>p \neq null</math> (<math>P</math> is a valid clause) begin 5:   <math>q = extendedSplitContradiction(p, D)</math> 6:   if <math>q == \emptyset</math> 7:     return "Unsat" 8:   else if <math>checkUnacceptable(q, S)</math> 9:     <math>backtrackingSCS(p)</math>; 10:    <math>p = change\_nonUnitClause(S)</math> 11:    goto 4 12:   else if <math>checkExitConditions()</math> 13:     <math>R = q \cup R</math> 14:     goto 23 15:   else 16:     <math>R = q \cup R</math> 17:     <math>D = p \cup D</math> 18:     if <math>checkSynergized(p)</math> 19:       goto 4 20:     else 21:       goto 5 22: end 23: <math>inner\_Simplification(R)</math> 24: foreach <math>c \in S</math> 25:   <math>forward\_Simplification(c, S)</math> 26: foreach <math>c \in R</math> 27:   <math>backward\_Simplification(S, c)</math> 28: <math>S = R \cup S</math> </pre>
--

Fig. 1. The pseudo-code for the S-CS dynamic deduction algorithm

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

*select\_unitClauses(S)*: select a unit clause set in clause set  $S$  according to unit clause selection strategy, and the number of the selected unit clauses is user-defined.

*select\_nonUnitClause(S)*: select a non-unit clause in clause set  $S$  according to non-unit clause selection strategy.

*extendedSplitContradiction(p, D)*: apply S-CS rule with the selected non-unit clause  $p$  and the input clause set  $D$ , construct a standard contradiction and get a generated S-CS clause.

*checkUnacceptable(q, S)*: check whether the clause  $q$  in clause set  $S$  is an acceptable clause, the conditions of an acceptable clause is described in Step 3 above.

*backtrackingSCS(p)*: In current deduction step, do cleanup tasks and return to the last deduction step according to the current input clause  $P$ .

*change\_nonUnitClause(S)*: reselected a new non-unit clause instead of the current selected clause in clause set  $S$ .

*checkExitConditions()*: check exit conditions of S-CS dynamic deduction algorithm.

*checkSynergized(p)*: check whether the input clause  $P$  contains substitution, or if it exists, check whether the number of times the clause reused exceeds the set threshold.

*inner\_Simplification(R)*: apply forward simplification, factoring rule and equality resolution in S-CS clause set  $R$ .

*forward\_Simplification(c, S)*: check whether clause  $c$  is a redundant clause using the clauses in the clause set  $S$ . If yes, delete the clause  $c$ .

*back\_Simplification(S, c)*: check whether the clauses in clause set  $S$  are redundant clauses using the clause  $c$ . If yes, delete the redundant clauses.

## 5 Lemma Filtration in the CSE\_E

CSE\_E generates a large number of lemmas, usually unit clauses, through the S-CS dynamic deduction. Those lemmas along with the original clauses are fed to E to continue the proof search, this is the essential way to combine CSE and E in CSE\_E 1.0. Lemma filtration is applied in CSE\_E 1.0 in order to improve the applicability of the lemmas to the negated conjecture clause (measured by the ratio of the applicable lemmas to the total number of lemmas) and the capability of CSE\_E 1.0. Lemma Filtration is mainly achieved by the following methods: 1) set the threshold of the number of literals in the lemma, and feed to E only the lemmas whose numbers of literals are less than the set threshold; 2) set the threshold of maximum term depth in the lemma, and feed to E only the lemmas whose maximum term depth are less than the set threshold. By default, CSE\_E 1.0 takes the maximum term depth of the original clause set as a reference and sets the threshold; 3) set the threshold of deduction distance for the lemmas, and feed to E only the lemmas whose deduction distances are less than the threshold.

The use of lemmas is related to the running mechanism of CSE\_E. CSE\_E divides the total running time and schedules the running of the two provers by time. It runs plain E first, if E does not find a proof it runs plain CSE, then if it does not find a proof, some clauses inferred in the CSE run as lemmas are added to the original clause set, and the combined clause set is handed back to E for further proof search.

