symmetry

MDPI

Article

CSE_E 1.0: An Integrated Automated Theorem Prover for First-Order Logic

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

1 School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China
2 National-Local Joint Engineering Lab of System Credibility Automatic Verification, Chengdu 610031, China
3 School of Mathematics, Southwest Jiaotong University, Chengdu 610031, China
4 School of Computing, Ulster University, Belfast BT37 0QB, Northern Ireland, UK
* Correspondence: caofeng@my.swjtu.edu.cn

Received: 12 August 2019; Accepted: 6 September 2019; Published: 8 September 2019

Abstract: First-order logic is an important part of mathematical logic, and automated theorem proving is an interdisciplinary field of mathematics and computer science. The paper presents an automated theorem prover for first-order logic, called *CSE_E* 1.0, which is a combination of two provers contradiction separation extension (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*, and (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 to 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 are unsolved by E, *CSE_E* 1.0 schedules the running of the two provers in time. It runs plain E first, and 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 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 (FOFdivision). 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, whose research involves mathematics and computer science, and has been successfully applied in many application areas [1–3]. However, there are still many unsolved problems in the recently-released version (TPTP-v7.2.0) of the TPTPbenchmark library [4,5]. In addition, based on the analysis of the results of CADEATP system competitions in the last few years [6], there is still much room for improvement of 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., [7–13]) or heuristic strategies [14–20] (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 [9], unit-resulting resolution [10] and hyper-resolution [11]. 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 to a 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 coming 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 [21] 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) synergised inference rule. The 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 the S-CS rule, because the synergised 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, synergised, guidable, and controllable deduction method [22]. 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 attributes of the generated clause to optimize proof search. When the constructed contradiction is limited to only two clauses, the S-CS rule degenerates to a general chains of binary resolution method; therefore, the binary resolution rule is a special case of the S-CS rule.

Eis a well-known equational automated theorem prover for first-order logic with powerful equality processing capability [23] based on superposition and rewriting. The prover has successfully participated in many ATP system competitions [24–26]. Over the past decade, many improvements and extensions of E have been developed from different perspectives, where some successful ones are E.T.[27] and E-Males [28,29]. 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 the S-CS rule can be applied effectively to E. The present work introduces a novel ATP system for first-order logic, which goes 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 the multi-clause dynamic deduction of the 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 the 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 at the FOFdivision and awarded the "best newcomer" prize at CASC-J9 [6]. Meanwhile, CSE outperformed some mature provers, such as Prover9 [6]. In this paper, CSE_E 1.0 is evaluated through benchmarks in terms of its applicability and capability.

The remainder of the paper is organized as follows. In Section 2, we briefly review the novel S-CS calculus and compare it with the binary resolution based on saturation algorithm, and the characteristics of S-CS dynamic deduction are analysed. 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 the S-CS rule and the lemmas' filtration method are proposed in

Sections 4 and 5 respectively. Section 6 shows the performance of CSE_E 1.0. The conclusion and follow-up research work are given in the last section.

2. Overview of the Multi-Clause S-CS Rule in CSE_E

Multi-clause S-CS-based dynamic deduction theory for first-order logic was introduced in [21], which gives the theoretical foundation for the CSE prover. This section provides a review of the basic concepts of the S-CS rule and then gives some theoretical analysis.

2.1. Definitions and Some Terminologies

Definition 1 ([21]). *Suppose a clause set $S = \{C_1, C_2, \ldots, C_m\}$ in first-order logic, where the following conditions hold:*

1. *There does not exist the same variables among $C_1, C_2, \ldots, C_m$ (if there exist the same variables, a rename substitution can be applied to make them different);*
2. *For each $C_i (i = 1, 2, \ldots, m)$, a substitution σ_i can be applied to C_i (σ_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:*
   - *$C_i^{\sigma_i} = C_i^{\sigma_i -} \vee C_i^{\sigma_i +}$, 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, and $C_i^{\sigma_i -}$ cannot be an empty clause; moreover,*
   - *For any $(x_1, \ldots, x_m) \in \prod_{i=1}^{m} C_i^{\sigma_i -}$, there exists at least one complementary pair among $\{x_1, \ldots, x_m\}$, and $\bigwedge_{i=1}^{m} C_i^{\sigma_i -}$ is called a separated standard contradiction (S-SC);*
   - *The resulting clause $\bigvee_{i=1}^{m} C_i^{\sigma_i +}$, denoted as $\mathbf{e}_m^{s\sigma}(C_1, C_2, \ldots, C_m)$ (here, "s" means "standard", $\sigma = \bigcup_{i=1}^{m} \sigma_i$), is called a standard contradiction separation clause (S-CS clause) of $C_1, C_2, \ldots, C_m$.*

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

Definition 2 ([21]). *Suppose a clause set $S = \{C_1, C_2, \ldots, C_m\}$ in first-order logic. $\Phi_1, \Phi_2, \ldots, \Phi_t$ is called a standard contradiction separation-based dynamic deduction sequence from S to a clause Φ_t, denoted as $\mathcal{D}^s$, if:*

