Induction with Subtheory Selection and Abduction

Pavel Brazdil and Alípio Jorge*

Abstract. Induction may often generate theories that contain redundant clauses. It is thus desirable that the induced theory is revised before it is handed over to an interpreter. In this paper we describe a particular solution to the problem, based on the notion of *explanation based subtheory selection.* It appears that this mechanism can also be used to perform abduction. The set of abducible facts can be processed by subtheory selection to generate a set of plausible consistent explanations. Abduction can thus be seen as particular form of a mechanism that is required in every inductive system..

1 INTRODUCTION

If we analyze the theories generated by various ILP systems, we note that sometimes the correct definition is accompanied by several other clauses which appear to be redundant. As these can lead to redundant computations at interpretation stage, they are in general undesirable.

This problem is in fact quite common and arises in all systems in which induction is repeated while some parameters are varied (set of examples, background knowledge etc.). The problem arises also in SKILit (Jorge & Brazdil, 1996), due to the fact that this system uses an iterative strategy (*iterative bootstrap induction)*, when inducing theories. This strategy is useful when attempting to induce recursive definitions on the basis of few examples. The strategy may however generate theories that contain redundant definitions.

How can the redundant (or other incorrect) definitions be deleted? This seems to be a simple problem, but a question arises how to do this efficiently. In section 3 we describe in detail our particular solution, referred to as *explanation based subtheory selection*.

It appears that this mechanism can also be exploited in abduction. The method of *explanation based subtheory selection* can be used to process the set of abducible facts and return one or more subsets, representing plausible and consistent explanations. Abduction can thus be seen as particular form of a mechanism that is required to complement an inductive system.

2 TWO USES OF SUBTHEORY SELECTION

The method of subtheory selection uses a given hypothetical theory (T) and returns a subset of that theory (T') satisfying certain criteria. The criteria used here are the common ones used in induction:

- T' should cover all positive examples from the given set E^+ (completeness),
- T' should not violate any constraints from T_c (consistency)[1],
- T' should satisfy certain quality criteria.

It is assumed that tests of coverage are done in conjunction with background theory T_{bk} and theory T_c consisting of certain domain constraints. We assume that both are given.

The method of subtheory selection can be used for two different purposes. First, to eliminate redundant (or incorrect) clauses generated by an inductive system. Second, it can be used to process a given set of abducibles and return one or more subsets. Each subset returned represents a plausible explanation for the given case.

2.1 Cleaning the Induced Theory with Subtheory Selection

In this section we show how the method of subtheory selection can be used to eliminate redundant clauses from a given theory.

Let us consider an example which involves an induced theory for *union(A,B,C)* shown in Fig.1, generated by the inductive subsystem of SKILit (Jorge & Brazdil, 1996) in one of the experiments. This theory represents an intermediate result which is passed to the subtheory selection subsystem. The final theory generated by SKILit is a subset of the one shown.

From the point of view of subtheory selection, the theory shown in Fig.1 represents a hypothetical theory T. It includes the correct definition of *union* (clauses c4,c5 and c6). Apart from this, it contains several other clauses (c1,c2,c3,c7), which are correct (i.e. do not cover any negative examples). These clauses are *redundant.*

The algorithm of subtheory selection requires also a background theory T_{bk}. Here this theory contains the definitions of all subsidiary predicates required when defining *union* (i.e. *member(A,B), not_member(A,B), dest(A,B,C)* and *const(A,B,C)*[2]). The sets of positive examples E^+ and negative examples E^- are also given.

The objective of subtheory selection is to eliminate the redundant clauses from the hypothetical theory T. As pointed out earlier we require that the criteria of completeness and consistency are satisfied.

```
c1: union(A,B,B):-
       dest(B,C,A).
c2: union(A,A,A).
c3: union(A,B,C):-
       dest(A,D,E), null(E),
       not_member(D,B),const(C,D,B).
c4: union(A,B,C):-
       dest(A,D,E), not_member(D,B),
       union(E,B,F), const(C,D,F).
c5: union(A,B,C):-
       dest(A,D,E), member(D,B),
```

* LIACC, University of Porto, Rua Campo Alegre 823, 4150 Porto, Portugal.
email: {pbrazdil, amjorge}@ncc.up.pt.

