Induction as Knowledge Integration

Benjamin D. Smith
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive M/S 525-3660
Pasadena, CA 91109-8099
smith@aig.jpl.nasa.gov

Paul S. Rosenbloom
Information Sciences Institute & Computer Science Dept.
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
rosenbloom@isi.edu

Abstract

Two key issues for induction algorithms are the accuracy of the learned hypothesis and the computational resources consumed in inducing that hypothesis. One of the most promising ways to improve performance along both dimensions is to make use of additional knowledge. Multi-strategy learning algorithms tackle this problem by employing several strategies for handling different kinds of knowledge in different ways. However, integrating knowledge into an induction algorithm can be difficult when the new knowledge differs significantly from the knowledge the algorithm already uses. In many cases the algorithm must be rewritten.

This paper presents KII, a Knowledge Integration framework for Induction, that provides a uniform mechanism for integrating knowledge into induction. In theory, arbitrary knowledge can be integrated with this mechanism, but in practice the knowledge representation language determines both the knowledge that can be integrated, and the costs of integration and induction. By instantiating KII with various set representations, algorithms can be generated at different trade-off points along these dimensions.

One instantiation of KII, called RS-KII, is presented that can implement hybrid induction algorithms, depending on which knowledge it utilizes. RS-KII is demonstrated to implement AQ-11 (Michalski 1978), as well as a hybrid algorithm that utilizes a domain theory and noisy examples. Other algorithms are also possible.

Introduction

Two key criteria for evaluating induction algorithms are the accuracy of the induced hypothesis and the computational cost of inducing that hypothesis. One of the most powerful ways to achieve improvements along both of these dimensions is by integrating additional knowledge into the induction process. Knowledge consists of examples, domain theories, heuristics, and any other information that affects which hypothesis is induced—that is, knowledge is examples plus biases.

A given single-strategy learning algorithm can utilize some knowledge very effectively, others less effectively, and some knowledge not at all. By using multiple strategies, an induction algorithm can make more effective use of a wider range of knowledge, thereby improving performance. However, even a multi-strategy learning algorithm can only make use of knowledge for which its strategies are designed.

In order to utilize new kinds of knowledge, the knowledge must either be recast as a kind for which the algorithm already has a strategy—for example, integrating type constraints into FOIL by casting them as pseudo negative examples (Quinlan 1990)—or the algorithm must be rewritten to take advantage of the new knowledge by adding a new strategy or modifying an existing one. The first approach—recasting knowledge—is limited by the expressiveness of the knowledge already used by the algorithm. If the new knowledge cannot be expressed in terms of the existing kinds of knowledge, then the new knowledge cannot be utilized. The second approach—rewriting an algorithm to utilize a new kind of knowledge—is difficult. It also fails to solve the underlying problem—if yet another kind of knowledge is made available, the algorithm may have to be modified once again.

What is needed is an easier way to integrate knowledge into induction. One approach for doing this exploits the observation that a knowledge fragment plus a strategy for using that knowledge constitutes a bias, since together they determine which hypothesis is induced. These biases can be expressed uniformly in terms of constraints and preferences on the hypothesis space. The induced hypothesis is the most preferred hypothesis among those that satisfy the constraints. New knowledge and strategies are integrated into induction by combining their constraints and preferences with those previously integrated.

This approach is formalized in a framework called KII. This framework represents constraints and preferences as sets, and provides set-based operations for integrating knowledge expressed in this way, and for inducing hypotheses from the integrated knowledge. Converting knowledge into constraints and preferences is handled by *translators* (Cohen 1992), which are written by the user for each knowledge fragment, or class of related knowledge fragments.

Since KII is defined in terms of sets and set operations, some set representation must be specified in order for KII to be operational. The set representation determines the kinds of knowledge that can be expressed, and also determines the computational complexity of integration and induction. Each set representation yields an instantiation of KII at a different trade-off point between expressiveness and computational complexity.

This approach is most similar to that of Russell and Grosof (Russell & Grosof 1987), in which biases are represented as determinations, and the hypothesis is deduced from the determinations and examples by a theorem prover. As in KII, the inductive leaps come from biases, which may be grounded in supposition instead of fact. A major difference between this system and KII is KII's ability to select different set representations, which allows different trade-offs to be made between expressiveness and cost. Determinations, by contrast, are at a fixed trade-off point, although one could imagine using restricted logics.

One advantage of KII's formal relationship between the set representation and the cost/expressiveness trade-off is that it allows formal analysis of these trade-offs. In particular, an upper limit can be established on the expressiveness of the set representations for which induction is even computable. This sets a practical limit on the kinds of knowledge that can be utilized by induction.

Among the set representations below this limit, there are a number that generate useful instantiations of KII. Most notably, Incremental Version Space Merging (Hirsh 1990) can be generated by using a boundary set representation for constraints (i.e., version spaces), and an empty representation for preferences; and an algorithm similar to Grendel (Cohen 1992) can be instantiated from KII by representing sets as antecedent description grammars (essentially context free grammars). These will be discussed briefly. A new algorithm, RS-KII, is instantiated from KII by representing sets as regular grammars. This algorithm seems to strike a good balance between expressiveness and complexity.

RS-KII can use a wide range of knowledge, and combine this knowledge in a number of ways. This makes it a good multi-strategy algorithm. RS-KII can use the knowledge and strategies of at least two existing algorithms, the Candidate Elimination Algorithm (Mitchell 1982) and AQ-11 with a beam width of one (Michalski 1978). It can also utilize additional knowledge, such as a domain theory and noisy examples. Although space limits us from discussing all of these in detail, the translators needed to implement AQ-11 are demonstrated, as well as those for the domain theory and noisy examples. When utilizing only the AQ-11 knowledge, RS-KII induces the same hypotheses as AQ-11 with a beam width of one, with a computational complexity that is only a little worse. When RS-KII utilizes the translators for the additional knowledge, RS-KII induces a more accurate hypothesis than AQ-11, and in much less time. RS-KII looks able to express and integrate other common knowledge sources and strategies as well, though this is an area for future research.

The Knowledge Integration Framework

This section formally describes KII, a Knowledge Integration Framework for Induction. The combination of a knowledge fragment and a strategy for using that knowledge can be considered a bias, which is expressed in terms of constraints and preferences over the hypothesis space. For instance, a positive example and a strategy that assumes the target concept is strictly consistent with the examples, would be translated as a constraint that is satisfied only by hypotheses that cover the example. A strategy that assumed noisy examples might be expressed as a preference for hypotheses that were most consistent with the example, but does not reject inconsistent hypotheses outright.

The biases are *integrated* into a single composite bias by combining their respective constraints and preferences. The composite bias, which includes the examples, wholly determines the selection of the induced hypothesis. If there are several hypotheses which the bias finds equally acceptable, any one may be selected arbitrarily as the target concept. This set is called the *solution set*. In this view, integration precedes induction, rather than being part of it. This separation makes it easier to integrate knowledge into induction, since the effects of each process are clearer.

KII formalizes these ideas as follows. Each bias is expressed as a triple of three sets, $\langle H, C, P \rangle$, where H is the hypothesis space, C is the set of hypotheses that satisfy the constraints of all the biases, and P is a set of hypothesis pairs, $\langle x, y \rangle$, such that x is less preferred than y by at least one of the biases. The solution set, from which the induced hypothesis is selected arbitrarily, is the set of most preferred hypothesis among those that satisfy the constraints—namely, the hypotheses in C for which no other hypothesis in C is preferable, according to P. Formally, $\{x \in C \mid \forall y \in C \ \langle x, y \rangle \notin P\}$.

KII provides several operations on knowledge expressed in this representation: *translation*, *integration*, *induction* (selecting a hypothesis from the solution set), and solution set *queries*. These operations, as well as the solution set itself, are defined in terms of set operations on H, C, and P. These operators are described in detail below.