## 6 Experimental and Performance Analysis

## 6.1 Experiment Setup

CSE\_E 1.0 combines CSE 1.1 with E 2.1. CSE 1.1 is implemented mainly in C++, and JAVA is used for scheduling running implementation of batch problems. The job dispatch between CSE and E is implemented in JAVA. The working system of CSE\_E 1.0 is available at <http://www.tptp.org/CASC/J9/Entrants.html>. CSE\_E 1.0 is tested on both the CADE ATP competition CASC-26 (2017) and the CASC-J9 (2018) FOF division problems in comparison with E 2.1 and E 2.2: 1) CSE\_E 1.0 runs for CASC-26 (2017) problems were made on a computer with a 3.4GHz Intel(R) Core (TM) i7-6700 processor and 8GB memory, and OS Ubuntu15.04 64-bit, with the default timeout of 300 seconds. The test results of E 2.1 are obtained in the same hardware environment and timeout set. 2) CSE\_E 1.0 participated the CASC-J9 (2018) FOF division competition and won the second place. The test experimental data of CASC-J9 FOF division problem is derived from CASC-J9 competition results directly.

## 6.2 Experimental Results and Analysis

### 6.2.1 Performance of CSE\_E 1.0 on CASC-26 FOF division problems

The CSE\_E 1.0 incorporates S-CS dynamic deduction with E 2.1, so it can generate a large number of S-CS clauses with few literals (most of them are unit clauses) and search for different deduction paths compared to E 2.1. Fig. 2 shows the comparison of solved problems by CSE\_E 1.0 and E 2.1 on CASC-26 FOF division problems. CSE\_E 1.0 has solved 398 problems with 8 more than E 2.1. The average time spent for the 390 problems solved by E 2.1 is 18.79 seconds, and 21.72 seconds spent for 398 theorems by CSE\_E 1.0. The average time spent for 390 problems by CSE\_E 1.0 is 16.54 seconds, 2.25 seconds less than that of E 2.1.

From the 90 second point onwards, the performance of CSE\_E 1.0 outperforms E 2.1, where CSE\_E 1.0 has solved 370 problems with 12 more than E 2.1. The average time of E 2.1 is 5.9 seconds, 8.5 seconds for CSE\_E 1.0. Within 200 seconds, CSE\_E 1.0 has solved 386 problems with 3 more than E 2.1, and the average time of E 2.1 is 14.6 seconds, 14.2 seconds for CSE\_E 1.0. Experimental results shown that CSE\_E 1.0 outperformed E 2.1 in terms of the capability and time efficiency.

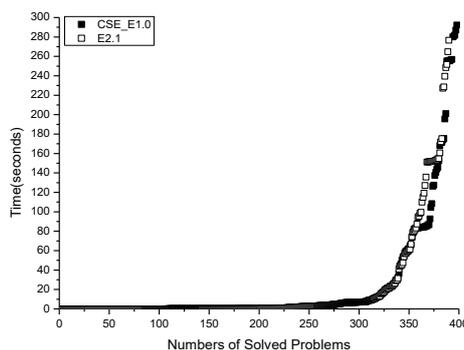


Fig. 2. Comparison on solved problems by CSE\_E 1.0 and E 2.1

In the total 110 unsolved problems by E 2.1, CSE\_E 1.0 can solve 14 out of 110

(12.7% of the total), where 6 problems were solved by CSE alone and 8 problems were solved by E with CSE lemmas added. Table 1 is the list of those 14 problems sorted by their used time.

Those 14 problems have the average rating 0.76. The clauses in these problems usually contain many literals, while CSE can improve the deduction efficiency due to the fact that it can better exploit the synergized effect of the input clauses, achieve better "literal elimination" effect by separating standard contradictions, and generate the S-CS clause with few literals. The average solving time by CSE\_E 1.0 is 230 seconds, so the time efficiency is acceptable.

