



Pacific Knowledge Systems

Challenges with Rules

A discussion on how the use of RippleDown technology enables domain experts to efficiently add rules to Laboratory Information Systems already in use. This paper explores the issues encountered by rules-based systems and how RippleDown overcomes these challenges.

Professor Paul Compton

University of New South Wales

December 2011

Contents

Introduction	3
Purpose of this white paper	3
Intended audience	4
The idea of rule-based systems.....	4
The difficulties experienced with rule-based systems.....	5
XCON.....	6
GARVAN-ES1.....	6
Other (older) systems	8
Recent systems	9
Summary	11
Why are rules difficult?	12
How experts provide knowledge	12
Developing simple rules	13
More complex rule systems.....	14
Conflict resolution	17
Debugging rules	18
Graphic User Interfaces.....	18
Verification.....	19
Validation	19
Imperative Debugging.....	19
Explanation	20
Why-not explanation	20
Explorative debugging.....	20
Refinement and theory revision	20
Algorithmic debugging	21
Sequential inference	21
Other knowledge acquisition research.....	21
Software engineering approaches	21
Ontologies	23
Machine learning	25
Knowledge elicitation	27
Summary	27
References.....	28

Introduction

Purpose of this white paper

There is clear evidence that RippleDown enables domain experts to gradually add thousands of rules to production systems that are already in use at the rate of a minute or two per rule (Compton, Peters et al. 2011). Is RippleDown needed to achieve this, or can rules be added at the rate of a few minutes per rule with other rule technologies? The paper discusses:

- Why rules, at first glance, seem very simple and should be able to added very rapidly,
- The real-world experience that shows this impression is misleading, and that the “industry standard” is between 30 minutes and a few hours per rule,
- The reason why rules are so much harder to add than one would expect, and
- Some of methods proposed for making rules more manageable.

The focus of the paper is rules built by human experts, but it also includes some discussion of machine learning as a way of avoiding the demands of adding rules.

Knowledge-based system projects can also fail because of lack of commitment by senior management, failure to consider the wider requirements and other project management or software engineering issues. The paper only considers the actual task of adding and maintaining rules, not these other issues that may arise with any IT project.

The central message of this paper is that, while any specific rule may be easy to create, especially with a good graphical user interface, the task of adding each new rule to a large Knowledge Base becomes increasing difficult and time-consuming without an in-built methodology which ensures that accuracy and consistency of the knowledge base is maintained.

Without such an in-built methodology, a knowledge base will never achieve sufficient complexity and size to be useful in a production environment, as it will be too expensive and difficult to maintain.

This issue is known as the knowledge engineering “bottleneck”, and is the main reason why expert systems have, in general, failed to live up to their early promise.

How RippleDown has successfully addressed this issue, is discussed in other white papers on this web site, particularly in the white paper “Experience with Long-Term Knowledge Acquisition” (Compton, Peters et al 2011).

Intended audience

This paper is intended for anyone who is considering developing a production system using rules and wants to get a clearer understanding of what is involved in adding and maintaining rules.

The idea of rule-based systems

Wikipedia provides a [simple introduction](#) to rule-based systems. A much more complete discussion is contained in the [Object Management Group standard](#) for rules (OMG 2009).

The OMG standard defines a rule as:

if [condition] then [action-list]

A knowledge base is simply a collection of such rules. The rules are processed by an inference engine which evaluates the rules against data.

The key design feature in early expert systems in the 1970's was the separation of rules from the inference engine, rather than having rules as part of procedural code. It was argued that this would make it much easier to add and maintain rules (Davis and King 1975). But this idea had already emerged much earlier (McCarthy... 1959). Rules that are separate from the inference engine are considered as declarative knowledge. That is, they are statements or declarations of the conclusions that should follow from the given facts rather than actual coding. The inference engine is the code that acts on these rules.

It might appear that declarative rules are readily implementable in procedural code simply as an IF . . . THEN statement. However, rules in a procedural program are designed to control the execution flow of the program. Although they can be used in a more declarative fashion to simply assert a new fact as true, if more than a few rules are added there is almost certain entanglement of their declarative function and program execution flow function. Since the code is procedural and executed in order, where does one add a new rule? That insight from the 1970's was that the best way to address this problem was to separate the inference engine, the procedural code that processed the rules, from the declarative statement of the rules.

This meant that the knowledge engineer or domain expert adding rules only edited the rules, not the underlying inference engine. Secondly the knowledge engineer did not have to directly manage the procedural flow. Rules can be added in any order and the inference engine figures out the processing sequence.

Expert systems fell out of favour as they did not achieve their early expectations or hype, but exactly the same ideas are behind the ever-increasing interest in business rule systems (Seer 2005). This time, the argument was to separate out the business rules from the data base trigger code, or “stored procedures” where they were normally embedded (Date 2000). The separation of business rules from inference is not a point of discussion but is central to the [Business Rules Manifesto](#) . A recent document from Microsoft presents in more detail the same traditional arguments for the separation of rules from the procedural code that processes the rules (Merritt 2004).

There are many rules engines available which can be used for both expert systems and business rules systems that are based on this separation of knowledge and inference. They include public domain system such CLIPS, Jess (derived from CLIPS), Drools as well as proprietary systems such as IBM’s iLog. Using any of these systems one finds that it is indeed trivial to create and add a rule taking only a few minutes.

The ease of creating rules can lead to the expectation that it must be reasonably easy to build a production-grade rule-based system. The question then arises: if rule-based systems are so easy to build, why aren’t they much more widely used?

