On semantic resolution with lemmaizing and contraction

Maria Paola Bonacina[⋆1] and **Jieh Hsiang** [⋆⋆2]

[1] Department of Computer Science
The University of Iowa, Iowa City, IA 52242-1419, USA
bonacina@cs.uiowa.edu
[2] National Taiwan University, Taipei, Taiwan
hsiang@csie.ntu.edu.tw

Abstract. Reducing redundancy in search has been a major concern for automated deduction. Subgoal-reduction strategies prevent redundant search by using *lemmaizing* and *caching*, whereas contraction-based strategies prevent redundant search by using *contraction* rules, such as *subsumption*. In this work we show that lemmaizing and contraction can coexist in the framework of *semantic resolution*. On the lemmaizing side, we define two meta-level inference rules for lemmaizing in semantic resolution, one for unit and one for non-unit lemmas, and we prove their soundness. Rules for lemmaizing are meta-rules because they use global knowledge about the derivation, e.g. ancestry relations, in order to derive lemmas. On the contraction side, we give contraction rules for semantic strategies, and we define a *purity deletion* rule for first-order clauses that preserves completeness. While lemmaizing generalizes success caching of model elimination, purity deletion echoes failure caching. Thus, our approach integrates features of backward and forward reasoning.

1 Introduction

Some of the most successful theorem-proving programs existing today implement either contraction-based strategies (e.g., [1, 11, 13]) or subgoal-reduction strategies (e.g., [2, 17]). Contraction-based strategies (e.g., [4, 5, 10, 16]) are *forward-reasoning* strategies, that prove the target theorem by deriving consequences from the axioms. (Most forward-reasoning resolution strategies do not differentiate between the target theorem (negated) and the axioms. They treat them as a single set of clauses and try to derive a contradiction from it.) The primary strength of these strategies, is that they apply eagerly contraction inference rules, such as *simplification* and *subsumption*, to delete *redundant* clauses. By effectively reducing redundancy, contraction-based strategies keep the size of the database in check and have been used successfully to prove many problems beyond the reach of other types of strategies (e.g., [1, 11, 13]).

[⋆] Supported in part by grant CCR-94-08667 of the National Science Foundation.
[⋆⋆] Supported in part by grant 85-2221-E-002-009 of the National Science Council.

The subgoal-reduction strategies are *linear*, *backward-reasoning* strategies. In such a strategy an inference step consists in reducing the current goal to a set of subgoals, starting from the input goal. Typical examples are *model elimination* [12], and the *Prolog technology theorem provers* [17]. A weakness of a pure subgoal-reduction strategy is that by concentrating only on the current goal it has no memory of previously solved goals. Therefore, if the same subgoals, or instances thereof, are generated at different stages, the strategy would solve them independently, repeating the same steps. More sophisticated subgoal-reduction strategies avoid such repetitions by using techniques of *lemmaizing*, that is, saving solved goals as lemmas. Lemmaizing for model elimination was presented already in [12]. However, its first implementation in [9] was less efficient than expected [2], because unrestricted lemmaizing generated too many lemmas. More recently, lemmaizing and *caching* [15] in Horn logic have been reintroduced successfully in the framework of Prolog technology theorem proving [3]. Caching comprises *success caching* and *failure caching.* The former is conceptually very close to lemmaizing: solutions are stored in a *cache* for fast retrieval, rather than being added as lemmas. The latter adds the capability of using the information that a goal has failed before to avoid trying to solve it again. Related techniques, called *memoing* or *tabling*, have been explored independently in logic programming [19]. The experimental work has been followed by the theoretical analysis of [14], which shows that lemmaizing and caching reduce from exponential to linear the amount of duplication in the search spaces of model elimination for problems in propositional Horn logic.

Our intent in this paper is to show that lemmaizing and caching are meta-level inferences that may apply to different types of strategies, including strategies that are not based on subgoal reduction. For this purpose, we consider *semantic resolution* strategies. The reason of this choice is that, among resolution strategies, semantic strategies are those that provide a general notion of "goal", by partitioning the database in a consistent set of "axioms" and a *set of support* of "goals". We define meta-rules for lemmaizing in semantic-resolution strategies and we give inference rules that implement them. Lemmaizing in model elimination then becomes a special case[3]. This generalization of lemmaizing is significant in at least two ways:

1. Semantic strategies require that all their inferences are *supported*, i.e. have a premise in the set of support. We observe that lemmaizing consists in generating lemmas from the complement of the set of support (e.g., from the axioms in model elimination), that is, lemmas are unsupported inferences. Semantic-resolution strategies may do forward or backward reasoning depending on how the set of support is defined. If supported inferences are forward inferences, lemmaizing adds backward reasoning to a forward-reasoning strategy; if supported inferences are backward inferences, lemmaizing adds forward reasoning to a backward-reasoning strategy. Therefore, our

[3] The treatment of model elimination in our approach has been omitted for reasons of space and may be found in [6].

treatment makes lemmaizing a general technique for combining forward and backward reasoning in semantic resolution.

2. We point out that lemmaizing is a *meta-level rule*. A derivation is made of inference steps, each justified by an inference rule. Lemmaizing derives a lemma based on a fragment of the current derivation. Therefore, it is an inference at the meta-level with respect to the basic inferences.

