PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT. 



# Efficient Induction of Version Spaces through Constrained Language Shift

Claudio Carpineto

Fondazione Ugo Bordoni
Via Baldassarre Castiglione 59, 00142 - Rome, ITALY
fubdpt5@itcaspur.bitnet

## Abstract

A large hypothesis space makes the version space approach, like any other concept induction algorithm based on hypothesis ordering, computationally inefficient. Working with smaller composable concept languages rather than one large concept language is one way to attack the problem, in that it allows us to do part of the induction job within the more convenient languages and move to the less convenient languages when necessary. In this paper we investigate the use of multiple concept languages in a version space approach. We define a graph of languages ordered by the standard set inclusion relation, and provide a procedure for efficiently inducing version spaces while shifting from small to larger concept languages. We apply this method to the attribute languages of a typical conjunctive concept language (i.e., a conjunctive concept language defined on a tree-structured attribute-based instance space) and compare its complexity to that of a standard version space algorithm applied to the full concept language. Finally we contrast our approach with other work on language shift, outlining an alternative highly-constrained strategy for searching the space of new concepts which is *not* based on constructive operators.

## 1 Introduction

Of all the algorithms for incremental concept induction that are based on the partial order defined by generality over the concept space, the candidate elimination (CE) algorithm [Mitchell 1982] is the best known exemplar. The CE algorithm represents and updates the set of all concepts that are consistent with data (i.e. the version space) by maintaining two sets, the set S containing the maximally specific concepts and the set G containing the maximally general concepts. The procedure to update the version space is as follows. A positive example prunes concepts in G which do not cover it and causes all concepts in S which do not cover the example to be generalized just enough to cover it. A negative example prunes concepts in S that cover it and causes all concepts in G that cover the example to be specialized just enough to exclude it. As more examples are seen, the version space shrinks; it may eventually reduce to the target concept provided that the concept description language is consistent with the data.

This framework has been later improved along several directions. The first is that of incorporating the domain knowledge available to the system in the algorithm; this has resulted in feeding the CE algorithm with analytically-generalized positive examples (e.g., [Hirsh 1989], [Carpineto 1990]), and analytically-generalized negative examples (e.g., [Carpineto 1991]). Another research direction is to relax the assumption about the consistency of the concept space with data. In fact, like many other learning algorithms, the CE algorithm uses a restricted concept language to incorporate bias and focus the search on a smaller number of hypotheses. The drawback is that the target concept may be contained in the set of concepts that are inexpressible in the given language, thus being unlearnable. In this case the sets S and G become empty: to restore consistency the bias must be weakened adding new concepts to the concept language [Utgoff 1986]. Thirdly, the CE algorithm suffers from lack of computational efficiency, in that the size of S and G can be exponential in the number of examples and the number of parameters describing the examples [Haussler 1988]. Changes to the basic algorithm have been proposed that improve efficiency for some concept language [Smith and Rosenbloom 1990].

In this paper we investigate the use of *multiple* concept languages in a version space approach. By organizing the concept languages into a graph corresponding to the relation *larger-than* implicitly defined over the sets of concepts covered by the languages, we have a framework that allows us to shift from small to larger concept languages in a controlled manner. This provides a powerful basis to apply a general divide-and-conquer strategy to improve the efficiency of a standard version space approach in which the concept description language is factorizable. The idea is to start out with the smallest concept languages (i.e., the factor languages) and, once the version spaces induced over them have become inconsistent with the data, to move along the graph of product languages to the maximally small concept languages that restore consistency. Working with smaller concept languages may greatly reduce the size of S and G, thus resulting in a neat improvement in efficiency. On the other hand, use of several languages in parallel and language shifts negatively affect complexity. Therefore the two main objectives of the paper are : (1) define a set of languages and a procedure for inducing version spaces after any language shift efficiently, (2) show that in some cases this method may be applied to reduce the complexity of the standard CE algorithm. Since this framework supports version-space induction over a set of concept languages, it can also be suitable to handle inconsistency when the original concept language is too small. More generally, it suggest an alternative approach to inductive language shift in which the search for useful concepts to be added to the concept language is not based on constructive operators. This aspect is also discussed in the paper.