The difficulties experienced with rule-based systems

Many of the reasons expert systems fail to reach routine or production use relate to the normal challenges in building and deploying any IT system: understanding organisational priorities and commitments, staff availability, requirements and other constraints such as interfacing etc. However, over and above this there is a problem known as the knowledge-engineering or knowledge-acquisition “bottleneck”. That is, although it is trivial to create a rule and add it to the system, knowledge-base developers discover that it far more difficult to ensure that the knowledge one accumulates through these rules will actually give the correct output, for example the correct diagnosis.

Matthew Fox in an article entitled “AI and expert system myths, legends and facts” (Fox 1990) summarises the difficulties as follows.

LEGEND: AI systems are easy to maintain. Using rules as a programming language provides programmers with a high degree of program decomposability; that is, rules are separate knowledge chunks that uniquely define the context of their applicability. To the extent that we use them in this manner, we can add or remove rules independently of other rules in the system, thereby simplifying maintenance.

Building rule-based systems differs from this ideal. Various problem-solving methods (including iteration) require that rules implementing these methods have knowledge of other rules, which breaks the independence assumption and makes the rule base harder to maintain. The much-heralded XCON system has reached its maintainability limit (about 10,000 rules). The complexity of rule interactions at this level exceeds maintainer abilities.

In the following we consider some the published information on the difficulties of building rules.

XCON

XCON was used to configure VAX computers against customer requirements, and was considered as probably the most outstanding expert system success in the early years of expert systems. The initial system was developed in Carnegie-Mellon University (CMU), but it then took over a year of training before DEC engineers could maintain XCON, then a system with 1,000 rules, and its maintenance demands meant they were unable to also maintain other expert systems introduced from CMU to DEC (Polit 1984). XCON eventually had 6,500 rules with 50% changed every year – which was found to be a major maintenance challenge (Soloway, Bachant et al. 1987). It apparently required 40 programmer/knowledge engineers to maintain XCON (Sviokla 1990) and as noted by Fox it reached the limit of maintainability at 10,000 rules (Fox 1990).

It should be noted that the problems with maintaining XCON were not because of the limitations of using earlier rule technology. XCON was built using the “Official Production System” (OPS) RETE architecture from Carnegie-Mellon (Forgy and McDermott 1977) and modern rule systems generally describe themselves as based on the OPS architecture. The most important contribution of the OPS architecture was to speed up inference using a RETE network (Forgy 1982; Forgy 1989), while the core inference conflict resolution strategies, used to decide which rule should be acted on next, were largely those outlined earlier (Davis and King 1975). Developments in [later generations](#) of OPS have still focused on improving performance, not the essential structure of the inference process. The relationship between the inference process and the knowledge engineering problems with rules will be considered below

GARVAN-ES1

GARVAN-ES1 was one of the first four medical expert systems to go into clinical use (Buchanan 1986). Although in routine use it was also an on-going research project, because of concerns about the reliability of expert systems, particularly in medicine. Expert systems can have a very high level of expertise in a domain but still make very

stupid mistakes (Lenat, Prakash et al. 1985). The output of GARVAN-ES1, diagnostic comments for thyroid laboratory tests, was continually monitored. Any omissions or errors resulted in rules being added or corrected. The cases for which changes were made were kept and used to test the system as further changes were made. Although on a much smaller scale (600 rules) than XCON, maintenance was still a major challenge (Compton, Horn et al. 1988) as shown in the following example.