1. *$\Phi_i \in S$, or,*
2. *there exist $r_1, r_2, \ldots, r_{k_i} < i, \Phi_i = \mathbf{e}_{k_i}^{s\theta_i}(\Phi_{r_1}, \Phi_{r_2}, \ldots, \Phi_{r_{k_i}})$, where $\theta_i = \bigcup_{j=1}^{k_i} \sigma_j$, σ_j is a substitution to $\Phi_{r_j}, j = 1, 2, \ldots, k_i, i = 1, 2, \ldots, t$.*

Example 1. *Let $S = \{C_1, C_2, C_3, C_4\}$ be a clause set in first-order logic, where $C_1 = P_1(a) \vee P_2(b, f(a))$,$C_2 = P_2(a, b) \vee P_2(b, f(a))$,$C_3 = \neg P_1(a) \vee P_1(b) \vee \neg P_2(x_1, x_2) \vee P_3(x_1, x_3)$,$C_4 = P_1(b) \vee \neg P_1(x_1) \vee \neg P_2(x_2, x_3) \vee \neg P_3(a, c)$. Here a, b, c are constants, f is a function symbol and x_1, x_2, x_3 are variables. Applying the S-CS rule to the four clauses C_1, C_2, C_3, C_4, we obtain an S-CS clause involving four clauses: $C_5 = C_4^s(C_1, C_2, C_3, C_4) = P_1(b) \vee P_2(b, f(a))$, then S-SC is: $P_1(a) \bigwedge P_2(a, b) \bigwedge(\neg P_1(a) \vee \neg P_2(a, b) \vee P_3(a, c)) \bigwedge(\neg P_1(a) \vee \neg P_2(a, b) \vee \neg P_3(a, c))$.*

Because the S-CS rule is a new inference mechanism, which is quite different from the conventional binary resolution methods, we would like to explain some related terminologies before introducing the related technologies in CSE_E.

1. Standard contradiction: This is a set of sub-clauses of selected literals from each input clause, and there exists at least one complementary pair of literals in it. Binary resolution only eliminates one complementary pair of literals, but the S-CS rule eliminates a standard contradiction.
2. S-CS clause: This is the disjunction of the non-selected literals of the input clauses (the selected literals of input clauses are used for building a standard contradiction) and is the generated clause by the S-CS rule.

3. Multi-clause deduction: The S-CS rule can handle more than two clauses in each deduction step. As Example 1 illustrates, the S-CS rule handles four input clauses C_1, C_2, C_3, C_4 in one deduction step, with three literals eliminated in C_3 and C_4, respectively.
4. Dynamic deduction: According to Definitions 1 and 2, when the S-CS rule handles multiple clauses, it contains many deduction steps, and the number of clauses involved in different deduction steps might be different. S-CS deduction provides a dynamic deduction process.
5. Synergised deduction: In the process of S-CS deduction, the input clauses play a synergised deduction role, i.e., they work together to form a standard contradiction and generate an S-CS clause. As Example 1 illustrates, the S-CS rule handles four input clauses C_1, C_2, C_3, C_4 in the deduction step, where $P_1(a)$ in C_1 helps to eliminate the literal $\neg P_1(a)$ in C_3 and the literal $\neg P_1(x_1)$ in C_4, and $P_2(a, b)$ in C_2 helps to eliminate the literal $\neg P_2(x_1, x_2)$ in C_3 and the literal $\neg P_2(x_2, x_3)$ in C_4.

2.2. Theoretical Analysis of the S-CS Rule

The state-of-the-art first-order logic ATP systems usually use the saturation algorithm [30,31] as the deduction framework, such as Vampire [32–34], Eprover[23,35] and iProver[36]. The saturation algorithm runs in an iterative manner. In each iteration, it selects a current clause in *passive* according to the 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. The S-CS rule is a multi-clause deduction method, so it is quite different from binary inference methods under the saturation algorithm.

It can be obtained from the analysis of Example 1 that the deduction is very complicated by binary resolution methods under the 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 standard contradiction, which is derived from the input clauses under the control of a reasonable strategy. In Example 1, four input clauses, with a total of 12 literals generating an S-CS clause, just contain two literals by the S-CS rule.

In addition, since it is generally impossible for the current first-order logic ATP system to cover all proof searches in a fixed runtime. Under the framework of the saturation algorithm, the current clause is deduced with the clauses in *active* by binary resolution methods, and the generated clauses will not be used in the current iteration. Therefore, the proof search of the binary resolution methods under the saturation algorithm with a given strategy is difficult to cover completely the proof search by S-CS dynamic deduction. On the other hand, the S-CS rule is a multi-clause dynamic deduction process and has good deduction characteristics, which can make full use of the synergised 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 each 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 of proof search by binary inference based on saturation to a certain extent.
2. Synergised deduction among multiple (two or more) clauses. Different from hyper-resolution and unit-resulting resolution, the S-CS dynamic deduction can make synergised deduction among multiple clauses, so it has a stronger capability of literal elimination. As shown in Example 1, three literals are eliminated in clauses C_3 and C_4 respectively by the S-CS rule. The bigger the standard contradiction is separated, the stronger the ability of literal elimination.
3. In the process of proof search, the S-CS clauses generated by S-CS dynamic deduction are usually difficult to obtain by binary inference under the saturation algorithm with a given strategy. This is because the S-CS clauses usually contain fewer literals, where the separated standard contradiction

in each deduction step 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 the heuristic strategy, the S-CS dynamic deduction is guidable and controllable and generates S-CS clauses by separating standard contradictions (the contradiction is a literal set). The S-CS clauses usually have few literals or are unit clauses. Because finding refutation for a problem in first-order logic needs to infer an empty clause, this means that it is getting closer to finding refutation if the generated clauses have a smaller number of literals, and thus, it can potentially improve the deduction efficiency.
5. Unit clauses can flexibly participate in S-CS dynamic deduction. According to the definition of the 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 is participating 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.

