Part III:
Extended Abstracts

JIGSAW: puzzling together RUTH and SPECTRE (Extended Abstract)

Hilde Adé¹ and Henrik Boström²

¹ Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium

² Department of Computer and Systems Sciences, Stockholm University
Electrum 230, S-164 40 Kista, Sweden

1 Introduction

In this work we present JIGSAW, which is a theory revision system within the Inductive Logic Programming (ILP) framework. In essence it is an enhanced version of the system RUTH [2], which is a non-interactive theory revisor built on the ideas presented by De Raedt and Bruynooghe [6]. The main contribution of this work is that we incorporated unfolding as a theory specialization operator into RUTH. This was done by integrating RUTH with SPECTRE [5] which is a theory specialization system based on unfolding and clause removal. The theory revision problem addressed by our system can be formalized as follows:

- **Given:**
 - an integrity constraint theory I
 - a new integrity constraint IC
 - a database D , satisfying I , but possibly not satisfying IC
- **Find:** a transaction $Trans$ such that database $Trans(D)$ satisfies $I \cup \{IC\}$

In our framework an *integrity constraint theory* is a set of integrity constraints. An *integrity constraint* is a clause of the form $p_1 \wedge \dots \wedge p_k \rightarrow q_1 \vee \dots \vee q_m$, ($k \geq 0$, $m \geq 0$), which can be read as: if p_1 and ... and p_k are true, then at least one of the q_j should be true. A *database* is a set of definite clauses. Finally, a *transaction* is a list of actions, where an action in our framework is of the form *add(fact)*, *add(clause)*, *delete(fact)*, *delete(clause)* or *replace(clause, set_of_clauses)*.

2 JIGSAW

The system JIGSAW is based on the overall structure of the RUTH system, and integrates the theory specialization technique of SPECTRE. In this section, we first briefly describe RUTH and SPECTRE, and then present JIGSAW.

RUTH. The system RUTH is a theory revision system built on ideas presented in [6]. However, it works non-interactively, replacing queries to an oracle by an intelligent search strategy based on an iterative deepening schema [7]. A full description of the system can be found in [2] and [3], and theoretical properties are described in [1].

```

procedure RUTH( I , D )
begin
  Trans  $\leftarrow \emptyset$ 
  Hypo  $\leftarrow \emptyset$ 
  while I is still violated by Trans(D) do
    select a violated constraint lc from I
    hypothesize an example Ex
    Hypo  $\leftarrow$  Hypo  $\cup$  Ex
    while Hypo  $\neq \emptyset$  do
      choose an example Ex from Hypo
      if Ex is an uncovered positive example
      then handle_positive(Trans,I,Hypo,Ex)
      else { Ex is a covered negative example }
             handle_negative(Trans,I,Hypo,Ex)
  Output : Trans(D)
end procedure RUTH

```

The top level algorithm of RUTH

The central idea in the algorithm is the following. Upon violation of an integrity constraint $p_1 \wedge \dots \wedge p_k \rightarrow q_1 \vee \dots \vee q_m$ with answer substitution θ , the system hypothesizes that at least one of the $p_i\theta$ should be false, or at least one of the $q_j\theta$ should be true in the transformed database. As in concept learning $p_i\theta$, resp. $q_j\theta$, is called a covered negative example, resp. an uncovered positive example. Transforming the database is done by adding or deleting clauses or facts.

The operators of RUTH are contained in the procedures *handle_positive* and *handle_negative*. For handling a covered negative example RUTH uses an MIS-like specialization operator. It constructs an SLD-tree for the example, hypothesizes that one of the clauses used in the proof is incorrect, and removes this clause from the database. To handle an uncovered positive example the system has three operators: (1) add the example as a ground fact to the database, (2) construct a maximally general clause that covers the positive example, and that is consistent with the integrity theory and (3) abduce new facts $l_i\theta$ using a clause $h \leftarrow l_1 \wedge \dots \wedge l_n$ of the database such that $h\theta = \text{Ex}$ and for which not all literals $l_i\theta$ are true.

SPECTRE. Boström and Idestam-Almqvist [5] have argued that the specialization of logic programs can be viewed as the pruning of SLD-trees. In their system SPECTRE, the actual pruning is done by applying the transformation rule *unfolding* [8] together with clause removal. The specialization of a logic program with respect to positive and negative examples is realized in the following way. As long as there is a clause in the program that covers a negative example, it is checked whether it covers any positive examples or not. If it covers no positive examples, then it is removed, otherwise it is unfolded. The choice of which literal to unfold upon is made using a computation rule, which is given as input

to the algorithm. In [5] the optimal choice of literal to unfold upon is approximated by selecting the literal that results in the minimal residual impurity of the resolvents when having applied unfolding.

JIGSAW. For the new system JIGSAW we keep the top level algorithm of RUTH as well as the iterative deepening schema. Changes are only introduced at the operator level : the procedure *handle_negative* is enhanced with an unfolding and clause removal operator. Thus the set of possible actions of RUTH (*add(clause)*, *delete(clause)*, *add(fact)*, *delete(fact)*) is extended with the action *replace(clause, set_of_clauses)*.

