# Creating Abstractions Using Relevance Reasoning

**Alon Y. Levy**
AT&T Bell Laboratories
AI Principles Research Department
600 Mountain Avenue, Room 2C-406
Murray Hill, NJ, 07974.
Email: levy@research.att.com

## Abstract

Reasoning with multiple levels of abstraction is a powerful method of controlling problem solving in complex domains. We consider the problem of simplifying a knowledge base by creating an abstraction that is tailored for a given set of queries. Our approach is based on associating formally an abstraction with some irrelevant detail that is removed from the knowledge base. We show how creating an abstraction and determining its utility amounts to automatically deciding which aspects of a representation are irrelevant to a query. As a result, we derive a general algorithm schema for automatically generating abstractions for a query. As an instance of the schema, we describe a novel algorithm for automatically abstracting a KB by projecting out relation arguments.

## Introduction

Abstraction is a pervasive phenomenon in human common sense reasoning and problem solving. From the early days of AI research, it was noted that if systems are going to reason effectively in complex domains, they too must be able to create automatically appropriate abstractions. This idea was the driving force of several early works (e.g., [Sacerdoti, 1974; Plaisted, 1981]) and has recently received renewed attention (e.g., [Ellman, 1992; Knoblock, 1990; Bacchus and Yang, 1992; Ellman, 1993]). The need for abstraction is rooted in the fact that a declarative representation is designed for a variety of queries and consequently, it is likely to be too detailed for any given query. Essentially, the idea proposed in these works is that instead of trying to solve a query with the given complex theory of the domain, a system should create a simpler, more abstract theory, and solve the query in that theory. Depending on the problem solving context, the abstract solution may suffice, or there may be an additional step of mapping the abstract solution back to a solution of the original problem.

The key issue in this approach is how to *automatically* create abstractions that are well suited for a given query (or set of queries). It is unreasonable to expect a representation designer to anticipate all possible queries and the abstractions that will be suited to each of them. For an abstraction to be useful it must reduce the cost of answering the query, i.e., the cost of creating the abstraction, solving the query in the abstract theory and mapping the solution back to the original solution (a process that may need to be iterated several times) should be less than solving the query with the original representation.

Intuitively, as noted in several works on abstraction and irrelevance-reasoning (e.g., [Giunchiglia and Walsh, 1992; Subramanian, 1989; Levy and Sagiv, 1993]), a *good* abstraction is one in which we remove from the theory knowledge that is *irrelevant* to the given query. If the detail removed is indeed irrelevant, then the solutions found in the simpler theory will map back to the original theory, and consequently, backtracking between abstraction levels will not be necessary. This paper makes these intuitions concrete by making a formal connection between irrelevance and abstractions and shows how to use it to automatically create abstractions. The key to our approach is that when we consider an abstraction, we articulate *what* is being removed from the theory in the process of abstraction. Deciding to use an abstract theory then amounts to deciding that the removed knowledge is indeed irrelevant and that removing it will yield a computationally simpler theory. The following simple example illustrates our approach.

**Example 1:** The following rules describe flight routes between cities in the U.S. The first and second arguments of *Flight* and *Route* denote the origin and destination, respectively. Their third arguments denote the costs of the flights, and the fourth arguments denote the airline. The fifth argument of *Route* denotes the number of legs in the route. The knowledge base also contains a set of ground atoms for the predicate *Flight*. Flight routes are composed using rules $r_2$ and $r_4$. Flight routes must always be on a single airline. Furthermore, we can only use a foreign airline

if the cost of the route is less than \$500. The atom $AirlineRoute(x, y, a, l)$ denotes that there is a route with $l$ legs from $x$ to $y$ that uses only airline $a$.

$r_1 : Flight(x, y, c, a) \wedge American(a) \Rightarrow Route(x, y, c, a, 1)$
$r_2 : Flight(x, z, c_1, a) \wedge Route(z, y, c_2, a, l) \wedge$
$American(a) \Rightarrow Route(x, y, c_1 + c_2, a, l + 1)$
$r_3 : Flight(x, y, c, a) \wedge \neg American(a) \wedge (c < 500) \Rightarrow$
$Route(x, y, c, a, 1)$
$r_4 : Flight(x, z, c_1, a) \wedge Route(z, y, c_2, a, l) \wedge$
$\neg American(a) \wedge (c_1 + c_2 < 500) \Rightarrow$
$Route(x, y, c_1 + c_2, a, l + 1)$
$r_5 : Route(x, y, c, a, l) \Rightarrow AirlineRoute(x, y, a, l)$