Translation Knowledge is converted from the form in which it occurs (its *naturalistic representation* (Rosenbloom *et al.* 1993)) into $\langle H, C, P \rangle$ triples by *translators* (Cohen 1992). Since knowledge is translated into constraints and preferences over the hypothesis space, the implementation of each translator depends on both the hypothesis space and the knowl-

edge. In the worst case, a different implementation is required for each pair of knowledge fragment and hypothesis space. Since there are a potentially infinite number of translators, they are not provided as part of the KII formalism, but must be provided by the user as needed.

Fortunately, closely related pairs of hypothesis space and knowledge often have similar translations, allowing a single translator to be written for all of the pairs. One such translator, which will be described in detail later, takes as input an example and a hypothesis space. The example can be any member of the instance space, and the hypothesis space is selected from a family of languages by specifying the set of features. The same translator works for every pair of example and hypothesis language in this space.

Integration Translated knowledge fragments are *integrated* by composing their $\langle H, C, P \rangle$ triples. A hypothesis can only be the induced hypothesis if it is accepted by the constraints of all of the knowledge fragments, and if the combined preferences of the knowledge fragments do not prefer some other hypothesis. That is, the induced hypothesis must satisfy the conjunction of the constraints, and be preferred by the disjunction of the preferences. This reasoning is captured in the following definition for the integration of two tuples, $\langle H, C_1, P_1 \rangle$ and $\langle H, C_2, P_2 \rangle$. The hypothesis space is the same in both cases, since it is not clear what it means to integrate knowledge about target hypotheses from different hypothesis spaces.

$$\text{Integrate}(\langle H, C_1, P_1 \rangle, \langle H, C_2, P_2 \rangle) = \langle H, C_1 \cap C_2, P_1 \cup P_2 \rangle \quad (1)$$

The integration operator assumes that the knowledge is consistent. That is, C_1 and C_2 are not mutually exclusive, and that $P_1 \cup P_2$ does not contain cycles (e.g., $a < b$ and $b < a$). Although such knowledge can be integrated, the inconsistencies will not be dealt with in any significant fashion. Mutually exclusive constraints will result in an empty solution set, and cycles are broken arbitrarily by assuming every element of the cycle is dominated. Developing more sophisticated strategies for dealing with contradictions is an area for future research.

Although KII does not deal with contradictory knowledge, it can deal with uncertain knowledge. For example, noisy examples and incomplete domain theories can both be utilized in KII. Translators for these knowledge sources are described later.

Induction and Solution Set Queries The integrated knowledge is represented by a single tuple, $\langle H, C, P \rangle$. The target concept is induced from the integrated knowledge by selecting an arbitrary hypothesis from the solution set of $\langle H, C, P \rangle$. KII also supports queries about the solution set, such as whether it is empty, a singleton, contains a given hypothesis, or is a subset of some other set. These correspond to the operations that have proven empirically useful for version spaces (Hirsh 1992), which can be thought of as solution sets for knowledge expressed as constraints.

It is conjectured that these four queries plus the ability to select a hypothesis from the solution set are sufficient for the vast majority of induction tasks. Most existing induction algorithms involve only the enumeration operator and perhaps an *Empty* or *Unique* query. The Candidate Elimination algorithm (Mitchell 1982) and Incremental Version Space Merging (IVSM) (Hirsh 1990) use all four queries, but do not select a hypothesis from the solution set (they return the entire set).

The queries and selection of a hypothesis from the solution set can be implemented in terms of a single *enumeration* operator. The enumeration operator returns n elements of a set, S, where n is specified by the user. It is defined formally as follows.

$$\begin{aligned} &\mathit{Enumerate}(S, n) \rightarrow \{h_1, h_2, \ldots h_m\} \\ &\quad \text{where} \\ &\quad m = \min(n, |S|), \{h_1, h_2, \ldots h_m\} \subseteq S \end{aligned}$$

Normally, S is the solution set of $\langle H, C, P \rangle$. It can sometimes be cheaper to compute the first few elements of the solution set from $\langle H, C, P \rangle$ than to compute even the intensional representation of the solution set from $\langle H, C, P \rangle$. Therefore, the S argument to the enumeration operator can be either a $\langle H, C, P \rangle$ tuple, or a set expression involving an $\langle H, C, P \rangle$ tuple and other sets. This allows the enumeration operator to use whatever optimizations seem appropriate. A different implementation of the enumerate operator is needed for different set representations of S, H, C, and P.

A hypothesis is induced by selecting a single hypothesis from the solution set. This is done with a call to $\mathit{Enumerate}(\langle H, C, P \rangle, 1)$. The emptiness and uniqueness queries are implemented as shown below, where S is the solution set of tuple $\langle H, C, P \rangle$, A is set of hypotheses in H, and h is a hypothesis in H.

- $\mathit{Empty}(S) \Leftrightarrow \mathit{Enumerate}(\langle H, C, P \rangle, 1) = \emptyset$
- $\mathit{Unique}(S) \Leftrightarrow |\mathit{Enumerate}(\langle H, C, P \rangle, 2)| = 1$
- $\mathit{Member}(h, S) \Leftrightarrow \mathit{Enumerate}(\langle H, C, P \rangle \cap \{h\}, 1) \neq \emptyset$
- $\mathit{Subset}(S, A) \Leftrightarrow \mathit{Enumerate}(\langle H, C, P \rangle \cap \overline{A}, 1) = \emptyset$

An Example Induction Task

An example of how KII can solve a simple induction task is given below. Sets have been represented extensionally in this example. Although this is not the only possible set representation, and is generally a poor one, it is the simplest one for illustrative purposes.

The Hypothesis Space The target concept is a member of a hypothesis space in which hypotheses are described by conjunctive feature vectors. There are three features `size`, `color`, and `shape`. The values for these features are `size` $\in$ {`small`, `large`, `any-size`}, `color` $\in$ {`black`, `white`, `any-color`}, and

- *TranPosExample*$(H, \langle z, c, s\rangle) \rightarrow \langle C, \{\}\rangle$ where
  $C = \{x \in H \mid x \text{ covers } \langle z, c, s\rangle\}$
  $= \{z, \texttt{any-size}\} \times \{c, \texttt{any-color}\} \times \{s, \texttt{any-shape}\}$
- *TranNegExample*$(H, \langle z, c, s\rangle) \rightarrow \langle C, \{\}\rangle$ where
  $C = \{x \in H \mid x \text{ does not cover } \langle z, c, s\rangle\}$
  $=$ complement of
  $\{z, \texttt{any-size}\} \times \{c, \texttt{any-color}\} \times \{s, \texttt{any-shape}\}$
- *TranPreferGeneral*$(H) \rightarrow \langle H, P\rangle$ where
  $P = \{\langle x, y\rangle \in H \times H \mid x \text{ is more specific than } y\}$
  $= \{\langle sbr, ?br\rangle, \langle sbr, s?r\rangle, \langle sbr, ??r\rangle, \langle swr, ?wr\rangle, \ldots\}$

Figure 1: Translators.

shape $\in$ {`circle`, `rectangle`, `any-shape`}. Hypotheses are described as 3-tuples from `size`×`color`×`shape`. For shorthand identification, a value is specified by the first character of its name, except for the any values which are represented by a "?". So the hypothesis ⟨`any-size`, `white`, `circle`⟩ would be written as *?wc*.

Instances are the "ground" hypotheses. An instance is a tuple ⟨*size*, *color*, *shape*⟩ where *color* $\in$ {`black`, `white`}, *size* $\in$ {`small`, `large`}, and *shape* $\in$ {`circle`, `rectangle`}.

Available Knowledge The available knowledge consists of three examples (classified instances), and an assumption that accuracy increases with generality. There are three examples, two positive and one negative. The two positive examples are $e_1 = swc$ and $e_2 = sbc$. The negative example is $e_3 = lwr$. The target concept is *s??*. That is, `size = small`, and color and shape are irrelevant.