[1] The requirement that T' should not cover any negative examples from E- can be reformulated and expressed using the notion of constraints.

[2] The meaning of *union(A,B,C)* is the usual one. List C represents the union of distinct elements found in lists A and B, in accordance with the requisites of *set union*. The predicate *member(A,B)* is true if element A is a member of list B; *dest(A,B,C)* is true if list A can be decomposed into head B and tail C; *const(A,B,C)* is true if head B and tail C give list A. The input variables are underlined.

```
        union(E,B,C).
c6: union([],B,B).
c7: union(A,[],A).
```

Fig. 1 Theory for *union* generated by SKILit (input to subtheory selection)

As expected, the subtheory selection returns theory T'= {c4,c5,c6} which represents the usual definition of *union*. The details of how the algorithm obtained this are given later.

2.2 Constructing Abductive Explanation with Subtheory Selection

Let us now consider the second usage of subtheory selection, related to processing a hypothetical theory containing a set of abducibles. In this case the method returns a subset of this theory which satisfies the criteria of completeness and consistency. The subset returned represents a plausible explanation for the given set of observations.

Let us consider one quite well known example involving a car which does not start. Suppose we observe that the headlights work. The objective is to return an *explanation*, sometimes also called a *reason* or *diagnosis* in ATMS literature (de Kleer, 1986).

If we approach this as a problem of subtheory selection, the observations represent the set E^+. Here we will use the predicate *part(P,S)* to express the fact that part *P* (e.g. *motor*, *headlights* etc.) is in state *S* (*working*, *not_working*). So in our case, set E^+ contains the following observations to be explained:

```
e1+: part(motor, not_working).
e2+: part(headlights, working).
```

The background theory T_{bk} shown below (Fig.2) contains diagnostic knowledge. Rule d1, for instance, expresses the fact that if both battery and lamps are working (i..e. not damaged or broken) the headlights will be functional (i.e. give light).

```
d1: part(headlights, working) :-
       part(lamps, working),
       part(battery, working).

d2: part(headlights, not_working) :-
       part(lamps, not_working).

d3: part(headlights, not_working) :-
       part(battery, not_working).

d4: part(motor, working) :-
       part(battery,working),
       part(ignition,working).

d5: part(motor, not_working) :-
       part(battery,not_working).

d6: part(motor, not_starts) :-
       part(ignition,not_working).
```

Fig.2 Theory T_{bk} containing diagnostic knowledge

Constraint theory C contains a constraint expressing the facts *part(X,working)* and *part(X,not_working)* cannot co-exist:

```
:- part(X,working), part(X,not_working).
```

The hypothetical theory T contains all possible abducibles, that is, all facts that can be used to complete the explanation of the given situation.

```
c1: part(lamps, working).
c2: part(lamps, not_working).

c3: part(battery,working).
c4: part(battery,not_working).

c5: part(ignition,working).
c6: part(ignition,not_working).
```

Fig. 3 Hypothetical Diagnostic Theory (Abducibles)

In this case, the subtheory selection returns theory T'= {c1,c3,c6}, which states that *lamps* and *battery* are working ok, but *ignition* is not. The details of how this answer was obtained are given in Section 3.

3 EXPLANATION BASED SUBTHEORY SELECTION

Our method of subtheory selection involves three stages:

- construction of atomic theories,
- evaluation of atomic theories,
- selection of the best combination of atomic theories.