Suppose we want to find whether there is a flight route between two cities $a$ and $b$ with $l$ legs on United Airlines (i.e., find whether $AirlineFlight(a, b, UA, l)$ is entailed by the KB). Since United is an American airline, the cost of the route is irrelevant to the query, and we can abstract the representation by projecting out the cost arguments from the relations *Flight* and *Route*. Intuitively, the irrelevance can be established by observing that the cost arguments play no role in the rules that are relevant to the query (i.e., the rules $r_1, r_2$ and $r_5$). In contrast, if the query would consider a foreign airline, the costs would be relevant because they impose additional constraints in $r_3$ and $r_4$. Therefore, we can rewrite our rules as follows:

$r'_1 : Flight'(x, y, a) \wedge American(a) \Rightarrow Route'(x, y, a, 1)$
$r'_2 : Flight'(x, z, a) \wedge Route'(z, y, a, l) \wedge American(a) \Rightarrow$
$Route'(x, y, a, l + 1)$
$r'_5 : Route'(x, y, a, l) \Rightarrow AirlineFlight(x, y, a, l)$

Rules $r_3$ and $r_4$ are irrelevant to the query and are therefore removed in the abstract KB. We also project out the third argument in the ground atoms of *Flight*. The resulting KB may yield a significantly smaller search space. For example, consider the difference between the rules $r_2$ and $r'_2$. In rule $r_2$, if we fail to join a ground atom $Flight(x, z, c_1, a)$ with a ground atom $Route(z, y, c_2, a)$, a backward chainer may still try to join the atom $Flight(x, z, c'_1, a)$ with an appropriate atom of *Route* for every value $c'_1$ it finds, and will fail on all of them. In contrast, rule $r'_2$ will not try other costs for the same flight route.[1] Furthermore, solutions including foreign flights will be ignored completely. ∎

In the next section we formally define the notions of irrelevance (e.g., we define what it *means* for an argument of a relation to be irrelevant to a query) and show how irrelevance claims provide a logical justification for creating abstractions suited for a set of queries. As a result, we derive a general algorithm schema for automatically generating abstractions for a query, which is based on identifying aspects of the representation that are irrelevant to a query. As an instance of the schema, we describe a novel algorithm for automatically abstracting a theory by projecting relation arguments. Finally, we argue for the advantages of viewing abstractions as a problem of irrelevance reasoning.

[1] Although in some simple cases these repetitions can be eliminated by employing some method of dependency directed backtracking, such methods will not be as general as projecting out arguments and will also have additional costs associated with maintaining the dependencies.

## Irrelevance and Abstractions

In our discussion, we assume that our domain is represented by a knowledge base (KB) $\Delta$ of clauses. We denote by $\mathcal{R}$, $\mathcal{O}$ and $\mathcal{F}$ the set of predicate symbols, object constants and function symbols used in $\Delta$, respectively. For readability, in our examples we write the clauses as rules, whenever possible. The meaning of the clauses are given via interpretations in which they are satisfied. An interpretation is a mapping from the symbols in $\Delta$ to our conceptualization of the domain. It maps elements of $\mathcal{O}$ to objects in our domain and elements of $\mathcal{R}$ and $\mathcal{F}$ to relations and functions on our domain. A *model* of $\Delta$ is an interpretation that satisfies all the clauses in $\Delta$. Intuitively, the set of models of $\Delta$ represents the possible states of the domain that we consider possible, given the constraints expressed in the clauses of $\Delta$.

In our discussion of irrelevance we will consider the possible derivations of a query formula from the KB. For the purpose of our discussion, we assume that inferences are made by the resolution rule of inference. A resolution proof $D$ of a clause $C_0$ can be viewed as a tree, in which the root is the derived clause, and the children of a clause $C$ are the clauses that were resolved to obtain $C$. We denote the set of leaves of the tree of $D$ by $Base(D)$. The set $Base(D)$ represents a "support set" for $C_0$ in $\Delta$. It should be emphasized that although we use resolution in our discussion, the results can easily be applied to other sound inference rules.

We begin by defining the meaning of *irrelevance claims*, i.e., claims stating that a *subject* $s$ is irrelevant to a query $q$, w.r.t. a KB $\Delta$. Previous work on irrelevance (e.g., [Subramanian, 1989; Levy and Sagiv, 1993]) considered formal definitions of irrelevance for the case where $s$ is a clause (or set of clauses). In order to use irrelevance to justify abstractions, we need to define irrelevance of other subjects. For instance, in Example 1, we based our abstraction on the irrelevance of predicate arguments. Other irrelevance subjects are shown in Table 1.