The satisfiability (SAT) problem in propositional logic is a finite problem [37]. The current mainstream solvers adopt DPLL[38] or CDCL[39] as the solution framework. By analysing the conflicts of variable assignment, the learnt clauses are generated to avoid the same path search in the subsequent search process. If all the variables are assigned without conflicts, the problem is solved. The state-of-the-art first-order logic ATP systems adopt the saturation framework [30,31], and the new clauses are generated by the input clauses with different inference rules. If an empty clause is generated, the problem is proven.

VSIDS[40] is an effective strategy in the SAT solver, which is used to prevent falling into the local path search. Its idea is to add a certain score to the variables during the solving process and use the score to guide the selection of decision variable. The deletion of learnt clauses in the SAT solver also adopts the same method [41,42] as above. The S-CS dynamic deduction algorithm in CSE_E 1.0 takes a similar idea as that in VSIDS, that is two attributes (acceptable deduction weight and unacceptable deduction weight) of the selected input clauses and the literals (used for standard contradiction construction) are dynamically adjusted in the S-CS deduction process and used to guide the selection of clauses and literals, thereby making more extensive use of different input clauses and avoiding the same proof search to a certain extent.

3. Heuristic Strategies in the CSE_E

For ATP systems, the design of heuristic strategies is usually related to the used inference rules. CSE_E designs the heuristic strategies based on the S-CS rule, and we would like to explain some related terminologies before introducing the heuristic strategies in CSE_E.

1. Acceptable deduction: The acceptable deduction means the S-CS deduction step generates an S-CS clause that meets the user-defined requirements (e.g., number of literals, maximum term depth).
2. Unacceptable deduction: The unacceptable deduction means the S-CS deduction step generates an S-CS clause that does not meet the user-defined requirements (e.g., number of literals, maximum term depth).
3. Deduction distance: This is a measure of the distance of the selected clause to a negated conjecture clause according to the unified complementary relationship between literals.
4. Backtracking mechanism: If the S-CS deduction step is an unacceptable deduction, the backtracking mechanism makes the deduction return to the last S-CS deduction step and then reselect a new input clause to participate in S-CS deduction.

Based on the characteristics of the S-CS rule, CSE_E uses different heuristic strategies to generate promising lemmas. The following criteria are considered: (1) make full use of the synergised effect among multiple clauses and make up the loss of the proof search of binary inference under the saturation algorithm; (2) through the control of S-CS deduction and the 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 the 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 and used to guide setting the threshold of the number of literals in S-CS clause.
2. The maximum term depth in an original clause: The maximum term depth is recorded and used to guide setting the threshold of the term depth of the S-CS clause and evaluating the 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 the substitutions in the process of S-CS dynamic deduction and the selection of the unit clause.

3.1. Unit Clause Selection Strategy

This strategy is implemented mainly based on the following unit clause attributes: (1) deduction weight: the 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: this 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: this describes the maximum term depth 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: this 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, a larger number of variables priority and predicate with a 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: the 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; (2) unacceptable deduction weight: the 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; (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: the number of literals in an input clause can reflect the number of literals in the S-CS clause to a certain extent; (5)deduction distance: the deduction distance is a measure of a clause and describes the possibility of the clause to be applicable for inference rules 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, a 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: the acceptable deduction weight of a literal is the number of times the literal has been used for an acceptable deduction; (2) unacceptable deduction weight: the 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, and 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: this 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

The S-CS rule is a multi-clause and synergised deduction method that generates an S-CS clause by separating a standard contradiction, which is a subset of the literals form the input clauses, and infers the disjunction of the non-selected literals of the input clauses as an S-CS clause. The selection of input clauses and literals from the input clauses constructing standard contradictions, as well as the generated S-CS clause are controllable. The S-CS clause control strategy is used to organize the attributes of S-CS clause: it first sets the thresholds of the attributes of the S-CS clause. If one of the attributes 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: The number of literals is an important indicator for clause evaluating, especially for the newly 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 one, 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 parameter, and the other is automatically set according to the maximum number of literals in different original clause sets when facing different problems.
2. Maximum term depth: when a variable in a 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 efficiency of the S-CS dynamic deduction. CSE also supports two methods to set the threshold of the maximum term depth in the S-CS clause: one is user-defined by the strategy parameter, and the other is automatically set according to the maximum term depth in different original clause sets when facing different problems.