Translators The first step is to translate the knowledge into constraints and preferences. Three translators are constructed, one for each type of knowledge: the positive examples, negative examples, and the generality preference. These translators are shown in Figure 1. Since the hypothesis space is understood, $\langle H, C, P\rangle$ tuples will generally be referred to as just $\langle C, P\rangle$ tuples for the remainder of this illustration.

The examples are translated in this scenario under the assumption that they are correct; that is, the target concept covers all of the positive examples and none of the negatives. Positive examples are translated as constraints satisfied only by hypotheses that cover the example. Negative examples are translated similarly, except that hypotheses must *not* cover the example. The bias for general hypotheses is translated into a $\langle C, P\rangle$ pair where C is H (it rejects nothing), and $P = \{\langle x, y\rangle \in H \times H \mid x \text{ is more specific than } y\}$. Hypothesis x is more specific than hypothesis y if x is equivalent to y, except that some of the values in y have been replaced by "any" values. For example, *swr* is more specific than *?wr*, but there is no ordering between *?wc* and *swr*.

Integration and Induction Examples e_1 and e_2 are translated by *TranPosExample* into $\langle H, C_1, \emptyset\rangle$ and $\langle H, C_2, \emptyset\rangle$, respectively. Example e_3 is translated by *TranNegExample* into $\langle H, C_3, \emptyset\rangle$. The preference for general hypotheses is translated into $\langle H, H, P_4\rangle$. These tuples are integrated into a single tuple, $\langle H, C, P\rangle = \langle H, C_1 \cap C_2 \cap C_3 \cap H, \emptyset \cup \emptyset \cup \emptyset \cup P_4\rangle$. This tuple represents the combined biases of the four knowledge fragments.

A hypothesis is induced by selecting one arbitrarily from the solution set of $\langle H, C, P\rangle$. This is accomplished by calling *Enumerate*$(\langle H, C, P\rangle, 1)$. The solution set consists of the undominated elements of C with respect to the dominance relation P. C contains three elements, *s??*, *??c* and *s?c*. P prefers both *s??* and *??c* to *s?c*, but there is no preference ordering between *s??* and *??c*. The undominated elements of C are therefore *s??* and *??c*. One of these is selected arbitrarily as the induced hypothesis.

Instantiating KII

In order to implement KII, specific set representations for H, C, and P are necessary. These representations can be as simple as an extensional set, or as powerful as arbitrary Turing machines. However, some representation is needed. The representation determines which knowledge can be expressed in terms of $\langle H, C, P\rangle$ tuples and integrated. It also determines the computational complexity of the integration and enumeration operations, which are defined in terms of set operations. By instantiating KII with different set representations, algorithms can be generated at different tradeoff points between cost and expressiveness.

The space of possible set representations maps onto the space of grammars. Every computable set is the language of some grammar. Similarly, every computable set representation is equivalent to some class of grammars. These classes include, but are not limited to, the classes of the Chomsky hierarchy (Chomsky 1959)—regular, context free, context sensitive, and recursively enumerable (r.e.). The complexity of set operations generally increases with the expressiveness of the language class.

Allowing H, C, and P to be recursively enumerable (i.e., arbitrary Turing machines), would certainly provide the most expressiveness. Although $\langle H, C, P\rangle$ tuples with r.e. sets can be expressed and integrated, the solution sets of some such tuples are uncomputable, and there is no way to know which tuples have this property. This will be discussed in more detail below. Since it is impossible to enumerate even a single element of an uncomputable set, it is impossible to induce a hypothesis by selecting one from the solution set. There is clearly a practical upper limit on the expressiveness of the set representations.

It is possible to establish the most expressive languages for C and P that guarantee a computable solution set. This establishes a practical limit on the knowledge that can be integrated into induction.

By definition, the solution set is computable if and only if it is recursively enumerable. The solution set can always be constructed by applying a formula of set operations to C and P, as will be shown below. The most restrictive language in which the solution set can be expressed can be derived from this formula and the set representations for C and P by using the the closure properties of these set operations. Inverting this function yields the most expressive C and P representations for which the solution set is guaranteed to be at most recursively enumerable.

The solution set can be computed from C and P according to the equation $\overline{first((C\times C)\cap P)}\cap C$. The derivation is shown in Equation 2, below. In this definition, the function $first(\{\langle x_1,y_1\rangle,\langle x_2,y_2\rangle,\ldots\})$ is a projection returning the set of tuple first-elements, namely $\{x_1,x_2,\ldots\}$.

$$\begin{aligned} SolnSet(\langle H,C,P\rangle) &= \{x\in C \mid \forall_{y\in C}\ \langle x,y\rangle \notin P\} \\ &= \overline{\{x\in H \mid (x\in C \text{ and } \exists_{y\in C}\langle x,y\rangle\in P) \text{ or } x\notin C\}} \\ &= \overline{\{x\in H \mid x\in C \text{ and } \exists_{y\in C}\langle x,y\rangle\in P\}\cup\overline{C}} \\ &= \overline{first(\{\langle x,y\rangle\in C\times C \mid \langle x,y\rangle\in P\})}\cap C \\ &= \overline{first((C\times C)\cap P)}\cap C \end{aligned} \quad (2)$$

The least expressive representation in which the solution set can be represented can be computed from the closure of the above equation over the C and P set representations. To do this, it helps to know the closure properties for the individual set operations in the equation: intersection, complement, Cartesian product, and projection (first). The closure properties of intersection and complement are well known for most language classes, although it is an open problem whether the context sensitive languages are closed under complementation (Hopcroft & Ullman 1979). The closure properties of projection and Cartesian product are not known as such, but these operations map onto other operations for which closure properties are known.

The Cartesian product of two grammars, $A\times B$, can be represented by their concatenation, AB. The tuple $\langle x,y\rangle$ is represented by the string xy. The Cartesian product can also be represented by interleaving the strings in A and B so that $\langle x,y\rangle$ is represented by a string in which the symbols in x and y alternate. Interleaving can sometimes represent subsets of $A\times B$ that concatenation cannot, depending on the language in which the product is expressed. The closure properties of languages under Cartesian product depends on which approach is used. The following discussion derives limits on the languages for $C\times C$ and P. When the language for C is closed under Cartesian product, then the limits on $C\times C$ also apply to C, since both can be expressed in the same language. Otherwise, the limits on C have to be derived from those on $C\times C$ using the closure properties of the given implementation of Cartesian product. However, when C is not closed under Cartesian product, the language for C is necessarily less expressive than that for $C\times C$. The expressiveness limits on $C\times C$ therefore provide a good upper bound on the expressiveness of C that is independent of the Cartesian product implementation.

Regardless of the representation used for Cartesian product, projection can be implemented as a *homomorphism* (Hopcroft & Ullman 1979), which is a mapping from symbols in one alphabet to strings in another. Homomorphisms can be used to erase symbols from strings in a language, which is exactly what projection does—it erases symbols from the second field of a tuple, leaving only the symbols from the first field. A more detailed derivation of the properties for projection and Cartesian product can be found in (Smith 1995).

The closure properties of languages under projection, intersection, intersection with a regular grammar, and complement are summarized in Table 1. It should be clear that the solution set, $\overline{first((C\times C)\cap P)}\cap C$, is r.e. when $(C\times C)\cap P$ is at most context free, and uncomputable when it is any more expressive than that. For example, if $(C\times C)\cap P$ is context sensitive, then $first((C\times C)\cap P$ is r.e. The complement of a set that is r.e. but not recursive is uncomputable (Hopcroft & Ullman 1979), so the solution set, $\overline{first((C\times C)\cap P}$, is uncomputable. A complete proof appears in (Smith 1995).