Intuitively, a subject is irrelevant to a query if it can be removed from our conceptualization of the domain without affecting our ability to answer the query correctly. For instance, for the query described in Example 1, we can simplify our conceptualization by projecting out the cost column from the relations corresponding to flights and routes, and we would still be able to answer the query (whereas it would not be adequate for queries involving foreign airlines). However, in formalizing our intuition we must take into account that we cannot reason directly with our conceptualiza-

| | Abstraction mapping $s^f$ | Intended semantics $s^I$ |
|---|---|---|
| Predicate abstraction | Replace occurrences of $P_1, \ldots, P_n$ by $P$ | $P = P_1 \cup \ldots \cup P_n$ |
| | Replace occurrences of $P_1, \ldots, P_n$ by $P$ | $P = P_1 \cap \ldots \cap P_n$ |
| Object abstraction | $a_1, \ldots, a_n \rightarrow a$ | $P(a)$ iff $P(a_1) \wedge \ldots \wedge P(a_n)$ |
| Function abstraction | $f \rightarrow f'$ | $f'$ is an approximation of $f$ |

Table 1: Example irrelevance subjects

tion of the domain, but only with clauses in the KB that represent a set of intended models. Moreover, as noted by in [Subramanian, 1989], a purely model-theoretic account of irrelevance will not capture our intuitions about the notion. Finally, since our main goal in defining irrelevance is to provide a basis for automatically creating abstractions, we will consider definitions that involve the actual clauses in the KB.

Formally, an irrelevance subject $s$ is a pair $(s^f, s^I)$ specifying a syntactic *abstraction mapping* [Plaisted, 1981], $s^f$, on clauses and the intended mapping on interpretations $s^I$ (see Table 1). The mapping $s^I$ represents the simplification we intend to make to the conceptualization of the domain via the abstraction (see [Nayak and Levy, 1994] for a more detailed account of such simplifications).[2] As an example, consider a subject which is a set of predicate arguments. We denote such a subject by a list of pairs $(q_i, n_i)$, where $q_i$ is a predicate and $n_i$ is an integer less or equal to the arity of $q_i$. In our example, the irrelevant arguments are $\{(Flight, 3), (Route, 3)\}$. The mapping $s^f$ would map the literals of the form $Flight(x, y, c, a)$ to the literal $Flight(x, y, a)$ (and likewise for *Route*). The mapping $s^I$ would map the relation denoted by *Flight* to the relation resulting from projecting out its third column.

The mapping $s^f$ is defined on literals and extended in the natural way to clauses. Following [Plaisted, 1981], we require that $s^f$ satisfy the following restrictions: (1) if $L$ is a literal, then $s^f(\neg L) = \neg s^f(L)$, and (2) if a clause $C$ subsumes $D$, then $s^f(C)$ subsumes $s^f(D)$.

Applying the mapping $s^f$ to all the clauses in $\Delta$ may result in an inconsistent theory. Therefore, we will apply $s^f$ only to clauses that are *independent* of $s$, as we define below. The notion of independence will also form the basis for our definition of irrelevance.

**Definition 1:** *Let $\Delta$ be a knowledge base and $s = (s^f, s^I)$ be an irrelevance subject. A clause $\phi$ is independent of $s$ if for any interpretation $I$:*

$$I \models \Delta \Longrightarrow s^I(I) \models s^f(\phi).$$ ∎

Intuitively, a clause $\phi$ is independent of $s$ if $s^f(\phi)$ does not decrease the set of possible models, and therefore does not enable us to derive conclusions that did not follow from the original KB. For example, the rule $r_2$ is independent of the predicate arguments $\{(Flight, 3), (Route, 3)\}$, but it is not independent of $\{(Flight, 4), (Route, 4)\}$. To see this, consider the interpretation $I_1$ and its corresponding interpretation $s^I(I_1)$:

Flight: $\{(a, b, 100, UA), (b, c, 150, NW)\} \xrightarrow{s^I}$
$\{(a, b, 100), (b, c, 150)\}$

Route: $\{(a, b, 100, UA, 1), (b, c, 150, NW, 1)\} \xrightarrow{s^I}$
$\{(a, b, 100, 1), (b, c, 150, 1)\}$

American: $\{UA, NW\} \xrightarrow{s^I} \{UA, NW\}$.

While the interpretation $I_1$ satisfies $r_2$, $s^I(I_1)$ does not satisfy the rule resulting from projecting out $\{(Flight, 4), (Route, 4)\}$ from $r_2$:

$r_2'' : Flight(x, z, c_1) \wedge Route(z, y, c_2, l) \wedge \neg American(a) \Rightarrow$
$Route(x, y, c_1 + c_2, l + 1)$

which enables us to derive the incorrect conclusion $Route(a, c, 250, 2)$. Based on the notion of independence, we define irrelevance as follows:

