A New Method for Consequence Finding and Compilation in Restricted Languages

Alvaro del Val
Departamento de Ingeniería Informática
Universidad Autónoma de Madrid
28049 MadridΓSpain
delval@ii.uam.es
http://www.ii.uam.es/~delval

Abstract

SFK (skip-filtered, kernel) resolution is a new method for finding "interesting" consequences of a first order clausal theory Σ, namely those in some restricted target language $\mathcal{L}_T$. In its more restrictive form, SFK resolution corresponds to a relatively efficient SAT method, directional resolution; in its more general form, to a full prime implicate algorithm, namely Tison's. It generalizes both of them by offering much more flexible search, first order completeness, and a much wider range of inferential capabilities.

SFK resolution has many applications: computing "characteristic" clauses for task-specific languages in abduction, explanation and non-monotonic reasoning (Inoue 1992); obtaining LUB approximations of the input theory (Selman and Kautz 1996) which are of polynomial size; incremental and lazy exact knowledge compilation (del Val 1994); and compilation into a tractable form for restricted target languages, *independently* of the tractability of inference in the given target language.

Introduction

We introduce kernel deductionsΓwhich in the propositional case can be succintly described as the set of resolution proofs explored by Tison's method for computing the prime implicates of a propositional theory (Tison 1967). Kernel resolutionΓwhich resembles somewhat ordered resolutionΓis complete for first-order consequence finding. Skip-filtered kernel resolution (SFK for short) provides a very general way of restricting kernel resolution to make it complete for all consequences in some restricted target language.

A first order clausal theory Σ is written over some clausal language $\mathcal{L}$Γconsisting of all clauses that can be constructed using terms and predicates occurring in Σ. Among the implicates or clausal consequences of Σ (clauses $C \in \mathcal{L}$ such that $\Sigma \models C$)Γwe might be interested in all of themΓor only in those that belong to some target language $\mathcal{L}_T \subseteq \mathcal{L}$. These are the $\mathcal{L}_T$-implicates (clauses $C \in \mathcal{L}_T$ such that $\Sigma \models C$). Let $PI_{\mathcal{L}_T}(\Sigma)$ be the set of prime $\mathcal{L}_T$-implicates of ΣΓi.e. $\mathcal{L}_T$-implicates not properly (θ-)subsumed by any other $\mathcal{L}_T$-implicate of Σ. Then the restricted task of $\mathcal{L}_T$-consequence finding is (mainly) finding all clauses in $PI_{\mathcal{L}_T}(\Sigma)$.

Consequence finding (as opposed to refutation finding) has received relatively little attention in the theorem proving literature. HoweverΓand as convincingly argued by (Inoue 1992)Γthis relative disregard is no longer justifiedΓas consequence-finding has multiple applications. Inoue discusses finding the set of prime $\mathcal{L}_T$-implicates (what he calls "characteristic clauses") for certain task-specific languages $\mathcal{L}_T$ (e.g. involving only certain predicates)Γand its application to explanationΓabductionΓdiagnosisΓand non-monotonic reasoning. SFK resolution can be used for all of them.

A second set of more recent applications of consequence-finding arise from *knowledge compilation* (KC) (see (Cadoli and Donini 1997) for a review)Γwhere the knowledge base is preprocessed in an "offline phaseΓ" so as to make "online" query answering tractable. In *exact* KCΓall queries can be answered tractably after compilation; (del Val 1994Γ Marquis 1995) are examples of exact KC using full full consequence finding. In the main version of *approximate* KC (Selman and Kautz 1991ΓSelman and Kautz 1996)Γon the other handΓthe theory is approximated by a theory formulated in some target sublanguage $\mathcal{L}_T$. This approximation is equivalent to the set of all $\mathcal{L}_T$-consequences of the input theoryΓso *provided* inference in $\mathcal{L}_T$ is tractableΓall queries in $\mathcal{L}_T$ can be answered tractably.

Our contribution to KC is threefold. We show how to systematically obtain polynomial size approximations (e.g. obtain all implicates smaller than some constant)Γ thus repairing a weak spot of Selman and Kautz's LUB framework. SecondΓwe define an incremental version of the exact KC algorithm of (del Val 1994) using kernel resolutionΓwith support for lazy and query-directed compilation. FinallyΓby combining our exact KC algorithm with SFK resolutionΓwe provide a way to ensure tractable answers for any target language $\mathcal{L}_T$Γ*independently of whether inference in $\mathcal{L}_T$ is tractable.*

The structure of this paper is as follows. In section 1 we introduce kernel and SFK resolutionΓand prove them sound and complete for ($\mathcal{L}_T$)-implicate finding in predicate calculus. Section 2 discusses propositional

search strategies—in particular "bucket elimination" and incremental saturation—suitable for any form of SFK resolution. Section 3 introduces various applications of SFK-resolution: computing characteristic clauses and LUB approximations—polynomial size approximations—exact KC—and compilation for restricted target languages.