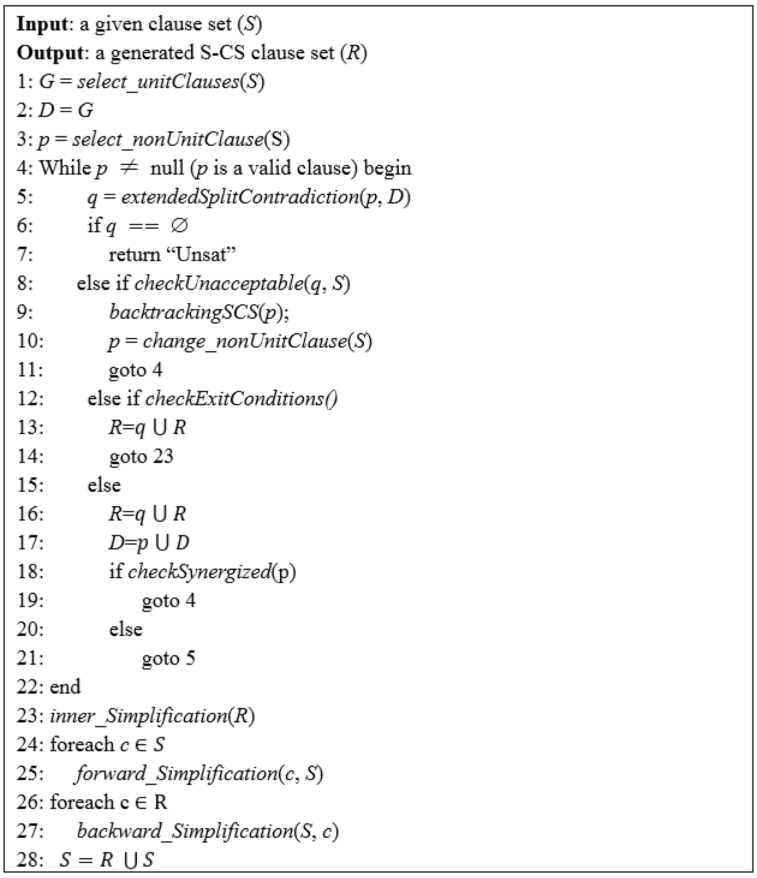
In summary, the selection of which preferences are used is specified as the strategy parameters, and some parameters (e.g., maximum number of literals in the S-CS clause) are set by analysing the attributes of the original clause set. The implementation supports the single-strategy mode and the multi-strategy mode. When trying to solve a problem, the single-strategy mode uses only one strategy, and the multi-strategy mode uses multiple strategies 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 using the previous strategies will be allowed to be used 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 clauses 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 participating 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 Figure 1 below. The corresponding algorithm is described in detail as follows:

- Step 1: According to the strategy parameter, denoted as K, of the number of selected unit clauses, a unit clause set as input clauses is selected by the unit clause selection strategy and added to S_u (a clause set, initially empty). The selected unit clause is denoted as C_i according to the order of the sequence number $i(i = 1, 2, \ldots, K)$.
- Step 2: Select an input clause by the non-unit clause selection strategy, and denote it as $C_j(j = K + 1, \ldots)$.
- Step 3: Apply the S-CS rule to the selected clause C_j; construct a standard contradiction with the input clauses; and get $C_m^{\sigma_m +}$ and $C_m^{\sigma_m -}$ (m is the serial number of the selected clause). Then, the S-CS clause is generated. If the S-CS clause is an empty clause, go to Step 5. Otherwise, the S-CS clause will be checked based on the control conditions as follows: (1) if the S-CS clause is a tautology or a redundant clause; (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 S-CS deduction is unacceptable. The backtracking algorithm will be performed, and go to Step 2.
  The backtracking algorithm is used to return to the last deduction step in the process of S-CS dynamic deduction and needs do some clean-up tasks. It mainly includes: (1) clear the substitutions generated in the current deduction step; (2) remove literals in the S-CS clause and the constructed standard contradiction, which was added in the current deduction step; (3) the unacceptable deduction weight of clause C_j increases by one, and the unacceptable deduction weights of literals in clause C_j that were used for standard contradiction construction increase by one; (4) set the sequence number of the proof search 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 one, and the acceptable deduction weights of literals in the clause C_j that were used for standard contradiction construction increases by one. The generated S-CS clause is represented as a lemma and added into 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 synergised deduction effect, do the following iterative processes: (1) If the clause C_j contains substitution, then record this deduction path to avoid the repetition, clear substitutions of C_j; reuse C_j; and go to Step 3. When the number of reused times of the clause C_j exceeds the set threshold, the sequence number j increases by one, and go to Step 2. (2) When the clause C_j does not contain substitution, the sequence number j increases by one, 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 into the clause set.



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

Figure 1. The pseudo-code for the standard contradiction separation (S-CS) dynamic deduction algorithm.