**Definition 2:** *Let $\Delta$ be a KB and $s = (s^f, s^I)$ be an irrelevance subject and $\psi$ be a query.*

*The subject $s$ is* weakly irrelevant *to $\psi$ (denoted by $WI(s, \psi, \Delta)$) if there is* some *derivation $D$ of $\psi$ such that all the clauses in $Base(D)$ are independent of $s$. The subject $s$ is* strongly irrelevant *to $\psi$, (denoted by $SI(s, \psi, \Delta)$) if for* all *derivations $D$ of $\psi$, all the clauses in $Base(D)$ are independent of $s$.* ∎

Note that these definitions can be viewed as instances in the space of definitions of irrelevance proposed in [Levy and Sagiv, 1993]. It is more useful to state and derive irrelevance claims that hold with respect to a set of knowledge bases. Formally, if $\Sigma$ is a set of KBs, we define $WI(s, \psi, \Sigma)$ to hold if $WI(s, \psi, \Delta)$ for every $\Delta \in \Sigma$ (and similarly for $SI$).

Consider Example 1, where $\Sigma$ is the set of KBs consisting of the rules $r_1$–$r_5$ and *some* set of ground unit clauses of the predicate *Flight*. The predicate arguments $s = \{(Flight, 3), (Route, 3)\}$ are strongly irrelevant to ground queries that are instances of $q = American(a) \wedge AirlineRoute(x, y, a, l)$, because, as the query-tree in Figure 1 shows, only the rules $r_1$, $r_2$ and $r_5$ and ground unit clauses can appear in derivations of the query and these are all independent of $s$ (note that ground unit positive clauses are independent of any set of predicate arguments). If we add the rule

$r_6 : Flight(x, z, c_1, a) \wedge Route(z, y, c_2, a, l) \wedge$
$(c_1 + c_2 < 500) \Rightarrow Route(x, y, c_1 + c_2, a, l + 1)$

[2] Note that $s^I$ is not uniquely determined by $s^f$, as shown by the first two entries in Table 1.

which is not independent of $s$, then $s$ would be only weakly irrelevant to the query (since $r_6$ is redundant and therefore, if there is a derivation of the query, there will be a derivation with $r_6$ and one without it).

Our definitions enable us to give a logical justification for creating abstractions. The following theorem states that if $s$ is weakly irrelevant to a query, then the KB resulting from abstracting all the independent clauses will be sufficient for answering the query.

**Theorem 3:** *Let $\Delta$ be a knowledge base and let $s = (s^f, s^I)$ be an irrelevance subject such that $s^f$ is an abstraction mapping. Let $\Delta_s$ be the KB defined by:*

$$\Delta_s = \{s^f(\phi) \mid \phi \in \Delta \text{ and } \phi \text{ is independent of } s\}.$$

*Let $q$ be a query and suppose $WI(s, q, \Delta)$ holds. Then*

$$\Delta \vdash q \Longrightarrow \Delta_s \vdash s^f(q)$$

*and, if $s^f(q) = q$ then*[3]

$$\Delta_s \models q \Longrightarrow \Delta \models q. \blacksquare$$

Note that the second part of the theorem does not depend on the inference mechanism used. Furthermore, if our inference rules are complete (e.g., refutation resolution), then the above theorem implies

$$\Delta \vdash q \Longleftrightarrow \Delta_s \vdash q \text{ and } \Delta \models q \Longleftrightarrow \Delta_s \models q.$$

**Proof sketch:** The first half of the theorem follows from Plaisted [Plaisted, 1981]. For the second half, suppose $\Delta_s \models q$ and let $I$ be a model of $\Delta$, i.e., $I \models \Delta$. We need to show that $I \models q$. By the definition of independence and the construction of $\Delta_s$, we get $s^I(I) \models \Delta_s$ and therefore, $s^I(I) \models q$. However, since $I$ and $s^I(I)$ are identical for the symbols appearing in $q$, it follows that $I \models q$. $\blacksquare$

The importance of Theorem 3 is that it gives a logical justification for creating an abstraction that is especially fit for the specific set of queries. As we describe in the next section, it also gives us a method for developing algorithms for automatically creating abstractions. It is important to note that Theorem 3 provides a justification for using a specific abstract KB, namely $\Delta_s$. One advantage of $\Delta_s$ is that it can be efficiently generated from the original theory. However, we can sometimes add clauses to $\Delta_s$ to obtain a stronger theory (as done in [Tenenberg, 1990]), and therefore lose less information in the abstract KB. We do not discuss this extension here.