There are several ways to select C, P, and the implementation of Cartesian product, such that $(C\times C)\cap P$ is at most context free. The expressiveness of both C and P can be maximized by choosing one of C and P to be at most regular, and the other to be at most context free. This is because CFLs are closed under intersection with regular sets, but not with other CFLs. Regular sets are closed under all implementations of Cartesian product (both concatenation and arbitrary interleaving), and context free sets are closed under concatenation but only some interleavings. So if C is regular, any implementation of Cartesian product can be used, but if C is context free, then the choices are more restricted.

As a practical matter, C should be closed under intersection and P under union in order to support the integration operator. This effectively restricts C to be regular and P to be at most context free. This also maximizes the choices of the Cartesian product implementation. However, it is possible for C to be context free and P to be regular if the C set of at most one of the $\langle H,C,P\rangle$ triples being integrated is context free and the rest are regular. This follows from the closure of context free languages under intersection with regular grammars.

Other ways of selecting C and P are summarized in Table 2. This table assumes that C is closed under Cartesian product. As one interesting case, if the representation for P can express only the empty set, then the solution set is just C, so C can be r.e. The

Operations	Language					
	Regular	DCFL	CFL	CSL	recursive	r.e.
$\cap$	√			√	√	√
$\cap R$	√	√	√	√	√	√
complement	√	√		?	√	
projection (homomorphisms)	√		√			√

Table 1: Closure Under Operations Needed to Compute the Solution Set.

restriction that $(C \times C) \cap P$ be at most context free is still satisfied, since $(C \times C) \cap P$ is always the empty set, and therefore well within the context free languages.

RS-KII

Instantiating KII with different set representations produces algorithms with different computational complexities and abilities to utilize knowledge. One instantiation that seems to strike a good balance between computational cost and expressiveness represents H, C, and P as regular sets. This instantiation is called RS-KII.

RS-KII is a good multi-strategy algorithm, in that it can utilize various knowledge and strategies, depending on what knowledge is integrated, and how it is translated. Existing algorithms can be emulated by creating translators for the knowledge and strategies of that algorithm, and integrating the resulting $\langle H, C, P \rangle$ tuples. Hybrid multi-strategy algorithms can be created by translating and integrating additional knowledge, or by integrating novel combinations of knowledge for which translators already exist.

Creating algorithms by writing translators for individual knowledge fragments and integrating them together can be easier than writing new induction algorithms. Algorithms can be constructed modularly from translators, which allows knowledge fragments to be easily added or removed. By contrast, modifications made to an algorithm in order to utilize one knowledge fragment may have to be discarded in order to utilize a second fragment.

The remainder of this section demonstrates how RS-KII can emulate AQ-11 with a beam width of one (Michalski 1978), and how RS-KII can integrate additional knowledge, namely an overgeneral domain theory and noisy examples, to create a hybrid algorithm. AQ-11 with higher order beam widths is not demonstrated, since it is not clear how to express the corresponding bias as a regular grammar. This bias may require a more powerful set representation.

When using only the AQ-11 knowledge, RS-KII induces the same hypotheses as AQ-11, albeit at a slightly worse computational complexity. When utilizing the additional knowledge, RS-KII induces a more accurate hypothesis than AQ-11, and does so more quickly.

RS-KII translators can be written for other knowledge as well, though space restrictions prevent any detailed discussion. Of note, RS-KII translators can be constructed for all biases expressible as version spaces (for certain classes of hypothesis spaces) (Smith 1995). It also looks likely that RS-KII translators can be constructed for the knowledge used by other induction algorithms, though this is an area for future research.

Translators for AQ-11 Biases

The biases used by AQ-11 are strict consistency with the examples, and an user-defined *lexicographic evaluation function* (LEF). The LEF totally orders the hypotheses according to user-defined criteria. The induced hypothesis is one that is consistent with all of the examples, and is a (possibly local) maximum of the LEF. A translator is demonstrated in which the LEF is an information gain metric, as used in algorithms such as ID3 (Quinlan 1986).

Hypotheses are sentences in the VL_1 language (Michalski 1974). There are k features, denoted f_1 through f_k, where feature f_i can take values from the set V_i. A hypothesis is a disjunction of terms, a term is a conjunction of selectors, and a selector is of the form $[f_i \; rel \; v_i]$, where v_i is in V_i and rel is a relation in $\{<, \leq, =, \neq, \geq, >\}$. A specific hypothesis space in VL_1 is specified by the list of features and their values, and is denoted $VL_1(\langle f_1, V_1 \rangle, \ldots, \langle f_k, V_k \rangle)$.

An instance is a vector of k values, $\langle x_1, x_2, \ldots, x_k \rangle$, where x_i is a value in V_i. A selector $[f_i \; rel \; v_i]$ is satisfied by an example if and only if $x_i \; rel \; v_i$. A hypothesis *covers* an example if the example satisfies the hypothesis.

Strict Consistency with Examples A bias for strict consistency with a positive example can be expressed as a constraint that the induced hypothesis must cover the example. Similarly, strict consistency with a negative example constrains the induced hypothesis not to cover the example. Each of these constraints is expressed as a regular grammar that only recognizes hypotheses that satisfy the constraint. The regular expression for the set of VL_1 hypotheses covering an example, $Covers(H, e)$ is shown in Figure 2. The sets of values in COVERING-SELECTOR are all regular sets. For example, the set of integers less than

C	P	$(C \times C) \cap P$	$\overline{first((C \times C) \cap P)} \cap C$
$\leq$ regular	$\leq$ regular	$\leq$ regular	$\leq$ regular
$\leq$ regular	$\leq$ CFL	$\leq$ CFL	$\leq$ recursive
$\leq$ CFL	$\leq$ regular	$\leq$ CFL	$\leq$ recursive
$\geq$ CFL	$\geq$ CFL	$\geq$ CSL	uncomputable
		$>$ CFL	uncomputable

Table 2: Summary of Expressiveness Bounds.

$TranPosAQExample(VL_1(\langle f_1, V_1\rangle, \ldots, \langle f_k, V_k\rangle), \langle x_1, x_2, \ldots, x_k\rangle) \rightarrow \langle H, C, \{\}\rangle$

where
$H = VL_1(\langle f_1, V_1\rangle, \ldots, \langle f_k, V_k\rangle)$
$C = Covers(VL_1(\langle f_1, V_1\rangle, \ldots, \langle f_k, V_k\rangle), \langle x_1, \ldots, x_k\rangle)$

$TranNegAQExample(VL_1(\langle f_1, V_1\rangle, \ldots, \langle f_k, V_k\rangle), \langle x_1, x_2, \ldots, x_k\rangle) \rightarrow \langle H, C, \{\}\rangle$

where
$H = VL_1(\langle f_1, V_1\rangle, \ldots, \langle f_k, V_k\rangle)$
$C = Excludes(VL_1(\langle f_1, V_1\rangle, \ldots, \langle f_k, V_k\rangle), \langle x_1, \ldots, x_k\rangle)$

Figure 3: Example Translators for VL_1.

100 is $(0-9)|((1-9)(0-9))$. There is an algorithm that generates each of these sets given the relation and the bounding number, but it is omitted for brevity. The complement of $Covers(H, e)$ is $Excludes(H, e)$, the set of hypotheses in H that do not cover example e. These two regular grammars implement the translators for positive and examples in the VL_1 hypothesis space language, as shown in Figure 3. The translator takes as input the list of features and their values, and the example.

The LEF AQ-11 performs a beam search of the hypothesis space to find a hypothesis that maximizes the LEF, or is at least a good local approximation. AQ-11 returns the first hypothesis visited by this search that is also consistent with the examples. This is a bias towards hypotheses that come earlier in the search order.

