Integrity Constraints and Interactive Concept-Learning.

Luc De Raedt

Maurice Bruynooghe

Bern Martens

Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, 3001 Leuven, Belgium; email: lucdr@cs.kuleuven.ac.be

Abstract

We show how our interactive concept-learner Clint can be enhanced with capabilities to handle integrity constraints. In this version of Clint, the user may supply general first order logic clauses as integrity constraints to the system. The system will then assure that these constraints are satisfied by the learned knowledge base.

1 INTRODUCTION.

In (De Raedt & Bruynooghe 90c) we have analysed the differences between *incremental concept-learning* and *intensional knowledge base updating* (see e.g. Bry 90, Guessoum & Lloyd 90). One of the conclusions of this comparison was that integrity constraints are very useful for inductive concept-learning as they allow to make the learning process very knowledge intensive. Whereas the use of integrity constraints is common in database technology (see e.g. Gallaire et al. 84, Reiter 90), it is new for concept-learning. Using integrity constraints, one can easily specify properties about the target knowledge base to be learned.

We will show how our interactive concept-learner Clint (see De Raedt 91, De Raedt & Bruynooghe 89,90ab) can be modified to assure that the integrity constraints are satisfied by the learned knowledge base. When a learned knowledge base violates a given constraint, the system analyses the violation and generates membership questions to the user. The answers to these questions result in additional examples for the system, which are used as the starting point of a new learning phase by the system. The resulting system converges in the limit – under certain conditions (see De Raedt 91) – to a target knowledge base.

As an application of the technique, we will show how the extension of Clint to cope with a three-valued logic and an explicit negation described in (De Raedt & Bruynooghe, 90b) can be obtained as a special case of

the algorithm to handle integrity constraints.

This paper is organized as follows : in section 2, we present a logical framework for concept-learning, in section 3, we briefly review the basic principles of Clint, in section 4, we show how Clint can be adapted to cope with integrity constraints, in section 5, we discuss the relation with our technique to use a three-valued logic; and finally, in section 6, we conclude and touch briefly on related work.

2 PROBLEM-SPECIFICATION.

Clint is an interactive concept-learner, which combines many interesting features : it generates examples, copes with indirect relevance, uses knowledge, shifts its concept-description language bias, and identifies concepts in the limit. Clint uses a first order logical framework. Within this framework, a *concept* is a predicate, which is defined by a set of definite Horn-clauses (Lloyd 87), i.e. Prolog without negation. Horn-clauses in Clint are organized into sets satisfying certain syntactic restrictions (see De Raedt & Bruynooghe 90a). These sets are called *languages*. One such language requires that all variables that occur in the body of a clause also occur in the head of the clause. Because each language allows to formulate only a certain (finite) number of concept-descriptions, Clint uses a series of languages $L_1, L_2, \dots, L_M$, which is ordered according to growing expressiveness; for a formal definition of these languages, see (De Raedt & Bruynooghe 90a). When Clint discovers that it cannot describe the concept in the description language L_i , it will shift the language bias to L_{i+1} . *Examples* in Clint are ground facts. Positive examples are true in the intended interpretation, negative ones are false. Clint assumes also that there is an oracle which is able to answer membership questions. A membership question asks for the truth-value of a ground fact in the intended interpretation. An important notion in this context is the notion of coverage : a clause c covers an example e w.r.t. a given knowledge base KB iff $\exists \theta : head(c)\theta = e$ and $KB \cup \{c\} \models body(c)\theta$.

An integrity constraint is a general first order logic clause, such as e.g. $\text{parent}(X) \rightarrow \text{father}(X) \vee \text{mother}(X)$ or $\text{father}(X) \wedge \text{mother}(X) \rightarrow \text{false}$. To judge whether a knowledge base KB of concept-descriptions, satisfies an integrity constraint IC , it is necessary to consider the completion $\text{Comp}(KB)$ in the sense of (Clark 78). A knowledge base KB satisfies an integrity constraint IC iff $\text{Comp}(KB) \models IC$. Satisfaction of a constraint $A_1 \wedge \dots \wedge A_n \rightarrow B_1 \vee \dots \vee B_m$ is confirmed by failure of the general goal $\leftarrow A_1 \wedge \dots \wedge A_n \wedge \neg B_1 \wedge \dots \wedge \neg B_m$. When this goal succeeds, the constraint is violated. In that case, the query will generate an answer substitution θ for the above general goal. If the substitution θ does not ground the query, one can always obtain a substitution γ such that $\theta\gamma$ grounds the query and $\theta\gamma$ is a legal answer substitution for the query. In the algorithm we assume that such a ground substitution is generated. Integrity constraints are assumed to be true in the intended interpretation, which explains why they have to be satisfied by the learned knowledge base. Notice that examples are a special case of integrity constraints. Therefore we will also talk about the satisfaction and violation of examples. By now, we are ready to present the problem-specification of Clint :

