Copyright ©2000IEEE. Reprinted from (Proceedings of the 12th EEEE International Conference on Tools with AI, held in Vancouver, Canada, 2000)).
This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of The Robert Gordon University's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to pubs-permissions@ieee.org.

By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

Appears in *Proceedings of the 12TH IEEE Int. Conf. on Tools with Artificial Intelligence,* pp182–185, 2000, ISBN 0-7695-0909-6. ©IEEE 2000

Debugging Knowledge-Based Applications with a Generic Toolkit

Susan Craw Robin Boswell
School of Computer and Mathematical Sciences
The Robert Gordon University
St Andrew Street, Aberdeen, AB25 1HG, Scotland
{s.craw|rab}@scms.rgu.ac.uk

Abstract

Knowledge refinement tools assist in the debugging and maintenance of knowledge based systems (KBSs) by attempting to identify and correct faults in the knowledge that account for incorrect problem-solving. Most refinement systems target a single shell and are able to refine only KBSs implemented in this shell. Our KRUSTWorks toolkit is unusual in that it provides refinement facilities that can be applied to a number of different shells, and is designed to be extensible to new shells. This paper outlines the components of the KRUSTWorks toolkit and how it is applied to faulty KBSs. It describes its application to two real aerospace KBSs implemented in CLIPS and POWERMODEL to demonstrate its flexibility of application.

1. Background

Knowledge refinement tools correct faulty knowledge based systems (KBSs) in reaction to examples of incorrect problem solving. They assist in detecting and removing faults while the KBS is being developed, but also when updating the KBS if its specification changes over time [4].

Most refinement tools are restricted to KBSs implemented in a single shell or language. ODYSSEUS [11] refines Minerva KBSs by exploring the problem-solving strategy that is explicitly represented in control knowledge. CLIPS-R [7] acts on CLIPS KBSs by analysing the KBS's interaction with the user and the content of working memory when execution halts. We developed a refinement tool specifically for the PFES shell, designed to implement formulation applications [4]. Several refinement tools target Prolog clauses [8, 2]. In contrast, the KRUSTWorks toolkit described here allows the knowledge engineer to construct a KRUSTTool refinement tool, customised for a particular KBS. Thus, KRUSTWorks helps to refine KBSs implemented in a number of different shells, and can be extended to new shells as required.

Refinement tools often assume that the shell's inference uses pure logic, and ignore procedural features like conflict resolution strategies. Refinement tools constructed from KRUSTWorks represent and reason about non-logical features of rule execution by using generic KBS concepts that represent the knowledge and reasoning processes in a variety of KBSs. We exploit the fact that despite variations in syntax, there are a relatively small number of *types* of rule conditions and conclusions [6]. Tasks and problem-solving methods can also be organised into ontologies [5].

This paper describes the KRUSTWorks toolkit. We first outline its components and how the KRUSTTools it creates cover a range of KBSs (Section 2). We have created KRUSTTools for CLIPS and IntelliCorp's POWERMODEL, and these are evaluated on two real KBSs developed for diagnostic aerospace applications (Section 3). The paper concludes by summarising the lessons learned.

2. The KRUSTWorks Refinement Toolkit

KRUSTWorks has two types of component: core refinement procedures that are independent of the KBS; and a set of toolkits from which the knowledge engineer selects tools to suit their specific KBS. The algorithm applies standard refinement steps (Figure 1). Run the KBS on a particular training example, allocate blame to potentially faulty rules, and then propose repairs that prevent the faulty behaviour. However, KRUSTTools are unusual in generating many repairs and postponing the selection of the best until the refined knowledge bases (KBs) have been evaluated by executing them on further examples. This cycle is repeated iteratively for each training example. Once processed, each example is added to a *constraint* buffer; subsequent refined KBs performing incorrectly for constraint examples are rejected. Another project has developed a more sophisticated treatment of training examples by re-ordering and backtracking to previously generated refined KBs [12].

KRUSTWorks provides several toolkits for the refinement algorithm. *Communication functions* establish a two-