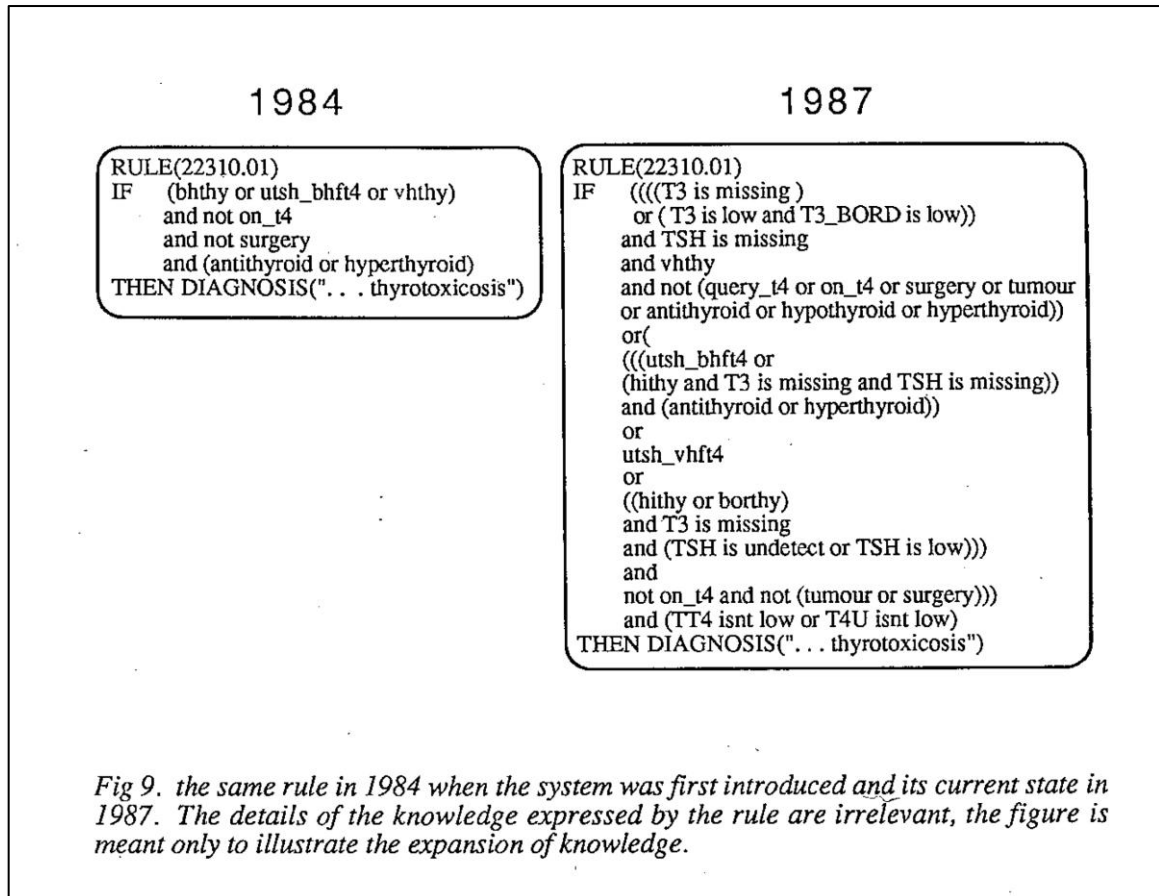


Figure 1, from (Compton, Horn et al. 1988).

The paper describes that the 1987 knowledge base was run with the 1987 rule above replaced by the 1984 rule (both shown in Fig. 1) and the knowledge base then tested on the validation cases. 3 of the validation cases fired the 1984 rule giving the correct answer, but 6 cases covered by the 1987 rule were missed by the 1984 rule and another 4 cases were incorrectly classified by the 1984 rule. This suggests that this rule was

probably changed 10 times between 1984 and 1987.

Obviously the rule is poorly coded with both negations and disjunctions, but this was the early days of expert systems and there was no expectation or anticipation that this rule would be changed 10 times. The scale of the changes required across the knowledge base can be seen in Fig 2. below showing cornerstone cases of various types. A cornerstone case was a case for which an error had been made and for which the rules were then edited. The figure shows the number of such cases of each diagnostic type which were incorrectly classified, and so which required changes to be made to the rules. The frustration of this maintenance experience lead to the first early steps towards RippleDown (Compton and Jansen 1988; Compton and Jansen 1990)

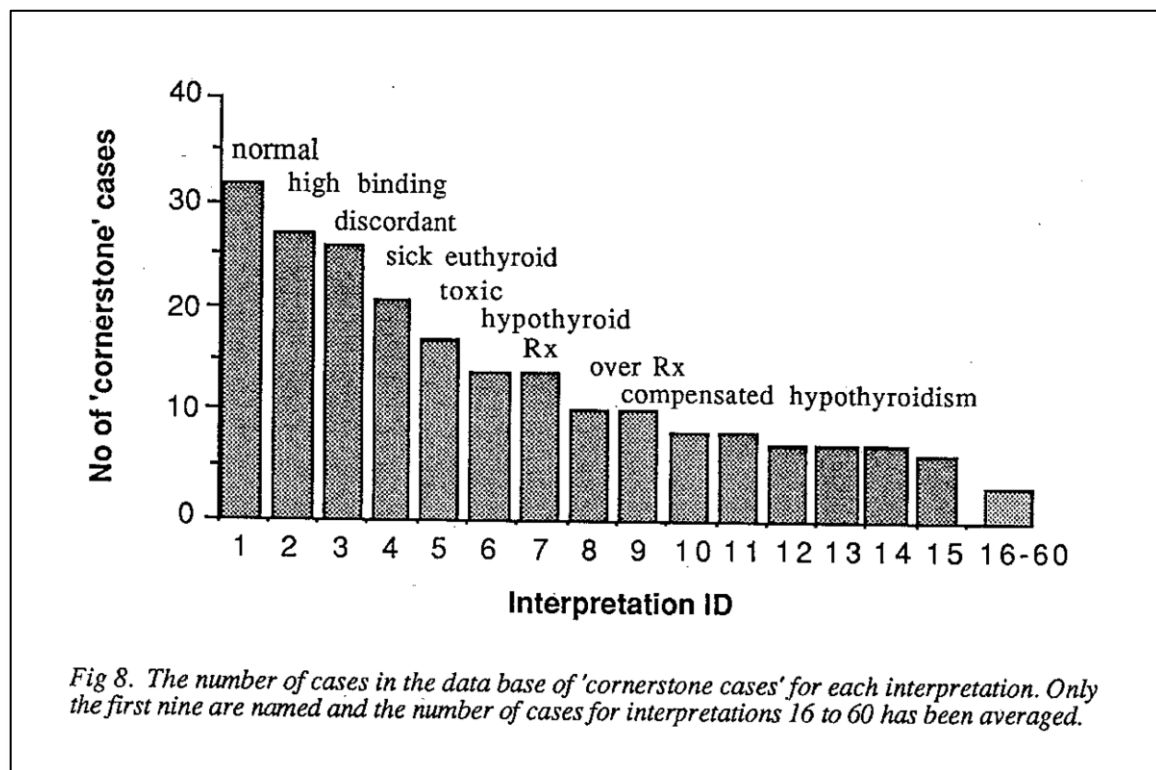


Figure 2, from (Compton, Horn et al. 1988).

Other (older) systems

Although there are numerous references to the difficulty of building and maintaining rules, there are surprisingly few detailed descriptions of acquisition and maintenance for individual systems apart from XCON and GARVAN-ES1. Bobrow et al (Bobrow, Mittal et

al. 1986) surveyed three well known systems, R1 (an earlier version of XCON), Pride and the Dipmeter advisor and concluded:

Expert Knowledge Has to Be Acquired Incrementally and Tested. Expert knowledge is not acquired all at once: The process of building an expert system spans several months and sometimes several years. In the course of this development, it is typical to expand and reformulate the knowledge base many times. In the beginning, this is because choosing the terminology and ways of factoring the knowledge base is subject to so much experimentation. In the middle phases, cases at the limits of the systems capabilities often expose the need to reconsider basic categories and organization. Approaches viable for a small knowledge base and simple test cases may prove impractical as larger problems are attempted. . . . Toy programs for a small demonstration can be built quickly-often in just a few months using current technology. However, for large-scale systems with knowledge spanning a wide domain, the time needed to develop a system that can be put in the field can be measured in years, not months, and in tens of worker-years, not worker-months.