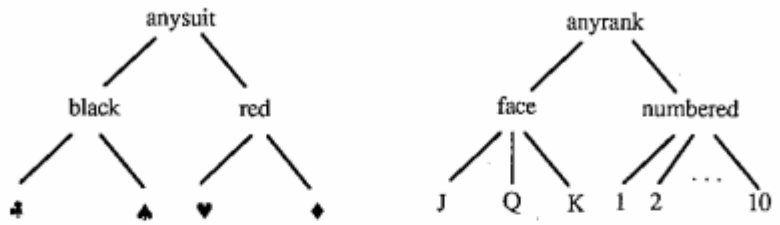


Fig. 1. Two concept languages in the playing cards domain.

The rest of the paper is organized as follows. In the next section we define a graph of conjunctive concept languages and describe the learning problem with respect to it. Then we present the learning method. Next, we apply the method to the factor languages of a conjunctive concept language defined on a tree-structured attribute-based instance space, and evaluate its utility. Finally we compare this work to other approaches to factorization in concept induction and to inductive language shift.

## 2 The learning problem

We first introduce the notions that characterize our learning problem. In the following concepts are viewed as sets of instances and languages as sets of concepts.

A concept $c_1$ is *more general than* a concept $c_2$ if the set of instances covered by $c_1$ is a proper superset of the set of instances covered by $c_2$.

A language $\mathbf{L}_1$ is *larger than* a language $\mathbf{L}_2$ if the set of concepts expressible in $\mathbf{L}_1$ is a proper superset of the set of concepts expressible in $\mathbf{L}_2$.

In the playing cards domain, which we shall use as an illustration, two possible concept languages are: $\mathbf{L}_1$ = {anysuit, black, red, ♣, ♠, ♥, ♦} and $\mathbf{L}_2$ = {anyrank, face, numbered, J, Q, K, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. The relation more-general-than over the concepts present in each language is shown in fig1.

The *product* $\mathbf{L}_{1,2}$ of two factor languages $\mathbf{L}_1$ and $\mathbf{L}_2$ is the set of concepts formed from the conjunctions of concepts from $\mathbf{L}_1$ and $\mathbf{L}_2$ (examples of product concepts are 'anyrank-anysuit', 'anyrank-black', etc). The number of concepts in the product language is therefore the product of the number of concepts in its factors. Also, a concept $c_{c1',c2'}$ in the language $\mathbf{L}_{1,2}$ is more general than (>) another concept $c_{c1'',c2''}$ if and only if $c_1' > c_1''$ and $c_2' > c_2''$.

With n initial languages it is possible to generate $\sum_{k=1,n} n! / (n - k)!\, k! = 2^n - 1$ product languages (see fig. 2). Moreover, given that the superconcept 'any' can always be added to each factor language, the relation larger than over this set of languages can be immediately established, for each product language is larger than any of its factor languages.

The learning problem can be stated as follows.

Given
A set of factor concept languages
A set of positive instances.
A set of negative instances.

Incrementally Find
The version spaces in the set of product concept languages that are consistent with data and that contain the smallest number of factors.

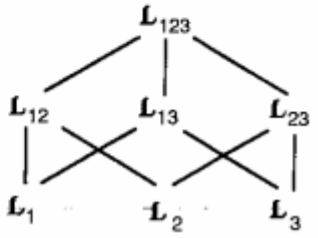


Fig. 2. The graph of product languages with three factor languages.