The functions in the pseudo-code (described in Figure 1) are remarked as follows:

select_unitClauses(S): select a unit clause set in clause set S according to the 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 the S-CS rule with the selected non-unit clause p and the input clause set D; construct a standard contradiction; and get the 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 are described in Step 3 above.

backtrackingSCS(p): in the current deduction step, do clean-up tasks and return to the last deduction step according to the current input clause p.

change_nonUnitClause(S): reselect a new non-unit clause instead of the current selected clause in clause set S.

checkExitConditions(): check whether the exit conditions of the S-CS dynamic deduction algorithm are satisfied.

checkSynergised(p): check whether the input clause p contains substitution; if it does, check whether the number of reused times of the clause exceeds the set threshold.

inner_Simplification(R): apply forward simplification, the factoring rule and equality resolution to the 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.

backward_Simplification(S, c): check whether the clauses in clause set S are redundant clauses using the clause c. If yes, delete the redundant clauses.

5. Lemma Filtration in *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, and 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 depths 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 set 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 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. In order to make *CSE_E* compatible with different running platforms and have good portability, CSE 1.1 was implemented mainly in C++, and JAVA was used for scheduling the running implementation of batch problems. The job dispatch between CSE and E was implemented in JAVA. The object-oriented languages make it easy to manage the various components in CSE 1.1. In order to give the clauses equal opportunity to participate in S-CS deduction, we preferentially selected clauses participating in deduction that rarely produced acceptable deduction. Based on theoretical and practical analysis, some strategies were adopted in *CSE_E*: smaller deduction weight priority for the unit clause selection strategy, smaller acceptable deduction weight priority for the non-unit clause selection strategy, smaller acceptable deduction weight priority for the literal selection strategy. The working system of *CSE_E* 1.0 is available at http://www.tptp.org/cgi-bin/SystemOnTPTP. *CSE_E* 1.0 was tested on both the FOF division problems of CADEATPcompetition CASC-26 (2017) and the CASC-J9 (2018) in comparison with E 2.1 and E 2.2: (1) *CSE_E* 1.0 runs for CASC-26 (2017) problems were done on a computer with a 3.-GHz Intel(R) Core (TM) i7-6700 processor and 8 GB memory and OS Ubuntu15.04 64-bit, with the default time-out of 300 seconds. The test results of E 2.1 were obtained in the same hardware environment and time-out set. (2) *CSE_E* 1.0 participated in the FOF division of the CASC-J9 (2018) competition and won second place. The experimental data of the FOF division problems of CASC-J9 by *CSE_E* 1.0 and E 2.2 were derived from the CASC-J9 competition results directly [6].

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 some different deduction paths compared to E 2.1. Figure 2 shows the comparison of solved problems by *CSE_E* 1.0 and E 2.1 on FOF division problems of CASC-26. *CSE_E* 1.0 solved 398 problems with eight more than E 2.1. The average time spent for the 390 problems solved by E 2.1 was 18.79 s and 21.72 s spent for

398 theorems by *CSE_E* 1.0. It is worth mentioning that the average time spent for 390 problems by *CSE_E* 1.0 was 16.54 s, 2.25 s less than that of E 2.1.

Figure 2 shows that *CSE_E* 1.0 and E 2.1 respectively solved 200 problems in a short period of time (less than 1 s), and their curves were almost coincident. From the 90-second point onwards, the performance of *CSE_E* 1.0 outperformed E 2.1, where *CSE_E* 1.0 solved 370 problems with 12 more than E 2.1. The average time of E 2.1 was 5.9 s and 8.5 s for *CSE_E* 1.0. Within 200 s, *CSE_E* 1.0 solved 386 problems with three more than E 2.1, and the average time of E 2.1 was 14.6 s and 14.2 s for *CSE_E* 1.0. From the overall curve of Figure 2, *CSE_E* 1.0 was basically at the right bottom of E 2.1 and could solve more problems when it was close to the set runtime. Experimental results showed that *CSE_E* 1.0 outperformed E 2.1 in terms of the capability and time efficiency.

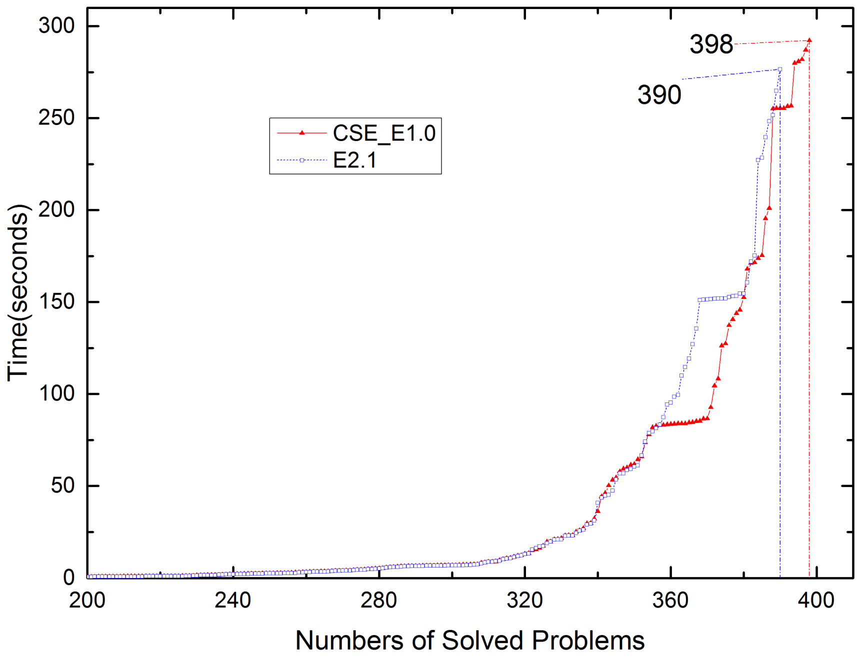


Figure 2. Comparison of solved problems by *CSE_E* 1.0 and E 2.1. CSE, contradiction separation extension.