Kernel resolution

Kernel resolution generalizes Tison's (1967) well-known algorithm for computing prime implicates. Tison's algorithm processes propositional variables in order—and is based on the idea that once all resolvents on a given literal have been computed—no further resolutions on that literal are needed.

Procedure Tison-PI(Σ)
$PI := \Sigma$;
for each variable x_i in $[x_1..x_n]$ **do**
  for each pair of clauses $x_i C, \overline{x_i} D \in PI$ **do**
    if their resolvent $E = C \cup D$ is not a tautology **and**
      E is not subsumed by any clause of PI
    then remove from PI any clause F subsumed by E
      $PI := PI \cup \{E\}$
return PI

We can view Tison's algorithm as engaged in an implicit proof enumeration task. For each clause C it generates—we can trace back the deduction of C through its parent and ancestor clauses—obtaining the usual resolution tree for C. Seeing the method in this way allows us to separate the proof finding task from the particular enumeration technique implicit in Tison's method.

Specifically—in Tison's method a clause C can be resolved exactly upon literals which are later in the ordering ("larger") than the literal resolved upon to obtain C (or upon any literal if C is an input clause). We call this "usable" set of literals—the *kernel* of C—which we denote $k(C)$. The remaining literals (those smaller than the literal resolved upon) we call the *skip*—denoted $s(C)$—because neither they nor any of their descendants will be resolved upon. We write a clause C as an ordered pair $A[B]$ where $C = A \cup B$—$s(C) = A$ and $k(C) = B$.

As an example—with the obvious ordering—if $x_1\overline{x_2}$ and x_2x_3 are clauses in the input then we would obtain the clause x_1x_3. Because x_1 has already been processed by Tison method when the clauses are resolved upon x_2 and $\overline{x_2}$—we know that x_1x_3 will never be resolved upon x_1—and may be resolved upon x_3. We record this information by writing the clause as $x_1[x_3]$.

Kernel resolution deductions are simply resolution deductions in which every literal resolved upon is in its clause's kernel—just as in Tison's method. We extend this definition slightly to deal with the first order case directly. Formally:

Definition 1 *Let $p_1, \ldots, p_n$ be an ordering of the predicate symbols. We say that p_i has* ordinal *i. Let l_i and l_j literals with predicate symbols p_i and p_j respectively. We say that l_j is larger or equal than l_i iff $i \leq j$.*

Definition 2 *A kernel deduction of a clause C from a set of clauses Σ is a sequence of clauses of the form $S_1[K_1], \ldots, S_n[K_n]$ such that:*

1. *$C = S_n \cup K_n$*
2. *For every k, $S_k \cup K_k$ is not a tautology.*
3. *For every k, either:*
   (a) *$K_k \in \Sigma$ and $S_k = \emptyset$ (input clause); or*
   (b) *$S_k[K_k]$ is a factor of $S_j[K_j]$, $j < k$;*[1] *or*
   (c) *$S_k \cup K_k$ is a resolvent of two clauses $S_i \cup K_i$ and $S_j \cup K_j$ $(i, j < k)$ such that:*
      i. *the literals resolved upon to obtain $S_k \cup K_k$ are in, respectively, K_i and K_j; and*
      ii. *K_k is the set of all literals of $S_k \cup K_k$ which are larger or equal than the literals resolved upon, according to the given ordering, and S_k is the set of smaller literals.*

The kernel index *of $S_k[K_k]$ is the ordinal i of the predicate p_i resolved upon to obtain $S_k[K_k]$ (or 0 if $S_k[K_k]$ is an input clause). The* kernel depth *of the deduction is the kernel index of its conclusion C.*

The "clausal meaning" of a kernel clause $S_k[K_k]$ is simply given by $S_k \cup K_k$; the crucial aspect is that resolutions are only permitted upon kernel literals—condition 3.b.ii—and that the literal resolved upon partitions the literals of the resolvent into those smaller (the skip) and those larger or equal (the kernel) than the literal resolved upon—condition 3.b.iii. The same partition can be induced from the kernel index.