## 3. The learning method

In this approach concept learning and language shift are interleaved. We process one instance at a time, using a standard version space approach to induce consistent concepts over *each* language of the current set (initially, the n factor languages). During this inductive phase some concept languages may become inconsistent with the data. When every member of the current set of languages has become inconsistent with data, the language shifting algorithm is invoked. It iteratively selects the set of maximally small concept languages that are larger than the current ones (i.e. the two-factored languages, the three-factored languages, etc.) and computes the new version spaces in these languages. It halts when it finds a consistent set of concept languages (i.e. a set in which there is at least one consistent concept language); then it returns control to the inductive algorithm to process additional examples.The whole process is iterated as long as the set of current languages can be further specialised (i.e. until the n-factored language has been generated). We call this algorithm Factored Candidate Elimination (FCE) algorithm. The top-level FCE algorithm is presented in table 1.

The core of the algorithm is the procedure to find the new consistent version spaces in the product languages (in italics in table 1). The difficulty is that the algorithm for inducing concepts over a language (the inductive algorithm) is usually distinct from the algorithm for adding new terms to the language itself (the language-shifting algorithm). In

Table 1: The top-level FCE algorithm

| | |
|---|---|
| Input: | An instance set {I}.<br>A set of partially ordered concept languages {L} formed by n given one-factored languages and their products. |
| Output: | The version spaces in the set of languages {L} that are consistent with {I} and that contain the smallest number of factors. |
| Variables: | $\{L_S\}_k$ is the subset of (unordered) languages in {L} which have k factors.<br>$\{VS\}_k$ is a set of version spaces, with $\|VS_k\| = \|L_{Sk}\|$.<br>$\{L_S,VS\}_k$ is the set of pairs obtained pairing the corresponding elements in $\{L_S\}_k$ and $\{VS\}_k$. |
| Function: | CE(i,l,vs) takes an instance, a concept language and a version space and returns the updated version space. |

```
FCE({I},{L})
        K=1.
        {VS}1 = {LS}1.
        For each instance i in {I},
                For each (ls,vs) in {LS,VS}k
                        vs = CE(i,ls,vs).
                If all the version spaces in {VS}k are empty
                        Then  Repeat
                                If K=n
                                    Then Return failure
                                K=K+1.
                                For each ls in {LS}k,
                                    find the new version space vs associated with it.
                              Until at least one vs is not empty.
```

general, the inductive algorithm has to be run again over the instance set after any change made by the language-shifting algorithm ([Utgoff 1986], [Matheus and Rendell 1989], [Pagallo 1989], [Wogulis and Langley 1989]). In this case, however, in defining the procedure to induce the new consistent concepts after any language shift, we take advantage of the features of the particular inductive learning algorithm considered (i.e. the CE algorithm) and of the properties of language "multiplication". The two key facts are that the CE algorithm makes an explicit use of concept ordering and that concepts in any product language preserve the order of concepts in its factors. This makes it possible to modify the basic CE algorithm with the aim of computing the set of consistent concepts in a product language as a function of some appropriate concept sets induced in its factors.

The concept sets computed in each factor language which will be utilized during language shift are the following. First, for each language we compute the set S*. S* contains the most specific concepts in the language that cover *all* positive examples, regardless of whether or not they include any negative examples. Second, for each language *and each negative example*, we compute the set G*. G* contains the most general concepts in the language that do not cover the negative example, regardless of whether or not they include all positive examples.

These operations can be better illustrated with an example. Let us consider again the playing cards domain and suppose that we begin with the two concept languages introduced above - rank ($L_1$) and suit ($L_2$). Let us suppose the system is given one positive example - the Jack of spades - and two negative examples - the Jack of hearts and the Two of spades. We compute the two corresponding version spaces (one for each language), the sets S* (one for each language), and the sets G* (one for each language and for each negative example) in parallel. In particular, the sets S* and G* can be immediately determined, given the ordering over each language's members. The inductive phase is pictured in fig.3 (f stands for face, b for black,etc).