This bias can be expressed as an $\langle H, C, P\rangle$ tuple in which $C = H$ (i.e., no hypotheses are rejected), and P is a partial ordering over the hypothesis space in which $\langle a, b\rangle$ is in P if and only if hypothesis a comes after hypothesis b in the search order (i.e., a is less preferred than b).

The search order of a beam search is difficult, and perhaps impossible, to express as a regular grammar. However, with a beam width of one, beam search becomes hill climbing, which can be expressed as a regular grammar.

In hill climbing, single selector extensions of the current best hypothesis are evaluated by some evaluation function, f, and the extension with the best evaluation becomes the next current best hypothesis. Given two terms, $t_1 = a_1 a_2 \ldots a_n$ and $t_2 = b_1 b_2 \ldots b_m$, where a_i and b_i are selectors, t_1 is visited before t_2 if the first $k-1$ extensions of t_1 and t_2 are the same, but on the k^{th} extension, either t_1 has a better evaluation than t_2, or t_1 has no more selectors. Formally, there is either some extension $k \leq \min(m, n)$ such that for all $i < k$, $a_i = b_i$ and $f(a_1 \ldots a_k) > f(b_1 \ldots b_k)$, or $m < n$ and the first m selectors of t_1 and t_2 are the same.

This is equivalent to saying that the digit string $f(a_1) \cdot f(a_1 a_2) \cdot \ldots \cdot f(a_1 a_2 \ldots a_n)$ comes before the digit string $f(b_1) \cdot f(b_1 b_2) \cdot \ldots \cdot f(b_1 b_2 \ldots b_n)$ in dictionary (lexicographic) order. This assumes that low evaluations are best, and that the evaluation function returns a unique value for each term—that is, $f(a_1 \ldots a_m) = f(b_1 \ldots b_m)$ if and only if $a_i = b_i$ for all i between one and m. This can be ensured by assigning a unique id to each selector, and appending the id for the last selector in the term to the end of the term's evaluation. The evaluations of two terms are compared after each extension until one partial term either has a better evaluation, or terminates.

A regular grammar can be constructed that recognizes pairs of hypotheses, $\langle h_1, h_2\rangle$, if h_1 is visited before h_2 in the search. This is done in two steps. First, a grammar is constructed that maps each hypothesis onto digit strings of the kind described above. The digit strings are then passed to a regular grammar that recognizes pairs of digit strings, $\langle d_1, d_2\rangle$, such that d_1 comes before d_2 in dictionary order. This is equivalent to substituting the mapping grammar into the dictionary ordering grammar. Since regular grammar are closed under substitution, the resulting grammar is also regular (Hopcroft & Ullman 1979).

The digit string comparison grammar is the simpler of the two, so it will be described first. This grammar recognizes pairs of digit strings, $\langle x, y\rangle$, such that x comes before y lexicographically. A special termination symbol, #, is appended to each string, and the resulting strings are interleaved so that their symbols alternate. The interleaved string is given as input to the grammar specified by the regular expression EQUAL* LESS-THAN ANY*, where EQUAL = (00|11|##), LESS-THAN = (01|#0|#1) and ANY = (0|1|#). This expression assumes a binary digit string, but can be easily

$$
\begin{array}{l}
Covers(VL_1(\langle f_1, V_1\rangle, \langle f_2, V_2\rangle, \ldots, \langle f_k, V_k\rangle), \langle x_1, x_2, \ldots, x_k\rangle) \rightarrow G \text{ where} \\
\quad \mathrm{G} = (\text{ANY-TERM or})^* \text{ COVERING-TERM (or ANY-TERM)}^* \\
\quad \text{ANY-TERM} = \text{SELECTOR}^+ \\
\quad \text{COVERING-TERM} = \text{COVERING-SELECTOR}^+ \\
\quad \text{SELECTOR} = \text{“[”} \ f_1\ (< \mid \leq \mid = \mid \neq \mid \geq \mid >)\ V_1\ \text{“]”} \ \mid \\
\qquad\qquad\qquad\qquad \text{“[”} \ f_2\ (< \mid \leq \mid = \mid \neq \mid \geq \mid >)\ V_2\ \text{“]”} \ \mid \\
\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
\qquad\qquad\qquad\qquad \text{“[”} \ f_k\ (< \mid \leq \mid = \mid \neq \mid \geq \mid >)\ V_k\ \text{“]”} \\
\quad \text{COVERING-SELECTOR} = \{[f_i \,\#\, v] \mid x_i \,\#\, v \text{ and } \# \in \{<, \leq, =, \neq, \geq, >\}\} \\
\quad = \text{“[”} \ f_1 < \{v \in V_1 \mid v \geq x_1\}\ \text{“]”} \mid \ldots \mid \text{“[”} \ f_k < \{v \in V_k \mid v \geq x_k\}\ \text{“]”} \mid \\
\qquad \text{“[”} \ f_1 \leq \{v \in V_1 \mid v > x_1\}\ \text{“]”} \mid \ldots \mid \text{“[”} \ f_k \leq \{v \in V_k \mid v > x_k\}\ \text{“]”} \mid \\
\qquad \text{“[”} \ f_1 = x_1\ \text{“]”} \mid \qquad\qquad \ldots \mid \text{“[”} \ f_k = x_k\ \text{“]”} \mid \\
\qquad \text{“[”} \ f_1 \neq (V_1 - \{x_1\})\ \text{“]”} \mid \qquad\qquad \ldots \mid \text{“[”} \ f_k \neq (V_k - \{x_k\})\ \text{“]”} \mid \\
\qquad \text{“[”} \ f_1 \geq \{v \in V_1 \mid v < x_1\}\ \text{“]”} \mid \ldots \mid \text{“[”} \ f_k \geq \{v \in V_k \mid v < x_k\}\ \text{“]”} \mid \\
\qquad \text{“[”} \ f_1 > \{v \in V_1 \mid v \leq x_1\}\ \text{“]”} \mid \ldots \mid \text{“[”} \ f_k > \{v \in V_k \mid v \leq x_k\}\ \text{“]”}
\end{array}
$$

Figure 2: Regular Expression for the Set of VL_1 Hypotheses Covering an Instance.

extended to handle base ten numbers.

The mapping of a hypothesis onto a digit string is accomplished by a Moore machine—a DFA that has an output string associated with each state. Recall that the digit string for a term, $a_1 a_2 \ldots a_m$, is $f(a_1) \cdot f(a_1 a_2) \cdot \ldots \cdot f(a_1 a_2 \ldots a_m)$). The machine takes a hypothesis as input. After reading each selector, it outputs the evaluation string for the current partial term. So after seeing a_1, it prints $f(a_1)$. After seeing a_2 it prints $f(a_1 a_2)$, and so on until it has printed the digit string for the term. When the end of the term is encountered (i.e., an *or* symbol is seen), the DFA returns to the initial state and repeats the process for the next term. The evaluation function must return a fixed-length string of digits.

A Moore machine can only have a finite number of states. It needs at least one state for each selector. It must also remember enough about the previous selectors in the term to compute the term's evaluation. Since terms can be arbitrarily long, no finite state machine can remember all of the previous selectors in the term. However, the evaluation function can often get by with much less information.

For example, when the evaluation function is an information metric, the evaluation of a partial term, $a_1 a_2 \ldots a_k$, depends only on the number of positive and negative examples covered by the term. This can be represented by 2^n states, where n is the number of examples. In this case, a state in the Moore machine is an n digit binary number, where the i^{th} digit indicates whether or not the example is covered by the term. In the initial state, all of the examples are covered. When a selector is seen, the digits corresponding to examples that are not covered by the selector are turned off. The binary vector for the state indicates which examples are covered, and the output string for the state is the information corresponding to that coverage of the examples.[1] When an *or* is seen, the DFA prints a zero to indicate end-of-term, and returns to the initial state.

