SOLAR: A Consequence Finding System for Advanced Reasoning

Hidetomo Nabeshima¹, Koji Iwanuma¹, and Katsumi Inoue²

¹ University of Yamanashi 4-3-11 Takeda, Kofu-shi 400-8511, Japan
{nabesima,iwanuma}@iw.media.yamanashi.ac.jp
² Kobe University Rokkodai-cho, Nada-ku, Kobe 657-8501, Japan
inoue@eedept.kboe-u.ac.jp

1 Introduction

SOLAR is an efficient first-order consequence finding system based on a connection tableau format with Skip operation. Consequence finding [1, 2, 3, 4] is a generalization of refutation finding or theorem proving, and is useful for many reasoning tasks such as knowledge compilation, inductive logic programming, abduction. One of the most significant calculus of consequence finding is SOL [2]. SOL is complete for consequence finding and can find all minimal-length consequences with respect to subsumption. SOLAR (SOL for Advanced Reasoning) is an efficient implementation of SOL and can avoid producing non-minimal/redundant consequences due to various state of the art pruning methods, such as skip-regularity, local failure caching, folding-up (see [5, 6]).

SOLAR also achieves a good performance as a theorem prover. For 1,921 problems in TPTP v2.5.0 library which do not contain the equality, the experimental results show that SOLAR can solve 52% problems within 300 CPU seconds for each problem, whereas 50% are solved by OTTER 3.2. SOLAR is written in Java, and thus has the desirable features of high programmability, extensibility, reusability, and platform independence. Hence SOLAR can easily be incorporated into many AI programs. According to our knowledge, SOLAR is the first sophisticated implementation of first-order consequence finding calculus in the world.

2 Consequence Finding Procedure SOL

Consequence finding [1, 2, 3, 4] is a computation problem for finding important consequences from an axiom set, and is a generalization of refutation finding or theorem proving. However, in practice, the set of theorems derivable from an axiom set might be infinite, even if it is restricted to containing only the consequences that are minimal with respect to subsumption. Toward more practical automated consequence finding, Inoue [2] reformulated and restricted the attention to the problem for finding only “interesting” consequence formulas, called
characteristic clauses. The concept of characteristic clauses is useful for various reasoning problems of interest to AI, such as nonmonotonic reasoning, abduction, knowledge compilation (see [2, 1] for details), inductive logic programming [7, 8], multi-agent systems [6], bioinformatics [9] and distributed knowledge bases [10].

Inoue [2] proposed SOL-resolution for mechanically finding characteristic clauses within first-order logic, which can be viewed as either an extension of Loveland's model-elimination-like calculus [11] with Skip operation or a generalization of Siegel's propositional production algorithm [12]. Compared with other calculi, SOL-resolution can focus on generating only the characteristic clauses rather than all logical consequences. SOL-resolution is one of the most advanced and significant calculi for the consequence finding problem.

The original SOL-resolution [2] was given in a model-elimination-like chain format [11]. Iwanuma et al. [5] reformulated SOL-resolution within the framework of connection tableaux [13, 14] and proposed various complete pruning methods [5, 6] for enhancing the efficiency of SOL tableaux such as skip-regularity, local failure caching, folding-up.

We give a brief view of SOL tableaux. A production field \( \mathcal{P} \) is a pair, \( (L, Cond) \), where \( L \) is a set of literals and is closed under instantiation, and \( Cond \) is a certain condition to be satisfied. When \( Cond \) is not specified, \( \mathcal{P} \) is denoted as \( (L) \). A clause \( C \) belongs to \( \mathcal{P} = (L, Cond) \) if every literal in \( C \) belongs to \( L \) and \( C \) satisfies \( Cond \). When \( \Sigma \) is a set of clauses, the set of logical consequences of \( \Sigma \) belonging to \( \mathcal{P} \) is denoted as \( \text{Th}_{\mathcal{P}}(\Sigma) \). A production field \( \mathcal{P} \) is stable if, for any two clauses \( C \) and \( D \) such that \( C \) subsumes \( D \), \( D \) belongs to \( \mathcal{P} \) only if \( C \) belongs to \( \mathcal{P} \). The stability of a production field is important in practice [2], and we assume in this paper that production fields are stable.

Example 1. Let \( L = L^+ \cup L^- \) be the set of all literals in the first-order language, where \( L^+ \) and \( L^- \) are the positive and negative literals in the language, respectively. The following are examples of stable production fields.

1. \( \mathcal{P}_1 = (L) \): \( \text{Th}_{\mathcal{P}_1}(\Sigma) \) is the set of logical consequences of \( \Sigma \).
2. \( \mathcal{P}_2 = (L^+) \): \( \text{Th}_{\mathcal{P}_2}(\Sigma) \) is the set of all positive clauses derivable from \( \Sigma \).
3. \( \mathcal{P}_3 = (L^- , \text{length is fewer than } k) \): \( \text{Th}_{\mathcal{P}_3}(\Sigma) \) is the set of negative clauses implied by \( \Sigma \) consisting of fewer than \( k \) literals.

On the contrary, \( \mathcal{P}_4 = (L, \text{length is more than } k) \) is not a stable production field. For example, if \( k = 2 \) and \( L = \{\neg P, Q, R\} \), then \( C = \neg P \lor Q \) subsumes \( D = \neg P \lor Q \lor R \), and \( D \) belongs to \( \mathcal{P}_4 \) while \( C \) does not.

Given a set of clauses \( \Sigma \), a newly added clause \( C \) and a production field \( \mathcal{P} \), an SOL-deduction from \( \Sigma + C \) and \( \mathcal{P} \) satisfies the following:


1. Soundness: If a clause \( S \) is derived by an SOL-deduction from \( \Sigma + C \) and \( \mathcal{P} \), then \( S \) belongs to \( \text{Th}_{\mathcal{P}}(\Sigma \cup \{C\}) \).
2. Completeness: If a clause \( F \) does not belong to \( \text{Th}_{\mathcal{P}}(\Sigma) \) but belongs to \( \text{Th}_{\mathcal{P}}(\Sigma \cup \{C\}) \), then there is an SOL deduction of a clause \( S \) from \( \Sigma + C \) and \( \mathcal{P} \) such that \( S \) subsumes \( F \).