The three instances cause both of the version spaces to reduce to the empty set. The next step is therefore to shift to the set of maximally small concept languages that are larger than $L_1$ and $L_2$ (in this case the product $L_{12}$) and check to see if it contains any concepts consistent with data. The problem of finding the version space in the language $L_{12}$ can be subdivided into the two tasks of finding the lower boundary set $S_{12}$ (i.e. the set of the most specific concepts in $L_{12}$ that are consistent with data) and the upper boundary set $G_{12}$ (i.e. the set of the most general concepts in $L_{12}$ that are consistent with data).

Computation of $S_{12}$

Because a product concept contains an instance if and only if all of its factor concepts contain the instance, the product of $S_1$* and $S_2$* returns the most specific factor concepts that include all positive instances. By discarding those that also cover negative examples, we get just the set $S_{12}$. If the set becomes empty, then the product language is also inconsistent with the data. More specific concepts, in fact, cannot be consistent because they would rule out some positive example. More general concepts cannot be consistent either, for they would cover some negative examples. In our example, as there is only one positive example, the result is trivial : $S_{12} = \{J\spadesuit\}$.

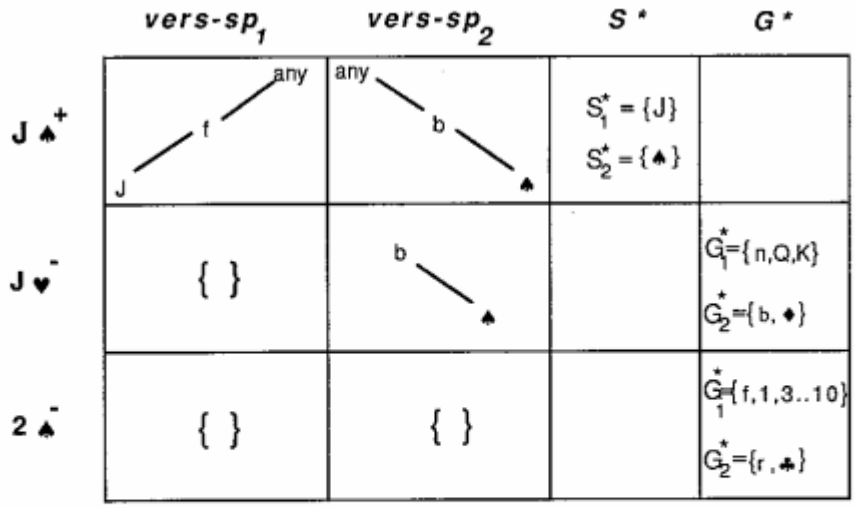


Fig. 3. Concept sets computed during the inductive phase.

Computation of G12

Rather than generating and testing for consistency all the product concepts more general than the members of $S_{12}$, the set $G_{12}$ is computed using the sets G* As for each negative example there must be *at least one* factor concept in each consistent product concept which does not cover the negative example, and because we seek the maximally general consistent product concepts, the idea is to use the members of the sets G* as upper bounds to find the factor concepts present in such maximally general product concepts.

The algorithm is as follows. It begins by dropping from the sets G* the elements that cannot generate factor concepts that are more general than those contained in $S_{12}$. Then, it (a) finds all the conjunctions of concepts in the reduced sets G* such that each negative instance is ruled out by at least one concept, and (b) checks if there are more general consistent conjunctions. Step (a) requires conjoining each factor concept in each G* (it will rule out at least one negative example) with all the combinations of factor concepts in the other G*'s which rule out the remaining negative examples. Step (b) requires generalising (with the value 'any') the factor concepts in the conjunctions found at the end of step (a) which do not contribute to rule out any negative example.The resulting set of conjunctions, if any is found, coincides with the set $G_{12}$, in that there cannot be more general product concepts consistent with data. However, it may not be possible to find a consistent concept conjoining the members of the G*'s. In this case we are forced to specialise the members of the G*'s to the extent required so that they rule out more negative instances, and to iterate the procedure (in the limit, we will get the set $S_{12}$).