In the second part of the paper, we describe how contraction inference rules can be incorporated into semantic-resolution strategies. Furthermore, we define a generalized notion of *purity deletion*, and show how it provides additional power in reducing redundancy in these strategies. Roughly speaking, purity deletion is similar to failure caching, although in a forward-reasoning setting.

In summary, one can have a semantic-resolution strategy that features both contraction and lemmaizing, that are two strengths of contraction-based and subgoal-reduction strategies respectively. Furthermore, contraction-based strategies, unlike subgoal-reduction strategies, are equipped with tools, such as contraction and indexing, to deal with a database of generated and kept clauses. Therefore, while lemmaizing in semantic resolution will certainly need to be restricted, contraction-based strategies might be less sensitive than subgoal-reduction strategies to the risk of generating too many lemmas.

The rest of the paper is organized in the following way. In Section 2 we give a brief summary of semantic resolution and how it plays a role in terms of forward and backward reasoning methods. Section 3 contains the treatment of lemmaizing as meta-level inference rules for semantic resolution. In Section 4 we define concrete inference rules for lemmaizing in strategies with set of support. In Section 5 we show how to incorporate contraction rules in semantic strategies with lemmaizing, and we present purity deletion. For reasons of space, we have left all the proofs in a longer version of this paper [6].

2 Semantic resolution strategies

In *semantic resolution*[4] the application of resolution to a set of clauses S is controlled by a given interpretation I, in such a way that the consistent subset $T \subset S$ that contains the clauses satisfied by I is not expanded. Only resolution steps with at most one premise from T are allowed: a clause in either T or $S-T$, called *nucleus*, resolves with one or more clauses in $S-T$, called *electrons*, until a resolvent that is false in I, and therefore belongs to $S-T$, is generated. Semantic resolution may also assume an ordering on predicate symbols, and then require that the literal resolved upon in an electron contains the greatest predicate symbol in the electron.

Hyperresolution is semantic resolution where the interpretation I is defined based on sign: in *positive* hyperresolution, I contains all the negative literals, T contains the non-positive clauses, $S-T$ contains the positive clauses, and the electrons are positive clauses (from $S-T$) that resolve with all the negative

[4] References were omitted in this section for brevity. They may be found in [6].

literals in the nucleus (from T) to generate a positive hyperresolvent. *Negative* hyperresolution is defined dually. Hyperresolution is more restrictive than generic semantic resolution, because resolution steps where both nucleus and electron are in $S - T$ may not happen (e.g., two positive clauses do not resolve).

In *resolution with set of support* a *set of support* (SOS) is a subset of S such that $S - SOS$ is consistent. Only resolution steps with at most one premise from $S - SOS$ are allowed and all generated clauses are added to SOS. To keep the notation uniform, we use $T = S - SOS$ for the consistent subset in all strategies. Resolution with set of support fits in the paradigm of semantic resolution, under the interpretation that the clauses in T are true, the clauses in SOS and all their descendants are false. *Positive resolution* and *negative resolution* are sometimes considered supported strategies where SOS contains the positive or negative clauses, respectively. However, they are not semantic strategies in the proper sense, because they do not partition the clauses based on an interpretation, with the provision that the consistent set is not expanded.

The original idea of *set-of-support strategy* was that the axioms of a problem usually form a consistent set and that a strategy should not expand such a set, but rather work on the goals. In this interpretation, T contains the axioms, SOS contains the goal clauses (the clauses obtained from the transformation into clausal form of the negation of the target theorem) and the effect of working with a set of support is that most of the work done by the strategy is done on the goals, yielding *backward-reasoning* strategies. The general definitions of semantic resolution and resolution with set of support, however, imply neither backward reasoning nor forward reasoning. For instance, if the axioms are non-negative clauses and the goals are negative clauses, the positive strategies are forward-reasoning strategies and the negative strategies are backward-reasoning strategies compatible with the set-of-support strategy. This is the case in Horn logic. In general, the partition of S into T and SOS based on the distinction between axioms and goals may not agree with the partition based on sign (e.g., the goals may not be negative clauses), so that hyperresolution and the set-of-support strategy are not always compatible.

Linear resolution can be regarded as a linear refinement of resolution with set of support. Given a set of clauses $S = T \cup \{C_0\}$ with a selected *top clause* C_0, the strategy builds a linear derivation, where at step i clause C_{i+1} is generated by resolving the *center clause* C_i with a *side clause*, either a clause in T (an *input clause*), or a clause C_j such that $j < i$ (an *ancestor clause*). If T is consistent and C_0 is the negation of the target, the center clauses form the set of support, and the only needed resolution steps between clauses in SOS are the resolutions with ancestors. Because the strategy is linear, it makes the backward-reasoning character more pronounced: there is a notion of *current goal*, the most recently generated center clause, and each step consists in reducing the current goal to a subgoal. We call such linear, backward-reasoning strategies *subgoal-reduction strategies*.

Linear resolution, however, requires keeping the ancestors around. *Linear input resolution*, where all side clauses are input clauses, is complete for Horn logic, but not for first-order logic. On the other hand, *model elimination* [12] enjoys the advantage of being a linear input strategy that is complete for first-order logic. Roughly speaking, ancestor-resolution inferences are made unnecessary by saving the literals resolved upon in the goals as *framed literals*, and allowing the latter to resolve away subgoal literals[5]. It follows that each step involves either the current goal and an input clause (analogous to an input resolution step) or the current goal only. Therefore, subgoal-reduction strategies based on model elimination usually work on a stack of goals, rather than on a database of clauses, and at each step focus exclusively on the current goal, on top of the stack. The search plan is depth-first search with backtracking, and iterative deepening (DFID) to ensure refutational completeness. Finally, since the axiom set of a problem is static, these strategies yield fast implementations using the Warren Abstract Machine.

