A Case Study in the Use of Theory Revision in Requirements Validation

T.L. McCluskey
School of Computing and Mathematics,
The University of Huddersfield,
HD1 3DH, UK
lee@zeus.hud.ac.uk.

M.M. West
School of Computing and Mathematics,
The University of Huddersfield,
HD1 3DH, UK
m.m.west@zeus.hud.ac.uk.

Abstract

Research emanating from Artificial Intelligence has throughout its history contributed to techniques and ideas in Software Engineering. We describe in this paper a case study showing the use of theory revision to the refinement of a formally specified requirements model. In a previous project we were contracted to create a precise model of the complex criteria governing the separation of aircraft profiles in Atlantic Airspace. During that work it became clear that the (automated) validation of the model was of the utmost importance, and in our current project we have used machine learning tools to provide extra support in bug identification, bug removal and maintenance of such a requirements model. In this paper we give an overview of the domain, identify a relevant learning bias which makes search for revisions tractable, and describe a systematic approach for the application of theory revision to such a model. We illustrate the approach with results of experiments where theory revision techniques have identified and removed errors, and induced a new part of the model.

Keywords Theory Revision, Machine Learning and Software Engineering, Requirements Model, Automated Validation.

1 INTRODUCTION

Promoting and maintaining the quality of requirements specifications has a vital role in the engineering of software. Some software projects, such as those involving safety-critical elements, necessitate that precise, mathematical specifications of their requirements domains be constructed. Such 'requirements models' must be *validated* to satisfy certain major quality objectives such as accuracy, completeness, usability, and understandability, and during the model's lifetime it is likely to be incrementally updated, and will require re-validation. Validation and maintenance of realistic domain models is a very time consuming, expensive process where the role of support tools in vital. The process is best carried out using diverse techniques, and one of the most useful techniques is to *test* an animated form of the model. Even when an animated version is available, however, it is not easy to pinpoint the causes of bugs and subsequently provide the correct revision that eliminates them.

In this work we view a precise requirements model as an imperfect theory of the requirements domain that needs to undergo refinement to remove bugs or to reflect changes in the domain, and we formulate the problem as one of *theory revision*. The case study uses an air traffic control requirements model developed in a previous project called FAROAS (McCluskey et al. 1995). The model represents aircraft separation criteria and conflict prediction procedures relating to airspace over the North East Atlantic, and is recorded in the 'Formal Methods Europe Applications Database'[1]. The model's 'conventional' support environment had been used for verification and validation of models written as a set of axioms in *many sorted first order logic* (Meinke and Tucker 1993) – here abbreviated to msl. During the current IMPRESS project we extended the environment to include machine learning tools which perform blame assignment, explanation-based generalisation and theory revision (TR). We show in this paper how we overcame the in-

[1] web site http://www.cs.tcd.ie/FME

tractability problems in fielding TR by firstly focusing on likely faulty axioms sets using a blame assignment algorithm, then targeting for revision the ordering relations between values of *ordinal* sorts. We describe a method and a class of revision operator that has been successfully used to (a) find and remove bugs from the requirements model, and (b) to construct a new part of the model to cope with the changing of criteria for vertical separation between subsonic aircraft. Thus TR can be seen as a useful embedded component within a requirements validation regime for high integrity systems.

2 THE ATC DOMAIN

2.1 DOMAIN DESCRIPTION AND ACQUISITION

'Shanwick' is a large area of airspace in the eastern half of the North Atlantic, managed by air traffic control centres in Shannon, Ireland and Prestwick, Scotland. Controllers must organise this airspace daily, taking into account such factors as weather and the desired flight paths of aircraft companies. They plan the four dimensional flight profiles of aircraft crossing this airspace in good time before the aircraft reaches the boundary, and for this task require a precise definition of aircraft separation criteria, and an algorithm for predicting conflicts. The controllers are supported in their safety-critical work by a computer system which performs predication and resolution of conflicts between pairs of flight profiles, and our involvement came about as part of the research and development concerning the requirements specification of a replacement for their current flight data processing system.