In the first stage the method constructs so called *atomic theories*. These theories are the simplest theories identified, each covering at least one positive example. In the second stage the atomic theories are *evaluated* on all available examples and the best ones are identified. This enables to construct the final theory in the third stage. The method employed is a *constructive* method. Although the given hypothetical theory T is used as a source, the final theory T' is built up from scratch.

3.1 Construction of Atomic Theories

The objective of the first stage of subtheory selection is to process the given theory T and return a set of atomic theories T^*. Each atomic theory should cover at least one positive example from E^+ and should be consistent with all negative examples E^-. Theory T_{bk} represents background knowledge.

```
Input:  T,Tbk, E+,E-;
Output: T*.
T*= Ø
While E+ ≠ Ø do:
  Select ei+∈ E+.
    If ei+ cannot be derived with
      any existing atomic theory,
      (i.e. if  Tj*∈T*∧ Tj*∪Tbk ⊬ ei):

     Construct depth limited
       derivations of ei+.

     For each distinct derivation j do:
        Construct atomic theory Tij*
           from clauses in the derivation.

        If it does not cover any
           negative examples, add it to T*.
```

```
      End For
    End If

    Eliminate ei+ from E+
        (E+= E+ - {ei+})
End While
```

Fig. 4 Constructing Atomic Theories

The positive examples are processed one by one, similarly as in the covering algorithm. In each case the system verifies whether the example chosen is already covered by one of the existing atomic theories. If this is so, the system does not attempt to construct any new theories for efficiency reasons. If none of the existing atomic theories serves, the system attempts to construct new ones. The system searches for *all* possible proofs within a given depth bound (Fig.4). Each atomic theory constructed contains all the clauses used in the corresponding derivation.

Although the induced theories by SKILit do not usually cover any of negative examples, the algorithm was designed to handle negative examples as well. Should *new* negative examples be given, the algorithm can be used to clean up the current theory. For this reason we require that the atomic theories constructed should not cover any negative examples.

The depth bound plays quite an important role. It limits the search space and it is thus practicable to search for *all* proofs within a given depth bound. As non-terminating subprograms are admitted as input, the depth-bound enables us also to identify so called *suspect* atomic theories and exclude them. These are the theories which give rise to derivations that reach the depth bound[3].

3.2 Evaluation of Atomic Theories

In the second stage of subtheory selection all existing atomic theories are evaluated and characterized. The evaluation involves calculating the following three parameters: relative coverage (Cov), compression (Comp) and weighted mean solution length saving (MeanSol).

The first parameter, *relative coverage (Cov),* quantifies the proportion of the positive examples covered, on scale 0 to 1 (value 1 is represents total coverage). The second parameter, *compression (Comp),* represents an estimate of compression that would be achieved if the positive examples covered were replaced by the corresponding atomic theory.

The third parameter, *weighted mean solution length saving,* is calculated separately for each positive example covered as follows. Suppose that three atomic theories cover a particular example in question using varying number of steps (say, 20,40 and 10). The maximum number of steps (40) establishes a reference point. Then *solution length saving = (1-actual solution steps / maximum solution steps).* (In our case we get: .5, 0, .75). This calculation is then repeated for all positive examples and then we calculate *mean solution length saving (MeanSol).* The three measures are combined heuristically as follows*: Quality=(1/3)*Cov +(1/3) *Comp +(1/3)*Cov* MeanSol.*

For all purposes here the method described seems to be satisfactory. However, the first two parameters could be better combined. We plan to consider various MDL-based methods (Rissanen,1982) for evaluating theories reported recently in literature to see if the method could be improved.

[3] In general, the depth bound may be reached if the problem is complex and cannot be solved using a given number of steps. In order to overcome the problem, we intend to employ a variable depth bound in future, which would be extended by iterative deepening.

3.3 Selection of Atomic Theories

After the atomic theories have been characterized, the selection process is quite simple. The algorithm goes through all positive examples once more, attempting to cover them. However, the method does not use *clauses* at this stage, but rather *atomic theories*. Among the possible atomic theories the system always selects the best one.