3 Generation of lemmas

In this section we present our treatment of lemmaizing. In Sections 3.1 and 3.2 we give meta-rules for lemma generation in the class of semantic strategies. We assume derivations in the form

$$(T_0; SOS_0) \underset{\mathcal{C}}{\vdash} (T_1; SOS_1) \underset{\mathcal{C}}{\vdash} \ldots (T_i; SOS_i) \underset{\mathcal{C}}{\vdash} \ldots,$$

where $\mathcal{C}$ is any strategy in the class, all generated resolvents are added to the SOS component, but the T component is not assumed to be constant, because it may be modified by contraction or lemmaizing. In Section 4 we give inference rules that implement the meta-rules for resolution with set of support.

3.1 Generating unit lemmas

Intuitively speaking, if $T \cup \{\neg L\} \models \Box$, then $T \models L\sigma$ for some substitution σ, and $L\sigma$ can be treated as a lemma of T and be added to T. Generalizing this idea slightly, if a clause $C\sigma$ is deduced from $\neg L \vee C$ using T alone (without using any other clause in SOS), it means that $T \cup \{\neg L \vee C\} \models C\sigma$. Then $L\sigma$ can also be added as a lemma to T.

There is a caveat, however. For this argument to be sound, it is necessary for the C in $\neg L \vee C$ not to take any part in the derivation of $C\sigma$ from $T \cup \{\neg L \vee C\}$. More precisely, the derivation of $C\sigma$ from $T \cup \{\neg L \vee C\}$ does not include any resolution or factoring step with a selected literal[6] in C. This is necessary to make sure that the existence of a derivation of $C\sigma$ from $T \cup \{\neg L \vee C\}$ implies the existence of a derivation of $L\sigma$ from T. If the derivation from $T \cup \{\neg L \vee C\}$ involves literals in C, then the existence of a derivation of $L\sigma$ from T is not

[5] Model elimination may be presented in many ways, e.g. as a refinement of linear resolution or as a tableaux-based method. We refer to [6] for more details.

[6] A selected literal is a literal resolved upon in a resolution step or unified in a factoring step.

ensured, because the steps involving the literals in C may not be reproducible in a derivation from T. The following definitions will capture this requirement.

Definition 3.1 *Let C be a clause and C' be a binary resolvent or a factor of C. The relation $A \mapsto B$ holds if A is a literal in C different from the selected literal(s) in generating C', B is a literal in C' and $B = A\sigma$, where σ is the most general unifier of the inference generating C'.*

The relation $A \mapsto B$ captures the inheritance of literals that are not selected. By using the transitive closure $\mapsto^*$ of $\mapsto$, we can represent inheritance of literals through a sequence of steps:

Definition 3.2 *Given a resolution derivation $S \vdash^* S'$, where S and S' are sets of clauses, a clause $C' \in S'$ is a* strict descendant *of a clause $C \in S$, if for every literal $A' \in C'$ there is a literal $A \in C$, such that $A \mapsto^* A'$.*

Definition 3.3 *Let S be a set of clauses. $C\sigma$ is* linearly derived from $\neg L \vee C$ by using S *if there is a linear resolution derivation with input set of clauses S, top clause $\neg L \vee C$ and last center clause $C\sigma$. We denote such a derivation by $\neg L \vee C \mathrel{|\!\!\leadsto}_S C\sigma$.*
If the derivation is a linear input derivation (i.e., all side clauses come from S) and $C\sigma$ is a strict descendant of $\neg L \vee C$, we say that $C\sigma$ is strictly linearly derived from $\neg L \vee C$ by using S *and we write $\neg L \vee C \mathrel{|\!\!\leadsto}^h_S C\sigma$.*

Coming back to the SOS strategy, given sets SOS and T, $\neg L \vee C \mathrel{|\!\!\leadsto}^h_T C\sigma$ indicates that $C\sigma$ is derived from $\neg L \vee C$ and T, and that in the derivation, no literals in C and no clauses in SOS are involved in any of the inference steps. We have now all the elements to write the first meta-rule for lemma generation:

Definition 3.4 Unit Lemmaizing*: if $\neg L \vee C \mathrel{|\!\!\leadsto}^h_T C\sigma$, then add lemma $L\sigma$ to T.*

Then we prove the soundness of Unit Lemmaizing:

Theorem 3.1 *If $\neg L \vee C \mathrel{|\!\!\leadsto}^h_T C\sigma$, then $T \models L\sigma$.*