O'Leary conducted a survey of people who had developed expert systems for accounting and finance (O'Leary 1991). The major problems identified by the respondents were the incompleteness of the knowledge base and the related problems of ensuring correctness. There is no information on the size of these knowledge bases or how long they had been in use, but there is clear identification of the problems.

Recent systems

Zacharias conducted a similar survey of modern rule-system developers (Zacharias 2008). 76 people responded to the survey with 64 answering most questions. They were asked to respond with reference to the largest knowledge base they had worked on in the last five years. On average the respondents had over 6.6 years experience developing knowledge based systems and used a range of rule technologies and, for slightly over 50% of the knowledge bases, domain experts added the rules. 60% of the respondents indicated that their knowledge bases *frequently* failed to give the correct result and 34% indicated that *sometimes* the incorrect results was given. The biggest need was identified as debugging/verification tools to correct such errors. This is the identical experience to the earlier examples given above and is essentially the same conclusion reached in O'Leary's 1991 survey (O'Leary 1991).

The Zacharias survey provides a comparison point with RippleDown that is worth noting. Zacharias provides summary data on the mean and median size of the knowledge bases

and the mean and median time in man-months taken to develop the knowledge bases. In the Table 1 the time to add a rule in the Zacharias data is found from dividing the mean time in person months by the mean knowledge base size (38 mins per rule) or the median time by the median knowledge base size (385 mins per rule). It was assumed a person-month was 20 seven hour days per month. In calculating the total number of rules it was also assumed that the 64 respondents who responded to most of the survey questions also provided data on knowledge base size, so the total number of rules was calculated by multiplying the average number of rules by 64. The data on RippleDown comes from logs of user activity (Compton, Peters et al. 2011). RippleDown logs provide the time users were logged on to the knowledge builder adding rules to deal with a case and revalidate (debug) the knowledge base. Each case may require several comments, and so may require several rules. The logs record the total time for rules for a case, rather than the time for an individual rule, but in the table below we assume it is the time for an individual rule. . The logged time includes interruptions while building rules, resulting in the average time to build a rule, 136 secs being almost double the median time of 78 secs.

Rule-building techniques	Time to add rules	Number of rules
Various standard techniques (Zacharias 2008)	38 to 385 mins per rule	126,016 (estimate)
RippleDown [®]	78 to 136 secs per rule	57,626

Table 1: data from Zacharias (2008) and Compton, Peters et al (2011)

The comparison here is more qualitative than quantitative as Zacharias' survey is based on user surveys rather than empirical measurements.; it covers a range of knowledge based sizes and technologies; there is no information on the use these systems; and finally the number of respondents who provided numerical data is an estimate. On the other hand the RippleDown data underestimates the number of rules added as it only records the number of cases for which rules were added and overestimates the time taken because the logs include time spent on interruptions.

However, despite these reservations in making a comparison, there seems to be a striking difference in that the rules which could be built in just one hour using RippleDown would seem likely to take three days with other technologies.

The earlier discussion referred to the deceptive ease with which rules could be added to a rule-based system. Zacharias' respondents similarly identified the ease of rule addition as one of the strengths of rule-based systems, but they also identified the problems of debugging rules to ensure the rule base gave the right answer as the major problem with rules. Elsewhere Zacharias writes:

The One Rule Fallacy: Because one rule is relatively simple and because the interaction between rules is handled automatically by the inference engine, it is often assumed that a rule base as a whole is automatically simple to create. However, it is an illusion to assume that rules created in isolation will work together automatically in all situations. Rules have to be tested in their interaction with other rules on as many diverse problems as possible to have a chance for them to work in novel situations (Zacharias 2009)

However, the issue is not just that the knowledge base has to be tested, but the difficulty of knowing how to make the appropriate changes. As noted by Jacob and Froscher:

Changing a knowledge base of an expert system built with typical current technology requires a knowledge engineer who understands the design of the system and the structure of the knowledge base thoroughly; most often, this means only the original author of the system. (Jacob and Froscher 1990)

This has been true since the start of expert systems. As Clancey notes in the abstract of his paper on the epistemology of MYCIN.

However, experience with one of these programs, MYCIN, indicates that the (*production rule*) representation has serious limitations: people other than the original rule authors find it difficult to modify the rule set. (Clancey 1983)

Summary

Although rules are easy to create in the sense of writing an individual rule and adding it to the knowledge base; experience shows that it is a far more difficult task to ensure a collection of rules will give the answers desired across a domain. In particular, it can be very difficult to ensure that the addition of the new rule does not have unintended side effects when used in conjunction with the existing rules.

Why are rules difficult?