Definition 3 *A clause D θ-subsumes a clause C just in case there exists a substitution σ such that $D\sigma \subseteq C$ and D has no more literals than C.*

Definition 4 *We write $\Sigma \vdash_k C$ iff there exists a kernel deduction of a clause D that θ-subsumes C from Σ.*

Example 1 Suppose $\Sigma_1 = \{C_1, C_2, C_3, C_4\}$ is as depicted below. The left column shows the computation of Tison–PI(Σ_1) under the natural ordering—with each step explained in the middle column.

$C_1 = x_1x_2$ $C_2 = \overline{x_2}x_3$ $C_3 = \overline{x_1}x_4$ $C_4 = \overline{x_3}x_4$	input Σ_1	$C'_1 = [x_1x_2]_{(0)}$ $C'_2 = [\overline{x_2}x_3]_{(0)}$ $C'_3 = [\overline{x_1}x_4]_{(0)}$ $C'_4 = [\overline{x_3}x_4]_{(0)}$
$C_5 = x_2x_4$	Resolve(C_1, C_3, x_1)	$C'_5 = [x_2x_4]_{(1)}$
$C_6 = x_1x_3$	Resolve(C_1, C_2, x_2)	$C'_6 = x_1[x_3]_{(2)}$
$C_7 = x_3x_4$	Resolve(C_2, C_5, x_2)	$C'_7 = [x_3x_4]_{(2)}$
$C_8 = \overline{x_2}x_4$	Resolve(C_2, C_4, x_3)	$C'_8 = \overline{x_2}[x_4]_{(3)}$
$C_9 = x_1x_4$	Resolve(C_4, C_6, x_3)	$C'_9 = x_1[x_4]_{(3)}$
$C_{10} = x_4$	Resolve(C_4, C_7, x_3)	$C'_{10} = [x_4]_{(3)}$

After obtaining C_{10}—all clauses containing x_4 are deleted—since they are now subsumed. The resulting

[1] There cannot be two literals with the same predicate symbol one of which is in S_j and the other in K_j. Thus there is no possible ambiguity in defining the factoring operation.

set of prime implicates is $\{C_1, C_2, C_6, C_{10}\}$. We can describe this process as computing kernel deductionsΓas shown in the right columnΓwhere the rightmost numbers are kernel indexes. ■

Proposition 1 *Every resolution deduction carried out by Tison's method from a set of ground clauses Σ can be written as a kernel deduction from Σ.*

ThusΓusing the completeness of Tison's method for computing prime implicatesΓwe immediately obtain:

Corollary 2 (Ground soundness/completeness) *Ground kernel deductions are sound and complete for consequence-finding, that is, for any ground clause C and set of ground clauses Σ, $\Sigma \models C$ iff $\Sigma \vdash_k C$.*

Using standard lifting techniques from (Slagle *et al.* 1969ΓMinicozzi and Reiter 1972) we can show:

Theorem 3 (FOL soundness/completeness) *Let Σ be a set of clauses, C a clause. $\Sigma \models C$ iff $\Sigma \vdash_k C$.*

In additionΓfirst order kernel resolution satisfies:

Proposition 4

1. *Every descendant D of a clause C is such that $[s(C)]\sigma \subseteq s(D)$ for some substitution σ.*
2. *Descendants of skipped literals are never resolved upon.*

Property 4.1 is crucial to SFK-resolutionΓthe generalization of kernel resolution to focus on restricted target languages which is discussed next. As a special caseΓground skipped literals of a clause stay in all its descendants. Property 4.2 is crucial for applications in knowledge compilation.

Skip-filtered kernel resolution

As already suggestedΓwe are often interested in finding only the consequences of the input theory in some restricted "target language" $\mathcal{L}_T$Γrather than all consequences. Skip–filtered kernel resolutionΓor SFK resolution for shortΓis a restriction of kernel resolution which achieves exactly this.

Definition 5 *An $\mathcal{L}_T$-SFK resolution deduction of C from Σ is a kernel deduction of C from Σ satisfying the following additional restriction:*

*4. For every k: S_k or some factor of S_k is in $\mathcal{L}_T$ (in which case we say that $S_k[K_k]$ is $\mathcal{L}_T$-*acceptable*).*

We write $\Sigma \vdash_k^{\mathcal{L}_T} C$ when there is a $\mathcal{L}_T$-SFK resolution deduction from Σ of some D which θ-subsumes C.