Table 1. List of the 14 problems solved by CSE\_E 1.0, but unsolved by E 2.1

Problem	Rating	Time(seconds)	Problem	Rating	Time(seconds)
GEO502+1	0.59	168	SWB025+1	0.72	171.2
SWB093+1	0.79	173.9	SWB027+1	0.79	175.4
BOO109+1	0.57	195.5	GEO511+1	0.9	201
GEO495+1	0.79	255.1	GEO531+1	0.66	255.3
GEO532+1	0.76	255.4	GEO507+1	0.66	256.5
SWB081+1	0.79	256.777	GEO506+1	0.83	279.931
AGT011+2	0.86	280.739	SWB010+1	0.9	292.181

### 6.2.2 Performance of CSE\_E 1.0 on CASC-J9 FOF division problems

We conducted the comparative analysis on CSE\_E 1.0 and E 2.2 using the CASC-J9 competition results (see Fig. 3). CSE\_E 1.0 solved 362 out of 500 problems with 12 problem more than E 2.2. The average time spent for the 350 problems solved by E 2.2 is 25.6 seconds, 26.5 seconds spent for 362 problems by CSE\_E 1.0. The average time spent for 350 problems by CSE\_E 1.0 is 18.4 seconds, which is 7.2 seconds less than that of E 2.2. Fig. 3 shows that from the 80 seconds point onwards, the performance of CSE\_E 1.0 outperforms E 2.2, where CSE\_E 1.0 solved 315 problems with 5 more than E 2.2. The average time of E 2.2 is 7.9 seconds, and 8.6 seconds for CSE\_E 1.0. Within 160 seconds, CSE\_E 1.0 solved 346 problems with 7 problems more than that of E 2.2, and the average time of CSE\_E 1.0 is 16.2 seconds, with 3.2 seconds less than that of E 2.2. Experiments show that CSE\_E 1.0 outperforms E 2.2 in terms of the capability and time efficiency.

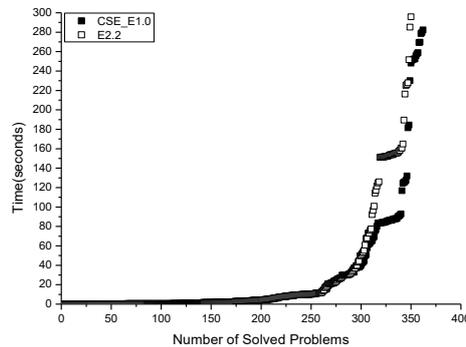


Fig. 3. Comparison on solved problems by CSE\_E 1.0 and E 2.2

Table 2 shows that CSE\_E 1.0 can solve 14 (9.3%) of the total 150 unsolved problems by E 2.2, where 3 problems were solved by CSE alone and 11 problems were solved by E with CSE lemmas added. Those solved problems have the average rating being 0.74. The average solving time by CSE\_E 1.0 is 249 seconds, so the time efficiency is acceptable.

Table 2. List of the 14 solved problems by CSE\_E 1.0 but unsolved by E 2.2

Problems	Rating	Time(seconds)	Problems	Rating	Time(seconds)
AGT018+1	0.62	181.56	SWB027+1	0.79	184.4
GEO511+1	0.9	230.05	BOO109+1	0.57	248.16
SWB016+1	0.69	251.05	SWB094+1	0.79	251.95
SWB082+1	0.79	252.25	SWW189+1	0.9	252.46
SWB098+1	0.79	257.68	SWV038+1	0.28	269.39
SCT170+3	0.9	269.74	GEO089+1	0.62	278.67
SCT139+1	0.83	279.7	GEO506+1	0.83	282.36

Otherwise, CSE\_E 1.0 has solved one problem GEO506+1 that has not been solved by any other competition provers in CASC-J9 (see Table 3). This problem has a large number of literals and much more variables. CSE\_E 1.0 can make full use of clauses containing variables, so can perform the literal elimination operation by means of separating standard contradictions.