In our example there are just two factors and only two negative instances. The initial sets G* are:

| | | |
|---|---|---|
| J ♥ - | {n, Q, K} | {b, ♦} |
| 2 ♠ - | {f, 1, 3,.., 10} | {r, ♣} |

The simplification with the set $S_{12}$ returns:

| | | |
|---|---|---|
| J ♥ - | { } | {b} |
| 2 ♠ - | {f} | { } |

Step (a) in this case reduces to the *union of* the conjunction of $G_1$* relative to instance 1 and $G_2$* relative to instance 2 *and* the conjunction of $G_1$* relative to instance 2 and $G_2$* relative to instance 1. The result ({fb}) does not need be generalized (step (b)) for both 'f' and 'b' contribute to rule out (at least) one negative example. Also, in this case, the specialisation procedure is not needed because we have been able to find a consistent conjunction: $G_{12}$={fb}. The overall version space in the language $\mathbf{L}_{12}$ is shown in fig.4.

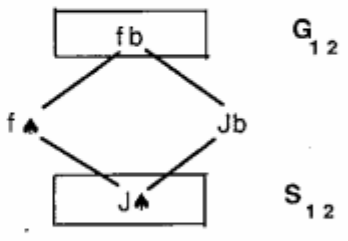


Fig.4. The version space in the product language after the constructive phase.

## 4 Evaluation

There are two ways in which the factored CE algorithm (FCE) can be used to reduce the complexity of the standard CE algorithm. Either we use a graph-factoring algorithm [Subramanian and Feigenbaum, 1986] to find the factors of a given concept space (provided that it is factorable), or we choose a concept language that can be naturally decomposed into factor languages. Here we evaluate the utility of the FCE algorithm with respect to a simple but widely used concept language that has this property. We consider a conjunctive concept language defined on a tree-structured attribute-based instance space. We assume the number of attributes be $n$, each attribute with $l$ levels and branching factor $b$ (the case can be easily extended to nominal and linear attributes, considering that a nominal attribute can be converted in a tree-structured attribute using a dummy root 'any-value', and that a linear attribute can be considered as a tree-structured attribute with branching factor = 1). Each term of the concept space is a conjunction of n values, one for each attribute; the total number of terms in the concept space is $[(b^l - 1) / (b - 1)]^n$. It is worth noting that with such a concept language the set S of the version space will never contain more than one element [Bundy *et al.* 1985]. Even in this case, however, Haussler [1988] has shown that the size of the set G can still be exponential, due to its fragmentation.

In the following we compare the CE algorithm applied to this full conjunctive concept language to the FCE algorithm applied to its attribute languages. While their relative performances are equivalent , in that in order to find all the concepts consistent with data in the full concept language it suffices to eventually compute the boundaries of the n-factored version space, their time complexity may strongly vary. The gain/loss in efficiency ultimately depends on the number of instances that each intermediate language is able to account for before it becomes inconsistent. In the best case all the induction is done within the smallest languages,and language shift to larger languages is not necessary. In the worst case no consistent concepts are induced in the smaller languages, so that all the induction is eventually done within the full concept language.

To make a quantitative assessment we have to make assumptions about a number of factors in addition to the structure of factor and product languages, including target concept location, training instance distribution, cost of matching concepts to training instances. We consider the *worst case* convergency to the target concept in the full concept language. This amounts to say that after the first positive instance (the first instance must be positive in the CE algorithm) there are only negative instances, and that each of them causes only one concept to be removed from the version space until it shrinks to the target concept (i.e., the first positive instance). In terms of the full concept language ordering this means that general concepts are removed earlier than any of their more specific concepts. Furthermore, we assume that the generality of the attribute values in the concepts dropped from the version space decreases uniformly. More precisely, we assume that if an attribute value in a dropped concept is placed at level k in the corresponding attribute tree, then the values of that attribute in the remaining consistent concepts are placed at most at level k+1. This presentation of training instances has the effect of maximizing the amount of instances that each intermediate language can take in before it becomes inconsistent.

As for the cost of matching concepts to instances and other concepts we assume that it is the same in all languages.