This Moore machine is parameterized by the list of examples and the evaluation function f. This machine is substituted into the regular expression for comparing digit strings. The resulting DFA takes recognizes a pair of hypotheses, $\langle h_1, h_2\rangle$, if and only if h_1 comes before h_2 in the hill climbing search.

Although the machine has an exponential number of states, they do not need to be represented extensionally. All that must be maintained is the current state (an n digit binary number). The next state can be computed from the current state and a selector by determining which examples are not covered by the selector, and turning off those bits. This requires at most $O(n)$ space and $O(mn)$ time to evaluate a hypothesis, where n is the number of examples, and m is the number of selectors in the hypothesis.

The translator for this knowledge source takes as input the hypothesis space, the list of examples, and an evaluation function, f. The function f takes as input the number of covered and uncovered examples, and outputs a fixed length non-negative integer. The translator returns $\langle H, H, P\rangle$, where P is the grammar described above. $\langle H, H, P\rangle$ prefers hypotheses that are visited earlier by hill climbing with evaluation function f. This kind of bias is used in a number of induction algorithms, so this translator can be used for them as well.

Although the logic behind the LEF translator is

[1] Since information is a real between -1 and 1, and the output must be a fixed-length non-negative integer, the output string for a state is the integer portion of $(info + 1.0) * 10^6$, where *info* is the information of the example partitioning represented by the n digit number for that state.

rather complex, the translator itself is fairly straightforward to write. The Moore machine requires only a handful of code to implement the next-state and output functions, and the digit-string comparison grammar is a simple regular expression. The design effort also transfers to other biases. The evaluation function can be changed, so long as it only needs to know which examples are covered by the current term, and the basic design can be reused for translators of similar biases.

Some of the difficulty in designing the LEF translator may be because the bias is designed for use in a hypothesis space search paradigm, and does not translate well to RS-KII. Bear in mind that the beam-search is an approximation of another bias, namely that the induced hypothesis should maximize the LEF. Finding a maximal hypothesis is intractable, so AQ-11 approximates it with a beam search. This particular approximation was chosen because it is easy to implement in the hypothesis-space search paradigm. However, RS-KII uses a different paradigm, so a different approximation of the "maximize the LEF" bias that is easier to express in RS-KII may be more appropriate.

Translators for Novel Biases

The following translators are for biases that AQ-11 does not utilize, namely consistency with one class of noisy examples, and an assumption that the target hypothesis is a specialization of an overgeneral domain theory.

Noisy Examples with Bounded Inconsistency Bounded inconsistency (Hirsh 1990) is a kind of noise in which each feature of the example can be wrong by at most a fixed amount. For example, if the width value for each instance is measured by an instrument with a maximum error of ±0.3mm, then the width values for these instances have bounded inconsistency.

The idea for translating examples with bounded inconsistency is to use the error margin to work backwards from the noisy example to compute the set of possible noise-free examples. One of these examples is the correct noise-free version of the observed example, into which noise was introduced to produce the observed noisy example. The target concept is strictly consistent with this noise-free example.

Let e be the noisy observed example, E be the set of noise-free examples from which e could have been generated, and let e' be the correct noise-free example from which e was in fact generated. Since it is unknown which example in E is e', a noisy example is translated as $\langle H, C, \emptyset \rangle$, where C is the set of hypotheses that are strictly consistent with one or more of the examples in E. Hypotheses that are consistent with none of the examples in E are not consistent with e', and therefore not the target concept. This is the approach used by Hirsh (Hirsh 1990) in IVSM to translate noisy examples with bounded inconsistency.

$$\begin{aligned} &\mathit{TranPosExampleBI}(H, \langle \delta_1, \delta_2, \ldots, \delta_k \rangle, \\ &\qquad \langle x_1, x_2, \ldots, x_k \rangle) \rightarrow \langle H, C, P \rangle \\ E &= [x_1, \pm\delta_1] \times [x_2, \pm\delta_2] \times \ldots [x_k, \pm\delta_k] \\ &\quad \text{where } [x_i, \pm\delta_i] = \{v \mid x_i - \delta_i \leq v \leq x_i + \delta_i\} \\ C &= \bigcup_{e_i \in E} C_i \text{ s.t. } \langle C_i, \emptyset \rangle = \mathit{TranPosAQExample}(H, e_i) \end{aligned}$$

Figure 4: RS-KII Translator for Positive Examples with Bounded Inconsistency.

This suggests the following RS-KII translator for examples with bounded inconsistency. The set of possible noise-free examples, E, is computed from the noisy examples and the error margins for each feature. Each example, e_i, in this set is translated using one of the RS-KII translators for noise-free examples—either $\mathit{TranPosAQExample}(H, e_i)$ or $\mathit{TranNegAQExample}(H, e_i)$—which translates example e_i into $\langle H, C_i, \emptyset \rangle$. C_i is the set of hypotheses that are strictly consistent with e_i. The translator for the bounded inconsistent example returns $\langle C = \bigcup_{i=1}^{|E|} C_i, \emptyset \rangle$. C is the set of hypotheses consistent with at least one of the examples in E.

The set E is computed from the observed example, $\langle x_1, x_2, \ldots, x_k \rangle$, and the error margins for each feature, $\pm\delta_1$ through $\pm\delta_k$, as follows. If the observed value for feature f_i is x_i, and the error margin is $\pm\delta_i$, then the correct value for feature f_i is in $\{v \mid x_i - \delta_i \leq v \leq x_i + \delta_i\}$. Call this set $[x_i, \pm\delta_i]$ for short. Since instances are ordered vectors of feature values, E is $[x_1, \pm\delta_1] \times [x_2, \pm\delta_2] \times \ldots \times [x_k, \pm\delta_k]$.

A translator for examples with bounded inconsistency based on this approach is shown in Figure 4. It takes as input a VL_1 hypothesis space (H), the error margin for each feature ($\pm\delta_1$ through $\pm\delta_k$) and an instance. Negative examples are translated similarly, except that $\mathit{TranNegAQExample}(H, e_i)$ is used.

Domain Theory A domain theory encodes background knowledge about the target concept as a collection of horn-clause inference rules that explain why an instance is a member of the target concept. The way in which this knowledge biases induction depends on assumptions about the correctness and completeness of the theory. Each of these assumptions requires a different translator, since the biases map onto different constraints and preferences.

A translator for a particular overgeneral domain theory is described below. The theory being translated is derived from the classic "cup" theory (Mitchell, Keller, & Kedar-Cabelli 1986; Winston *et al.* 1983), and is shown in Figure 5. It expands into a set of sufficient conditions for `cup(X)`, as shown in Figure 6. The translator assumes that the target concept is a specialization of the theory. In this case, the actual target concept

```
cup(X)          :-  hold_liquid(X), liftable(X),
                    stable(X), drinkfrom(X).
hold_liquid(X)  :-  plastic(X) | china(X) | metal(X).
liftable(X)     :-  small(X), graspable(X).
graspable(X)    :-  small(X), cylindrical(X) |
                    small(X), has_handle(X).
stable(X)       :-  flat_bottom(X).
drinkfrom(X)    :-  open_top(X).
```

Figure 5: CUP Domain Theory.

```
1. cup(X)  :-  plastic(X), small(X), cylindrical(X),
               flat_bottom(X), open_top(X).
2. cup(X)  :-  china(X), small(X), cylindrical(X),
               flat_bottom(X), open_top(X).
3. cup(X)  :-  metal(X), small(X), cylindrical(X),
               flat_bottom(X), open_top(X).
4. cup(X)  :-  plastic(X), small(X), has_handle(X),
               flat_bottom(X), open_top(X).
5. cup(X)  :-  metal(X), small(X), has_handle(X),
               flat_bottom(X), open_top(X).
6. cup(X)  :-  china(X), small(X), has_handle(X),
               flat_bottom(X), open_top(X).
```

Figure 6: Sufficient Conditions of the CUP Theory.

is "plastic cups without handles," which corresponds to condition one, but this information is not provided to the translator. All the translator knows is that the target concept can be described by a disjunction of one or more of the sufficient conditions in the cup theory.