We will consider only target languages which are *closed under θ-subsumption (c.u.s.)*Γi.e. such that for any $C \in \mathcal{L}_T$Γif D θ-subsumes C then $D \in \mathcal{L}_T$. If a language $\mathcal{L}_S$ is not c.u.sΓone can always *close it off*Γby defining the language $cus(\mathcal{L}_S) = \{D \in \mathcal{L} \mid \exists C \in \mathcal{L}_S : D \ \theta\text{-subsumes } C\}$.

There are many languages which are c.u.sΓe.g. the set of Horn clausesΓsets of clauses which use only some subset of the predicatesΓthe set of clauses with less than k literals. For many othersΓclosing off is a useful operation; e.g. consider $cus(\{C\})$Γwhere C is a single clauseΓor $cus(\mathcal{L}_S)$ when $\mathcal{L}_S$ is some subset of ground clauses of $\mathcal{L}$.

Theorem 5 (FOL soundness/completeness for $\mathcal{L}_T$) *Suppose $\mathcal{L}_T$ is c.u.s. $\forall C \in \mathcal{L}_T$: $\Sigma \models C$ iff $\Sigma \vdash_k^{\mathcal{L}_T} C$.*

Corollary 6 *Let $SFK_{\mathcal{L}_T}(\Sigma)$ be the set of unsubsumed clauses generated by any exhaustive form of SFK resolution. $PI_{\mathcal{L}_T}(\Sigma) = SFK_{\mathcal{L}_T}(\Sigma) \cap \mathcal{L}_T$.*

Note that c.u.s. implies that the empty clause $\square$ is in $\mathcal{L}_T$. There are therefore two extreme special cases of this theorem. If $\mathcal{L}_T = \mathcal{L}$ is the full languageΓthen SFK-resolution is complete for consequence finding; and if $\mathcal{L}_T = \mathcal{L}_\square = \{\square\}$ then SFK resolution is a satisfiability methodΓwhere the only "interesting" potential implicate of Σ is the empty clause.

Subsumption

Deletion of subsumed clauses is a very powerful pruning technique for any search strategy. SFK-resolution remains complete under the following *replacement policy*: if C θ-subsumes DΓthen delete DΓand update C's kernel index to be the minimum of the indexes of C and D; semanticallyΓthis amounts to possibly enlarging the kernel of the subsumerΓadding literals that were already skipped. For certain search strategies (see BE below)Γwe can use a stronger *deletion policy*Γby which we just delete clauses as soon as they become subsumed (as usualΓchecking for forward subsumption before backward subsumption); but this policy destroys completeness in other cases (e.g. IS below).

Propositional search strategies

Kernel resolution differs from other consequence finding procedures in that it does not prejudge the search strategy. Any exhaustive search of the space of SFK deductions will doΓincluding standard saturation methods (Chang and Lee 1973) and lazy evaluation strategies. We introduce in this section two propositional exhaustive search procedures: "Bucket eliminationΓ" a generalization for SFK resolution of the methods of (Dechter 1998); and an alternative method of "incremental saturation" which processes input clauses incrementallyΓ and is specially suited for situations where the input is initially only partially specified.

All the results in this section assume $\mathcal{L}_T$ is c.u.s. (without loss of generalityΓas seen above)Γand apply only to the ground case. The reason for the latter restriction is that in FOL both methods fail to guarantee *fairness*Γi.e. that every derivable clause is eventually derivedΓand are therefore incomplete for FOL.

Indexing functions

Before we present both methodsΓwe will discuss the basic data structures we use. Consider an ordered array of buckets $b[x_i]$Γone for each propositional variable

$x_1, \ldots, x_n$. Each bucket $b[x_i]$ contains clauses containing occurrences of x_iΓaccording to some *indexing function* $I_{\mathcal{L}_T}$. $I_{\mathcal{L}_T}$ is a function that maps each clause into a subset of the clause's variables; it determines which clauses go to which buckets by the rule $C \in b[x_i]$ iff $x_i \in I_{\mathcal{L}_T}(C)$.

Definition 6 ($\mathcal{L}_T$-acceptable indexing) *An indexing function $I_{\mathcal{L}_T}$ is $\mathcal{L}_T$-acceptable iff $x_i \in I_{\mathcal{L}_T}(C)$ whenever resolving C on x_i can generate some $\mathcal{L}_T$-acceptable resolvent as a child.*

The long version of this paper discusses indexing functions in more detailΓas the complexity of search is a direct function of bucket size. For any $\mathcal{L}_T$Γwe will use the $\mathcal{L}_T$-acceptable function $I^o_{\mathcal{L}_T}(C) =$ {kernel variables of the largest prefix $l_1 \ldots l_k$ of C s.t. $l_1 l_2 \ldots l_{k-1} \in \mathcal{L}_T$}Γ where C is assumed sorted in ascending kernel order.