We can now analyse the complexity in the two approaches. As done in [Mitchell 1982], the time complexity bounds indicate bounds on the number of comparisons between concepts and instances, and comparisons between concepts.

CE algorithm with full conjunctive concept language. Let q be the number of negative instances, g the largest size of G. Following [Mitchell 1982], in our case the key term is $O(g^2q)$. The maximum size of G is given by the largest number of unordered concepts that can be found in the version space after the first positive instance. This number turns out to be $O(n^2l)$. To illustrate, first we must note that the version space after the first positive instance will contain the concepts more general than the instance, therefore the admissible values for each attribute will be the l values in the attribute tree that are placed in the chain linking the attribute value in the instance to the root of the attribute tree. When n = 2 there are at most l ways to choose a pair of values from two ordered sets of size l in such a way that the pairs are unordered. When n increases, this number comes to be multiplied by n / (n -2)! 2! . In fact, considering that two n-factored concepts are unordered if they contain at least two factor concepts with different orderings, all the possible unordered n-factored concepts can be obtained considering the same combinations as in the l original unordered concepts for each possible way of choosing a pair of attributes from among the n attributes. The maximum size of G is therefore $O(n^2l)$. The complexity of the CE algorithm is $O(n^4l^2q)$.

FCE algorithm with attribute languages. In this case several concept languages are active at once. For each negative instance we have to update in parallel at most $\max_k [n! / (n-k)!k!]$, that is $O(n^2)$, version spaces. Given our hypothesis on instance distribution, the g value of the intermediate version spaces will be 1 for the one-factored languages, 2 for the two-factored languages, .., n for the n-factored languages. The largest value of g is n, and the relative complexity factor for each version space is therefore $O(n^2)$. Thus the time taken to induce version spaces within the set of active languages is at most $O(n^2n^2q) = O(n^4q)$.

The total time complexity can be calculated adding the time taken by language shift to the time taken by concept induction alone. The cost of shifting the concept languages is given by the number of language shifts ($2^n$) multiplied by the cost of any single language shift. The time taken by any single language shift becomes constant if we modify the FCE algorithm's inductive phase by labelling each member of each G* and any of its more specific concepts with all the negative instances it does not cover. In this way, in fact, the operations described in the procedure to compute the G set in any product language will no longer involve any matching between concepts and instances. On the other hand, the cost of labelling must now be added to the cost of language shift. The labelling we introduced requires matching each negative instance against the members of n G*'s (we keep only the G*'s relative to the initial factor languages), where each G* contains only one member (in our case, in fact, as there is only one positive instance, we can immediately remove the concepts that are not more general than the positive instance from the G*'s,

at an additional cost of O(qnbl)), and repeat for all the l more specific concepts of each member of G* (i.e., the concepts contained in the chain of admissible values relative to that G*'s factor language). Therefore labelling takes in all O(qnl) + O(qnbl) = O(qnbl). The time complexity of language shift is $O(2^n)$ + O(qnbl). The overall time complexity is therefore $O(n^4q)$ + $O(2^n)$ + O(qnbl), which, for practical values of n, b, and l, approximates to $O(n^4q)$.

In sum, we have $O(n^4l^2q)$ in the CE algorithm versus $O(n^4q)$ in the FCE algorithm. The effect of using the FCE algorithm with the chosen instance distribution appears to be that of blocking the fragmentation of G due to l. It is also worth noting that the factor $O(n^2)$ in the FCE algorithm due to the presence of multiple languages can be reduced by reducing the number of intermediate product languages employed. This would, on the other hand, be counteracted by an increase of the factor $O(n^2)$ due to the g of the intermediate languages. Here is a trade-off between using few concept languages and using many concept languages in a given range. The fewer the concept languages, the less the amount of computation devoted to parallel induction and language shift. The more the concept languages, the more likely it is that a smaller amount of induction will be done within the largest concept languages, which are the least convenient. Experimentation might help investigate this kind of trade-off .