The final theory T' is built from scratch, starting with an empty theory. At each step the best atomic theory is added. However, before doing that we verify whether it is consistent with other information. Suppose that, at some stage, we are adding, say, T_{ij}^*. In this case we require that $T' \cup Tij^* \cup T_{bk} \cup Tc$. is consistent. If it is not, that is, if $T' \cup Tij^* \cup T_{bk}$ violate any constraint in Tc, the system backtracks and attempts to select the next best alternative.

3.4 Eliminating Non-Terminating Theories

Non-terminating theories get filtered out. As non-terminating theories are difficult to identify using analytic methods, we let our system construct all candidate atomic theories and select the ones with the best quality. As has been pointed out earlier, quality is judged according to certain parameters. Potentially non-terminating atomic theories get excluded on the grounds of unacceptable solution length.

4 EXPERIMENTAL RESULTS

4.1 Processing the Induced Theory Concerning *Union*

Let us analyze how the system processed the hypothetical theory for *union*, shown in Fig.1. In the first stage the system identified various atomic theories that explain the given positive examples. In this case, four atomic theories were identified: T_1^*={c4,c5,c6}, T_2^*={c3,c5}, T_3^*={c2}, T_4^*={c1,c5}. The atomic theories were then qualified according to their quality. Theory T_1^* was identified as a clear winner, as it covers all the positive examples in E^+ (the second best theory is T_2^* covering only half of them). This affected the final quality. T_1^* had a score of 0.60, while T_2^* achieved only 0.37. As T_1^* covers all positive examples the final theory $T' = T_1^*$.

4.2 Processing the Induced Theory Concerning *Multiply*

Let us analyze another case which involves processing the induced theory for multiplying integers. The theory induced by SKILit is shown in Fig.5. As in the previous case, the theory shown represents an intermediate result which is passed over to the subtheory selection algorithm.

The theory shown includes several redundant clauses, all of which are *correct*. (One exception is clause *c9*, but is was added manually to test the capabilities of the system). Clauses *c1, c3, c5* and *c6* express quite obvious properties of *multiply: 0*B=0, A*0=0, A*1=A, 1*B=B*. Clause *c1* includes a condition *plus(A,B,B)* which is true if *A=0*. As we have specified the induction problem, we did not include *zero(0)* in the background knowledge. This is why system did not use just *zero(0)* here, but rather *plus(A,B,B)* which is true whenever *A=0*.

Some clauses shown represent rather specific methods for computing the product of two integers (e.g. clauses *c2,c4*). Clause *c4*, for instance, can be used to calculate *A*2*. The body of this clause seems unnecessarily complicated, but there is a good reason for that. As it happens, we but did not include *two(2)* in the background knowledge. However, this did not preclude SKILit from generating quite an interesting solution. The predicates *pred(B,D), pred(A,E), plus(E,D,A)* express the fact *(A-1)+(B-1)=A*, which can be simplified into *B=2*. Had the predicate *two(2)* been in the background knowledge, the system would have used the condition *two(B)* instead. It is interesting that the system found quite an ingenious way of overcoming this problem.

Clause *c7* is somewhat special. It expresses the *commutativity* of multiplication (i.e. the two inputs can be interchanged without effecting the result). Although this clause can be very useful in the process of inducing the correct recursive definition, it can lead to non-terminating computation. Clause c8 represents the correct recursive clause. This clause was generated by SKILit on second iteration of bootstrap induction.

c1: multiply(A,B,A):-
plus(A,B,B).

```
c2: multiply(A,A,B):-
       pred(A,C), one(C), plus(A,A,B).

c3: multiply(A,B,B):-
      plus(A,B,A).

c4: multiply(A,B,C):-
      pred(B,D),  pred(A,E),
      plus(E,D,A), plus(A,A,C).

c5: multiply(A,B,A):- one(B).
```

c6: multiply(A,B,B):- one(A).

```
c7: multiply(A,B,C):-
     pred(A,D),multiply(B,A,C).
```

c8: multiply(A,B,C):-
pred(A,D), multiply(D,B,E),
plus(E,B,C).

```
c9: multiply(A,B,C) :-
      plus(A,B,C).
```