3.2 Generating non-unit lemmas

The derivation $\neg L \vee C \mathrel{|\!\!\leadsto}^h_T C\sigma$ in the condition for Unit Lemmaizing satisfies the restrictions that all side clauses of the (linear) derivation come from T and that the literals in C are not selected in the derivation. In Horn logic, since linear input resolution is complete and factoring is not necessary, Unit Lemmaizing is the only form of lemmaizing. In first-order logic, one may have a derivation $\neg L \vee C \mathrel{|\!\!\leadsto}_{T \cup SOS} C\sigma$, in which members of the set SOS are also used and $C\sigma$ is not necessarily a strict descendant of C. This condition leads to a more general meta-rule for lemmaizing, that may generate also non-unit lemmas. The general form of a lemma will be $(L \vee F)\sigma$, or $(\neg F \supset L)\sigma$, where $\neg F\sigma$ is the "premise" for $L\sigma$ to hold. Operationally, F contains those subgoals of $\neg L$ that are resolved in the derivation $\neg L \vee C \mathrel{|\!\!\leadsto}_{T \cup SOS} C\sigma$ by using SOS or C, but cannot be resolved by using T only. They are formally defined as follows:

Definition 3.5 *Given a derivation* $\neg L_1 \vee \cdots \vee \neg L_k \vee C \mid\!\leadsto_{T \cup SOS} C\sigma$*, for all* i*,* $1 \leq i \leq k$*, the* residue *of* $\neg L_i$ *in* T*, denoted by* $R_T(\neg L_i)$*, is defined as follows: let* $D = L' \vee Q_1 \vee \ldots \vee Q_m$ *be the clause that resolves with* $\neg L_1 \vee \cdots \vee \neg L_k \vee C$ *upon* $\neg L_i$ *and* L'*; then*

$$R_T(\neg L_i) = \begin{cases} \neg L_i & \textit{if } D \in SOS, \\ false & \textit{if } D \in T \textit{ and } m = 0, \\ R_T(Q_1) \vee \ldots R_T(Q_m) & \textit{if } D \in T \textit{ and } m \geq 1, \\ \neg L_i & \textit{if } \neg L_i \textit{ is removed by factoring} \\ & \textit{with a literal in } C, \\ R_T(\neg L_j) & \textit{if } \neg L_i \textit{ is removed by factoring} \\ & \textit{with } \neg L_j \textit{ for } 1 \leq j \neq i \leq k. \end{cases}$$

Example 3.1 *Assume that* $\neg L \vee C \mid\!\leadsto_{T \cup SOS} C\sigma$ *is made of the following steps:*

1. $\neg L \vee C$ *resolves with* $L \vee P$ *generating* $P \vee C$*,*
2. $P \vee C$ *resolves with* $\neg P \vee Q \vee R$*, generating* $Q \vee R \vee C$*,*
3. $Q \vee R \vee C$ *resolves with* $\neg Q$*, generating* $R \vee C$*,*
4. $R \vee C$ *resolves with* $\neg R$*, generating* C*.*

We now analyze the residue $R_T(\neg L)$*, according to different situations. If* $L \vee P \in SOS$*, then* $R_T(\neg L) = \neg L$*. If* $L \vee P \in T$*, then* $R_T(\neg L) = R_T(P)$*. In the latter case, if* $\neg P \vee Q \vee R \in SOS$*, then* $R_T(\neg L) = R_T(P) = P$*. On the other hand, if* $\neg P \vee Q \vee R \in T$*, then* $R_T(\neg L) = R_T(P) = R_T(Q) \vee R_T(R)$*. Since* $R_T(Q) = Q$ *if* $\neg Q \in SOS$ *and* $R_T(Q) = false$ *if* $\neg Q \in T$*, and the same is true for* R*, the value of* $R_T(\neg L)$ *in the last case can be determined by the different combinations of* $R_T(Q)$ *and* $R_T(R)$*.*

A meta-rule for *Generalized Lemmaizing* can then be formulated:

Definition 3.6 *If* $\neg L \vee C \mid\!\leadsto_{T \cup SOS} C\sigma$*, then add lemma* $(L \vee R_T(\neg L))\sigma$ *to* T*.*

Unit Lemmaizing is the special case of Generalized Lemmaizing where $R_T(\neg L)$ is $false$, and thus $(L \vee R_T(\neg L))\sigma$ reduces to $L\sigma$. If $R_T(\neg L)$ is $\neg L$, it means that $\neg L$ itself cannot be resolved in T and therefore no lemma should be added. Indeed, in such a case $(L \vee R_T(\neg L))\sigma$ is a tautology.

Proposition 3.1 *Given a derivation* $\neg L \vee C \mid\!\leadsto_{T \cup SOS} C\sigma$*, if literal* $\neg L$ *can be eliminated only by a resolution step with a side clause from* SOS *or by factoring, then no non-trivial lemma can be generated.*

We conclude with the soundness of Generalized Lemmaizing:

Theorem 3.2 *If* $\neg L \vee C \mid\!\leadsto_{T \cup SOS} C\sigma$*, then* $T \models (L \vee R_T(\neg L))\sigma$*.*

4 Inference rules for Generalized Lemmaizing

In this section we assume that the underlying strategy is resolution with set of support and we give a set of inference rules for resolution and factoring that implement our meta-rules for lemmaizing within such a strategy. In the inference rules the expression $[F]_L$ is used to denote that F, a conjunction of literals, is a potential list of premises for resolving away the literal L completely using only clauses in T. In other words, F is part of the residue of L. When F in $[F]_L$ contains the entire residue of L, the lemma $\neg L \vee F$ can be generated. Some of the inference rules generate a resolvent with literals labelled by a subscript L, such as Q_L. These are subgoals produced while resolving away L, and they themselves need to be resolved away before a lemma concerning L can be generated. We call a literal with subscript L an *L-subgoal*.