Bucket elimination

Bucket elimination (BE) processes buckets $b[x_1]$Γ$b[x_2]$Γ ...Γin order. It computes in step i all resolvents on x_i obtained from clauses in $b[x_i]$; among theseΓit selects the $\mathcal{L}_T$-acceptable resolventsΓand adds them to their "corresponding" buckets by calling $I_{\mathcal{L}_T}$. Variable x_i is "eliminated" when finished with its bucketΓin the sense that it will never be revisited: no further resolutions upon x_i are needed.

BE exhaustively explores the space of SFK deductions in breadth-first search relative to the *kernel depth* of SFK-derivations (see Def. 2). This allows us to show:

Theorem 7 (Ground completeness of BE) *Ground bucket elimination with any $\mathcal{L}_T$-acceptable indexing function $I_{\mathcal{L}_T}$ is a terminating and exhaustive search strategy for SFK resolution and therefore for propositional $\mathcal{L}_T$-consequence finding.*

Furthermore, ground BE remain complete under the deletion policy *for subsumed clauses.*

Consider now the two extreme cases of theorem 5 under the light of theorem 7. FirstΓ when $\mathcal{L}_T = \mathcal{L}$ then every possible resolvent is $\mathcal{L}$-acceptableΓ which suggests the implementation $I_{\mathcal{L}_T}(C) = I_{\mathcal{L}}(C) = k(C)$. The resulting bucket elimination procedure corresponds exactly to Tison-PIΓ where clauses can be resolved upon any of their kernel literals. On the other handΓ if $\mathcal{L}_T = \mathcal{L}_\square = \{\square\}$ then $I_{\mathcal{L}_T}(C) = I_{\mathcal{L}_\square}(C) =$ {smallest kernel variable of C}Γsince resolving on any other kernel variable cannot yield any $\mathcal{L}_\square$-acceptable resolvents. Bucket elimination with this indexing function is identical to *directional resolution* (DR)ΓDechter and Rish's (1994) version of the original Davis-Putnam (1960) satisfiability procedure Because DR is a relatively efficient satisfiability method for semi-structured problemsΓit is encouraging that it is at the same time the simplest form of SFK resolution. In the long paper we'll reinforce this point by providing a wide class of *realistic* problems for which DR is tractable.

Example 2 Let $\Sigma_2 = \{x_1x_2x_3, x_1x_2\overline{x_3}, x_1\overline{x_2}x_3, x_1\overline{x_2x_3}\}$Γ with the natural ordering. Satisfiability of Σ_2 can be determined without computing a single resolvent by DRΓi.e. by $\mathcal{L}_\square$-BEΓas $I_{\mathcal{L}_\square}$ leaves all buckets empty except $b[x_1]$Γwhich does not generate any resolvent.

Note that the unit implicate x_1 is not in the output $DR(\Sigma_2)$ of DR. FurthermoreΓthe obvious attempt to obtain a *tractable* answer to the query whether $\Sigma \models x_1$ failsΓas $DR(\Sigma_2) \cup \{\overline{x_1}\}$ is not unit refutable.

Let now $\mathcal{L}_T = \mathcal{L}_1$ consist of all clauses with at most one literal. The indexing function $I^o_{\mathcal{L}_1}(C)$ selects the kernel variables among the first two literals of C. In this caseΓinitially $b[x_1] = b[x_2] = \Sigma_2$. $\mathcal{L}_1$-BE yields the clauses $x_1[x_3]$ and $x_1[\overline{x_3}]$ when processing $b[x_2]$. Both clauses are indexed in $b[x_3]$Γso when processing x_3 we obtain $x_1[\ ]$Γthe only $\mathcal{L}_1$-implicate of Σ_2. ■

Incremental saturation

A different kind of strategy is "incremental saturationΓ" where each input clause C is processed in turnΓ until the store of clauses is "saturated" under $\mathcal{L}_T$-SFK resolutionΓthat isΓuntil no unsubsumed $\mathcal{L}_T$-acceptable resolvents can be generated from the available clauses.