Fig. 5 Theory for *multiply* generated by SKILit

The correct definitions are identified with the help of the given positive examples (these are the same examples that were used also in induction). These include *e1=multiply(3,7,21), e2=multiply(1,9,9)* among others. The negative examples include $e1^-$*=multiply(1,7,0)*, $e2^-$ *=multiply(0,4,1)*.

In the first stage the algorithm searched for useful atomic theories. In the process of doing this, several theories were identified as *suspect*: {c5,c7,c8}, {c6,c7,c8} and {c4,c7,c8}. We notice that all three theoris contain clause *c7* that expresses the commutativity of *multiply*. This clause can give rise to non-terminating computations.

Having eliminated the suspect theories from consideration, the system has no difficulty to identify the following atomic theories as candidates for inclusion in T':

T_1^*={c6,c8} covering 9/10 positive examples, but with reduced computation costs;

T_2^*={c1,c8} covering 10/10 positive examples, but with somewhat larger computation costs.

Theory T_1^* appears to be somewhat ahead of T_2^* because of the shorter solutions. As it does not cover all the examples, the T_2^* is also added to T' and so the final theory is T' ={c1,c6,c8}.

4.3 Processing the Abducible Theory Concerning Car Failure

Let us consider the hypothetical diagnostic theory containing the abducibles. In the first stage the system identified the following atomic theories:

T_1^*={c4}={part(battery,not_working)}
T_2^*={c6}={part(ignition,not_working)}

both of which represent alternative explanations of the fact that the motor is not working ($e1^+$: part(motor, not_working)) and, in addition also:

T_3^*={c1,c3}={part(lamps,working), part(battery,working)}

which explains the fact that headlights work ($e2^+$:part(headlights, working)). In the second stage the atomic theories get qualified. In the third stage, the system attempts to construct a global theory that explains all observations in E^+. The only final theory that is consistent is $T'=T_2^* \cup T_3^*$= {part(ignition,not_working), part(lamps,working), part(battery, working)}.

5 RELATION TO OTHER WORK

The problem of *selecting a subset of clauses* from a given set has been extensively studied in ATMS (de Kleer, 1986). The input to ATMS system is a theory T, background knowledge T_{bk} and query q. The objective is to find a subset T' of T, such that:

T' enables to answer q (i.e. $T' \cup T_{bk} \models q$) and

T' and T_{bk} are not inconsistent (i.e. $T' \cup T_{bk} \not\models$).

In literature, theory T' is often called *the explanation of q*, or sometimes also *the reason for q*.

The method incorporated in TRACY (Bergadano et al., 1995) satisfies the requisites of ATMS above. In addition it provides a constructive method to build T' (a subset of T) which resembles the method described here. Our method differs from TRACY in that it seeks the set of atomic theories which are used as basic building blocks. These are characterized using several parameters, some of which are performance oriented. This enables us to construct theories that not only cover the given positive examples, but are capable of computing the required outputs relatively efficiently.

In the work reported here we have not considered specialization of theories using *goal reduction*. A theory can be specialized by selecting a clause and a literal in its body, selecting another clause that matches this literal and resolving on this literal. This operator was used in various systems, including the *discrimination algorithm* (Brazdil, 1984), ML-SMART (Bergadano and Giordana, 1988), FOCL (Pazzani et al., 1991), and more recently also in SPECTRE (Boström and Idestam-Almquist, 1994; Boström, 1995).

We were reluctant to build a very general system which would mix generalisation and specialization in an unconstrained manner. Previous research in the area of program transformation has shown that although many programs can be generated using unfold / fold transformations, the search space is often unmanageably large for an automated system to explore.