## 5 Relation to factorization in concept induction

Factorization with smaller concept languages in the CE algorithm has been first explored in [Subramanian and Feigenbaum 1986] and [Genesereth and Nilsson 1987]. Although we were inspired by their work, our goals, methods and assumptions are different. First, in [Subramanian and Feigenbaum 1986] and [Genesereth and Nilsson 1987].language factorization has been used with the aim of improving efficiency during the phase of experiment generation, whereas we have investigated its utility during the earlier and more important stage of version-space induction from given examples and counter-examples. Second, while they have primarily addressed the problem of factoring a version space and assessing credit over its factors, we have focussed on language shift during version-space induction over a set of available factor and product languages. Third, their approach relies on the assumption that the given concept langage is factorable into *independent* concept languages[1]. By contrast, when applying the FCE algorithm directly to the attribute languages of a conjunctive concept language it is not necessary the attribute languages be independent. For example, the two factor languages we have used as an illustration throughout the paper (L1 and L2) happen to be two independent languages[2]; however, we could well apply the FCE algorithm to the concept language $\mathbf{L_B}$ we introduced earlier along with the concept language $\mathbf{L_C}$ = {anyrank, odd, even, 1, 3, 5, 7, 9, J, K, 2, 4, 6, 8, 10, Q}, these two languages being not independent (the intersection of the concept "2" in $\mathbf{L_B}$ and the concept "odd" in $\mathbf{L_C}$ is empty, for instance). Using non-independent factor languages, as their product may contain a large number of empty or redundant concepts, may badly affect the performance when the FCE algorithm is applied to recover from inconsistency due to use of small concept languages. But it does not seem to affect the result when the FCE algorithm is used to improve efficiency with respect to the full conjunctive concept language.

## 6 Relation to inductive language shift

As mentioned earlier, the FCE algorithm can also be seen as a method for introducing new concepts to overcome the limitations of a set of restricted concept languages (i.e., the factor languages). It does so by creating another set of larger concept languages (i.e., the product languages) to constrain the search for new useful concepts. This is a significant departure from the search strategy usually employed in most approaches to inductive language shift. Regardless of the specific goal pursued - many systems deal with improvement of some quality measures of the learned descriptions rather than with their correctness - "the problem of new terms" [Dietterich *et al.* 1982] or "constructive induction" [Michalski 1983] is in general tackled by defining a set of appropriate *constructive operators* and carrying out a depth-first search through the space of the remaining concepts to find useful (e.g., consistent, more concise, more accurate) extensions to be added to the given language. Furthermore, since the number of admissible extensions is generally intractably large, most of the approaches to constructive induction rely on various heuristics to reduce the number of candidate additional concepts and/or to cut down the search (e.g, [Matheus and Rendell 1989], [Pagallo 1989], [Wogulis and langley 1989]).

By contrast, we compute and keep all the admissible language extensions (in a given set of extensions) that restore consistency with data, rather than considering one or few plausible language extensions at a time. Just as the relation *more general than* that is implicitly defined over the terms of a concept language may allow efficient representation and updating of all consistent concepts [Mitchell 1982], so too the relation *larger than* that is implicitly defined over a set of languages may provide the framework to efficiently organize the small-to-large breadth-first search of useful languages. These considerations suggest that an alternative abstract model for language shift can be formulated, in which the search for new concepts, rather than being based on the use of constructive operators, is driven by the ordering of a set of candidate concept languages (work in preparation).

---

[1] Two concept languages $L_A$ and $L_B$ are *independent* if membership in any of the concepts from $L_A$ does not imply or deny membership in any of the concepts in $L_B$. This definition implies that for every concept a in $L_A$ and every concept b in $L_B$ the intersection of a and b is neither empty nor equal to either concept.Two independent concept languages are unordered with respect to the larger-than relation.

[2] It is often the case that attribute choice reflects independencies in the world, thus giving rise to actual independent factor languages.