How experts provide knowledge

One aspect of the problem is that experts such as pathologists never provide a final and complete rule, but always an over-generalisation. A response to the question: how to diagnose pregnancy from laboratory results, might be: *IF elevated hCG THEN pregnant*. However, this rule fails to take into account that a male might have an elevated hCG because of an hCG secreting tumour, so the rule should have been *IF female AND elevated hCG THEN pregnant*. But females can also have hCG secreting tumours so the rule should probably also have conditions relating to whether the woman is of reproductive age and probably include conditions relating to the doctor or type of clinic that requested the hCG measurement. Was it requested by an oncologist or an obstetrician – and if a quantitative hCG was ordered by an obstetrician, was it because they were concerned that the pregnancy might be ectopic or perhaps failing? Going further, if an obstetrician ordered the hCG test, perhaps no advice should be given as they will know perfectly well how to interpret the result for themselves and may in fact resent anyone else's attempt to interpret the test results for them

Although the expert's initial rule was almost useless, it does not indicate any failure in the pathologist in communicating their knowledge. Pathologists and all medical practitioners deal with individual patients and individual patient reports. If the data indicates the patient is clearly a young woman and pregnancy is being queried, it is almost certain that a pathologist building a "pregnancy" rule will omit any rule conditions related to the patient being a female of child bearing years. Such an omission is only to be expected, because the pathologist is providing advice in the context of the particular patient, that is a young woman who may be pregnant.

In the 1980s and 1990s there was considerable debate whether expert systems were even possible, because of the intrinsically incomplete way in which experts provided knowledge. Winograd and Flores argued that knowledge was always provided in a context (Winograd and Flores 1987). Clancey argued that knowledge is not something in the head to be mined, a common metaphor in expert system development, but is always constructed in and for a particular context (Clancey 1997 – this book summarised his many papers on situated cognition over the previous decade). It was suggested by Compton and Jansen (1990) that when an expert provides reasons for a conclusion, what they are really doing is giving the reasons why that conclusion is the correct conclusion in the context, i.e. that is compared to other conclusions that might be under consideration either implicitly or explicitly. In the hCG examples above, if hCG is

measured in a male the conclusions under consideration are related to an hCG secreting tumour, not pregnancy. At an epistemological level what the expert is doing is giving the reasons that disprove the other possible conclusions, rather than a definitive and complete proof of the conclusion independent of the context. This derives from Popper's notion of falsification. That is, hypotheses are never proven, and can never be completely proven; rather, alternative hypotheses are disproven.

The debate about situated cognition in the late 80s and early 90s coincided with a drying up of research funds because of the failure of expert systems to live up to their promise. The net effect of this was largely a move to different problems, but still based on the hope or expectation that somehow complete and stable knowledge would be able to be provided, with relatively few researchers trying to develop technologies that accepted the intrinsic incompleteness of knowledge. Regardless of the philosophical merits of the arguments about situated cognition, John McCarthy had already identified that knowledge was always incomplete. His statement of the problem was that a robot could never be programmed to deal with every possible situation that might arise in the real world (McCarthy 1977). There is always the possibility of some unexpected eventuality, just as in diagnosis there is always the possibility of some other explanation, no matter how remote (e.g. perhaps even falsified data). In our human discourse most of us implicitly accept that we are never completely certain, so our explanations that a particular conclusion is correct tend to rely on evidence that the other conclusions under consideration are not possible.

Developing simple rules

The simplest form of rule system is one where rule conditions relate directly to the data that is input and the conclusions of a rules is a final conclusions to be output. There are no intermediate conclusions on the way to a final conclusion. A simple rules system is essentially a decision table, where on a single pass all rules fire that are satisfied by the data.

To maintain such a system one has to deal with the problems noted above, that is, there will be cases that have not been covered, requiring new rules to be added. The new rules will be too general, so that one of more rules may have to be narrowed to exclude a case and another rule (or rules) added to interpret the excluded case. The rule creation or editing is relatively straightforward, but the challenge is that a new rule might inadvertently cover cases that it shouldn't and are already correctly covered by existing rules. Conversely, narrowing a rule might inadvertently exclude cases already correctly covered. An example of this from (Compton, Horn et al. 1988) was shown in Fig1 above.

To ensure such errors do not occur there has to be some sort of validation against test cases, .e.g (Baumeister 2010).

There is a further challenge: Who is going to make the changes? If we look at the sequence of changes to the hCG interpretation above, is the pathologist going to refer these changes one by one to the IT department – and similarly for every other area of knowledge covered? Or are pathologists themselves going to make the changes? If so, what sort of a rule-editing interface is needed, and what sort of a user interface is required for validating rules against test cases? How are the validation cases accumulated, and critically what is the process for identifying that the rules need to be changed? Certainly with a simple flat rule system it is very easy to change rules. But to do it again and again needs a complete system that fits seamlessly into the organisation workflow and requires minimal extra work.

More complex rule systems

Clancey identified that people generally tend to build classification systems which use intermediate conclusions and he defined this form of reasoning or problem solving as heuristic classification (Clancey 1985). Intermediate conclusions represent higher-level concepts or abstractions of the raw data in the case. They are conclusions which are defined by rules but then, in turn, can be used in subsequent rules as rule conditions. In this way the inference process chains or cycles through sequences of rules. Using intermediate conclusions may greatly reduce the number of rules to be added as can be seen in the following example:

Imagine rules for interpreting thyroid results. A very useful concept for interpreting thyroid results is “on thyroxine”, that is, a flag indicating whether the patient is on thyroxine medication. This concept may be implicit in the case data, rather than presented to the system as an explicit data item. For example, the referring clinician’s clinical notes on the request might refer to thyroxine replacement therapy. This thyroxine information can be written in many different ways, so either the reception clerk has to correctly interpret the clinical note as referring to thyroxin replacement when entering the test order, or simple rules like those below are required to deduce the concept (i.e. the intermediate) from the raw clinical note that was entered:

IF clinical_note contains “thyroxin” THEN *on_thyroxine*

IF clinical_note contains “treated” AND *thyroid_request* THEN *on_thyroxine*

IF clinical_note contains “orox” THEN *on_thyroxine*

IF clinical_note contains “dxrt” AND *thyroid_request* THEN *on_thyroxine*

IF clinical_note contains “thryoxin” THEN *on_thyroxine*

And there are likely to be other similar rules

Note that the words “treated” and “dxrt” could possibly refer to other treatments, but if thyroid measurements have been requested, they will refer to the patient being on thyroxine. Hence we can use a second intermediate “thyroid_request” to determine this:

IF FT3 requested THEN *thyroid_request*

IF TT3 requested THEN *thyroid_request*

IF TT4 requested THEN *thyroid_request*

IF TSH requested THEN *thyroid_request*

IF thyroid antibodies requested THEN *thyroid request*

Finally there will be rules related to hormone levels. E.g.