Table 3. One solved problem by CSE\_E 1.0, but unsolved by other competition provers

Problems	Rating	Number of formulae	Maximal formula depth	Number of variables	Maximal term depth	Number of atoms
GEO506+1	0.83	143	22	564	3	595

The above comparison analysis shows that CSE\_E 1.0 stands on the shoulders of the plain E, and is able to effectively improve E in terms of capability and time efficiency, and the S-CS dynamic deduction can be effectively applied to first-order logic ATP systems.

## 7 Conclusions and Future Work

CSE\_E 1.0 combined an S-CS based dynamic deduction method with E 2.1. It is a first-order logic ATP system that optimizes E from the inference mechanism aspect for the first time. Experimental results have shown that CSE\_E 1.0 outperforms E 2.1 and E 2.2 in both capability and time efficiency to some extent, which illustrates that S-CS rule can improve the performance of first-order logic ATP systems.

For combining the S-CS dynamic deduction method with E effectively, the high-quality S-CS clauses and the effective lemma selection are very important. How to effectively filter the lemmas according to different problems is also a future research work. In addition, optimizing the S-CS dynamic deduction algorithm is the core and ongoing work. CSE\_E 1.0 is a preliminary attempt to combine the S-CS based dynamic deduction with the state-of-the-art prover (e.g., E). There is still a lot of work that needs to be done in order to improve the efficiency of CSE\_E. We hope that the ongoing development of CSE\_E can solve more and more hard problems or real-world problems

with improved performance.

**Acknowledgments.** This paper is supported by the National Natural Science Foundation of China (Grant No.61673320), the Fundamental Research Funds for the Central Universities (Grant No. 2682017ZT12, 2682018CX59, 2682018ZT25).

## References

- [1] Pavlov V, Schukin A, Cherkasova T.: Exploring Automated Reasoning in First-Order Logic: Tools, Techniques and Application Areas. *Communications in Computer & Information Science* 394(1), 102-116 (2013)
- [2] Kovacs L, Voronkov A.: Finding Loop Invariants for Programs over Arrays Using a Theorem Prover. In: M. Chechik and M. Wirsing. (eds.) *FASE 2009, LNCS*, vol. 5503, pp. 470–485. Springer.
- [3] Sutcliffe G.: TSTP Solution Domains. <http://www.tptp.org/cgi-bin/SeeTPTP?Category=Solutions>. last accessed 2019/5/16
- [4] Sutcliffe G.: CASC Solution Domains. <http://tptp.org/CASC/>. last accessed 2019/5/16
- [5] Robinson J.A.: A machine-oriented logic based on the resolution principle. *Journal of the ACM* 12(1), 23-41 (1965)
- [6] Bachmair L, Ganzinger H.: Rewrite-based Equational Theorem Proving with Selection and Simplification. *Journal of Logic & Computation* 4(3), 217-247 (1991)
- [7] Loveland D W.: A linear format for resolution. In: *Proceedings of the IRIA Symposium on Automatic Demonstration*. pp. 147-162 (1970)
- [8] Chang C L.: The unit proof and the input proof in theorem proving. *Journal of the Association for Computing Machinery* 17(4), 698-707 (1970)
- [9] Robinson J A.: Automatic deduction with hyper-resolution. *International Journal of Computing & Mathematics* 1(3), 227-234 (1965)
- [10] Overbeek R, Mccharen J, Wos L.: Complexity and related enhancements for automated theorem-proving programs. *Computers & Mathematics with Applications* 2(1), 1-16 (1976)
- [11] Slaney J, Paleo B W.: Conflict resolution: a first-order resolution calculus with decision literals and conflict-driven clause learning. *Journal of Automated Reasoning* 12(4), 1-24 (2016)
- [12] Reger G, Tishkovsky D, Voronkov A.: Cooperating Proof Attempts. In: Felty, A.P. and Middeldorp A. (eds.) *CADE 2015, LNCS*, vol. 9195, pp. 339–355. Springer.
- [13] Schulz S, Möhrmann M.: Performance of Clause Selection Heuristics for Saturation-Based Theorem Proving. In: Olivetti N. and Tiwari A. (Eds.) *IJCAR 2016, LNCS*, vol. 9706, pp. 330–345. Springer.
- [14] Meng Jia, Paulson L C.: Lightweight relevance filtering for machine-generated resolution problems. *Journal of Applied Logic* 7(1), 41-57 (2009)
- [15] Voronkov A.: Sine Qua non for large theory reasoning. In: Bjørner N. and Sofronie-Stokkermans V. (Eds.): *CADE 2011, LNCS*, vol. 6803, pp. 299–314. Springer.
- [16] Schulz S.: Simple and Efficient Clause Subsumption with Feature Vector Indexing. In: Bonacina, M.P., Stickel, M.E. (eds.) *Automated Reasoning and Mathematics: Essays in Memory of William W. McCune, LNCS*, vol. 7788, pp. 45–67. Springer.
- [17] Xu Y, Liu J, Chen S W, Zhong X M, He X X.: Contradiction separation based dynamic multi-clause synergized automated deduction. *Information Sciences* 462(1), 93-113 (2018)