As our inductive system is quite powerful, we have decided that specialization should play a somewhat supplementary role. For this reason we use just one specialization operator - *dropping one of the alternative definitions* of a particular predicate - and formulated the algoritm of subtheory selection around that notion.

6 CONCLUSIONS

In this paper we have described two uses of subtheory selection. One, which is used to complement the process of induction. Here, subtheory selection is used to eliminate those parts of the induced theory that appear to be redundant.

The second use is in abduction. We have demonstrated that subtheory selection can be exploited to eliminate some elements from the set of abducibles. The method of subtheory selection proceeds by constructing a set of atomic theories, each representing a potential explanation of one particular observation. These are ordered taking into account various performance oriented criteria. The global theory (explanation) is built up from individual explanations. Our study thus suggests that abduction could be seen as a particular form of a mechanism that is required to complement the functioning of a typical inductive system.

ACKNOWLEDGEMENTS

We express our gratitude to CEC for their support under ILPNET and MLNet.

REFERENCES

[1] Bergadano F. and Giordana A.: (1988): A Knowledge Intensive Approach to Concept Induction, in *Machine Learning, Proceedings of the Fifth International Conference*, J.Laird (ed.), Morgan Kaufmann.

[2] Bergadano F., Gunetti D., Nicosia M. and Ruffo G.: "Learning Logic Programs with Negation as Failure", in *Advances of Inductive Logic Programming*, L.de Raedt (ed.), IOS Press, Amsterdam, 1996.

[3] Boström H. (1995): "Covering vs. Divide-and-Conquer for Top-Down Induction of Logic Programs", in *IJCAI-95 Proceedings*, Montréal, Canada, C.Mellish (ed.), Morgan Kaufmann.

[4] Boström H. and Idestam-Almquist P. (1994): "Specialization of Logic Programs by Pruning SLD-Trees", in *Proceedings of the 4th International Workshop on Inductive Logic Programming,* vol. 237 of GMD-Studien, GMD, Germany.

[5] Brazdil P. (1984): "Use of Derivation Trees in Discrimination", presented at ECAI-84, Pisa, Italy. See also "Problem 93", in *Prolog by Example: How to Learn, Teach and Use It*, Coelho H. and J.Cotta (eds.), Springer Verlag, 1988, pp. 167-172.

[6] Brazdil P. and Jorge A.: "Learning by Refining Algorithm Sketches", in *Proceedings of ECAI-94,* T.Cohn (ed.), J.Wiley, 1994.

[7] Brazdil P.: "Use of Derivation Trees in Discrimination", presented at ECAI-84, Pisa, Italy. See also "Problem 93", in *Prolog by Example: How to Learn, Teach and Use It*, Coelho H. and J.Cotta (eds.), Springer Verlag, 1988, pp. 167-172, 1984.

[8] Darlington J. and Burstall R. (1976): "A System that Automatically Improves Programs", *Acta Informatica, Vol. 6,* 41-60.

[9] Jorge A. and Brazdil P.: Architecture for Iterative learning of Recursive Definitions, in *Advances of Inductive Logic Programming*, L.de Raedt (ed.), IOS Press, Amsterdam, 1996.

[10] de Kleer, J.: "An Assumption Based Truth Maintenance System", *Artificial Intelligence 28*, 127-162, 1986.

[11] Pazzani M., Brunk C. and Silverstein G. (1991): "A Knowledge-Intensive Approach to Learning Relational Concepts, in *Machine Learning: Proceedings of Eighth International Workshop,* L.Binbaum and G.Collins (eds.), Morgan Kaufmann.

[12] Quinlan R.: "Learning Logical Definitions from Relations", in *Machine Learning 5*, 239-266, 1990.

[13] Rissanen J.: A Universal prior for integers and estimation of Minimum Description Length, *Annals of Statistics*, 11:416-431, 1982.