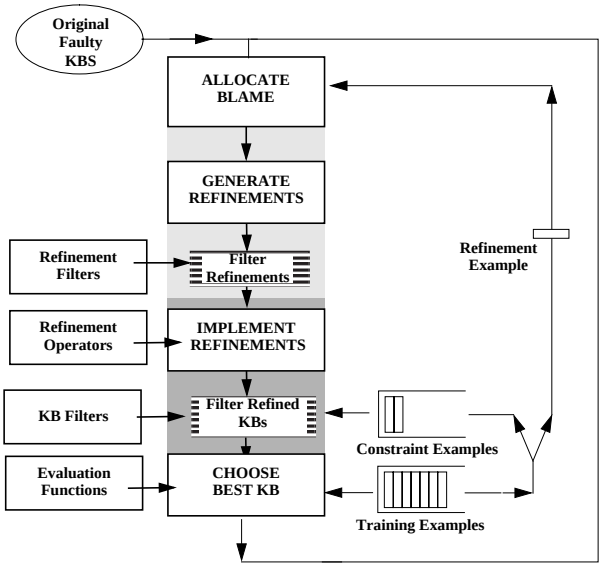

Figure 1. Refinement Algorithm

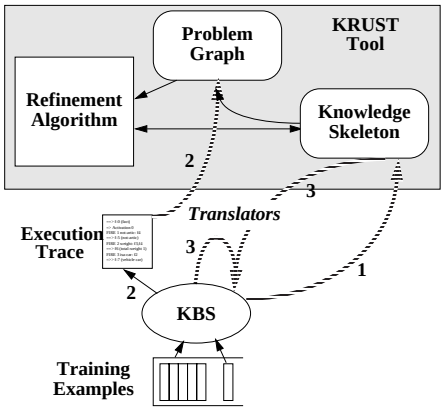

Figure 2. KRUST and KBS Processes

way communication between the KRUSTTool and the KBS via RPC-calls, pipes, or shared files. *Refinement Filters* remove refinements before the changes are implemented; e.g. conflicting or "poorest" according to some criteria (complexity, depth of rule in proof tree). *Refinement Operators* implement refinements and are associated with the various types of knowledge found in KBSs. *KB Filters* remove refined KBs that are unsatisfactory when executed; e.g. those that fail on constraint examples. *Evaluation Functions* select the best refined KB; e.g. the one with the highest accuracy on the training examples. The selection from toolkits is often dependent on the KBS to be refined; e.g. refinement operators depend on the type of knowledge in the KB. But the choice may be dependent on the refinement task; e.g. few training examples may demand more sophisticated evaluation functions.

Figure 2 illustrates the resulting KRUSTTool. The core *refinement algorithm* reasons about the static knowledge represented in the *knowledge skeleton* and the knowledge applied during problem solving as found in the *problem graph*. Firstly the shell-specific translator transforms the KB into the internal knowledge skeleton (1). For each training example, the KBS applies its problem-solving, and generates an execution trace from which the problem graph is formed (2). The refinement algorithm reasons from the knowledge skeleton and problem graph, applies refinement operators to change the knowledge skeleton, and the changed knowledge updates the actual KBS (3). Unlike many refinement tools (e.g. [8]) we separate the refinement and KBS processes so that different KBSs can easily be refined by the same basic architecture.

2.1. Representing Static Knowledge

There are only a small number of roles performed by rule conditions and conclusions [6]. An internal format is defined for each of the basic knowledge types and the content of the rules is captured in the knowledge skeleton. The knowledge hierarchy and associated refinement operators are described in [1]. We distinguish between comparisons assignments and goals. Different goal types also exist; e.g. OAV triples, AV tuples. Goals succeed by matching with observables or facts inferred by other rules. All three occur as rule conditions, but only assigments and goals appear as rule conclusions. Comparison and goal conditions succeed or fail, and so can be refined to affect the activation of that rule, but assignments always succeed. Refinement operators are defined for each knowledge type: a comparison `?temp<50` is generalised by increasing the value `50`; we specialise the goal `(temperature ?motor ?temp)` by instantiating `?motor` or `?temp`.

Rule syntax varies in different shells. When faced with a new shell, the knowledge engineer defines a grammar for the rules, describing items in terms of the standard knowledge types, but adding new types to the hierarchy if needed. The grammar underpins 2 shell-specific translators between the KBS and the knowledge skeleton. A similar approach analyses the execution trace for the problem graph.

2.2. Representing Problem Solving

The problem graph captures the reasoning as rules are applied to observables to infer the solution. Nodes represent rule activations or facts (those observed initially or inferred by applying rules). Edges link fact nodes to the rule nodes whose conditions they match, and rule nodes to the fact nodes they conclude. In addition to this part extracted from the trace, further nodes are added by finding rule chains in the knowledge skeleton that connect the ob-

servables and the *desired* solution. The trace part is guaranteed correct, but the added nodes and edges require a simulation of potential KBS behaviour and may introduce inaccuracies. Stalker [2] relies on the correctness of this simulation, but we implement and test all refinements in the KBS itself. Reasoning about simulated behaviour is common in planning applications [10] and so it is appropriate here for planning refinements. Refinement tools capture reasoning in different ways: CLIPS-R [7] groups traces which share initial sequences of rule firings; ODYSSEUS [11] explores all instantiations of the explicit problem solving strategy. Our problem graph and its applicability to a range of problem solving methods is described in [3].