## Automatically Creating Abstractions

The importance of the formulation presented in the previous section is that we can now clearly address the problem of automatically creating abstractions for a given set of queries, by automatically deriving irrelevance claims. Specifically, to use Theorem 3 we need to automatically derive claims of the form $WI(s, q, \Delta)$. One way of deriving such a claim is to find a subset $\Gamma$ of $\Delta$, such that $q$ will necessarily have a derivation that does not include clauses in $\Gamma$, and such that the clauses in $\Delta - \Gamma$ are all independent of $s$. It follows from the previous section that we can abstract $\Delta$ by $s^f(\Delta - \Gamma)$. Therefore, a general method for automatically creating abstractions has two steps (1) automatically find $\Gamma$ (i.e., a set of irrelevant clauses) and (2) automatically detect independence of a clause (i.e., the independence of $\Delta - \Gamma$ from $s$). These steps are discussed in the following sections.

[3]Note that this restriction effectively means that the irrelevance subject does not appear explicitly in the query.

## Determining Irrelevance

Methods for detecting irrelevance of clauses to a query are described in [Levy and Sagiv, 1993; Levy, 1993] and [Subramanian, 1989]. For example, we can use the *query-tree* [Levy and Sagiv, 1992] (shown in Figure 1) to show that only that the rules $r_1, r_2$ and $r_5$ and ground unit clauses involving American airlines can be used in derivations of the query. Consequently, all other clauses (including the rules $r_3$ and $r_4$) are irrelevant to the query.

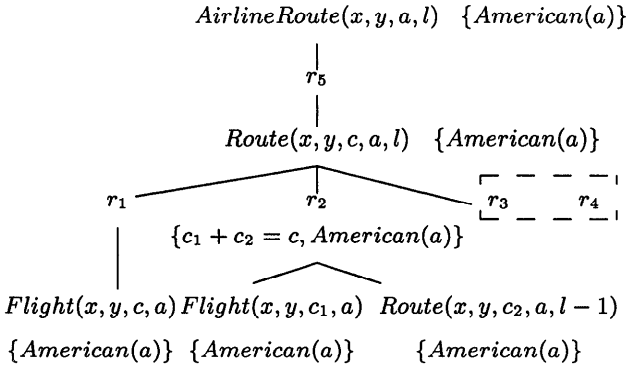


Figure 1: An example of a query-tree showing the possible symbolic derivations of $AirlineRoute(x, y, a, l) \wedge American(a)$. Note that the semantics of the interpreted predicates are taken into consideration in the construction of the query-tree. The literals shown in the brackets of each node denote the constraints that need to be satisfied by facts generated at this node, and are used as a criterion for terminating the tree (e.g., the nodes $Route(x, y, c_2, a, l-1)$ and $Route(x, y, c, a, l)$ have the same constraints and therefore only the latter is expanded). Note that rules $r_3$ and $r_4$ are not expanded because they would yield an unsatisfiable set of constraints ($\{American(a), \neg American(a)\}$).

Several aspects of the query-tree make it especially useful in our context. First, recall that determining irrelevance of a clause requires that we can decide that there is some derivation of the query that does not use it. For irrelevance reasoning to be of practical use, we must be able to determine irrelevance without actually solving the query. To that end, the query-tree considers only part of the KB in its reasoning. Specifically,

it considers only the rules in the KB, and high level constraints on the ground unit clauses that may appear (e.g., all flight costs are positive). Consequently, when it decides that a clause is irrelevant, the conclusion holds for all KBs that have the given set of rules, independent of the ground unit clauses. Furthermore, irrelevance is determined w.r.t. a *set* of queries, and these may involve disjunctions and conjunctions of literals. Second, in its irrelevance reasoning, the query-tree considers the semantics of some predicates (e.g., order predicates, $<, >, \leq, \geq, \neq$, or *sort* predicates, e.g., *American*). In many applications, considering the semantics of such predicates enables us to find interactions between clauses and therefore to deem clauses irrelevant. Finally, the query-tree can be built in time that is linear in the number of rules in the KB.

The query-tree actually detects *strongly irrelevant* clauses, i.e., clauses that are not part of *any* derivation of the query (note that strong irrelevance is a sufficient condition for weak irrelevance). In fact, under certain conditions (e.g., function-free or non-recursive rules) the query-tree will find *all* the irrelevant clauses. As pointed out in [Levy and Sagiv, 1993], removing strongly-irrelevant clauses is guaranteed not to slow down inferences (and usually to speed them up significantly), whereas removing weakly irrelevant clauses may actually cause inference to be slowed down. In our context this observation is important because creating an abstraction based on strong irrelevance guarantees that using the abstract theory will result in more efficient inference.

Algorithms for detecting weakly irrelevant clauses are described in [Subramanian, 1989; Levy, 1993]. For general clause form knowledge bases, connection graph methods [Kowalski, 1975; Sickel, 1976; Chang, 1979] provide sufficient conditions for strong and weak irrelevance.