In the FAROAS project, we created a precise requirements model (called the *CPS*) of the *conflict prediction* of aircraft flight profiles through the Shanwick airspace, together with a software support environment. Knowledge sources used were manuals of air traffic control, existing computer systems documentation, and air traffic control officers themselves. The current CPS contains a kernel of 300 - 400 axioms in msl representing aircraft profile separation criteria and a conflict prediction method; the total number of axioms in an instance of the model, which includes airspace and short term flight information for a day's set of profiles, exceeds two thousand. The model is structured into 23 sorts, and is enriched with real and natural numbers. An example of an axiom in the CPS is provided in Figure 1. This represents the condition for a vertical separation of 2,000 feet, where segments

```
(Segment1 and Segment2
are_subject_to_oceanic_cpr) =>
[(the_min_vertical_sep_Val_in_feet_required_for
Flight_level1 of Segment1
and Flight_level2 of Segment2) =  2000 <=>
[[(both Segment1 and Segment2
 are_flown_at_subsonic_speed)
 & (one_or_both_of  Flight_level1 and
Flight_level2 are_above  FL 290) ] or
 [(one_or_both_of Segment1 and Segment2
   are_flown_at_supersonic_speed) &
 (one_or_both_of Flight_level1 and
Flight_level2 are_at_or_below  FL 430) ] ] ]
```

Figure 1: Condition for a Minimum Vertical Separation of 2000 feet

are roughly 'straight' components of an aircrafts profile. Either the two aircraft are both subsonic and are flying above FL 290 (29,000 feet) or one or both are supersonic and are flying at or below FL 430.

2.2 A CONVENTIONAL SUPPORT ENVIRONMENT

The CPS is highly structured, with axioms containing very complex conditions, but the support of an integrated tools environment alleviates its analysis and manipulation. In the FAROAS project diverse validation was carried out using tight syntactic checking, semantic internal consistency checks, expert inspection, simulation and batch testing. The most complex tool in the environment is a translator program which inputs the CPS (or more generally a set of wffs in msl), together with a syntactic definition of the tailored msl language expressed in grammar rules. It parses the wffs and outputs an animation of them by translating them into what we call 'EF' (execution form). This is similar to general clausal form, except clauses may contain nested negation and disjunction in their bodies. EF obeys the syntax rules of Prolog and is executable by a Prolog interpreter. This parsing and translation process takes less than 5 minutes for all of the CPS, and its translated form we term CPS_{EF} [2].

Flight profiles are input to the software environment as msl axioms and are translated into EF. Although in theory *any* part of the CPS can be tested, virtually all of the instances we obtained were for the 'top level'

[2]all software tools reported in this paper are implemented in Sicstus Prolog and were tested using a SUN SPARC station 4 processor with 32MB memory

conflict axiom defining the mixfix *conflict predicate*:

SegmentX of Profile1 and SegmentY of Profile2
are_in_oceanic_conflict

A day's worth (500 - 800) of cleared aircraft profiles, where each profile is cleared with (say) the last 20 cleared aircraft in chronological order, results in approximately 10,000 instances classified as false for the conflict axiom, where the SegmentX and SegmentY are existentially quantified variables representing segments of an aircrafts profile.

In the rest of this paper we use the following notation: a classified instance that is labelled true and which CPS_{EF} classifies as true is called truly positive ('TP'), and denoted e^{TP}, whereas one that executes to false is called falsely negative('FN') and denoted e^{FN}. A classified instance labelled false which executes to false using CPS_{EF} is called truly negative ('TN'), whereas one that executes to true is called falsely positive ('FP'); these are denoted e^{TN} and e^{FP} respectively. Early phases of validation during the FAROAS project involving syntax checking and painstaking expert inspection increased the accuracy and completeness of the CPS so that dynamic testing of the conflict axiom resulted in a large number of TN, with a smaller (about 5 per cent) but significant number of FP. Although investigation of the set FP helped to find bugs, it became clear that more powerful tools for bug identification and removal were needed when building up and maintaining such a complex, precise domain model.

3 APPLICATION OF THEORY REVISION

3.1 RATIONALE

The principle objectives of the *current* project, IMPRESS, were to test the use of ML to help improve the quality (in terms of accuracy and completeness) of a formalised requirement specification written in msl and to increase the quality of the CPS itself. The focus was not only on bug removal but also on maintenance, to support the inevitable changes in the requirements model. Since we started with an existing symbolic domain model, the principle ML paradigm we decided to use was theory revision (Wrobel 1996). Our initial formulation was as follows:

Revisable theory: a subset of CPS_{EF} clauses. We can keep some parts of the CPS_{EF} immune or 'shielded' from the revision process, as they were adequately validated using other processes. For example, it may be assumed that the 'top level' axioms, i.e. those defining the basics of separation in terms of vertical and horizontal dimensions, are correct. The target concept is the conflict predicate shown above.