In the total 110 unsolved problems by E 2.1, *CSE_E* 1.0 could solve 14 out of 110 (12.7% of the total), where six problems were solved by CSE alone and eight problems by E with CSE lemmas added. Table 1 is the list of those 14 problems sorted by the time used.

Those 14 problems had an average rating of 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 synergised 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 was 230 s, so the time efficiency was acceptable.

Table 1. List of the 14 problems solved by *CSE_E* 1.0, but unsolved by E 2.1.

Problem	Rating	Time (s)	Problem	Rating	Time (s)
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 Figure 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 was 25.6 s and 26.5 s spent for 362 problems by *CSE_E* 1.0. It is worth mentioning that the average time spent for 350 problems by *CSE_E* 1.0 was 18.4 s, which was 7.2 s less than that of E 2.2.

Figure 3 shows that *CSE_E* 1.0 and E 2.2 respectively solved 150 problems in a short period of time (less than 2 s), and their curves were almost coincident. From the 80-s point onwards, the performance of *CSE_E* 1.0 outperformed E 2.2, where *CSE_E* 1.0 solved 315 problems with five more than E 2.2. The average time of E 2.2 was 7.9 s and 8.6 s for *CSE_E* 1.0. Within 160 s, *CSE_E* 1.0 solved 346 problems with seven problems more than that of E 2.2, and the average time of *CSE_E* 1.0 was 16.2 s, with 3.2 s less than that of E 2.2. From the overall curve of Figure 3, *CSE_E* 1.0 was at the right bottom of E 2.2. Experimental results showed that *CSE_E* 1.0 outperformed E 2.2 in terms of the capability and time efficiency.

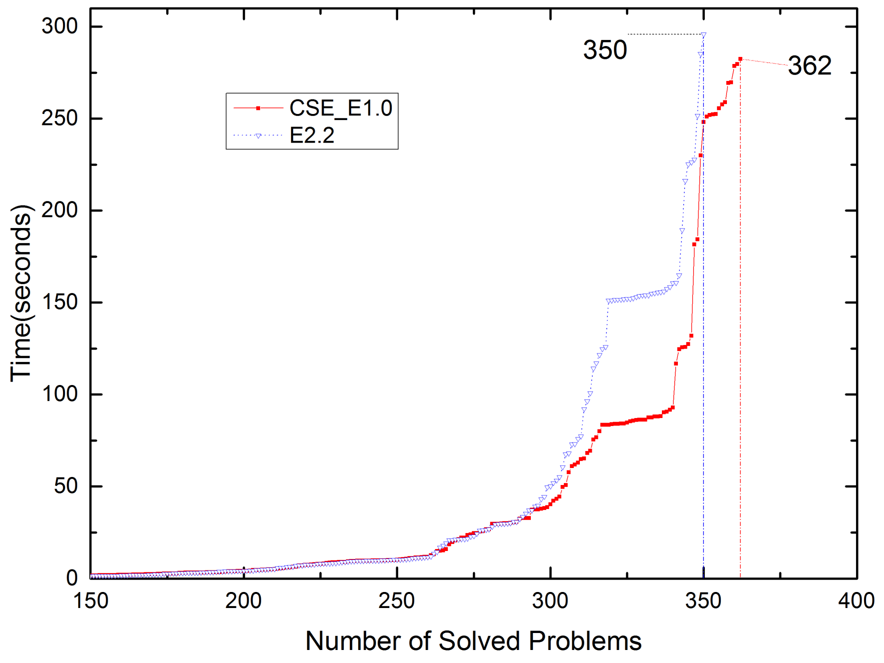


Figure 3. Comparison on solved problems by *CSE_E* 1.0 and E 2.2

Table 2 (sorted by the used time) shows that *CSE_E* 1.0 could solve 14 (9.3%) of the total 150 unsolved problems by E 2.2, where three problems were solved by CSE alone and 11 problems solved by E with CSE lemmas added. Those solved problems had an average rating of 0.74. The average solving time by *CSE_E* 1.0 was 249 s, so the time efficiency was acceptable.

Table 2. List of the 14 problems solved by *CSE_E* 1.0, but unsolved by E 2.1.

Problem	Rating	Time (s)	Problem	Rating	Time (s)
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, according to the results of the CASC-J9 competition, *CSE_E* 1.0 solved one problem, GEO506+1, that had not been solved by any other competition provers at CASC-J9 [6] (listed in Table 3). This problem had a large number of literals and much more variables. *CSE_E* 1.0 can make full use of clauses containing variables, so it could 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 at CASC-J9.