Incremental saturation (IS) adds a new input clause C to a $\mathcal{L}_T$-saturated set of clauses Σ as follows. We use a set of parallel buckets $bn[x_i]$ for each x_iΓto index the newly generated clauses separately. Begin by indexing C with $I_{\mathcal{L}_T}$ in the bn buckets. Then process buckets one by oneΓstarting from the bucket $bn[x_j]$ of the smallest $x_j \in I_{\mathcal{L}_T}(C)$. For each variable $x_k \geq x_j$Γcompute all resolvents that can be obtained by resolving clauses of $bn[x_k]$ with clauses of $bn[x_k]$ and $b[x_k]$Γand index the $\mathcal{L}_T$-acceptable resolvents among them in the bn buckets.

The procedure is thus driven by the parallel buckets; we never resolve together two clauses in $b[x_i]$Γi.e. two "old" clauses. When finishedΓthe parallel buckets should be emptied into the original bucketsΓto prepare the mechanism for receiving more clauses.

IS can be generalized to add a set of clauses Γ by indexing all clauses of Γ in the bn arrayΓand starting with the smallest variable x_j with non-empty $bn[x_j]$.

Theorem 8 (Ground completeness of IS) *Ground incremental saturation with any $\mathcal{L}_T$-acceptable indexing function $I_{\mathcal{L}_T}$ is a terminating and exhaustive search strategy for SFK resolution and therefore for propositional $\mathcal{L}_T$-consequence finding.*

IS comes with two important benefits over BE. FirstΓ IS can incrementally compute $\mathcal{L}_T$-implicatesΓas new input clauses are received. There is no need to recompute everything anew if the input grows. SecondΓIS supports queries about the *new $\mathcal{L}_T$-implicates* of a theory after adding a clause (set). To show thisΓdefine the set of *new prime $\mathcal{L}_T$-implicates induced by adding Γ to Σ* as:

$$nPI_{\mathcal{L}_T}(\Sigma, \Gamma) = PI_{\mathcal{L}_T}(\Sigma \cup \Gamma) \setminus PI_{\mathcal{L}_T}(\Sigma).$$

ThusΓif Σ is $\mathcal{L}_T$-saturatedΓadding Γ to Σ with IS yields $nPI_{\mathcal{L}_T}(\Sigma, \Gamma)$Γby simply intersecting the (unsubsumed) newly generated clauses with $\mathcal{L}_T$.

But IS also has its drawbacks with respect to BE. FirstΓIS must use a weaker subsumption policy than

BE namely *replacement* instead of deletion. Second IS unlike BE is limited in its ability to dynamically order variables and to use global information about the structure of the theory in deciding an ordering.

It is not difficult to provide examples where IS requires exponentially more time and space than BE and conversely depending on the strategy used by each.

Applications

We discuss very briefly in this section some of the applications of SFK resolution.

1. Task-specific languages. There are many AI tasks e.g. abduction diagnosis reason-maintenance and non-monotonic reasoning which can be cast in terms of restricted sets of consequences of the input theory. Let V be any set of literals (typically closed under instantiation) and consider the language $\mathcal{L}_V = \{C \in \mathcal{L} \mid C \subseteq V\}$. (Inoue 1992) discusses multiple applications of the sets $PI_{\mathcal{L}_V}$ and $nPI_{\mathcal{L}_V}$ in these areas. For example abduction can be captured in terms of clauses which contain only "assumables;" circumscription in terms of clauses which involve only "fixed" predicates and positive "minimized" literals; diagnosis in terms of clauses containing positive "abnormality" literals.

2. LUB approximations. For a given target language $\mathcal{L}_T$ the $\mathcal{L}_T$-LUB (lowest upper bound) approximation of a first order theory Σ is the strongest theory $\Sigma_{lub} \subseteq \mathcal{L}_T$ entailed by Σ (Selman and Kautz 1991 Selman and Kautz 1996). Σ_{lub} is unique modulo logical equivalence and equivalent to $PI_{\mathcal{L}_T}(\Sigma)$ (see (del Val 1995 del Val 1996) for the general case). Thus SFK-resolution can be used to generate LUB approximations.

SFK-resolution however does not compute very concise LUBs. For vocabulary-based languages we can do better. Specifically suppose that we put all symbols in V last in the ordering and that we allow all literals whose symbol is in V. We can modify BE so that it stops after processing all symbols preceding the first symbol from V. Until then every resolvent with a non-empty skip is discarded (since it would contain symbols not in V) i.e. the algorithm works essentially as DR. The set of $\mathcal{L}_T$-clauses obtained by this procedure can be shown to be equivalent to the $\mathcal{L}_V$-LUB. The analogous procedure for other search strategies (including FOL) is straightforward: resolve only on the first literal of each clause provided it's symbol is not in V.