Training Instances: The main source is a day's worth of cleared flight profiles supplied directly by the UK National Air Traffic Services. The conflict predicate can be executed, and when instantiated with pairs of cleared flight profiles should return false. The nature of the application skews the training somewhat as it is driven by FPs only. However, experiments have also been conducted with other, lower-level predicates as target concepts, such as those involved in vertical conflict. Instances associated with these conflicts are classified into FNs and TPs as well as TNs and FPs.

Learning Biases: the language used for the CPS_{EF} is strongly typed, which provides a useful constraint in the generalising or specialising of predicates. Also we assume a minimal revision bias: we know from other forms of validation that its structure mirrors the requirements domain, and so we assume only minimal revisions are necessary.

Given the general problem outlined above, we implemented a standard, simple TR algorithm with operators such as 'add antecedent' and 'delete clause'. However we only confirmed that a 'mainstream' approach to TR would be impracticable. Even given the biases, the potential space of revisions is enormous, and 'hill-climbing' with traditional TR operators appears out of the question. The CPS_{EF} executes the conflict axiom at an average rate of about one test per minute and results in a batch of tests taking perhaps days to execute!

We also investigated using TR tools, available via ftp, but came to the conclusion that we would need to build our own environment (West et al. 1996). This was based on the need for a flexible tool base given we were embarking on a research project, and the need for tool integration, particularly with our existing validation tools from the earlier FAROAS project. Moreover, the existing tools we examined were not powerful enough for our use. For example, FORTE (Richards and Mooney 1995), though well tested, could not cope with negation or functors. Both the latter are important features of the CPS. Also, while tools presented in the literature had been tested on theories of the order of 10's of predicates calls within a similar number of non-atomic clauses, the CPS_{EF} contains c.2,000 predicate calls within more than 300 non-atomic clauses.

3.2 ORDINAL SORTS

The key to our approach lay in the introduction of a further bias. Although the sorts comprising universes of objects are distinct, each sort can be characterised as either *ordered* or not (Birkhoff 1967). The sorts which are ordered are termed *ordinal* in this paper, and those which are not are termed *nominal*. Associated with each ordinal sort X is an *arbitrary* binary, transitive, ordering relationship we call '$\succeq$'. Examples of ordinal sorts are *Flight Level, Time* and *Latitude*, where primitive order relations are for example 'is above', 'is later than', 'is west of'. Examples of nominal sorts are *Aircraft, Airspace, Segment, Profile*. Technical specifications such as the CPS include many references to ordinal sorts, and our experience in the validation phase had shown that very often clauses involving comparisons and limits were to blame. For example, of the 17 *primitive* order relations defined in the CPS's grammar, there are 204 occurrences of them within the current version of the CPS.

Each axiom in the CPS has as its *variable* domain:

$$X_1 \times \ldots \times X_n \times D_1 \times \ldots \times D_m, n, m \geq 0.$$

where each X_i is an ordinal sort and each D_j is a nominal sort. We will focus on axioms containing '$\succeq$': examples are $x \succeq a, x_1 \succeq x_2$, where x, x_1, x_2 are ordinal variables and a a constant, limiting value of some appropriate sort. The axiom involving ordering might be an equation defining a function, $\mathcal{F}$ which returns different values for different subsets of its domain $X_1 \times \ldots \times X_n \times D_1 \times \ldots \times D_m$, or a predicate, $\mathcal{P}$. The statements involved in the definition of $\mathcal{P}$ return 'true' or 'false' for different subsets of its domain $X_1 \times \ldots \times X_n \times D_1 \times \ldots \times D_m$. If we factor out the X_i from the D_j components, for each main predicate and function, for each tuple $(d_1, \ldots, d_m)$ of values, there is defined an n dimensional region $\mathcal{R}(d_1, \ldots, d_m)$ – the domain of applicability of the predicate or function. For the remainder of the paper, we shall shorten this to $\mathcal{R}$.

When the CPS is translated to executable form, each axiom becomes a Prolog clause. The regions described above, for the main axioms, now become regions for Prolog clauses, where tuples of variables now become tuples of Prolog variables. In the case where a wff is an equation, its domain is extended by the returned term.