The inference rules are separated into three categories, one for resolution, one for factoring, and one for lemmatization.

4.1 The resolution rules

Several different resolution rules are needed since, in a set of support strategy, the sets SOS and T play different roles.

In the first rule for resolution, literal L in $L \vee C$ is resolved with a non-unit clause from T, and a lemma involving $L\sigma$ is initiated:

Resolution with lemma initiation

$$\frac{(T \cup \{\neg L' \vee D\}; SOS \cup \{L \vee C\})}{(T \cup \{\neg L' \vee D\}; SOS \cup \{L \vee C, (D_{L\sigma} \vee [false]_{L\sigma} \vee C)\sigma\})} \quad L\sigma = L'\sigma$$

where L and L' are literals, C and D are disjunctions of literals and σ is the most general unifier of L and L'. $D_{L\sigma}$ has the same literals as D, except that they are labelled by a subscript $L\sigma$. These are the $L\sigma$-subgoals, that need to be resolved away before a lemma concerning $L\sigma$ can be generated. The expression $[false]_{L\sigma}$ means that at this stage the premise of a potential lemma $L\sigma$ is empty. In this inference rule we assume that neither of the clauses involved in the resolution step has any subscripted literals. If the T-clause is a unit clause, no lemma is initiated, and plain unit resolution applies, because lemmaizing would produce an instance of the unit clause in T.

In the second rule for resolution, $\neg L$ is resolved with a clause from SOS:

Plain resolution

$$\frac{(T; SOS \cup \{L' \vee D, \neg L \vee C\})}{(T; SOS \cup \{L' \vee D, \neg L \vee C, (D \vee C)\sigma\})} \quad L\sigma = L'\sigma$$

We assume that neither of the two clauses involved in the resolution step has any subscripted literals. In this case, if one were to produce a residue for L, it would be L itself (by the first case of Definition 3.5), which would result in a lemma that is a tautology (Proposition 3.1) Thus, there is no need to initiate a lemma.

The next two rules have the condition $P\sigma = L'\sigma$:

Residue extension

$$\frac{(T; SOS \cup \{\neg L' \vee Q, P_L \vee D_L \vee C \vee [F]_L\})}{(T; SOS \cup \{\neg L' \vee Q, P_L \vee D_L \vee C \vee [F]_L, (Q_{L\sigma} \vee D_{L\sigma} \vee C \vee [F \vee P]_{L\sigma})\sigma\})}$$

In this rule, F is the lemma for L being constructed, and $P_L \vee D_L$ is the disjunction of the L-subgoals to be solved. Since P, an L-subgoal, is resolved with a clause in SOS, the remaining literals coming from that clause also become part of the set of L-subgoals, and P has to be added to the residue list. We remark that the clause $\neg L' \vee Q$ may also be labelled. For instance, $\neg L' \vee Q$ may have the form $\neg L'_M \vee E_M \vee B \vee [H]_M$. Then, the above rule may generate two resolvents: $(E_{L\sigma} \vee B_{L\sigma} \vee D_{L\sigma} \vee C \vee [F \vee P]_{L\sigma})\sigma$ and $(D_{M\sigma} \vee C_{M\sigma} \vee E_{M\sigma} \vee B \vee [H \vee \neg L']_{M\sigma})\sigma$.

Subgoal elimination

$$\frac{(T \cup \{\neg L' \vee Q\}; SOS \cup \{P_L \vee D_L \vee C \vee [F]_L\})}{(T \cup \{\neg L' \vee Q\}; SOS \cup \{P_L \vee D_L \vee C \vee [F]_L, (Q_{L\sigma} \vee D_{L\sigma} \vee C \vee [F]_{L\sigma})\sigma\})}$$

This rule is similar to *residue extension* except that the resolved literal P is not added to the residue list. This is because the clause which resolves P away is from T.

4.2 The factoring rules

Similar to the resolution rules, the factoring rules need to consider the behaviour of the L-subgoals and residues. All the following rules have the condition $P\sigma = P'\sigma$:

Residue extension factoring

$$\frac{(T; SOS \cup \{P_L \vee D_L \vee [F]_L \vee C \vee P'\})}{(T; SOS \cup \{P_L \vee D_L \vee [F]_L \vee C \vee P', (D_{L\sigma} \vee [F \vee P]_{L\sigma} \vee C \vee P')\sigma\})}$$

This rule says that if an L-subgoal is eliminated by factoring with a "normal" SOS-literal, then it needs to be considered as part of the residue of L.

Subgoal deletion factoring

$$\frac{(T; SOS \cup \{P_L \vee D_L \vee P'_L \vee C\})}{(T; SOS \cup \{P_L \vee D_L \vee P'_L \vee C, (D \vee P'_{L\sigma} \vee C)\sigma\})}$$

This rule says that, when factoring between two L-subgoals, one of them can be eliminated. This rule corresponds to the fifth case in Definition 3.5.

Plain factoring

$$\frac{(T; SOS \cup \{P \vee D_L \vee P' \vee C\})}{(T; SOS \cup \{P \vee D_L \vee P' \vee C, (D_L \vee P' \vee C)\sigma\})}$$

4.3 The lemma generation rule

Lemmaizing

$$\frac{(T; SOS \cup \{[F]_L \vee C\})}{(T \cup \{\neg L \vee F\}; SOS \cup \{C\})} \quad C \textit{ does not contain any } L\textit{−subgoals}$$

In the rule for lemmaizing, all the subgoals of the literal L have been solved, and therefore $\neg L$ with its residue is turned into a lemma.