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 stood on the shoulders of the plain E and was able to improve E effectively in terms of capability and time efficiency, and the S-CS dynamic deduction could be effectively applied to first-order logic ATP systems.

7. Conclusions and Future Work

CSE_E 1.0 combines 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 showed that *CSE_E* 1.0 outperformed E 2.1 and E 2.2 in both capability and time efficiency to some extent, which illustrated that the 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 proof search under effective heuristic strategy and the high-quality S-CS clauses are very important. How to filter effectively 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 much 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.

Author Contributions: Conceptualization, F.C.; methodology, F.C. and Y.X.; validation, J.L. and S.C.; writing, original draft preparation, F.C.; writing, review and editing, J.L., S.C. and X.N.; supervision, Y.X.; funding acquisition, Y.X.

Funding: This research was funded by the National Natural Science Foundation of China Grant Number 61673320 and the Fundamental Research Funds for the Central Universities Grant Numbers 2682017ZT12, 2682018CX59 and 2682018ZT25.

Acknowledgments: We thank Stephan Schulz for allowing and supporting the combination of CSE and E to form a new strong ATP system.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Pavlov, V.; Schukin, A.; Cherkasova, T. Exploring Automated Reasoning in First-Order Logic: Tools, Techniques and Application Areas. In Proceedings of the 4th International Conference on Knowledge Engineering and the Semantic Web, St. Petersburg, Russia, 7–9 October 2013; pp. 102–116.
2. Kovacs, L.; Voronkov, A. Finding Loop Invariants for Programs over Arrays Using a Theorem Prover. In Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering, York, UK, 22–29 March 2009; pp. 470–485.
3. Nipkow, T.; Brinkop, H. Amortized Complexity Verified. *J. Autom. Reason.* **2019**, *62*, 367–391. [CrossRef]
4. Sutcliffe, G. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *J. Autom. Reason.* **2017**, *59*, 483–502. [CrossRef]
5. TSTP Solution Domains. Available online: http://www.tptp.org/cgi-bin/SeeTPTP?Category=Solutions (accessed on 16 May 2019).

6. CASC Solution Domains. Available online: http://tptp.org/CASC/ (accessed on 16 May 2019).
7. Robinson, J.A. A machine-oriented logic based on the resolution principle. *J. ACM* **1965**, *12*, 23–41. [CrossRef]
8. Bachmair, L.; Ganzinger, H.; Lynch, C.; Snyder, W. Basic paramodulation and superposition. In Proceedings of the 11th International Conference on Automated Deduction, New York, NY, USA, 15–18 June 1992; pp. 462–476.
9. Loveland, D.W. A linear format for resolution. In *Automation of Reasoning 2: Classical Papers on Computational Logic*; Siekmann, J.H., Wrightson, G., Eds.; Springer: Berlin/Heidelberg, Germany, 1983; pp. 399–416.
10. Chang, C.L. The unit proof and the input proof in theorem proving. In *Automation of Reasoning 2: Classical Papers on Computational Logic*; Siekmann, J.H., Wrightson, G., Eds.; Springer: Berlin/Heidelberg, Germany, 1970; pp. 331–341.
11. Robinson, J.A. Automatic deduction with hyper-resolution. *Int. J. Comput. Math.* **1965**, *1*, 227–234.
12. Overbeek, R.; Mccharen, J.; Wos, L. Complexity and related enhancements for automated theorem-proving programs. *Comput. Math. Appl.* **1976**, *2*, 1–16. [CrossRef]
13. Slaney, J.; Mccharen, B.W. Conflict resolution: A first-order resolution calculus with decision literals and conflict-driven clause learning. *J. Autom. Reason.* **2016**, *12*, 1–24.
14. Reger, G.; Tishkovsky, D. Cooperating Proof Attempts. In Proceedings of the 25th International Conference on Automated Deduction, Berlin, Germany, 1–7 August 2015; pp. 339–355.
15. Schulz, S.; Möhrmann, M. Performance of Clause Selection Heuristics for Saturation-Based Theorem Proving. In Proceedings of the 8th International Joint Conference on Automated Reasoning, Coimbra, Portugal, 27 June–2 July 2016; pp. 330–345.
16. Meng, J.; Paulson, L.C. Lightweight relevance filtering for machine-generated resolution problems. *J. Appl. Log.* **2009**, *7*, 41–57. [CrossRef]
17. Chvalovský, K.; Jakubův, J.; Suda, M., Urban J. ENIGMA-NG: Efficient Neural and Gradient-Boosted Inference Guidance for E. In Proceedings of the 27th International Conference on Automated Deduction, Natal, Brazil, 27–30 August 2019; pp. 197–215.
18. Furbach, U.; Krämer, T.; Schon, C. Names Are Not Just Sound and Smoke: Word Embeddings for Axiom Selection. In Proceedings of the 27th International Conference on Automated Deduction, Natal, Brazil, 27–30 August 2019; pp. 250–268.
19. Rawson, M.; Reger, G. Old or Heavy? Decaying Gracefully with Age/Weight Shapes. In Proceedings of the 27th International Conference on Automated Deduction, Natal, Brazil, 27–30 August 2019; pp. 462–476.
20. Piotrowski, B.; Urban, J. ATPboost: Learning Premise Selection in Binary Setting with ATP Feedback. In Proceedings of the 9th International Joint Conference on Automated Reasoning, Oxford, UK, 14–17 July 2018; pp. 566–574.
21. Xu, Y.; Liu, J.; Chen, S.W.; Zhong, X.M.; He, X.X. Contradiction separation based dynamic multi-clause synergised automated deduction. *Inf. Sci.* **2018**, *462*, 93–113. [CrossRef]
22. Xu, Y.; Chen, S.W.; Liu, J.; Zhong, X.M.; He, X.X. Distinctive features of the contradiction separation based dynamic automated deduction. In Proceedings of the 13th International FLINS Conference, Belfast, UK, 21–24 August 2018; pp. 725–732.
23. Schulz, S. System description: E 1.8. In Proceedings of the 19th International Conference on Logic for Programming Artificial Intelligence and Reasoning, Stellenbosch, South Africa, 14–19 December 2013; pp. 735–743.
24. Sutcliffe, G. The 7th IJCAR Automated Theorem Proving System Competition-CASC-J7. *AI Commun.* **2015**, *28*, 683–692. [CrossRef]
25. Sutcliffe, G. The 8th IJCAR Automated Theorem Proving System Competition-CASC-J8. *AI Commun.* **2016**, *29*, 607–619. [CrossRef]
26. Sutcliffe, G. The 9th IJCAR Automated Theorem Proving System Competition-CASC-J9. *AI Commun.* **2018**, *31*, 495–507. [CrossRef]
27. Kaliszyk, C.; Schulz, S.; Urban, J.; Vyskočil, J. System Description: E.T. 0.1. In Proceedings of the 25th International Conference on Automated Deduction, Berlin, Germany, 1–7 August 2015; pp. 389–398.
28. Kühlwein, D.; Schulz, S.; Urban, J. E-MaLeS 1.1. In Proceedings of the 24th International Conference on Automated Deduction, New York, NY, USA, 9–14 June 2013; pp. 407–413.
29. Daniel, K.; Urban, J. MaLeS: A Framework for Automatic Tuning of Automated Theorem Provers. *J. Autom. Reason.* **2013**, *55*, 91–116.