## 7. Conclusion

We have presented the FCE algorithm for efficiently inducing version spaces over a set of partially-ordered concept languages. The utility of this algorithm is twofold: improving the efficiency of version-space induction if the initial concept language is decomposable into a set of factor languages, and inducing consistent version spaces if a set of concept languages inconsistent with data is initially available. In this paper we have focussed on the former. We have applied the FCE algorithm to the task of inducing version spaces over a conjunctive concept language defined on a tree-structured attribute-based instance space, and we have evaluated when it leads to a reduction in complexity.

## Acknowledgements

Part of this work was done while at the Computing Science Department of the University of Aberdeen, partially supported by CEC SS project SC1.0048.C(H). I would like to thank Derek Sleeman and Pete Edwards for their support and for useful discussions on this topic. The work was carried out within the framework of the agreement between the Italian PT Administration and the Fondazione Ugo Bordoni

## References

[Bundy *et al.* 1985] A. Bundy, B. Silver, D. Plummer. An analytical comparison of some rule-learning problems. *Artificial Intelligence*, Vol. 27, No. 2 (1985), pp. 137-181.

[Carpineto 1990] C. Carpineto. Combining EBL from success and EBL from failure with parameter version spaces. In *Proc. 9th ECAI*, Pitman, London, 1990, pp. 138-140.

[Carpineto 1991] C. Carpineto. Analytical negative generalization and empirical negative generalization are not cumulative: a case study. In *Proc. EWSL-1991, Lecture Notes on Artificial Intelligence*, Springer-Verlag, Berlin, 1991, pp. 81-88.

[Dietterich *et al* 1982] T. Dietterich, B. London, K. Clarkson, R. Dromey. Learning and inductive inference. In Cohen & Feigenbaum (Eds.) *The Handbook of Artificial Intelligence*, Morgan Kaufmann, Los Altos, 1982.

[Genesereth and Nilsson 1987] M. Genesereth, N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, Los Altos, 1987.

[Haussler 1988] D. Haussler. Quantifying inductive bias: Artificial Intelligence learning algorithms and Valiant's learning framework. *Artificial Intelligence*, Vol. 36, No.2 (1988), pp. 177-221.

[Hirsh 1989] H. Hirsh. Combining Empirical and Analytical Learning with Version Spaces. In *Proc. 6th Int.Workshop on Machine Learning*. Morgan Kaufmann, Los Altos, 1989, pp. 29-33.

[Matheus and Rendell 1989] C. Matheus, L. Rendell. Constructive induction on decision trees. In *Proc. 11th IJCAI*, Detroit, Morgan Kaufmann, Los Altos, 1985, pp. 645-650 .

[Michalski 1983] R. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, Vol. 20, 1983, pp. 111-161.

[Mitchell 1982] T. Mitchell. Generalization as Search. *Artificial Intelligence*, Vol. 18, 1982, pp. 203-226.

[Pagallo 1989] G. Pagallo. Learning DNF by Decision Trees. In *Proc.11th IJCAI*, Morgan Kaufmann, Los Altos, pp. 639-644.

[Smith and Rosenbloom 1990] B. Smith, P. Rosenbloom. Incremental Non-Backtracking Focusing: A Poliniomally Bounded Generalization Algorithm for Version Spaces. In *Proc. 8th AAAI*, Morgan Kaufmann, Los Altos, pp. 848-853.

[Subramanian and Feigenbaum 1986] D. Subramanian, J. Feigenbaum. Factorization in Experiment Generation. In *Proc. 5th AAAI*, Morgan Kaufmann, Los Altos, 1986, pp. 518-522.

[Utgoff 1986] P. Utgoff. Shift of bias for inductive concept learning. In Michalski *et al.* (Eds), *Machine Learning II*. Morgan Kaufmann, Los Altos, 1986, pp. 107-148.

[Wogulis and Langley 1989] J. Wogulis, P. Langley. Improving efficiency by learning intermediate concepts. In *Proc. 11th IJCAI*, Morgan Kaufmann, Los Altos, pp. 657-662.