Given :

- a knowledge base KB of concept-descriptions
- a set of positive and negative examples E (there may be examples of several predicates in E)
- a set of integrity constraints IT (the integrity constraint theory; IT is assumed to be consistent)
- a series of concept-description languages $L_1, \dots, L_M$
- an oracle willing to answer membership questions
- a novel example or integrity constraint IC supplied by the user (IC is assumed to be consistent with IT)

Find : a knowledge base KB' such that 1) all integrity constraints and examples in $IT \cup E \cup \{IC\}$ are satisfied; and 2) all clauses in KB' belong to L_i for some i .

This problem-specification is a generalization of the one given in (De Raedt & Bruynooghe 89,90a) because of the use of integrity constraints. The reasons for introducing constraints can be summarized as follows (see also the literature on databases) : 1) constraints are not restricted to definite Horn-clauses while clauses in the knowledge base, i.e. concept-definitions, are; 2) when adding constraints to the knowledge base as inference rules, one would obtain a much more complicated knowledge base with many redundancies; this would decrease the efficiency of deductions; and 3) certain constraints should in no case be used as inference rules. Think for example about the following rule : $\text{human}(X) \wedge \text{age}(X,Y) \rightarrow Y < 150$. One does not

want to use such rules to derive facts for the predicate $<$. For a further discussion of these issues we refer to the database literature.

3 CLINT.

When integrity constraints are not considered, there are two cases in which Clint has to modify the knowledge base : an implied negative example and an unimplied positive example.

The first case is treated in a similar way as in Shapiro's Model Inference System (Shapiro 83) : a clause that is incorrect in the intended interpretation is located by asking membership questions about the nodes of a proof tree that proves that the example is satisfied; once the clause is located, it is retracted and positive examples that were covered by it are reconsidered.

The second case occurs when an unimplied positive example p is encountered. Clint uses a two-step procedure to induce a maximally general clause that covers the example. The induced clause is then asserted in the knowledge base. In the first step, Clint computes a starting clause c that implies the example when it is added to the knowledge base. This starting clause is derived by means of a function $SC(KB, p, L)$, which computes the set of most specific clauses in the concept-description language L that cover the positive example p w.r.t. the current knowledge base KB . Clint considers first the clauses in $SC(KB, p, L_1)$. If there is no clause c in $SC(KB, p, L_1)$ such that c covers no negative examples, Clint will shift the language bias by considering $SC(KB, p, L_2)$, ... This process continues until a clause is found that is consistent with the negative examples or all languages have been considered. If a consistent clause is found, it is passed to the second step of the generalization procedure. If no consistent clause is found in any of the given languages, the treatment of the example p is postponed until the knowledge base KB is modified; for a complete discussion of postponing, see (De Raedt 91). Postponing an example occurs when the system does not have enough knowledge to derive a consistent clause covering the example. When an example is postponed, the knowledge base is *temporarily* left as it is, but the system remembers that the knowledge base does not satisfy the example and when it has acquired new knowledge by processing other examples, the postponed example is reconsidered. Eventually, this should result in a knowledge base that satisfies the postponed example. Examples that are not postponed are *active*.

In the second step of the generalization procedure, the consistent starting clause is minimally generalized such that the generalized clause covers an example e that is not covered by the starting clause and the generalized clause is consistent with the negative examples. If a generalized clause is found that satisfies these conditions, one such example e is presented to the user

for classification. If the user confirms that e is true in the intended interpretation, the system generalizes, i.e. the generalized clause is made the starting clause and the generalization procedure is invoked on this new starting clause. When the example is negative, the example is stored and the system backtracks to other generalizations of the starting clause that fulfil the necessary requirements. This process continues until it is impossible to generalize the starting clause any further without covering negative examples. At that point, it is asserted in the knowledge base.

In (De Raedt 91), it has been shown that Clint identifies a set of concepts in the limit from any presentation of the examples, provided that the concepts can be described in terms of the languages of the series and the initial knowledge base. This means that when the system is supplied with enough examples and the concepts are learnable, there will be a finite time after which the learned knowledge base is correct in the intended interpretation. Clearly, at that point, no example will be postponed.