Example 4.1 *If $T = \{P \vee R, \neg R\}$ and $SOS = \{\neg P \vee \neg Q\}$, the first resolvent is $\neg Q \vee R_{\neg P} \vee [false]_{\neg P}$. The new ($\neg P$)-subgoal in the resolvent resolves with $\neg R$ of T and derives $\neg Q \vee [false]_{\neg P}$. By the Lemmaizing rule, the last resolvent becomes $\neg Q$ and a lemma P can be added to T. Note that since set-of-support forbids resolution among members of T, the same lemma cannot be obtained from T directly.*

Example 4.2 *If T contains $\neg P \vee \neg Q$ and SOS contains $P \vee \neg Q$, then $\neg Q_P \vee [false]_P \vee \neg Q$ is inferred by Resolution with lemma initiation. A factoring step generates the factor $[false \vee \neg Q]_P \vee \neg Q$. Since $\neg Q$ is the residue of P, the lemma $\neg P \vee \neg Q$ is generated. In this case the lemma is already in T so that it is not added.*

5 Eliminating redundancy in contraction-based strategies

Forward-reasoning resolution strategies often adopt contraction inference rules such as *clausal subsumption* and *clausal simplification* to reduce the database of clauses. Since space explosion is usually the critical factor deciding whether a successful derivation is possible, the more power contraction exhibits, the more effective the proof method is. The combination of the set-of-support strategies and contraction strategies seems to have been implemented in some provers including OTTER, but it is rarely studied in the theorem-proving literature. In this section we discuss two results. First we present schemes of inference rules to incorporate contraction in semantic strategies, including strategies with lemmaizing. Then we introduce a notion of *purity* with which one can utilize unresolvable literals to detect and delete redundant clauses. The latter is similar to "failure caching" in the Prolog technology theorem proving framework.

5.1 Incorporating contraction in semantic strategies with lemmaizing

We start with an example which shows that naïve contraction in a semantic strategy may destroy its completeness:

Example 5.1 *Assume a semantic strategy featuring clausal simplification and a contraction-first search plan. Given $T = \{\neg P, P \vee Q\}$ and $SOS = \{\neg Q\}$, the strategy looks for contraction steps first, and it contracts $P \vee Q$ in T to P, by* *clausal simplification by $\neg Q$. It follows that $T = \{\neg P, P\}$ and $SOS = \{\neg Q\}$. The set T has become inconsistent and no refutation can be found by a semantic strategy, since resolution between clauses in T is not allowed.*

Intuitively, if contraction "moves" the inconsistency of $T \cup SOS$ into T, the semantic strategy becomes incomplete, because a semantic strategy assumes that T is consistent, and therefore is not able to detect an inconsistency in T. Thus, if I is the interpretation controlling the semantic strategy, a clause generated by contraction should be added to T only if the clause is true in I. The following schemes of inference rules for contraction capture this idea:

Contraction of SOS

$$\frac{(T; SOS \cup \{C\})}{(T; SOS \cup \{D\})} \quad \textit{C is contracted to D}$$

If a clause in SOS is contracted, the resulting clause is also false in I and belongs to SOS.

Contraction of T by T

$$\frac{(T \cup \{C\}; SOS)}{(T \cup \{D\}; SOS)} \quad \textit{C is contracted to D by clauses in T}$$

If a clause in T is contracted by clauses in T, the resulting clause is also true in I and therefore belongs to T.

Contraction of T by SOS

$$\frac{(T \cup \{C\}; SOS)}{(T; SOS \cup \{D\})} \quad \textit{C is contracted to D by clauses in SOS and } I \not\models D$$

$$\frac{(T \cup \{C\}; SOS)}{(T \cup \{D\}; SOS)} \quad \textit{C is contracted to D by clauses in SOS and } I \models D$$

If a clause in T is contracted by clauses in SOS, one needs to resort to the definition of I in order to decide the affiliation of the new clause. Otherwise, incompleteness such as shown in Example 5.1 may occur. For instance, for hyperresolution, D is placed according to the sign of its literals. In resolution with set of support, all clauses descending from SOS clauses are regarded as false clauses and belong to SOS. Thus, the above two rules can be combined into the following simpler scheme:

$$\frac{(T \cup \{C\}; SOS)}{(T; SOS \cup \{D\})} \quad \textit{C is contracted to D by clauses in SOS}$$

In case of deletion rules such as subsumption, C is simply deleted and no clause D is generated.

Example 5.2 *If the strategy is resolution with set of support and clausal simplification is applied to $T = \{\neg P, P \vee Q\}$ and $SOS = \{\neg Q\}$ of Example 5.1, according to the above schemes, we get $T = \{\neg P\}$ and $SOS = \{\neg Q, P\}$. Resolution of $\neg P \in T$ and $P \in SOS$ completes the proof.*

The following theorem summarizes the compatibility of contraction with semantic strategies:

Theorem 5.1 *Let I_1 denote a resolution inference system, I_1' a semantic restriction of I_1, and I_2 a set of sound contraction inference rules. If I_1' is refutationally complete and $I_1 \cup I_2$ is refutationally complete, then $I_1' \cup I_2$, where the contraction rules in I_2 are applied according to the above schemes, is also refutationally complete.*

For instance, I_2 can contain clausal simplification and clausal subsumption. Results on the completeness of forward-reasoning (ordered) resolution strategies with contraction (i.e., the completeness of $I_1 \cup I_2$) may be found in the literature (e.g., [4, 10]).