- [18] Xu Y, Chen S W, Liu J, et al.: Distinctive features of the contradiction separation based dynamic automated deduction. In: Jun Liu, Jie Lu, Yang Xu, et al, (Eds.) FLINS2018, CEIS, vol. 11, pp. 725-732. World Scientific.
- [19] Schulz S.: System description: E 1.8. In: McMillan K., Middeldorp A., and Voronkov A. (Eds.) LPAR-19, LNCS, vol. 8312, pp. 735–743. Springer.
- [20] Sutcliffe G.: The 7th IJCAR Automated Theorem Proving System Competition-CASC-J7. *AI Communications* 28(4), 683–692 (2015)
- [21] Sutcliffe G.: The 8th IJCAR Automated Theorem Proving System Competition-CASC-J8. *AI Communications* 29(5), 607–619 (2016)
- [22] Sutcliffe G. The 9th IJCAR Automated Theorem Proving System Competition-CASC-J9. *AI Communications* 31(6), 495–507 (2018)
- [23] Kaliszyk C, Schulz S, Urban J, et al.: System Description: E.T. 0.1. In: Felty, A.P. and Middeldorp A. (eds.) CADE 2015, LNCS, vol. 9195, pp. 389–398. Springer.
- [24] Kühlwein, D., Schulz, S., Urban, J.: E-MaLeS 1.1. In: Bonacina M.P. (Ed.) CADE 2013, LNCS, vol. 7898, pp. 407–413. Springer.
- [25] Daniel K, Urban J.: MaLeS: A Framework for Automatic Tuning of Automated Theorem Provers. *Journal of Automated Reasoning* 55(2), 91-116 (2013)
- [26] Denzinger J, Kronenburg M, SCHULZ S.: DISCOUNT - A distributed and learning equational prover. *Journal of Automated Reasoning* 18(2), 189-198 (1997)
- [27] Mccune W, Wos L.: Otter - the CADE-13 competition incarnations. *Journal of Automated Reasoning* 18(2), 211-220 (1997)
- [28] Riazanov A, Voronkov A.: The design and implementation of vampire. *AI Communications* 15(2), 91-110 (2002)
- [29] Kovács L, Voronkov A.: First-order theorem proving and vampire. In: N. Sharygina and H. Veith (Ed.) CAV 2013, LNCS, vol. 8044, pp. 1–35. Springer.
- [30] Biere A, Dragan I, Kovács L, et al.: Experimenting with SAT solvers in Vampire. In: Gelbukh A, Espinoza FC, GaliciaHaro SN (Ed.) MICAI 2014, LNCS, vol. 8856, pp. 431–442. Springer.
- [31] Schulz S.: E - a brainiac theorem prover. *AI Communications* 15(2), 111-126 (2002)
- [32] Korovin K.: iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In: A. Armando, P. Baumgartner, and G. Dowek (Ed.) IJCAR 2008, LNCS, vol. 5195, pp. 292–298. Springer.