IF *on_thyroxine* AND FT3 is high and TSH is undetectable THEN . . .

IF *on_thyroxine* AND FT3 is high and TSH is normal THE . . .

Etc etc

In these last two rules both “FT3 is high” and “TSH is undetectable” are further intermediates as they are abstractions of the numerical data provided by the laboratory. There might be still further intermediates relating to the possible patterns in combinations of thyroid results: *hypo_results*, *hyper_results*, *compensated results* etc. The use of intermediate conclusions has enormous potential to reduce the number of rules required. For example if there are 10 rules which give the same intermediate conclusion, and this intermediate is used in another 10 rules combined with other rule conditions a total of 20 rules will be developed, whereas without intermediates 100 rules would have been required.

Although the use of intermediates seems highly desirable in reducing the number of rules required, they make the problem of adding expert knowledge much more difficult. Without intermediates there is no question of which rule needs to be fixed; it is the rule that gave the wrong conclusion for the case. However, if there is a chain of intermediates, the expert has to decide whether to fix a rule giving an intermediate as a

conclusion somewhere along the chain of intermediates, or to fix the final rule. Regardless of whether the expert fixes an intermediate rule or a final rule, there are other indirect consequences. The figure below shows a sequence of rules that fire for a case marked by the red circles. If the expert or knowledge engineer narrows the initial rule giving the intermediate conclusion so that the final rule will not fire, this may also change the circumstances under which all the other final rules using that intermediate will fire. Vice-versa, if the expert or knowledge engineer changes the final rule using the intermediate, this change will apply to all the other circumstances under which the intermediate has been concluded. For none of these will the final conclusion now be given.

Similarly, if the expert widens the definition of the initial rule giving the intermediate conclusion, additional final rules may now fire, possibly misclassifying cases. And again if the expert widens the final rule, this will fire for all cases where the intermediate is given.

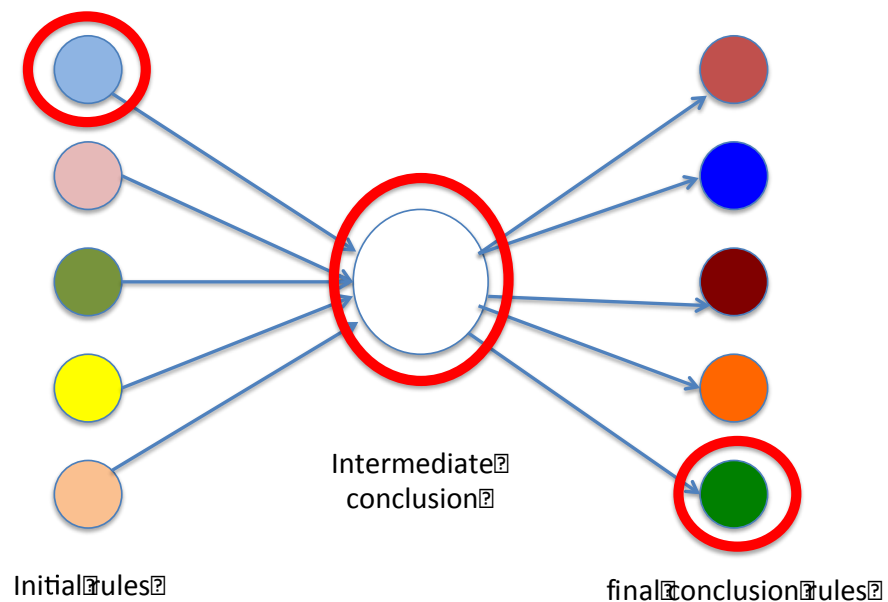


Figure 3: Different rules giving the same intermediate conclusion are shown by different coloured circles on the left. Different rules giving different conclusions, but using the same intermediate, along with other conditions are shown on the right. The red circles show the sequence of rules that fired on a particular case: the top initial rule giving the intermediate conclusion, and the bottom final rules giving a particular conclusion based on the intermediate as well as other features of the case.

This makes the apparently simple task of adding a rule much more complex. To be sure there are no untoward effects of adding the rule, the knowledge engineer and the expert have to be aware of all the potential effects of a change. Zacharias provides a detailed discussion of the issues in debugging such knowledge bases and comments:

The Problem of Interconnection: Because rule interactions are managed by the inference engine, everything is potentially relevant to everything else. This complicates fault localization, because bugs appear in seemingly unrelated parts of the rule base. Another consequence is that even a single fault can cause a large portion of (or even all) test cases to fail. (Zacharias 2009)

Conflict resolution

A further complexity is conflict resolution. A key idea behind the development of rule-based systems is that there is no information about the state of the system being reasoned about, other than the current facts, including both the data provided and any intermediate conclusions that have been asserted so far. This leads to the constraint that only one rule at a time should fire and change the state of the system (Davis and King 1975). Since a number of rules might be satisfied by the current data a conflict resolution strategy is needed to decide which rule takes precedence. There are number of standard conflict resolution strategies e.g. rules that use the most recently asserted intermediate conclusions have priority; the most specific rule has priority, or alternatively the most general rule has priority etc etc. The most powerful strategy is “salience”. That is, a rule has a number attached to it representing its priority, and the rule with the highest salience takes precedence.