The above schemes and theorem remain valid for resolution with set of support and lemmaizing with lemmas generated according to the rules of Section 4.1. One only needs to take care that whenever a contraction inference rule eliminates a subscripted literal in an SOS clause by applying another SOS clause, the residue is updated. We give an inference rule for clausal simplification as an example:

Clausal simplification of SOS by SOS

$$\frac{(T; SOS \cup \{Q, \neg Q'_L \vee C \vee [F]_L\})}{(T; SOS \cup \{Q, C \vee [F \vee \neg Q']_L\})} \quad Q\sigma = Q' \; for \; some \; \sigma$$

where Q is a unit clause which may be either positive or negative. It is presented here as positive just for convenience. If Q' is not subscripted, the inference rule works in the same way but no residue is added. Notice that the difference between this clausal simplification rule and the subgoal elimination rule of Section 4.1 is that clausal simplification replaces a clause by another one, whereas subgoal elimination adds a new clause.

5.2 Purity deletion

One technique used quite effectively in subgoal-reduction strategies that has not been used at all in forward reasoning is the notion of *failure caching.* Failure caching says that if a goal literal fails, then it can be used to fail any similar goals in the future. Since there is usually no notion of a goal in a forward-reasoning strategy, it is little wonder that failure caching has not been used in this context.

A goal literal (in a subgoal-reduction strategy) fails if it does not unify with any literal of opposite sign in the given set of clauses. One can in fact adopt this idea to get the opposite effect of lemmaizing. To be more precise, if a literal cannot be resolved away, then obviously it cannot play a role in any derivation of refutation. Therefore, any clause which contains such a literal can be deleted. In fact, this idea already existed in the Davis-Putnam procedure [8], in which such a literal was called a *pure* literal. We adopt the name and generalize the notion by the following inductive definition:

Definition 5.1 *Let S be a set of clauses and A be a literal occurring in S.*

1. *If for all clause $C \in S$ there is no literal $B \in C$, such that $A\sigma = \neg B\sigma$ for some substitution σ, then A is* pure *in S.*
2. *If for all the clauses $C \in S$ that contains a B such that $A\sigma = \neg B\sigma$ for some σ, C contains an instance of a pure literal, then A is* pure *in S.*

Condition 1 (*basic purity*) is the basis of the definition, and Condition 2 is the inductive case, that represents a sort of *transitive closure* of purity. Its logical justification is that a clause that contains a pure literal is not necessary for the refutation, and therefore a literal that can resolve only with unnecessary clauses is also unnecessary, or pure.

Proposition 5.1 *If a literal A is pure, then any instance of A is also pure.*

Our inference rule states that any clause which contains a pure literal can be deleted:

Purity deletion

$$\frac{S \cup \{C\}}{S}\ \exists A \in C,\ A \textit{ is pure}$$

If clauses that contain pure literals are deleted, more literals may become pure. The inductive case of the definition of purity captures the propagation of basic purity caused by the application of the Purity Deletion rule: if all the clauses that A may resolve with contain a pure literal, all such clauses will be deleted by the Purity Deletion rule and therefore A will become pure according to basic purity. The inductive part of the definition "anticipates" this propagation effect. Therefore, in order to show that clauses containing pure literals can be deleted while preserving refutational completeness, it is sufficient to consider the basis of the definition of purity.

In propositional logic, if S is an unsatisfiable set of clauses and A is pure in S, then $S' = S - \{C|A \in C\}$ is also unsatisfiable. This is because if S' were satisfiable, there would be a Herbrand model I of S'. Since neither A nor $\neg A$ appears in S', neither of them needs to be in I. Then $I \cup \{A\}$ would be a model of S, contradicting the fact that S is unsatisfiable. The same reasoning does not apply in this form to a set of first-order clauses for the following reason: if A is a ground first-order literal, the fact that neither A nor $\neg A$ appears in S' does not imply that neither of them is in I, because A may be in the Herbrand base of S' even if it does not occur in S', and therefore either A or $\neg A$ may be in I. A mapping of ground first-order atoms into propositional variables, however, is sufficient to extend the reasoning to sets of ground first-order clauses:

Lemma 5.1 *Let S be a finite set of ground first-order clauses and $A(\bar{t})$ be a pure literal in S. If S is unsatisfiable, then $S' = S - \{C|A(\bar{t}) \in C\}$ is unsatisfiable.*

By applying the Herbrand theorem, the lemma can then be lifted to a set of general first-order clauses:

Theorem 5.2 *Let S be a finite set of first-order clauses and $A(\bar{t})$ a pure literal in S. If S is unsatisfiable, then $S' = S - \{C | A(\bar{t}) \in C\}$ is unsatisfiable.*

In summary, a pure literal in the forward-reasoning context has some similarity with a failed goal of subgoal-reduction strategies, since both notions are based on the impossibility of unifying the literal. Indeed, instances of pure literals are pure, like instances of failed goals also fail. In this sense, purity deletion echoes failure caching in forward-reasoning strategies.