3.3 SIMPLE REVISIONS

Given a concept (for example the conflict predicate) and a set of positive instances of the concept, translated to EF, then the set of proof trees of the instances involve a set of clauses. Consider a clause C from this set, where its (Prolog) variables are the tuple `X1, ... ,Xn, D1, ... ,Dn`, where the `X`'s and `D`'s are ordinal and nominal respectively (where $n > 0$). Each instance of C is associated with an n-tuple of ordinal variables $(x_1, \ldots, x_n) = \mathbf{x}$. We should expect positive instances to have $\mathbf{x} \in \mathcal{R}$. The region $\mathcal{R}$ is defined by logical expressions $\mathcal{E}(\mathbf{x})$ involving ordinal variables $\mathbf{x}$ and is not necessarily connected. In a similar manner a clause C which does *not* succeed and which is involved in a failed proof tree (or trace) of a negative instance will have $\mathbf{x} \notin \mathcal{R}$. In order for instances to *fail* where they previously succeeded, and vice-versa, then region $\mathcal{R}$ is revised to become region $\mathcal{R}'$, for clause C. We classify revision operators that may change a clause containing an ordinal literal into two: *simple* and *composite*. Simple operators involve deletion and addition of antecedents from a clause, as in conventional TR, although the antecedents are restricted to occurrences of order relations This kind of operator is mainly for finding and possibly correcting bugs in the model. For example, the condition $x \succeq y$ may be either removed or replaced by $y \succeq x$. This latter is akin (in 2-D geometrical terms) to examining reflections of the region about a straight line.

3.4 COMPOSITE REVISIONS

The second kind of revision is designed to clarify requirements involving complex conditions involving limiting values, which might not have been captured initially from the expert sources, and also to cope with changing requirements. We first deal with *specialisation*. Suppose C to be a candidate for revision, or *revision point*, where C contains antecedents of the form $x \succeq a$, a a constant. Further, suppose C succeeds with instances θ_iC in proofs of some training instances $e_i^{FP} \in FP$, with tuple $\mathbf{x_i}$ the ordinal variables of θ_iC. Suppose also that C is successful with instances ϕ_jC in proof tree of training instances e_j^{TP}; the tuples $\mathbf{y_j}$ are the ordinal variables of ϕ_jC. Failure of C would ensure the removal of some instances e_i^{FP} and in order that C should fail, we need to *revise* $\mathcal{R}$ to $\mathcal{R}'$.

$$\forall \mathbf{x_i} : \mathbf{x_i} \notin \mathcal{R}'.$$

However, in order that C should safely succeed for correctly classified instances, then tuples $\mathbf{x}$ associated

with e^{TP} should *not* be removed from $\mathcal{R}$. Thus

$$\forall \mathbf{y_j} : \mathbf{y_j} \in \mathcal{R}'.$$

We calculate the two sets of tuples:

$$S^{FP} = \{\mathbf{x_i} \mid \mathbf{x_i} \text{ ordinal variables of } \theta_i \mathrm{C}\},$$
$$S^{TP} = \{\mathbf{y_j} \mid \mathbf{y_j} \text{ ordinal variables of } \phi_j \mathrm{C}\}. \quad (1)$$

This allows for the fact that the mis-classification of some/all of the instances e^{FP} may have arisen from another clause. Recalling that the variables of C are $\mathbf{x} = x_1 \ldots x_n$, we denote the minimum and maximum values of variable component x_i of the S^{FP} variables by min_i^{FP}, max_i^{FP} respectively. In a similar manner the minimum and maximum values of components of the S^{TP} variables are respectively min_i^{TP}, max_i^{TP}.

We induce the following, that for instances $\theta_i C$ to fail, the new specialised region is $\mathcal{R}$ less an n dimensional interval $\mathcal{R}_{FP}$ bounded by min_i^{FP}, max_i^{FP}. We have

$$\mathcal{R}_{FP} = \{(x_1 \ldots x_n) \mid min_1^{FP} \succeq x_1 \succeq max_1^{FP} \wedge \ldots \wedge min_n^{FP} \succeq x_n \succeq max_n^{FP}\} \quad (2)$$

However for instances $\phi_j C$ to succeed, $\mathcal{R}'$ must include an n dimensional interval $\mathcal{R}_{TP}$ bounded by min_i^{TP}, max_i^{TP}:

$$\mathcal{R}_{TP} = \{(x_1 \ldots x_n) \mid min_1^{TP} \succeq x_1 \succeq max_1^{TP} \wedge \ldots \wedge min_n^{TP} \succeq x_n \succeq max_n^{TP}\} \quad (3)$$

We have

$$\mathcal{R}' = (\mathcal{R} \setminus \mathcal{R}_{FP}) \cup \mathcal{R}_{TP} \quad (4)$$