4 CONSTRAINTS IN CLINT.

Clint has been enhanced with two different techniques to handle constraints: the *lazy* method and the *eager* one (see De Raedt 91).¹

4.1 USING INTEGRITY CONSTRAINTS.

The general principle for using constraints for concept-learning is rather simple. If a constraint $p_1 \wedge \dots \wedge p_k \rightarrow q_1 \vee \dots \vee q_m$ is violated for a substitution θ we have that: 1) the definition of one of the predicates p_i in the body of the constraint is too general (i.e. $KB \models p_i\theta$ but should not); or 2) the definition of one of the predicates q_j in the head of the constraint is too specific (i.e. $KB \not\models q_j\theta$ but should).

This observation reveals that violated constraints can be used to generate examples that are not satisfied by the knowledge base. If there is only one literal in a constraint, it is clear that the violated instance of the literal itself is an example that is not satisfied by the knowledge base. However, if there is more than one literal in a violated constraint, the responsible examples have to be identified by performing credit-assignment on a violated instance of a constraint. In interactive learners, the credit-assignment procedure queries the oracle for the truth-value in the intended interpretation of the instances of the literals in the violated constraint.

As an example of credit-assignment, consider the following constraint: $\text{is_allowed_to_drive}(X,Y) \wedge \text{isa}(Y,\text{car}) \rightarrow \text{license}(X,\text{car})$. Let us assume that the constraint is violated for $\theta = \{X=\text{luc},$

$Y=\text{dyane}\}$. The source of the violation can be located by asking the oracle for the intended interpretation of the corresponding facts. The error is then the fact for which the intended interpretation is different from the truth-value according to the knowledge base. So, we have that $\text{is_allowed_to_drive}$ or isa is too general or license is too specific.

4.2 THE LAZY METHOD.

The examples and the discussion in the previous section motivate the following straight-forward *lazy* method to cope with constraints in the context of Clint.

When a constraint is violated, credit is assigned to the literals in a violated instance of the constraint. This allows to derive examples that are not satisfied by the given knowledge base. These examples are passed to Clint, which treats them as usual. Once Clint finishes learning from these examples, it verifies whether the constraints are satisfied by the resulting knowledge base. Violations at that point are treated in the same manner. This process of verifying constraints, deriving examples from violated instances of constraints, learning from examples by Clint, ... is repeated until the knowledge base satisfies all constraints. Then, the user is queried for additional constraints (or examples).

There is however one complication with this approach: the treatment of postponed examples. When an example is postponed for a predicate p , one should not expect the constraints mentioning the predicate p to be satisfied. Also, the predicates (and the constraints) that depend on the postponed predicate may be affected.

Definition 1 A predicate p depends on a predicate q w.r.t. a given knowledge base iff there is a clause in the definition of p that mentions q , or there is a clause in the definition of p that mentions a predicate r that depends on q .

Definition 2 A constraint c depends on a predicate p iff there is a literal in c of the predicate p , or there is a literal in c of a predicate q that depends on p .

Constraints which depend on a predicate for which a postponed example exists are postponed. They are reactivated when all examples causing their postponement are reactivated.

Definition 3 A constraint c is postponed iff c is an example that is postponed, or there is a postponed example $p(t_1, \dots, t_n)$ such that c depends on p .

Clearly, if a constraint is not postponed, it is *active*. Notice that the class of active constraints evolves when examples become postponed or active.

¹Here we will only present the lazy method.

```

procedure IC_CLINT
  while the user wants to input constraints
    read the next constraint (or example) c
    call handle_constraint(c)

procedure handle_constraint(c : constraint)
  if c is active
    then if c is violated for a ground substitution  $\theta$ 
      then while there is no violated literal  $l \in c\theta$ 
        ask the oracle to classify a literal
         $l' \in c\theta$  with unknown int. interpretation
        select a violated ex.  $p(t_1, \dots, t_n) \in c\theta$ 
        supply it as an example to Clint
        this activates the normal learning proc
        for all constraints c that depend on p
          call handle_constraint(c)
        Reconsider the postponed examples
        this may (re)activate constraints

```

Observe that examples and constraints are now treated uniformly at the top-level by Clint. At certain points however examples are handled in a different way than other constraints.