## Determining Independence of Predicate Arguments

Algorithms for determining independence of a clause are specific to a given type of irrelevance subject. In this section we describe a novel algorithm for determining independence of an irrelevance subject consisting of a set of predicate arguments. First we describe a syntactic condition for checking whether a clause $C$ is independent of a given subject $s$. This condition can be used in conjunction with the algorithms of the previous section to determine irrelevance of $s$. However, a more interesting question is how we can automatically find the maximal set of irrelevant predicate arguments, given the set of relevant clauses. We describe an algorithm that uses the syntactic condition to find such a maximal set.

We use the following notation in this section. Given a clause $C$, we denote by $Neg(C)$ and $Pos(C)$ the negative literals and the positive literals in $C$, respectively (e.g., if $C$ is $\{\neg P(x), Q(x)\}$ then $Neg(C)$ is $\{\neg P(x)\}$ and $Pos(C)$ is $\{Q(x)\}$. We assume that $C$ is not redundant, i.e., there is no subset of $C$ that is logically equivalent to $C$, and that $C$ is not a tautology.

**Theorem 4:** *A clause $C$ is independent of the set of predicate arguments $s = \{(P_1, i_1), \ldots, (P_n, i_n)\}$ if the following conditions hold for $1 \leq j \leq n$.*

*If the predicate $P_j$ occurs in $Neg(C)$, then the argument in position $i_j$ of that occurrence:*

*A1: must be a variable (i.e., not a constant or a functional term).*

*A2: the variable must appear at most once in $Neg(C)$.*

*A3: if that variable also appears (by itself, or part of a functional term) in position $k$ of a predicate $Q$ in $Pos(C)$, then $(Q, k) \in s$.* ∎

The proof of the theorem (given in [Levy, 1993]) proceeds by case analysis, showing that for every model $I$ of $C$, $s^I(I)$ will be a model of $s^f(C)$. As an example, the clause $\{\neg P(x, y, z), \neg Q(x), R(y)\}$ is independent of $\{(P, 3)\}$. It is not independent of $\{(Q, 1), (P, 1)\}$ (violates A2) or of $\{(P, 2)\}$ (violates A3).

Given an argument $(P, i)$ that appears in $C$, we can determine the unique minimal set of arguments, $PC(C, P, i)$, such that $(P, i) \in PC(C, P, i)$ and $C$ is independent of $PC(C, P, i)$. This is done by iteratively adding the arguments that are required to be in $s$ by condition A3, and checking whether the final set violates A1 or A2. Note that there may be no such set $PC(C, P, i)$. In that case, we say that $(P, i)$ is *needed* in $C$.

We use the conditions of Theorem 4 to devise the following algorithm that finds the *maximal* set of irrelevant predicate arguments w.r.t. a query. Given a set of relevant clauses $\Gamma$, the algorithm (shown in Figure 2) finds the maximal set of predicate arguments $s$ that does not include any argument appearing in the query (which are assumed to be relevant), such that all clauses in $\Gamma$ are independent of $s$. The algorithm maintains a list of irrelevant arguments, which initially includes all the arguments of all predicates, except those appearing in the query. It makes one pass over the clauses in $\Gamma$ and either removes arguments from the list of irrelevant arguments, or adds conditions for the inclusion arguments in the list. These conditions specify a set of additional arguments that must be included (implied by A3). Finally, it removes from the irrelevant list any argument whose conditions are not satisfied.

Consider the application of the algorithm to the rules $r_1, r_2, r_5$ in Example 1, with the query $AirlineFlight(x, y, a, l) \wedge American(a)$. The set $\mathcal{R}$ initially includes all the arguments of *Flight* and *Route*. When considering the rule $r_1$, the algorithm adds the argument $(Route, i)$ to the preconditions of $(Flight, i)$, for $i = 1, \ldots, 3$, and removes the argument $(Flight, 4)$ from $\mathcal{R}$. Considering rule $r_2$, the algorithm removes the arguments $(Flight, 2)$, $(Route, 1)$ and $(Route, 4)$ from $\mathcal{R}$. As a consequence, the argument $(Flight, 1)$ is

```
procedure find-irrelevant-arguments(Γ, q)
begin /* Γ are the clauses and q is the query. */
    P = The predicates appearing in Γ, and not in q.
    R = {(P,i) | P ∈ P and i is an argument of P }.
    for every s ∈ R, Preconditions(s)= {}.
    for every C ∈ Γ do:
        for every (P,i) ∈ R
            if P appears in C and (P,i) is needed in C
              then remove (P,i) from R.
              else
                if PC(C,P,i) ⊄ R then
                  remove (P,i) from R.
                else add {PC(C,P,i) − {(P,i)}}
                  to Preconditions ((P,i)).
    repeat
        if (P,i) ∈ R and (Q,j) ∈ Preconditions ((P,i))
            and (Q,j) ∉ R
            then remove (P,i) from R.
    until no changes are made to R.
return R.
end.
```

Figure 2: Algorithm for finding a maximal set of irrelevant predicate arguments.

removed from $\mathcal{R}$ because its precondition was removed. Finally, considering rule $r_5$, the arguments $(Route, 2)$ and $(Route, 5)$ are removed from $\mathcal{R}$ because the arguments $(AirlineFlight, 2)$ and $(AirlineFlight, 4)$ are not members of $\mathcal{R}$. Therefore, the algorithm returns that the arguments $(Flight, 3)$ and $(Route, 3)$ are irrelevant to the given query.