($\mathcal{R}_{TP} = \mathcal{R}_{FP}$ is the limiting case, where all of the misclassified instances have arisen from another clause.) In order to accomplish the revision, we specialise the clause C as follows: every occurrence in the unrevised body of C of the logical expression $\mathcal{E}(x_1 \ldots x_n)$, should be replaced in the revised body of C by $\mathcal{E}'(x_1 \ldots x_n)$, which is defined:

$$(\mathcal{E}(x_1 \ldots x_n) \wedge \neg\, (min_1^{FP} \succeq x_1 \succeq max_1^{FP} \wedge \ldots \wedge min_n^{FP} \succeq x_n \succeq max_n^{FP})) \vee (min_1^{TP} \succeq x_1 \succeq max_1^{TP} \ldots \wedge min_n^{TP} \succeq x_n \succeq max_n^{TP}) \quad (5)$$

Generalisation can be explained in a similar manner: in order for instances e^{FN} to succeed, their $\mathbf{x}$ components are added to the region. However instances e^{TN} must still fail. We calculate sets S^{FN}, S^{TN} and regions $\mathcal{R}_{FN}, \mathcal{R}_{TN}$ in an analogous manner to S^{FP}, S^{TP} in (1) and $\mathcal{R}_{FP}, \mathcal{R}_{TP}$ in (2).

We induce the following, that for some FN instances to succeed and *all* the TN instances to fail, then the new generalised region is

$$\mathcal{R}' = (\mathcal{R} \cup \mathcal{R}_{FN}) \setminus \mathcal{R}_{TN} \quad (6)$$

In order to accomplish this, we generalise the clause C, so that every occurrence in the unrevised body of C of the logical expression $\mathcal{E}(x_1 \ldots x_n)$, should be replaced by $\mathcal{E}'(x_1 \ldots x_n)$, in an analogous manner to (5). In the next section we explain how these simple and composite revisions were applied to CPS_{EF}.

4 EXPERIMENTS WITH TR TOOLS

We report experiments involving two kinds of data set:

1. The first data-set consists of training instances from a day's cleared flight profiles recorded in January 1995. This data was used with the object of testing our current techniques using 'simple' ordinal operators. When tested, the errors in the CPS as measured by this training set were 33 in 5070, having been previously reduced by other techniques. Use of TR with simple operators further reduced the errors to 1 in 5070.

2. 'Reduced separation for vertical minima' (RVSM) criteria have recently been introduced for certain types of aircraft in North Atlantic airspace. A days cleared flight profiles were provided (from April 1997), where clearance is subject to the new revised criteria (of flight levels) for vertical separations for pairs of aircraft.The new criteria involved flight level *intervals* for both aircraft and was not captured by our current theory. 'Simple' ordinal operators were not suitable for revisions of the type investigated, so this data-set was used for independently testing 'composite' ordinal operators. After the CPS was revised using simple operators, it was then re-revised using training instances from post-RVSM data and composite operators. All 121 errors resulting from the data cleared by the changed separation standard were eliminated by the method.

4.1 THE METHOD AND RESULTS

The method shown here is a general one for revision of a theory, Γ, containing significant ordinal variables,

although it is based on experiments with the CPS. We have implemented TR in a manner based on the 'geometrical' discussion above, and integrated it into our legacy software environment (McCluskey et al. 1995). This architecture has had to be both flexible and experimental in response to the inherent complexity of formal specifications of the size and expressiveness of the CPS. For example, we have had to develop a form of blame assignment that can cope with proof trees from general clausal form logic programmes. This involves first unfolding and then transforming negative literals using De Morgan's laws and is detailed in (West et al. 1997).

4.1.1 An Error Removal Experiment

The algorithm for simple ordinal revisions is based on conventional theory revision techniques and used hill-climbing based on the accuracy of the theory Γ. It is shown in Figure 2. The *potential* of a clause C is the number of instances in which it succeeds in a proof tree, and the negative potential is the number of instances in which it fails in a proof trace; this notion was used in the description of the FORTE tool (Richards and Mooney 1995).

1. Collect training set of instances of a concept, L, known to contain misclassified instances.

2. Classify training instances into TNs, FPs, FN's, TPs and calculate accuracy.

3. Run blame assignment on instances in FN giving set of potential-pairs, P1.
$P1 = \{(C, N) \mid C$ revisable clause in Γ and N is the potential of $C\ \}$
Find subset OP1 of P1:
$OP1 = \{(C, N) \mid (C, N) \in P1 \wedge C$ contains an ordinal relation $\}$.

4. Repeat step 3 for instances in FP giving set of potential-pairs, P2, and subset OP2.
Let $OP = OP1 \cup OP2$.

5. Revision points $= \{C \mid (C, N) \in\}$ OP.
Apply each simple TR operator to each revision point, in order of C with largest potential. Implement the best revision.