The procedure `handle_constraint` only takes into account active constraints. The class of active constraints evolves because of possible changes in the set of postponed examples in the last statement. Given a violated active constraint, the procedure `handle_constraint` performs credit assignment in order to find an example that is not satisfied by the current knowledge base. This is realized by asking the oracle for the truth-value of literals in a violated instance of the constraint. Note that if the violated constraint contains only one literal (as in the case of examples), its truth-value is known. Then the oracle need not be bothered with questions. In any case, the while loop allows to identify a violated example. This example is then passed to the normal procedures of Clint specified in (De Raedt, 91), (De Raedt & Bruynooghe, 89). Because the knowledge base will be modified by Clint in order to satisfy the example, the predicates depending on the example may also be affected. Therefore, the system verifies whether these depending examples and constraints are still satisfied by the modified knowledge base. If not, appropriate action is taken. In general, it is a good heuristic to test examples before general constraints as this usually requires less questions to the oracle. After verifying the possibly affected constraints, the system reconsiders the postponed examples.

Let us now illustrate the algorithm on a small example using the following knowledge base.

```

female(alice) ←      male(luke) ←
female(kate)  ←      male(morris) ←
female(rose)  ←      male(jeff)  ←
female(mary)  ←      male(steve) ←

```

```

parent(jeff, kate) ←  parent(alice, kate) ←
parent(alice, luke) ← parent(mary, luke) ←
parent(rose, mary)  ← parent(morris, jeff) ←
parent(alice, steve) ←

```

```

father(X,Y) ← parent(X,Y) ∧ female(Y)

```

Notice that this knowledge base contains an erroneous definition: the clause for `father`. Assume also that `parent`, `male` and `female` are correct in the intended interpretation. This means that no questions will be asked about these predicates. Suppose the user now enters the following two integrity constraints:

- (1) $\text{mother}(X,Y) \wedge \text{father}(X,Y) \rightarrow \text{false}$
- (2) $\text{parent}(X,Y) \rightarrow \text{father}(X,Y) \vee \text{mother}(X,Y)$

Constraint (2) is violated for `parent(mary, luke) → father(mary, luke) ∨ mother(mary, luke)`. Therefore Clint asks whether `father(mary, luke)` is true. As the answer to the question is no and Clint knows that `parent(mary, luke)` is true, it decides that `mother(mary, luke)` must be true. This example is supplied to the basic procedures of Clint, which initiates a normal learning phase. The starting clause for `mother(mary, luke)` in L_1 is `mother(X,Y) ← female(X) ∧ male(Y) ∧ parent(X,Y)`. The generalization procedure generalizes by deleting `{male(Y)}` and asking for the truthvalue of `mother(rose, mary)`. As the value is true, Clint generalizes and asks for the truthvalue of `mother(morris, jeff)` (deleting `{female(X)}`). Since the answer is false, there are no further generalizations that satisfy the conditions. Therefore Clint asserts: `mother(X,Y) ← female(X) ∧ parent(X,Y)`. Now, Clint detects that constraint (1) is violated: `mother(rose, mary) ∧ father(rose, mary) → false`. As it already knows that `mother(rose, mary)` is true, it decides that `father(rose, mary)` must be false. This leads to retracting the incorrect clause for `father`. Then, Clint detects that constraint (2) is violated: `parent(morris, jeff) → father(morris, jeff) ∨ mother(morris, jeff)`. Therefore Clint asks whether `father(morris, jeff)` is true and as the answer is yes, this example is supplied to the normal procedures of Clint. This results in learning a correct definition of `father` after asking the additional membership-question `father(jeff, kate)` (positive) and taking into account the negative example `father(rose, mary)`. After adding the resulting clause `father(X,Y) ← male(X) ∧ parent(X,Y)` to the knowledge base, all known constraints are satisfied, and the learning process terminates.

5 AN APPLICATION : MULTI-VALUED LOGIC.

In (De Raedt & Bruynooghe, 90b), we argued that negation as failure is not suitable for inductive learning because it acts as if everything is known. As an alter-

native, we proposed to use a three-valued logic with an explicit negation. The logic has been realized by learning for each predicate p two sets of Horn-clauses : one (p) that defines the conditions under which p is true and one ($\text{not-}p$) that defines the conditions under which p is false. Note that p and $\text{not-}p$ are considered to be two different predicates ! Because we now have two different predicates, a (pos./neg.) example $p(t_1, \dots, t_n)$ also corresponds to an example for two different predicates : (pos./neg.) $p(t_1, \dots, t_n)$ and (neg./pos.) $\text{not-}p(t_1, \dots, t_n)$. In this way an example $p(t_1, \dots, t_n)$ can be classified in 4 different ways, by using a simple meta-interpreter :