Salience seems attractive in that, by simply assigning a number, a particular rule from a set of rules can be given priority, without trying to edit rule conditions. But the same type of problem arises: although it is trivial to set the salience for the candidate rules for a particular set of data, what happens to the salience relationship with all the other rules? If out of rules A, B, C & D, rule B is given a higher salience value, is this an appropriate salience value with respect to a group of rules B, C, E & F that might apply to another case.

As noted above, the surveys of O’Leary (O’Leary 1991) and Zacharias (Zacharias 2008) both identified that the biggest problem in building and maintaining a knowledge base identified by developers is debugging, ensuring that the knowledge base gives the right answers. It seems clear that conflict resolution strategies can only increase the difficulty of debugging a knowledge base. because although the conflict resolution strategies control the inference order, they themselves cannot be directly controlled (except for

salience). Control is indirect e.g. by adding more conditions to a rule to give it a higher priority.

Although conflict resolution is still a standard, and is referred to in the OMG production rule representation (OMG 2009), and the 2008 OCEB white paper on production rule systems (Vincent 2008), it seems that it is of decreasing importance. Perhaps because conflict resolution increases the difficulty of an already difficult task of managing and debugging rules, providers of rule systems are emphasising simpler approaches. Rules engines such as IBM's Ilog (Jrules) and FICO's Blaze Advisor no longer specifically discuss conflict resolution <https://ssl.tibcommunity.com/message/1165> , but it is still explicitly promoted by Tibco the third major engine referred to in the 2008 OCEB white paper. e.g. <http://tibcoblogs.com/cep/2010/06/02/inference-rules-and-flexible-execution/> (The comments in this blog suggest that Tibco rule priority is the same as salience used in other rules engines.)

Debugging rules

Following Zacharias (Zacharias 2009) we use the term debugging to broadly describe the process of identifying errors and adjusting a rule base until it gives the correct answers for cases across the domain. This section of the whitepaper briefly summarises the standard techniques used to assist with this task and initially follows Zacharias' review. However, we also note the comment from Zacharias and Abecker:

Today there is no debugger for rule-based systems that takes into account the declarative nature of rules (Zacharias and Abecker 2007).

Graphic User Interfaces

Most development environments for knowledge-based systems or Semantic Web technology provide high-level graphical editors for various tasks and this is likely to include some sort of graphic user interface to assist with debugging rules by enabling the connections between rules to be explored. The simplest approach is just to show the dependencies between rules, where the conclusion of one rule is a condition in another rule. Other more sophisticated interfaces group rules in various ways, considered further below. A graphic user interface does not of itself identify errors and omissions in the knowledge base, all it does is enable a user to inspect the knowledge base more easily so they can identify problems. Such graphic displays are very useful, but have the Catch-22 that the larger the knowledge base i.e. when more help is needed, the more complex the graphic display and the more difficult it is to use.

For an interesting discussion of what types of rule visualisation might be useful, see (Baumeister, Menge et al. 2008)

Verification

The major literature on debugging knowledge bases is concerned with verification and validation, but actually neither of these directly contribute to the type of debugging that a knowledge-base developer is mainly concerned with.

Verification is the process of ensuring there are no inconsistencies or contradictions in the knowledge, no circularities, i.e. where inference can loop around through the same sequence of rules without terminating, and no deficiencies, i.e. situations for which the knowledge bases cannot reach a conclusion (Preece and Shinghal 1994; Preece 2001). Various forms of verification exist in some modern rules engines, e.g. [Jrules detects conflicts and redundancy](#). However, just because there are no logical inconsistencies in a knowledge base does not mean the knowledge base will give the appropriate answers for cases. The debugging challenge referred to by developers, is not simply to ensure the internal consistency of the knowledge bases, but to ensure the system's actual performance on cases is correct (Zacharias 2008).

Validation

The limited nature of verification has always been recognised, so verification literature always refers to the need for validation carried out by running the knowledge base on test cases. This seems obvious, but there are a number of issues that need to be addressed (Preece 1994; Preece 2001; Baumeister 2010). Knowledge-based system domains are invariably rich, so that it is not possible to provide test cases for every possible data configuration that will ever be seen. What sorts of cases are missed with the validation cases? What is the correctness on different types of cases; there is little value in a system that is highly accurate on cases where an error is of little consequence, but inaccurate on cases where an error is very costly. As discussed below, machine learning also has the challenge of assembling appropriate test cases.

Although validation is critical in identifying the limitations of a knowledge base, it does not provide a way of debugging the knowledge base to correct the errors; at best it simply identifies the cases where the system is erroneous

Imperative Debugging

According to Zacharias the most widely used form of debugging for rule bases is standard imperative debugging where break points are inserted to stop the program and inspect the current state (Zacharias 2009).

Zacharias points out that this contradicts the supposed advantage of separating inference from declarative knowledge and forces the user to understand the inference process in detail.

Explanation

Explanation has been proposed since the earliest days of expert systems as a way of being confident in the system's conclusion. Essentially a trace is provided of all the rules that fired on the way to reaching a conclusion, so you can see why a conclusion was given. Explanation is very useful in understanding the particular rule path that led to a conclusion, but as Zacharias points out it, it cannot tell you what will happen when a rule is changed, or which rule should be changed to correct the interpretation of a case (Zacharias 2009).