The translator takes the theory and hypothesis space as input, and generates the tuple $\langle H, C, \{\}\rangle$, where C is satisfied by hypotheses equivalent to a disjunct of one or more of the theory's sufficient conditions. In general, the hypothesis space language may differ from the language of the conditions, making it difficult to determine equivalence. However, for the VL_1 language of AQ-11, the languages are similar enough that simple syntactic equivalence will suffice, modulo a few cosmetic changes. Specifically, the predicates in the sufficient conditions are replaced by corresponding selectors. All disjuncts of the resulting conditions are VL_1 hypotheses. The mappings are shown in Figure 7. In general, the predicates are Boolean valued, and are replaced by Boolean valued selectors. To show that other mappings are also possible, the predicate `small(x)` is replaced by the selector `[size ≤ 5]`.

The grammar for C is essentially the grammar for the cup theory, with a few additional rules. First, the cup theory is written as a context free grammar that generates the sufficient conditions. If the grammar does not have certain kinds of recursion, as is the case in the CUP theory, then it is in fact a regular grammar. In this case, the grammar for C will also be regular. Otherwise, the grammar for C will be context free. This limits the theories that can be utilized by RS-KII. However, RS-KII could be extended to utilize

```
plastic(x)      →  [plastic = true]
china(x)        →  [china = true]
metal(x)        →  [metal = true]
has_handle(x)   →  [has_handle = true]
cylindrical(x)  →  [cylindrical = true]
small(x)        →  [size ≤ 5]
flat_bottom(x)  →  [flat_bottom = true]
open_top(x)     →  [open_top = true]
```

Figure 7: Selectors Corresponding to Predicates in CUP Theory.

```
C               →  TERM | C or TERM
TERM            →  CONDITION
CONDITION       →  CUP(X)

PLASTIC(X)      →  [plastic = true]
CHINA(X)        →  [china = true]
METAL(X)        →  [metal = true]
HAS_HANDLE(X)   →  [has_handle = true]
CYLINDRICAL(X)  →  [cylindrical = true]
SMALL(X)        →  [size ≤ 5]
FLAT_BOTTOM(X)  →  [flat_bottom = true]
OPEN_TOP(X)     →  [open_top = true]
```

Figure 8: Grammar for VL_1 Hypotheses Satisfying the CUP Theory Bias.

a context free theory by allowing the C set of at most one $\langle H, C, P\rangle$ tuple to be context free. This would be a different instantiation of KII, but still within the expressiveness limits discussed in the previous section.

Once the theory has been written as a grammar, rewrite rules are added that map each terminal predicate (those that appear in the sufficient conditions) onto the corresponding selector(s). This grammar generates VL_1 hypotheses equivalent to each of the sufficient conditions. To get all possible disjuncts, rules are added that correspond to the regular expression CONDITION (*or* CONDITION)*, where CONDITION is the head of the domain-theory grammar described above.

The grammar for C discussed above is shown in Figure 8. This grammar is a little less general than it could be, since it does not allow all permutations of the selectors within each term. However, the more general grammar contains considerably more rules, and permuting the selectors does not change the semantics of a hypothesis. In the following grammar, the nonterminal CUP(X) is the head of the cup domain-theory grammar, which has the same structure as the theory shown in Figure 5.

Enumerating the Solution Set

The solution set is a regular grammar computed from C and P, as was shown in Equation 2. A regular grammar is equivalent to a deterministic finite automaton

(DFA). One straightforward way to enumerate a string from the solution set is to search the DFA for a path from the start state to an accept state. However, the DFA computed by the solution-set equation from C and P can contain *dead* states, from which there is no path to an accept state. These dead states can cause a large amount of expensive backtracking.

There is a second approach that can reduce backtracking by making better use of the dominance information in P. The solution set consists of the undominated strings in C, where P is the dominance relation. Strings in this set can be enumerated by searching C with branch-and-bound (Kumar 1992). The basic branch-and-bound search must be modified to use a partially ordered dominance relation rather than a totally ordered one, and to return multiple solutions instead of just one. These modifications are relatively straightforward, and are described in (Smith 1995).

Although the worst-case complexity of branch-and-bound is the same as a blind search of the solution-set DFA, the complexity of enumerating the first few hypotheses with branch-and-bound can be significantly less. Since for most applications only one or two hypotheses are ever needed, RS-KII uses branch-and-bound.

Results

By combining biases, different induction algorithms can be generated. AQ-11 uses the biases of strict consistency with examples, and prefers hypotheses that maximize the LEF. When using only these biases, both RS-KII and AQ-11 with a beam width of one induce the same hypotheses, though RS-KII is slightly more computationally expensive. The complexity of AQ-11 with a beam-size of one is $O(e^4k)$, where e is the number of examples and k is the number of features. The complexity of RS-KII when using only AQ-11 biases is $O(e^5k^2)$. These derivations can be found in (Smith 1995), and generally follow the complexity derivations for AQ-11 in (Clark & Niblett 1989). RS-KII is a little more costly because it assumes that the LEF bias, encoded by P, is a partial order, where it is in fact a total order. This causes RS-KII to make unnecessary comparisons that AQ-11 avoids. One could imagine a version of RS-KII which used information about whether P was a total order or a partial order.

RS-KII's strength lies in its ability to utilize additional knowledge, such as the domain theory and noisy examples with bounded inconsistency. When the domain theory translator is added, RS-KII's complexity drops considerably, since the hypothesis space is reduced to a relative handful of hypotheses by the strong bias of the domain theory. The concept induced by RS-KII is also more accurate than that learned by AQ-11, which cannot utilize the domain theory. When given the four examples of the concept "plastic cups without handles," as shown in Table 3, AQ-11 learns the overgeneral concept

ID	class	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
e_1	+	t	f	f	f	t	5	t	t
e_2	+	t	f	f	f	t	3	t	t
e_3	−	f	t	f	f	t	4	t	t
e_4	−	t	f	f	t	t	1	t	t

f_1	plastic	f_5	has_handle
f_2	china	f_6	size
f_3	metal	f_7	flat_bottom
f_4	cylindrical	f_8	open_top

Table 3: Examples for the CUP Task.

```
[plastic = true] [cylindrical = true]
```

which includes many non-cups, whereas RS-KII learns the correct concept:

```
[plastic = true] [cylindrical = true]
[size ≤ 5] [flat_bottom = true]
[open_top = true]
```

The additional bias from the domain theory makes this the shortest concept consistent with the four examples.

RS-KII can also handle noisy examples with bounded inconsistency. For the cup domain, assume that the size can be off by at most one. Let the size feature of example e_2 be six instead of five. AQ-11 would fail to induce a hypothesis at all, since there is no hypothesis consistent with all four examples. When using the bounded-inconsistency translator for examples, RS-KII *can* induce a hypothesis, namely the same one learned above with noise-free examples. In general, noisy examples introduce uncertainty, which can increase the size of the solution set and decrease the accuracy of the learned hypothesis. Additional knowledge may be necessary to mitigate these effects. In this case, however, the domain theory bias is sufficiently strong, and the noise sufficiently weak, that no additional knowledge is needed.

The ability to utilize additional knowledge allows RS-KII to induce hypotheses in situations where AQ-11 cannot, and allows RS-KII to induce more accurate hypotheses. RS-KII can also make use of knowledge other than those shown here by writing appropriate translators.

Precursors to KII

KII has its roots in two knowledge integration systems, Incremental Version Space Merging (Hirsh 1990), and Grendel (Cohen 1992). These systems can also be instantiated from KII, given appropriate set representations. These systems and their relation to KII are described below.