6. Repeat from step 2, unless a maximum accuracy has been reached.

Figure 2: Algorithm for Simple Ordinal Operators

Using a day's worth of training instances (cleared flight profiles) we obtained 33 FPs and 5037 TNs out of 5070 runs of the conflict axiom. Because of the complexity of the criteria the revision was accomplished by focusing the revision space to the longitudinal separation criteria (i.e. concept L in Figure 2) rather than from the initial training instances. L was selected by studying the output of blame assignment for all the FPs, and the generalised explanation output for individual FPs. Longitudinal separation values in minutes can be 5,6,7,8,9,10,15,20 or 30, and the CPS contains formalised criteria for all of these. 75 new training instances were generated from proof trees and proof traces in which a longitudinal separation value of 10 minutes was assigned to two aircraft at least one of which is flying at subsonic speed. The training instances included 25 FN and 50 TP, the concept being:

the_basic_min_longitudinal_sep_Val_in_mins_
required_for(Segment1,Segment2) = 10.

The TP's were generated by re-running the day's worth of instances, and identifying those in vertical conflict that gave a longitudinal separation of 10 minutes, but were *not* in overall conflict according to both air traffic control officers and the CPS_{EF} (thus lowering the possibility of noisy data). The FN's of concept L are derived directly from the 33 false positives from the conflict predicate. The algorithm using simple reverse and dropping conditions operators returned a new theory with two clauses altered by both the operators; after revision, 74 of the training instances were covered, and only 1 (FN) uncovered. Significantly, one of the clauses that was revised, defining the predicate:

are_after_a_common_pt_from_which_profile_tracks
_are_same_or_diverging_thereafter_and_at_which
_both_aircraft_have_already_reported_by

has been subsequently identified as an incorrect reading of an ATC Manual.

4.1.2 A Requirements Change Experiment

The method for implementing composite ordinal operators is shown in Figure 3. Steps 1 and 2 are similar to those of the simple operator. If after step 2, FN is larger than FP, then generalisation of a clause C occurs in steps 4b .. 8b in a similar manner. Note that the driver for the algorithm is the *stability* of the clauses in OP, rather than the increase in accuracy of Γ.

1. Collect training set of instances of a concept, known to contain misclassified instances. Initialise: D = Deleted clauses = { } , A = added clauses = { }.

2. Classify training instances into TNs, FPs, FN's, TPs and calculate accuracy.

3a. Specialise Γ in a manner indicated by steps 3a .. 8a. Run blame assignment on instances in FP giving set of potential-pairs, P.
P = $\{(C, N) \mid C$ revisable clause in Γ and N is the potential of C $\}$.
Find subset OP of P:
OP = $\{(C, N) \mid (C, N) \in$ P $\wedge$ C contains an ordinal relation $\}$.

4a. Select pair (C, N) where N is maximum of $\{N \mid (C, N) \in\}$ OP.

5a. Calculate the n dimensional regions $\mathcal{R}_{FN}, \mathcal{R}_{TN}$ defined by (1), (2) and (3).

6a. If $\mathcal{R}_{FN}, \mathcal{R}_{TN}$ are not equal
set head of C' := head of C;
set body of C' := body of C with $\mathcal{E}$ replaced by $\mathcal{E}'$ from (5)
else
delete C from OP and repeat from step 4a.

7a. Replace C with C' and calculate accuracy.

8a. $D' = D \cup C$; $A' = A \cup C'$.

9. Repeat from step 1 until OP is stable or accuracy is 100 %.

Figure 3: Algorithm for Composite Ordinal Operators.

Because of the safety-critical nature of the application, and the fact that some data values may occur only rarely, it is necessary to check that a clause $C \in A$ originally arising from a function, remains defined over its intended domain. If this is the case, a *post-processing* phase is necessary.