The algorithm finds the maximal set of predicate arguments that satisfies conditions A1, A2 and A3. This follows from the observation that for every argument in the returned set, its precondition arguments (i.e., the arguments in $PC(C, P, i)$) are also in $\mathcal{R}$. Furthermore, every argument that was removed from $\mathcal{R}$ was either needed in some clause or required some other argument that is not a member of $\mathcal{R}$. The time complexity of the algorithm is bounded by $|\Delta|R^2$, where $|\Delta|$ is the number of clauses in $\Delta$ and $R$ is the sum of the number of argument of relations in $\Delta$.

## Conclusions and Related Work

We presented a formal connection between the notion of irrelevance and the creation of abstractions. At its core, it is based on associating an abstraction with some *detail* that it removes from the representation of the domain, and justifying the abstraction by observing that the detail is irrelevant to the query. To make this connection we extended previous work on irrelevance reasoning to consider irrelevance of new subjects (e.g., predicate arguments, predicate refinements). Using the connection, we presented a general method for automatically generating abstractions that are suited for a particular set of queries. As an instance of this method, we described a novel and efficient algorithm for automatically creating abstractions in which we remove irrelevant arguments of predicates. Abstraction by projecting out arguments was also suggested in [Hobbs, 1985; Subramanian, 1989], but no algorithm for doing so was given. Our algorithm is a generalization of a method for pushing projections [Ramakrishnan *et al.*, 1988] in datalog programs. Our algorithm handles arbitrary clauses and the semantics interpreted predicates. Additional instances of the general algorithm for creating abstractions can also be devised. For example, the work of Tenenberg [Tenenberg, 1990] effectively provides an algorithm for determining independence of a clause from a predicate refinement,[4] thereby yielding an algorithm for determining irrelevance of predicate refinements.

The computational savings gained by using abstractions has been demonstrated both theoretically and empirically (e.g., [Bacchus and Yang, 1992; Knoblock, 1991; Ellman, 1993]). In our case the savings achieved by abstractions will be maximized if we can identify large sets of queries for which we can create the same abstract KB, and therefore amortize the cost of creating the abstract KB over many queries. One of the key advantages of using the query-tree for relevance reasoning is that it enables us to create abstractions that are tailored for sets of queries. Experiments presented in [Levy, 1993] show that the cost of building the query-tree is negligible compared to the savings achieved by using it.

Studying abstractions in our framework offers several additional advantages. First, we can exploit domain knowledge (stated as irrelevance claims) in creating abstractions for a given query. We can either use such claims directly to justify abstractions or combine them with other methods to derive logical conclusions from them (e.g., using algorithms from [Subramanian, 1989; Levy and Sagiv, 1993]), and obtain justifications for additional abstractions. Second, it provides a framework for choosing and combining existing KBs that each make certain abstractions of the domain, by labeling the KBs with the irrelevance assumptions underlying their abstractions. An example of such a task arises in Compositional Modeling [Falkenhainer and Forbus, 1991; Iwasaki and Levy, 1994], where we need to combine descriptions of different aspects of a physical device to create an adequate and parsimonious model of the device. Similar situations arise in reasoning with contexts (e.g., [Guha, 1991]) and in heterogenous distributed knowledge based systems. Finally, our framework provides insight into the utility of reasoning with abstractions (e.g., abstractions based on strong-irrelevance are guaranteed to yield savings), and to composability of abstractions (by composing the irrelevance statements underlying them).

In their theory of abstraction, Giunchiglia and

[4] A predicate refinement is a set of predicates $P_1, \ldots, P_n$ whose union denotes a predicate $P$.