30. Denzinger, J.; Kronenburg, M.; SCHULZ, S. DISCOUNT—A distributed and learning equational prover. *J. Autom. Reason.* **1997**, *18*, 189–198. [CrossRef]
31. Mccune, W.; Wos, L. Otter—The CADE-13 competition incarnations. *J. Autom. Reason.* **1997**, *18*, 211–220. [CrossRef]
32. Riazanov, A.; Voronkov, A. The design and implementation of vampire. *AI Commun.* **2002**, *15*, 91–110.
33. Kovács, L.; Voronkov, A. First-order theorem proving and vampire. In Proceedings of the 25th International Conference on Computer Aided Verification, Saint Petersburg, Russia, 13–19 July 2013; pp. 1–35.
34. BiereIoan, A.; Kovács, D.; Voronkov, A. Experimenting with SAT solvers in Vampire. In Proceedings of the 13th Mexican International Conference on Artificial Intelligence, Tuxtla Gutiérrez, Mexico, 16–22 November 2014; pp. 431–442.
35. Schulz, S. E—A brainiac theorem prover. *AI Commun.* **2002**, *15*, 111–126.
36. Korovin, K. iProver—An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In Proceedings of the 4th International Joint Conference on Automated Reasoning, Sydney, Australia, 12–15 August 2008; pp. 292–298.
37. Menouer, T.; Baarir, S. Parallel Satisfiability Solver Based on Hybrid Partitioning Method. In Proceedings of the 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, Saint Petersburg, Russia, 6–8 March 2017; pp. 54–60.
38. Maric, F. Formalization and implementation of modern SAT solvers. *J. Autom. Reason.* **2009**, *43*, 81–119. [CrossRef]
39. Liang, J.H.; Ganesh, V.; Poupart, P.; Czarnecki, K. Learning Rate Based Branching Heuristic for SAT Solvers. In Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing, Bordeaux, France, 5–8 July 2016; pp. 123–140.
40. Moskewicz, M.W.; Madigan, C.F.; Zhao, Y.; Zhang, L.T.; Malik, S. Chaff: Engineering an efficient SAT solver. In Proceedings of the 38th Design Automation Conference, Las Vegas, NV, USA, 18–22 June 2001; pp. 530–535.
41. Goldberg, E.; Novikov, Y. BerkMin: A Fast and Robust Sat-Solver. In *Design, Automation, and Test in Europe*; Lauwereins, R., Madsen, J., Eds.; Springer: Dordrecht, The Netherlands, 2008; pp. 465–478.
42. Wu, G.F.; Chen, Q.S.; Xu, Y.; He, X.X. A Hybrid Learnt Clause Evaluation Algorithm for SAT Problem. *Int. J. Comput. Intell. Syst.* **2018**, *12*, 250–258. [CrossRef]