204 training instances (classified according to post-RVSM criteria) of the conflict axiom were used to revise the CPS using the algorithm in Figure 3. However, revisions were confined to ordinals of the form 'is_above'. When tested, there were found to be 121 FP instances, and 83 FN instances. The 'blame assignment pinpointed the clause
'`the_min_vertical_sep_Val_in_feet_required_for( A, B, C, D, 2000)`'
as a revision point and the results are shown below. (The 'limitvar' predicate is a device for marking variable occurrences.) As can be seen, for supersonic aircraft, the criteria is unaltered. The criteria for a vertical separation of 2000 feet are specialised; they exclude the region where both flight levels are between FL 330 and FL 370 as shown in the following result:

```
lengths of FN, FP, TN, TP

0 121 83 0

%% set P.
[potential(1,121),potential(2,121), ..,
potential(23,1),potential(26,121), ..]

%% list of revision points
[26]

New_accuracy = 100.0, Old_accuracy = 40.686
%%revised code for 2000
the_min_vertical_sep_Val_in_feet_required_for(
   A, B, C, D, 2000) :-
   (both_are_flown_at_subsonic_speed(B, D),
       (A is_above fl(290),  limitvar(1),
          (( not__(A is_at_or_above fl(330))
           ; not__(A is_at_or_below fl(370)))
           ; not__(C is_at_or_above fl(330))
           ; not__(C is_at_or_below fl(370)))
      ; C is_above fl(290),  limitvar(2),
           (( not__(A is_at_or_above fl(330))
           ;  not__(A is_at_or_below fl(370))
            )
           ; not__(C is_at_or_above fl(330))
           ; not__(C is_at_or_below fl(370))))
  ;
one_or_both_of_are_flown_at_supersonic_speed(
                                    B, D),
    (A is_at_or_below fl(430), limitvar(3),
   ; C is_at_or_below fl(430), limitvar(4))),!.
```

5 RELATED WORK

Some recent work has pointed to the similarities between the validation of requirements models and knowledge based systems development (McCluskey et al. 1996; Shaw and Gaines 1996), and hence the area of Knowledge Base Refinement (KBR) is related to our work. A detailed comparison of validation in software engineering and KBS is given in reference (Vermesan and Bench-Capon 1995), and the state of the art in *automated* KBS validation is surveyed in refer-

ence (Zlatareva and Preece 1994).

As far as we are aware our work is the first to apply machine learning techniques to formal specifications of requirements, although, as mentioned above, work most related to our own occurs in the field of KBR. Both areas have to adopt strategies to overcome the complexity pitfalls surrounding the use of TR (where theoretical results suggest that no polynomial algorithm exists to perform global optimisation in hill climbing algorithms (Greiner 1995)). In KRUST (Palmer and Craw 1996), for example, test cases are used one at a time to refine the KBS, in contrast to our focusing procedure, which uses multiple examples and a form of statistical blame assignment. In MOBAL, an environment for knowledge acquisition that has been used with a large security rule base, TR is also used but in restrained fashion and with limited success (see (Sommer et al. 1994) page 453). Experience with MOBAL is consistent with our experience that ML tools work well in the context of a *diverse* tools environment.

Imperfect theory refinement techniques have been well researched in the machine learning literature, including reviews (Wrobel 1996), and a text relating ML to Software Engineering (Bergadano and Gunetti 1996). The case where theories represent planning domains is described in reference (Tae and Cook 1996) and the case where theories are posed as Horn Clause models is described in reference (Richards and Mooney 1995). Machine learning in domains containing significant numerical components has previously been accomplished by using neural networks (Opitz and Shavlik 1997). *Constraint Inductive Logic Programming* (Anthony and Frisch 1997; Sebag and Rouveirol 1996) has been utilised for generalisation and specialisation of numerical predicates. Theory Patching (Argamon-Engelson and Koppel 1998) is described as a type of TR in which revisions are made to individual components of the theory. (The concern of the latter paper is to determine which classes of logical domain theories the theory patching problem is tractable.) Theory patching compares with our work on focusing on *ordinal revisions* and on shielding clauses which are not to be revised.

6 CONCLUSIONS AND FURTHER WORK

In this paper we have reported the application of theory revision techniques to the validation and maintenance of a substantial 'theory', the formal requirements model of an air traffic control application. The model is encoded in msl, is customised by a generative grammar, animated by a Prolog generator, and can be analysed using an integrated environment supporting a diverse range of validation techniques (McCluskey 1997). After overcoming problems to do with blame assignment in general clause form programs (West et al. 1997), we developed the method whereby batches of tests were used by blame assignment, and single tests were used by explanation-based tools, to identify axioms sets in which bugs were likely to reside. After acquiring classified instances for these faulty components, we used theory revision operators, targeting comparison operators acting on ordinal sorts, to identify and remove the bugs. Here we have shown two different experiments where bugs were identified and removed, and a new part of the model was induced. The project started with an error rate for the conflict predicate of several hundred errors per 10,000 tests. The application of ML techniques in general has lead us to establish the cause of all the errors shown up in our initial tests, and the error rates using code generated from the current version of our model have been cut by 2 orders of magnitude. Having said this, our success in fielding TR seems to depend on correctly predicting how fundamental the revisions are, and having the machinery available to bring about such a level of revision.