3. Polynomial size approximations. In computing LUB approximations it is important that Σ_{lub} has polynomial size. This can be achieved in the propositional case by further restricting the clauses of our target language to have no more than k-literals for a fixed k; in FOL one also needs to bound the complexity of terms. Let $\mathcal{L}_k$ be any such language. The algorithm available for this task GLUB-1 (Selman and Kautz 1996 del Val 1996) is quite unsatisfactory as it only forbids resolutions among pairs of clauses both of which are in $\mathcal{L}_k$. Thus e.g. for $\mathcal{L}_1$ only resolutions among pairs of unit clauses are forbidden and thus for satisfiable theories it is equivalent to resolution closure. SFK resolution is clearly much better.

4. Exact knowledge compilation. The task of exact knowledge compilation (del Val 1994) is to preprocess an input knowledge base Σ so that after the "off-line" compilation is completed every query can be answered in polynomial time with respect to the size of the compiled knowledge base. The idea is that the cost of compilation can be amortized over many queries.

We present here a version of the (ground) KC algorithm FPI_1 (del Val 1994). $\text{FPI}_1(\Sigma)$ computes $PI(\Sigma)$ with Tison's method storing in the compiled theory only some of these implicates specifically the kernel merge resolvents. A merge resolvent is a resolvent such that some of its literals called the merge literals occur in both parent clauses (and must therefore be "merged" in the resolvent). When some merge literals are in the resolvent's kernel we speak of a *kernel merge resolvent.*

The procedure IFPIA (Incremental Filtered Prime Implicates Algorithm) given below differs from FPI_1 only in the use of kernel deductions instead of Tison's method. IFPIA is compatible with any incremental search strategy for kernel deductions or with forms of lazy evaluation. In particular Σ *need to be given in its entirety before beginning execution.* If new clauses are added they can be processed just as with kernel deductions with whichever strategy we are using.

Procedure IFPIA(Σ)

1. Compute the closure under kernel resolution of Σ.
2. Return the unsubsumed clauses which are either: input clauses or kernel merge resolvents of subsumers of either.

Let $IFPIA(\Sigma)$ be the output of the algorithm. Using theorem 1 from (del Val 1994) and proposition 4.2 above we can easily obtain:

Theorem 9 *$IFPIA(\Sigma)$ is logically equivalent to Σ, and unit refutation complete. That is, for any clause C, $\Sigma \models C$ iff $IFPIA(\Sigma) \models C$ iff there exists a unit resolution refutation of $IFPIA(\Sigma) \cup \neg C$.*

5. KC for restricted languages. The completeness of IFPIA depends only on the completeness of the underlying consequence finding method namely kernel resolution. Accordingly if we replace kernel resolution by $\mathcal{L}_T$–SFK resolution we can ensure tractability just for the queries in $\mathcal{L}_T$. Formally let $\mathcal{L}_T$–IFPIA be as IFPIA but with $\mathcal{L}_T$–SFK resolution and let $IFPIA(\Sigma, \mathcal{L}_T)$ be its output.

Theorem 10 *$IFPIA(\Sigma, \mathcal{L}_T)$ is unit refutation complete with respect to $\mathcal{L}_T$ queries. That is, for any clause $C \in \mathcal{L}_T$, $\Sigma \models C$ iff $IFPIA(\Sigma\mathcal{L}_T) \models C$ iff there exists a unit resolution refutation of $IFPIA(\Sigma) \cup \neg C$.*

Note that the approach here is quite different from that of LUB approximations in the sense that $IFPIA(\Sigma, \mathcal{L}_T)$ need not be a subset of $\mathcal{L}_T$ nor be equivalent to the $\mathcal{L}_T$-LUB. And most crucially tractability of answers to $\mathcal{L}_T$-queries does no longer depend on the tractability of inference in $\mathcal{L}_T$.

Related work in consequence-finding

We have already mentioned directional resolution and Tison-PI as instances of SFK-BE. Bucket elimination is introduced in (Dechter 1998) as a uniform framework for dealing with a wide variety of inference problems. We extend BE for the special case of resolution on logical clauses by treating the indexing function and $\mathcal{L}_T$-acceptability criterion as parameters. Incremental saturationΓin turnΓis motivated in part by IPIAΓan alternative incremental version of Tison-PI introduced by (Kean and Tsiknis 1990). Neither IPIA nor most other propositional consequence finding methods have anywhere near the inferential and compilation capabilities of SFK resolution.