- $p(t_1, \dots, t_n)$ is true if p covers $p(t_1, \dots, t_n)$ and $\text{not-}p$ does not cover $\text{not-}p(t_1, \dots, t_n)$
- $p(t_1, \dots, t_n)$ is false if p does not cover $p(t_1, \dots, t_n)$ and $\text{not-}p$ covers $\text{not-}p(t_1, \dots, t_n)$
- $p(t_1, \dots, t_n)$ is inconsistent if p covers $p(t_1, \dots, t_n)$ and $\text{not-}p$ covers $\text{not-}p(t_1, \dots, t_n)$
- $p(t_1, \dots, t_n)$ is unknown if p does not cover $p(t_1, \dots, t_n)$ and $\text{not-}p$ does not cover $\text{not-}p(t_1, \dots, t_n)$

In (De Raedt & Bruynooghe, 90b), (De Raedt, 91) we demonstrated that an interesting three-valued logic can be obtained by excluding the value inconsistent. The logic allows to reason about the unknown and to learn from negative information. The algorithm given in (De Raedt & Bruynooghe, 90b) is a special purpose algorithm to enforce the exclusion of the value inconsistent. Using constraints, however, it is very easy to obtain the same result. One only has to declare for each predicate p the constraint $p \wedge \text{not-}p \rightarrow \text{false}$, to the system. The eager method used with these constraints behaves in exactly the same way as the version of Clint of (De Raedt & Bruynooghe, 90b).

6 CONCLUSIONS.

We have introduced the use of integrity constraints in concept-learning and have shown how our interactive concept-learner could be adapted to enforce constraint satisfaction. The soundness of the method has been proven in (De Raedt 91). We believe that integrity constraints are very useful for concept-learning, because they allow to specify very rich forms of knowledge that is often easy to acquire from a user. Therefore it seems promising to enhance other concept-learners with constraint handling abilities. One interesting further direction of research is concerned with the integration of constraint-handling abilities in non-interactive concept-learners.

As far as we know, the use of integrity constraints for concept-learning is new. There is however one approach that is somewhat related from the constraint handling point of view : the system Blip (Wrobel 89) In

Blip there are facilities to reason at the meta-level and the meta-meta-level. From a certain perspective, integrity constraints can also be seen as meta-level statements (see e.g. Reiter 90).

Acknowledgements

Maurice Bruynooghe and Bern Martens are supported by the Belgian National Fund for Scientific Research.

References.

- F. Bry. Intensional updates : abduction via deduction. In *Proceedings of the 7th Int. Conf. on Logic Programming*. The MIT press, 1990.
- K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, 1978.
- L. De Raedt. *Interactive Concept-Learning*. PhD thesis, Dept. of Computer Science, KULeuven, 1991.
- L. De Raedt and M. Bruynooghe. Towards friendly concept-learners. In *Proceedings of the 11th IJCAI*, pages 849–856. Morgan Kaufmann, 1989.
- L. De Raedt and M. Bruynooghe. Indirect relevance and bias in inductive concept-learning. *Knowledge Acquisition*, 2:365–390, 1990.
- L. De Raedt and M. Bruynooghe. On negation and three-valued logic in interactive concept-learning. In *Proceedings of the 9th ECAI*, pages 207–212. Pitman, 1990.
- L. De Raedt and M. Bruynooghe. Unifying incremental concept-learning and intensional knowledge base updating. Technical report, Dept. of Computer Science, KULeuven, 1990. submitted.
- H. Gallaire, J. Minker, and J.M. Nicolas. Logic and databasis : a deductive approach. *Computing Surveys*, 1984.
- A. Guessoum and J.W. Lloyd. Updating knowledge bases. *New Generation Computing*, 8:71–88, 1990.
- J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, 2nd edition, 1987.
- R. Reiter. On asking what a database knows. In J. Lloyd, editor, *Computational Logic*. Springer-Verlag, 1990.
- Ehud Y. Shapiro. *Algorithmic Program Debugging*. The MIT press, 1983.
- S. Wrobel. Demand driven concept-formation. In K. Morik, editor, *Knowledge Representation and Organization in Machine Learning*, volume 347 of *LNAI*. Springer-Verlag, 1989.