Walsh [Giunchiglia and Walsh, 1992] distinguish two classes of abstractions, *TD* and *TI*. Roughly, TD abstractions are those in which theorems derived in the abstract theory hold also in the original theory, whereas TI abstractions are those in which every theorem in the original theory will have a theorem in the abstract theory (but an abstract theorem need not have a corresponding theorem in the original theory). The abstractions we considered are TD abstractions (i.e., will not introduce wrong conclusions), but are also *TI w.r.t. the query*, i.e., a solution to the query in the original theory is guaranteed to have a corresponding solution in the abstract theory (but other derivable theorems may be lost in the abstract theory). This aligns with the intuition that removing irrelevant knowledge should not enable us to lose the ability to solve the query or to derive new false conclusions. Creating an abstract KB can also be viewed as an instance of *knowledge compilation* [Selman and Kautz, 1991]. The key difference in our work is that we compile the KB w.r.t. a given set of queries, and therefore we can determine exactly when the compiled KB is applicable. Knoblock [Knoblock, 1990] also considers automatic generation of abstractions that are suited for a specific query (i.e., planning goal), by removing preconditions of actions. His ALPINE system generates TI abstractions, but provides the planner with a condition that enables it to prune the search needed to refine an abstract solution.

TI-abstractions have been used as a means of controlling problem solving in complex domains, by using abstractions to structure the search space hierarchically (e.g., [Sacerdoti, 1974; Plaisted, 1981; Knoblock, 1990; Ellman, 1993]). In that work, the intuition (formally analized in [Bacchus and Yang, 1992]) is that although the information removed from one level of the hierarchy to the other is not always irrelevant, it will be irrelevant in most cases, and therefore, in these cases it will not be necessary to backtrack through the abstraction hierarchy. To apply our framework to this context we are currently considering an extension of irrelevance reasoning to handle *approximate* irrelevance claims that can be used to justify abstracting knowledge that is irrelevant with high probability.

## Acknowledgements

I would like to thank Tom Ellman, Hiroshi Motoda and Pandu Nayak for very useful discussions on the topics described in this paper.

## References

Bacchus, Fahiem and Yang, Qiang 1992. The expected value of hierarchical problem-solving. In *Proceedings of AAAI-92.* 369–374.

Chang, C. L. 1979. Resolution plans in theorem proving. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence.* 143–148.

Ellman, Thomas, editor 1992. *Working Notes of the Workshop on Approximation and Abstraction of Computational Theories.* American Association for Artificial Intelligence.

Ellman, Thomas 1993. Abstraction via approximate symmetry. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence.* 916–921.

Falkenhainer, Brian and Forbus, Ken 1991. Compositional modeling: Finding the right model for the job. *Artificial Intelligence* 51:95–143.

Giunchiglia, Fausto and Walsh, Toby 1992. A theory of abstraction. *Artificial Intelligence* 56 (3).

Guha, Ramanathan V. 1991. *Contexts: A Formalization and Some Applications.* Ph.D. Dissertation, Stanford University, Stanford, CA.

Hobbs, Jerry R. 1985. Granularity. In *Proceedings of IJCAI-85.* 432–435.

Iwasaki, Yumi and Levy, Alon Y. 1994. Automated model selection for simulation. In *Proceedings of AAAI-94.*

Knoblock, Craig A. 1990. Learning abstraction hierarchies for problem solving. In *Proceedings of AAAI-90.*

Knoblock, Craig A. 1991. Search reduction in hierarchical problem solving. In *Proceedings of AAAI-91.* 686–691.

Kowalski, Robert 1975. A proof procedure using connection graphs. *Journal of the ACM* 22(4): 572–595.

Levy, Alon Y. and Sagiv, Yehoshua 1992. Constraints and redundancy in Datalog. In *The Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS).* 67–80.

Levy, Alon Y. and Sagiv, Yehoshua 1993. Exploiting irrelevance reasoning to guide problem solving. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence.* 138–144.

Levy, Alon Y. 1993. *Irrelevance Reasoning in Knowledge Based Systems.* Ph.D. Dissertation, Stanford University, Stanford, CA.

Nayak, P. Pandurang and Levy, Alon Y. 1994. A semantic theory of abstractions: A preliminary report. Technical Report, AT&T Bell Laboratories.

Plaisted, D. 1981. Theorem proving with abstraction. *Artificial Intelligence* 16:47–108.

Ramakrishnan, Raghu; Beeri, Catriel; and Krishnamurthy, Ravi 1988. Optimizing existential datalog queries. In *Proceedings of PODS-88.* 89–101.

Sacerdoti, Earl D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5:115–135.

Selman, Bart and Kautz, Henry 1991. Knowledge compilation using horn approximations. In *Proceedings of AAAI-91.* 904–909.

Sickel, Susan 1976. A search technique for clause interconnectivity graphs. *IEEE Transactions on Computers* C-25(8):823–835.

Subramanian, Devika 1989. *A Theory of Justified Reformulations.* Ph.D. Dissertation, Stanford University, Stanford, CA.

Tenenberg, Josh D. 1990. Abstracting first order theories. In Benjamin, Paul, editor 1990, *Change of Representation and Inductive Bias.* Kluwer, Boston, Mass.