Many problems for future work remain, however. Most outstanding is the generalisation of our environment so that other customised msl models can be created and analysed using ML tools. Secondly, the TR algorithms for simple and composite revisions need to be further refined and perhaps merged. Also, the implications of using blame assignment which takes into account negative literals in proof trees needs to be fully evaluated.

Acknowledgements

The IMPRESS project is supported by an EPSRC grant, number GR/K73152. We would like to acknowledge the help of Chris Bryant, who implemented some of the theory revision tools, Julie Porteous, for help in the initial stages of IMPRESS, and Julia Sonander of NATS, for supplying aircraft profile data. Further, we would like to thank the referees for helpful suggestions, and additional references, which have improved this paper.

References

Anthony, S. and A. Frisch (1997). Generating Numerical Literals during Refinement. In N. Lavrac and S. Dzeroski (Eds.), *Inductive Logic Programming: Proceedings of the 7th International Workshop, ILP-97* , Volume 1297 of *Lecture Notes in Artificial Intelli-*

gence, pp. 61 – 76. Springer-Verlag.

Argamon-Engelson, S. and M. Koppel (1998). Tractability of theory patching. *Journal of Artificial Intelligence Research 8*, 39–65.

Bergadano, F. and D. Gunetti (1996). *Inductive Logic Programming, From Machine Learning to Software Engineering*. Cambridge, Massachusetts, US: MIT Press.

Birkhoff, G. (1967). *Lattice Theory*. (Third ed.). American Mathematical Society.

Greiner, R. (1995). The complexity of theory revision. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.

McCluskey, T. L. (1997). An integrated environment for the conflict prediction specification. Technical Report impress/2/03/1, School of Computing and Mathematics, University of Huddersfield, UK.

McCluskey, T. L., J. M. Porteous, Y. Naik, C. Taylor, and S. Jones (1995). A requirements capture method and its use in an air traffic control application. *Software - Practice and Experience 25*(1), 47–71.

McCluskey, T. L., J. M. Porteous, M. M. West, and C. H. Bryant (1996, September). The validation of formal specifications of requirements. In *Proceedings of the BCS-FACS Northern Formal Methods Workshop*, Ilkley, UK. Electronic Workshops in Computing Series, Springer.

Meinke, K. and J. Tucker (1993). *Many Sorted Logic and Its Applications*. Wiley.

Opitz, D. W. and J. W. Shavlik (1997). Connectionist theory refinement: Genetically searching the space of network topologies. *Journal of Artificial Intelligence Research 6*, 177–209.

Palmer, G. J. and S. Craw (1996). The role of test cases in automated knowledge refinement. In *ES96: The Sixteenth Annual Technical Conference of the British Computer Society Specialist Group on Expert Systems*, Cambridge, England, pp. 75–90.

Richards, B. L. and R. J. Mooney (1995, May). Automated refinement of first-order horn-clause domain theories. *Machine Learning 19*(2), 95–131.

Sebag, M. and C. Rouveirol (1996). Constraint inductive logic programming. In L. De Raedt (Ed.), *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pp. 277 – 294. IOS Press.

Shaw, L. and B. Gaines (1996). Requirements acquisition. *Software Engineering Journal 11*, 149–165.

Sommer, E., K. Morik, J. M. Andre, and M. Uszynski (1994). What online machine learning can do for knowledge acquisition - a case-study. *Knowledge Acquisition 6*(4), 435–460.

Tae, K. and D. Cook (1996). Experimental knowledge acquisition for planning. In *Proceedings of the 13th International Conference on Machine Learning: ML'96*, pp. 480–488.

Vermesan, A. and T. Bench-Capon (1995). Techniques for the verification and validation of knowledge-based systems: a survey based onthe symbol/knowledge level distinction. *Software Testing, Verification and Reliability 5*, 233–271.

West, M. M., C. H. Bryant, and T. L. McCluskey (1997). Transforming general program proofs: A meta interpreter which expands negative literals. In *Proceedings: LOPSTR '97*, Leuven, Belgium.

West, M. M., C. H. Bryant, T. L. McCluskey, and J. M. Porteus (1996). The use of machine learning in the validation of a formal requirements specification: the work of IMPRESS. Technical Report impress/1/01/1, School of Computing and Mathematics, University of Huddersfield, UK.

Wrobel, S. (1996). First order theory revision. In L. De Raedt (Ed.), *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pp. 14–33. IOS Press.

Zlatareva, N. and A. Preece (1994). State-of-the-art in automated validation of knowledge-based systems. *Expert Systems with Applications 7*(2), 151–167.