For the evaluation of JIGSAW the "Student Loan" domain from the UCI repository of Machine Learning Databases was used. We randomly generated 40 corrupted domain theories and sets of 20 randomly selected examples. We also tested JIGSAW on the smaller CUP domain as described in [4]. Because of space constraints we cannot fully report on the results. We limit ourselves to the discussion the advantages and also a caveat

There are several advantages in using an unfolding technique for theory specialization. First, plain clause removal might affect the proofs of positive examples, whereas the unfolding technique first specializes the selected clause, and only removes clauses that do not cover any positive examples. Second, an unfolding operator can speed up the theory revision process. One can easily imagine situations where one unfolding step has the same effect as the removal of a clause, followed by the construction of a new - more specific - clause. The following small experiment taken from the CUP domain illustrates both these aspects.

Rules in the Database	
cup(X) $\leftarrow$ liftable(X), stable(X), open_vessel(X)	
liftable(X) $\leftarrow$ weighs(X,Y), light(Y)	
stable(X) $\leftarrow$ part_of(X,Y), bottom(Y), small(Y)	
stable(X) $\leftarrow$ part_of(X,Y), bottom(Y), large(Y)	
stable(X) $\leftarrow$ part_of(X,Y), body(Y), part_of(X,Z), support(Z), above(Y,Z)	
open_vessel(X) $\leftarrow$ part_of(X,Y), concavity(Y), upward_pointing(Y)	
Solutions	
RUTH:	<i>delete</i> cup(X) $\leftarrow$ liftable(X), stable(X), open_vessel(X) <i>add</i> cup(X) $\leftarrow$ liftable(X), open_vessel(X), part_of(X,Y), bottom(Y), small(Y)
Time: 2.48 secs	
JIGSAW:	<i>replace</i> cup(X) $\leftarrow$ liftable(X), stable(X), open_vessel(X) <i>by</i> cup(X) $\leftarrow$ liftable(X), open_vessel(X), part_of(X,Y), bottom(Y), small(Y) <i>and</i> cup(X) $\leftarrow$ liftable(X), open_vessel(X), part_of(X,Y) body(Y), part_of(X,Z), support(Z), above(Y,Z)
Time: 0.06 secs	

Third, one can regard unfolding as a specialization technique that makes explicit use of the available background knowledge. Finally, one unfolding step can introduce more than one literal in the body of a clause. Hence one application of unfolding may correspond to several specialization steps. However, this can also

be seen as a disadvantage, since there is a danger of overspecialization. This was illustrated by a small number of experiments where RUTH found a solution but JIGSAW did not.

3 Conclusion

The system JIGSAW that we have presented in this work is an example of integrating independently developed techniques. Benefiting from the common logical framework of the ILP research field, we were able to extend the theory revision system RUTH. More specifically, we integrated SPECTRE's specialization operator together with the impurity measure used for formulating a preference criterion. Experimental results using the Student Loan domain and the CUP domain show that the integration effort is beneficial, and they point out the advantages (and also some caveats) concerning the use of an unfolding technique.

Acknowledgements. Research for this paper is funded by the ESPRIT BRA project Nr.6020 (Inductive Logic Programming). Part of the work reported was done during a visit of Hilde Adé to the University of Stockholm, made possible by a grant of the Belgian Fund for Scientific Research.

We wish to thank Gunther Sablon, Maurice Bruynooghe and Luc De Raedt for their comments on earlier versions of this paper.

References

1. H. Adé, L. De Raedt, and M. Bruynooghe. Theory Revision. In *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 179–192, 1993.
2. H. Adé, B. Malfait, and L. De Raedt. RUTH : an ILP Theory Revision System. In *Proceedings of the 8th International Symposium on Methodologies for Intelligent Systems (ISMIS94)*, 1994.
3. H. Adé, B. Malfait, and L. De Raedt. RUTH : an ILP Theory Revision System. Technical Report CW-194, Department of Computer Science, Katholieke Universiteit Leuven, 1994.
4. F. Bergadano and A. Giordana. A knowledge intensive approach to concept induction. In *Proceedings of the 5th International Workshop on Machine Learning*. Morgan Kaufmann, 1988.
5. H. Boström and P. Idestam-Almquist. Specialization of Logic Programs by Pruning SLD-Trees. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 31–48. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.
6. L. De Raedt and M. Bruynooghe. Belief updating from integrity constraints and queries. *Artificial Intelligence*, 53:291–307, 1992.
7. R. Korf. Depth-first iterative deepening : an optimal admissible search. *Artificial Intelligence*, pages 97–109, 1985.
8. H. Tamaki and T. Sato. Unfold/fold Transformations of Logic Programs. In *Proceedings of the Second International Logic Programming Conference*, pages 127–138, 1984. Uppsala University, Uppsala, Sweden.