3. Evaluation

KRUSTTools have been applied to various artificial and industrial KBSs; e.g. student loans [12], AstraZeneca's tablet formulation system TFS [4]. In each case a KRUSTTool was developed for that KBS. Here we evaluate KRUSTTools created from KRUSTWorks on 2 industrial KBSs: Nasa's MMU and ESA's AMFESYS.

Ideally, a refinement tool is applied during the development of a KBS. Various early faulty versions of AstraZeneca's TFS demonstrated the effectiveness of the PFES KRUSTTool [4]. But access to industrial KBSs during development is hard to achieve, so we obtained copies of AMFESYS and MMU, which we assumed to be correct, and introduced manual corruptions to each KB. One advantage of this approach is that the expert does not need to label examples; instead, we generated sample problems and used the original KBS to generate the "correct" outputs.

For each KBS, 5 faulty KBs were created each containing a single change: modified threshold, equality test, CLIPS field constraint, or disjunction; or an added condition. Further faulty KBs were generated by combining single faults into all possible groups of 2-3 faults. The final KB contained all 5 faults; 26 faulty KBSs in all. Faults were numbered and the corrupted KBSs assigned names based on their faults; e.g. MMU134 has faults 1, 3, 4.

Evaluating knowledge acquisition and refinement tools can be user-centred [9]; ease of acquisition is measured and new knowledge is critiqued by the expert. Since KRUSTTools currently do not interact with the user, we evaluate their performance by measuring the accuracy of the refined KB on unseen test problems. An n-fold cross-validation was performed. The example set for the application was randomly divided into 5 equal subsets, and repeatedly allocated to training and testing sets in the ratio 3:2 (10 experiments). For each experiment, the KRUSTTool was applied iteratively to the training examples. Both the initial (corrupt) KB and the final refined KB were evaluated on the testing set and the improved accuracy noted.

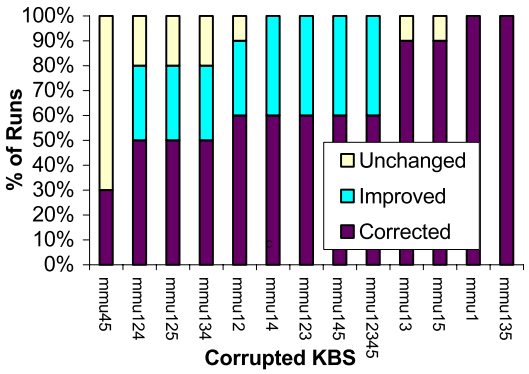

Figure 3. Improved error rates for MMU

3.1. MMU: Diagnosis of Manned Maneuvering Unit

NASA's Manned Maneuvering Unit (MMU) is a powered framework which fits around a space-suited astronaut and enables them to maneuver during space-walks. The MMU system (104 CLIPS rules) performs automatic fault diagnosis and recovery procedures for the MMU. Six examples came with MMU; having few test examples is quite common. We generated further examples manually, aiming to cover the problem space uniformly. The original KBS determined the "correct" diagnosis for all 80 examples.

Figure 3 allocates runs for each corrupt KBS into 3 classes: the refined KB is correct, has improved accuracy, is unchanged. Faults were not fixed in two situations: (1) no training examples exhibit the fault; and (2) the tool fixes the fault but no *testing* examples exhibit the fault, so the error rate is not improved. Several faults were under-represented like this (no examples for faults 3&5; and only 1 example highlights faults 2&4). For fault 1 there was ample evidence, and the tool always fixed the fault. Figure 3 shows the results for MMU45 and MMU KBs incorporating fault 1; the other KBs are always unchanged because of the lack of revealing training examples. Only 33% of refined MMU KBs were correct, and a further 11% were improved.

3.2. AMFESYS: Diagnosing Experiment Simulations

The European Space Agency's AMFESYS controlled the Automatic Mirror Furnace payload of the EURECA mission. Only the fault-diagnosis module (67 rules) is written as POWERMODEL *rules*. We generated 4 examples by running the full AMFESYS system and monitoring the observables presented to the fault-diagnosis module. Further examples, up to a total of 40, were created as for MMU.