In contrast to the propositional caseΓrelatively few first order methods are known which are complete for consequence-finding. The ones known to us include: unrestricted resolution (Lee 1967); m.s.l. (mergeΓ subsumptionΓlinear) resolution (Minicozzi and Reiter 1972); and SOL (skipΓorderedΓlinear) resolution (Inoue 1992). Other methods are complete only for finding the consequences in some restricted language; for exampleΓpositive hyperresolution finds all positive clauses entailed by the input theory (Slagle *et al.* 1969). Of more direct relevance to this paper is SOL resolutionΓ which is also complete for restricted $\mathcal{L}_T$ languages.

We do not know of any implementation of SOL resolution for full consequence findingΓwhereas Tison's methodΓ$\mathcal{L}$-BEΓis well regarded in practice for this task (de Kleer 1992ΓForbus and de Kleer 1993). The goal-directedness and low memory usage of linear resolution are advantageous for theorem provingΓbut this is not so clear for consequence findingΓwhere the computed clauses are typically not discarded.[2]

Kernel resolution resembles ordered resolution (Basin and Ganzinger 1996) in the use of an ordering of atoms or A-ordering. Ordered resolution can be seen as a sophisticated first order version of directional resolutionΓ or more exactly of $\mathcal{L}_\Box - SFK$ resolution. Kernel resolution generalizes directional resolution for consequence findingΓwhile ordered resolution generalizes DR to use much more sophisticated ground orderings. Presumably both aspects are quite compatibleΓso it should be possible to merge these two generalizations.

Discussion

We have introduced kernel resolutionΓ as a flexible method to compute consequences in restricted sublanguagesΓand discussed some of its applications. In subsequent workΓwe will show that it provides a theoretical framework that allows for orders of magnitude improvements in some exact knowledge compilation approachesΓ and to allow the latter to be applied to compilation for restricted languages.

[2]SOL resolution involves a large amount of redundancy for consequence-finding: a) every input clause has to be considered as a top clause, which leads to many essentially identical proofs; b) all possible orderings of non-top clauses have to be considered; c) very limited deletion of subsumed clauses in linear resolution (Stickel 1992), which has a much larger potential impact in consequence-finding applications.

References

David Basin and Harald Ganzinger. Complexity analysis based on ordered resolution. In *LCS'96*, 1996.

Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. *AI Communications*, 10:137–150, 1997. Printed in 1998.

Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, 1973.

M. Davis and H. Putnam. A computing procedure for quantification theory. *J. of the ACM*, 7(3):201–215, 1960.

Johan de Kleer. An improved incremental algorithm for generating prime implicates. In *AAAI'92*, 1992.

Rina Dechter. Bucket elimination: A unifying framework for structure-driven inference. In *Learning and Inference in Graphical Models.* 1998.

Alvaro del Val. Tractable databases: How to make propositional unit resolution complete through compilation. In *KR'94*, 1994.

Alvaro del Val. An analysis of approximate knowledge compilation. In *IJCAI'95*, 1995.

Alvaro del Val. Approximate knowledge compilation: The first order case. In *AAAI'96*, 1996.

Ken Forbus and Johan de Kleer. *Building Problem Solvers.* The MIT Press, 1993.

Katsumi Inoue. Linear resolution for consequence-finding. *Artificial Intelligence*, 56:301–353, 1992.

Alex Kean and George Tsiknis. An incremental method for generating prime implicants/implicates. *Journal of Symbolic Computation*, 9:185–206, 1990.

R.C.T. Lee. *A Completeness Theorem and a Computer Program for Finding Theorems Derivable from Given Axioms.* PhD thesis, UC Berkeley, 1967.

Pierre Marquis. Knowledge compilation using theory prime implicates. In *IJCAI'95*, 1995.

Eliana Minicozzi and Raymond Reiter. A note on linear resolution strategies in consequence-finding. *Artificial Intelligence*, 3:175–180, 1972.

Bart Selman and Henry Kautz. Knowledge compilation using Horn approximations. In *AAAI'91*, 1991.

Bart Selman and Henry Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, March 1996.

J. R. Slagle, C. L. Chang, and R. C. T. Lee. Completeness theorems for semantic resolution in consequence finding. In *IJCAI'69*, 1969.

Mark E. Stickel. A Prolog technology theorem prover: Implementation by an extended Prolog compiler. *Journal of Theoretical Computer Science*, 104:109–128, 1992.

P. Tison. Generalized consensus theory and application to the minimization of boolean circuits. *IEEE Transactions on Computers*, EC-16:446–456, 1967.