6 Discussion

Forward-reasoning resolution strategies for first-order logic often suffer from generating too many irrelevant clauses. In order to control the growth of the database of clauses, contraction inference rules are usually employed. Subgoal-reduction strategies, on the other hand, lack the ability of producing useful lemmas which may reduce the search effort. The question of how to combine the best of the two worlds has long been a challenge to the automated deduction community.

In this paper we address this question by showing how the technique of lemmaizing can be extended and used in the general context of semantic strategies. We show how contraction rules can be incorporated in semantic strategies, even with lemmaizing. We also presented a new redundancy deletion method for forward-reasoning strategies which is based on a notion of purity.

Lemmaizing is a concept used in the logic programming community for saving previous execution results for later use. It has been used effectively both in enhancing Prolog [19] and in Prolog technology theorem proving [3]. In this paper we have shown how to generalize lemmaizing to semantic resolution, and how to integrate lemmaizing and contraction in such a strategy. Our approach has a number of advantages. Unlike previous work on lemmatization which is mostly limited to model elimination, it provides a general way of adding lemmas to the complement of the set of support. Therefore, it provides a flexible way of adding some forward-reasoning ability to goal-oriented strategies or, vice versa, some backward-reasoning ability to forward strategies. Our method also works with non-unit lemmas, and it does not have the left-to-right order of evaluation restriction which is common with subgoal-reduction strategies such as model elimination. Lastly, our work on purity highlights an intuitive correspondence between purity deletion and failure caching. This is complementary to the intuitive correspondence between subsumption and success caching suggested in [18], and therefore reinforces the understanding that while contraction eliminates redundancy in contraction-based strategies, caching eliminates redundancy in subgoal-reduction strategies.

Acknowledgements

We would like to thank Mark Stickel for answering our questions on caching in model elimination.

References

1. S. Anantharaman and J. Hsiang, Automated Proofs of the Moufang Identities in Alternative Rings, *Journal of Automated Reasoning*, Vol. 6, No. 1, 76–109, 1990.
2. O. L. Astrachan and D. W. Loveland, METEORs: High performance theorem provers using model elimination, in R.S.Boyer (ed.), *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Kluwer Academic Publisher, Dordrecht, 1991.
3. O. L. Astrachan and M. E. Stickel, Caching and Lemmaizing in Model Elimination Theorem Provers, in D. Kapur (ed.), *Proc. of the 11th CADE*, Springer Verlag, LNAI 607, 224–238, 1992.
4. L. Bachmair and H. Ganzinger, On Restrictions of Ordered Paramodulation with Simplification, in M. E. Stickel (ed.), *Proc. of the 10th CADE*, Springer Verlag, LNAI 449, 427–441, 1990.
5. M. P. Bonacina and J. Hsiang, Towards a foundation of completion procedures as semidecision procedures, *Theoretical Computer Science*, Vol. 146, 199–242, July 1995.
6. M. P. Bonacina and J. Hsiang, On semantic resolution with lemmaizing and contraction, Tech. Rep., Dept. of Computer Science, University of Iowa, Sept. 1995.
7. C. L. Chang and R. C. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.
8. M. Davis and H. Putnam, A computing procedure for quantification theory, *Journal of the ACM*, Vol. 7, 201–215, 1960.
9. S. Fleisig, D. Loveland, A. Smiley and D. Yarmash, An Implementation of the Model Elimination Proof Procedure, *Journal of the ACM*, Vol. 21, 124–139, 1974.
10. J. Hsiang and M. Rusinowitch, Proving Refutational Completeness of Theorem Proving Strategies: the Transfinite Semantic Tree Method, *Journal of the ACM*, Vol. 38, No. 3, 559–587, July 1991.
11. D. Kapur and H. Zhang, RRL: a Rewrite Rule Laboratory, in E. Lusk, R. Overbeek (eds.), *Proc. of the 9th CADE*, Springer Verlag, LNCS 310, 768–770, 1988.
12. D. W. Loveland, A Simplified Format for the Model Elimination Procedure, *Journal of the ACM*, Vol. 16, No. 3, 349–363, July 1969.
13. W. W. McCune, Otter 3.0 Reference Manual and Guide, Tech. Rep. ANL-94/6, Mathematics and Computer Science Division, Argonne Nat. Lab., Jan. 1994.
14. D. A. Plaisted, The Search Efficiency of Theorem Proving Strategies, in A.Bundy (ed.), *Proc. of the 12th CADE*, Springer Verlag, LNAI 814, 57–71, 1994, and Tech. Rep. MPI-I-94-233, Max Planck Institut für Informatik.
15. D. A. Plaisted, Non-Horn Clause Logic Programming Without Contrapositives, *Journal of Automated Reasoning*, Vol. 4, No. 3, 287–325, 1988.
16. M. Rusinowitch, Theorem-proving with Resolution and Superposition, *Journal of Symbolic Computation*, Vol. 11, No. 1 & 2, 21–50, Jan./Feb. 1991.
17. M. E. Stickel, A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler, *Journal of Automated Reasoning*, Vol. 4, 353-380, 1988.
18. M. E. Stickel, PTTP and Linked Inference, in R. S. Boyer (ed.), *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Kluwer Academic Publishers, Dordrecht, 1991.
19. D. S. Warren, Memoing for logic programs, *Communications of the ACM*, Vol. 35, No. 3, 94–111, Mar. 1992.

This article was processed using the LaTeX macro package with LLNCS style