Figure 4 shows the results for all AMFESYS KBs. The AMFESYS KRUSTTool almost always fixed the faults: 73% of refined KBs are correct, and a further 22% are improved. Combinations of faults 1,2& 3 are always corrected. Faults

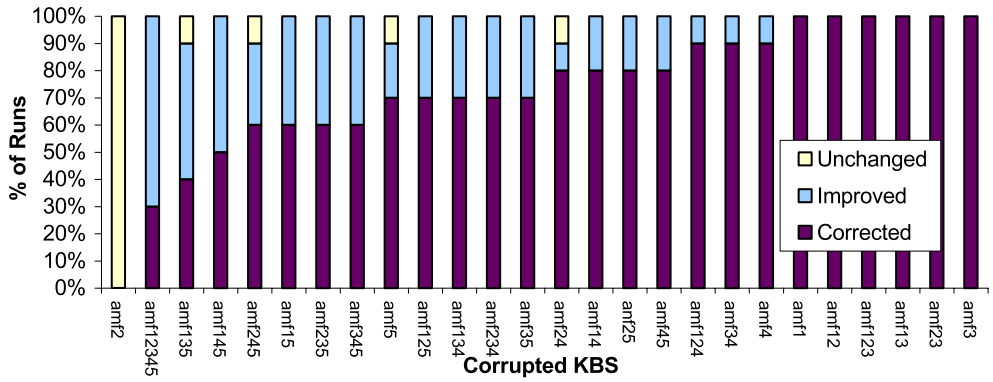

Figure 4. Improved error rates for AMFESYS

4&5 demonstrate (2) and degrade the performance when combined with other faults.

4. Conclusions

Most refinement tools target a single shell. KRUSTWorks uses generic representations for rules and the KBS's reasoning that enable us to build KBS-independent toolkits, applicable to a range of KBSs. We have presented the application of 2 tailored refinement tools to 2 industrial applications written in CLIPS and POWERMODEL. When the refinement tools failed to fix faults, the major cause was a lack of fault evidence in the available examples.

We also learned useful lessons relating to the refinement tool. Situations arose with MMU when a faulty refined KB disrupted the experiments: a pair of rules when overgeneralised looped indefinitely; other refinements caused runtime errors. Rather than attempt a complex semantic analysis to predict rule behaviour [10], we believe the refinement tool should simply execute the refined KBS. Thus we have implemented a KRUSTTool-KBS interface that limits the KBS execution resources or kills the process for a crashed KBS, and rejects the refined KB that causes the error. AMFESYS showed that it is possible to refine knowledge when the rules are only part of a larger system. This is important for the applicability of KRUSTWorks, since few industrial systems use rules exclusively.

Acknowledgements

This work is supported by EPSRC grant GR/L38387. We are grateful for software donated by ESA and IntelliCorp.

References

[1] R. Boswell and S. Craw. Knowledge modelling for a generic refinement framework. *Knowledge Based Systems*, 12(5-6):317–325, 1999.

[2] L. Carbonara and D. Sleeman. Effective and efficient knowledge base refinement. *Machine Learning*,37:143–181, 1999.

[3] S. Craw and R. Boswell. Representing problem-solving for knowledge refinement. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 227–234, Orlando, FL, 1999. AAAI Press/MIT Press.

[4] S. Craw, R. Boswell, and R. Rowe. Knowledge refinement to debug and maintain a tablet formulation system. In *Proceedings of the 9TH IEEE International Conference on Tools with Artificial Intelligence (TAI'97)*, pages 446–453, Newport Beach, CA, 1997. IEEE Press.

[5] D. Fensel, E. Motta, S. Decker, and Z. Zdrahal. Using ontologies for defining tasks, problem-solving methods and their mappings. In *Proceedings of the 10th European Knowledge Acquisition Workshop (EKAW97)*, pages 113–128, Sant Feliu de Guixols, Spain, 1997. Springer.

[6] V. M. Johnson and J. V. Carlis. Building a composite syntax for expert system shells. *IEEE Expert*, 12(6):60–66, 1997.

[7] P. M. Murphy and M. J. Pazzani. Revision of production system rule-bases. In *Machine Learning: Proceedings of the 11th International Conference*, pages 199–207, New Brunswick, NJ, 1994. Morgan Kaufmann.

[8] D. Ourston and R. Mooney. Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, 66:273–309, 1994.

[9] N. Shadbolt, K. O'Hara, and L. Crow. The experimental evaluation of knowledge acquisition techniques and methods: History, problems and new directions. *International Journal of Human-Computer Studies*, 51(4):729–755, 1999.

[10] D. E. Smith and M. A. Peot. Suspending recursion in causal-link planning. In *Proceedings of the Third International Conference on AI Planning Systems*, Edinburgh, Scotland, 1996. AAAI press.

[11] D. C. Wilkins. Knowledge base refinement as improving an incorrect and incomplete domain theory. In *Machine Learning: An Artificial Intelligence Approach Volume III*, pages 493–513. Morgan Kaufmann, San Mateo, CA, 1990.

[12] N. Wiratunga and S. Craw. Informed selection of training examples for knowledge refinement. In *Proceedings of the 12th European Knowledge Acquisition Workshop (EKAW2000)*, Juan Les Pins, France, 2000. Springer.