IVSM. Incremental Version Space Merging (IVSM) (Hirsh 1990) was one of the first knowledge integration

systems for induction, and provided much of the motivation for KII. IVSM integrates knowledge by translating each knowledge fragment into a version space of hypotheses consistent with the knowledge, and then intersecting these version spaces to obtain a version space consistent with all of the knowledge. Version spaces map onto $\langle H, C, P \rangle$ tuples in which C is a version space in the traditional $[S, G]$ representation, and P is the empty set (i.e., no preference information).

KII expands on IVSM by extending the space of set representations from the traditional [S,G] representation—and a handful of alternative representations (e.g., (Hirsh 1992; Smith & Rosenbloom 1990; Subramanian & Feigenbaum 1986))—to the space of all possible set representations. KII also expands on IVSM by allowing knowledge to be expressed in terms of preferences as well as constraints, thereby increasing the kinds of knowledge that can be utilized. KII strictly subsumes IVSM, in that IVSM can be cast as an instantiation of KII in which C is a version space one of the possible representations, and P is expressed in the *null representation*, which can only represent the empty set.

Grendel. Grendel (Cohen 1992) is another cognitive ancestor of KII. The motivation for Grendel is to express biases explicitly in order to understand their effect on induction. The biases are translated into a context free grammar representing the biased hypothesis space.[2] This space is then searched for a hypothesis that is strictly consistent with the examples, under the guidance of an information gain metric. Some simple information can also be encoded in the grammar.

Grendel cannot easily integrate new knowledge. Context free grammars are not closed under intersection (Hopcroft & Ullman 1979), so it is not possible to generate a grammar for the new knowledge and intersect it with the existing grammar. Instead, a new grammar must be constructed for all of the biases. KII can use set representations that are closed under intersection, which allows KII to add or omit knowledge much more flexibly than Grendel. KII also has a richer language for expressing preferences. Grendel-like behavior can be obtained by instantiating KII with a context free grammar for C.

Future Work

One prime area for future work is constructing RS-KII translators for other biases and knowledge sources, especially those used by other induction algorithms. This is both to extend the range of knowledge available to RS-KII, and to test the limits of its expressiveness with respect to existing algorithms.

A second area is investigating the naturalness of the $\langle H, C, P \rangle$ representation. In RS-KII, some of the knowledge in AQ-11 is easy to express as $\langle H, C, P \rangle$ tuples, but some, such as the LEF, is more awkward. Others, such as the beam search bias, cannot be expressed at all in RS-KII. One approach is to replace this hard-to-express knowledge with knowledge that achieves similar effects on induction, but is easier to express. Similar approaches are used implicitly in existing algorithms for knowledge that cannot be easily used by the search. For example, AQ11 approximates a bias for the best hypothesis with a beam search that finds a locally maximal hypothesis.

Finally, the space of set representations should be investigated further to find representations that will yield other useful instantiations of KII. In particular, it would be worth identifying a set representation that can integrate n knowledge fragments and enumerate a hypothesis from the solution set in time polynomial in n. This would provide a tractable knowledge integration algorithm. Additionally, the set representation for the instantiation effectively defines a class of knowledge from which hypotheses can be induced in polynomial time. This would complement the results in the PAC literature, which deal with polynomial-time learning from examples only (e.g.,(Vapnik & Chervonenkis 1971), (Valiant 1984), (Blummer *et al.* 1989)).

Conclusions

Integrating additional knowledge is one of the most powerful ways to increase the accuracy and reduce the cost of induction. KII provides a uniform mechanism for doing so. KII also addresses an apparently inherent trade-off between the breadth of knowledge utilized and the cost of induction. KII can vary the trade-off by changing the set representation. RS-KII is an instantiation of KII with regular sets that shows promise for being able to integrate a wide range of knowledge and related strategies, thereby creating hybrid multistrategy algorithms that make better use of the available knowledge. One such hybridization of AQ-11 was demonstrated. Other instantiations of KII may provide similarly useful algorithms, as demonstrated by IVSM and Grendel.

Acknowledgments

Thanks to Haym Hirsh for many helpful discussions during the formative stages of this work. This paper describes work that was supported by the National Aeronautics and Space Administration (NASA Ames Research Center) under cooperative agreement number NCC 2-538, and by the Information Systems Office of the Advanced Research Projects Agency (ARPA/ISO) and the Naval Command, Control and Ocean Surveillance Center RDT&E Division (NRaD) under contract number N66001-95-C-6013, and partially supported by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

[2] More precisely, they are are expressed as an antecedent description grammar.

References

Blummer, A.; Ehrenfect, A.; Haussler, D.; and Warmuth, M. 1989. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery* 36(4):929–965.

Chomsky, N. 1959. On certain formal properties of grammars. *Information and Control* 2.

Clark, P., and Niblett, T. 1989. The CN2 induction algorithm. *Machine Learning* 3(?):261–283.

Cohen, W. W. 1992. Compiling prior knowledge into an explicit bias. In Sleeman, D., and Edwards, P., eds., *Machine Learning: Proceedings of the Ninth International Workshop*, 102–110.

Hirsh, H. 1990. *Incremental Version Space Merging: A General Framework for Concept Learning.* Boston, MA: Kluwer Academic Publishers.

Hirsh, H. 1992. Polynomial-time learning with version spaces. In *AAAI-92: Proceedings, Tenth National Conference on Artificial Intelligence*, 117–122.

Hopcroft, J. E., and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley.

Kumar, V. 1992. Search, branch and bound. In *Encylopedia of Artificial Intelligence.* John Wiley & Sons, Inc., second edition. 1000–1004.

Michalski, R. 1974. Variable-valued logic: System VL_1. In *Proceedings of the Fourth International Symposium on Multiple-Valued Logic.*

Michalski, R. 1978. Selection of most representative training examples and incremental generation of VL_1 hypotheses: The underlying methodology and the descriptions of programs ESEL and AQ11. Technical Report 877, Department of Computer Science, University of Illinois, Urbana, Illinois.

Mitchell, T.; Keller, R.; and Kedar-Cabelli, S. 1986. Explanation-based generalization: A unifying view. *Machine Learning* 1:47–80.

Mitchell, T. 1982. Generalization as search. *Artificial Intelligence* 18(2):203–226.

Quinlan, J. 1986. Induction of decision trees. *Machine Learning* 1:81–106.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning* 5:239–266.

Rosenbloom, P. S.; Hirsh, H.; Cohen, W. W.; and Smith, B. D. 1993. Two frameworks for integrating knowledge in induction. In Krishen, K., ed., *Seventh Annual Workshop on Space Operations, Applications, and Research (SOAR '93)*, 226–233. Houston, TX: Space Technology Interdependency Group. NASA Conference Publication 3240.

Russell, S., and Grosof, B. 1987. A declarative approach to bias in concept learning. In *Sixth national conference on artificial intelligence*, 505–510. Seattle, WA: AAAI.

Smith, B., and Rosenbloom, P. 1990. Incremental non-backtracking focusing: A polynomially bounded generalization algorithm for version spaces. In *Proceedings of the Eighth Naional Conference on Artificial Intelligence*, 848–853. Boston, MA: AAAI.

Smith, B. 1995. *Induction as Knowledge Integration.* Ph.D. Dissertation, University of Southern California, Los Angeles, CA.

Subramanian, D., and Feigenbaum, J. 1986. Factorization in experiment generation. In *Proceedings of the National Conference on Artificial Intelligence*, 518–522.

Valiant, L. 1984. A theory of the learnable. *Communications of the ACM* 27(11):1134–1142.

Vapnik, V., and Chervonenkis, A. 1971. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications* 16(2):264–280.

Winston, P.; Binford, T.; Katz, B.; and Lowry, M. 1983. Learning physical descriptions from functional definitions, examples, and precedents. In *Proceedings of the National Conference on Artificial Intelligence*, 433–439. Washington, D.C.: AAAI.