Why-not explanation

The converse of explanation is a why-not explanation facility whereby one can understand why a rule which might have given the correct answer failed to fire for the case. There has been a range of promising research into why-not systems for specific applications e.g. (Bonatti, Olmedilla et al. 2006), but as Zacharias points out these are not yet practical systems and it is not clear whether they will be able to deal with a range of real-life errors (Zacharias 2009).

Explorative debugging

Explorative debugging relates to explanation and why-not explanation and is essentially a proposal to freely browse the connections between rules and rule conditions (Zacharias and Abecker 2007). Zacharias sees it having considerable potential, but is essentially a proposal. A related approach has suggested formal concept analysis as a way of browsing a knowledge base to explore how different rule conditions differentiate between different conclusions (Richards and Compton 1997).

Refinement and theory revision

Refinement and theory revision techniques take a knowledge base and a set of test cases some of which are not covered by the rules and iteratively refine the knowledge, either automatically or with expert help, until all cases are covered (Craw 1996). This is clearly an elegant approach, but in Zacharias' view "these approaches still have to prove they can be used efficiently and reliably" (Zacharias 2009).

RippleDown is closest to these refinement approaches as it uses a specific knowledge structure in the refinement process, which simplifies the expert's task.

Algorithmic debugging

Algorithmic debugging techniques aim to automatically divide a knowledge base into partitions which maximise interaction within a partition and minimise interaction between partitions. The idea is that this will enable faults in the knowledge base to be localised and minimise the debugging required. Although of considerable potential, Zacharias notes there has not been widespread adoption and there are no commercial systems that use algorithmic debugging (Zacharias 2009).

Sequential inference

The OMG production rule specification covers standard inference where the order of rule evaluation is not predetermined but is determined during inference. It also covers sequential inference, where all possible rule sequences have been predetermined. A set of rules can be converted into all possible rules sequences, but not necessarily vice-versa (OMG 2009). We note this because RippleDown also exploits a form of predetermined inference sequence as way of managing knowledge acquisition. However, the main argument for sequential rule engines is high performance and resolution of conflicts before inference, rather than debugging (Hicks 2007). Modern rule engines such as Blaze offer both traditional RETE inference, and also [compilation into sequential rules](#).

Other knowledge acquisition research

Debugging rules is part of knowledge acquisition research. Broadly, knowledge acquisition is about the challenge of getting experts to provide appropriate knowledge and then coding this into appropriate knowledge structures. Although not directly addressing the problem of debugging rules, much of this research can be seen as trying to minimise or avoid these problems.

Software engineering approaches

In terms of publications, the most significant research on improving how rule systems are built has been the development of a software engineering framework for knowledge engineering. We use the term “software engineering” as the focus has been on better ways of managing the process of building knowledge-based systems.

A key insight was that there are different types of problems for which different types of problem-solving or inference were appropriate. From this insight, various classifications of problem types have appeared. This commenced with Clancey's recognition that there were two broad task types, classification and construction (Clancey 1985). As the name suggests classification assesses what category a case belongs to; e.g. does the patient have disease X or disease Y. A construction task is the building or design of an artefact where the choices to be made depend on each other; e.g. in the classic example of XCON the selection of certain components for a VAX computer determined what other components were possible and vice-versa. The classification of the problem type then suggests specific methods that may be appropriate in solving that problem; e.g. (Puppe 1993; Schreiber, Akkermans et al. 1999). The idea was that an appropriate identification of the different knowledge tasks and the relevant problem-solving method for each task would greatly facilitate building the knowledge-based system, rather than just muddling through and adding rules without a framework or design.

For example Clancey in his elegant study of the structure of MYCIN points out that an important part of the knowledge in MYCIN is how it interacts with the user. E.g. is it more important to the referring physician to give them the interpretive comment "e.coli is causing meningitis" rather than "cryptococcus is causing meningitis" which may also be given by the Knowledge Base. (Clancey 1983). In MYCIN rather than being explicit, such control was embedded in the order of the rules. Clearly a system is more maintainable, particularly by others, if all the different forms of knowledge and how they are used is explicit.

The culmination of a software engineering approach was probably CommonKADS (Schreiber, Wielinga et al. 1994; Schreiber, Akkermans et al. 1999; Speel, Schreiber et al. 2001) which proposed that expert systems should be approached using a systematic software engineering process where a sequence of decisions about the various types of models to be developed would be made appropriately. This was a major step forward and it is clearly advantageous to use a carefully designed process in developing a project using rules, or any other type of software. However, there still remains the final step of populating the knowledge base with rules, and so there still remains the problem of debugging rules. As the CommonKADS authors themselves note:

Although methodologies such as CommonKADS support the knowledge acquisition process in a number of ways (e.g. by providing modelling constructs and template models) experience shows that conceptual modelling remains a difficult and time-consuming activity (Speel, Schreiber et al. 2001).

They note further:

This is not to say that it is impossible in principle to make tacit knowledge explicit, but it is difficult and the standard knowledge engineering repertoire does not include techniques to support this (Speel, Schreiber et al. 2001).

Tacit knowledge can mean completely non-verbal knowledge such as how to balance while riding a pushbike, but here more broadly means implicit knowledge that experts don't articulate e.g. that only women of reproductive years can be pregnant.

Baumeister et al. also point out that while the use of something like CommonKADS is highly advisable for a large project involving a number of people, it is a very heavyweight method if something more lightweight and agile is required (Baumeister, Seipel et al. 2009).

Ontologies

The idea of problem-solving methods, rather than just rules and inference, was closely linked to the development of idea of ontologies. The ontology was a formal representation of the entities and their properties and relationships in the domain and a task-appropriate problem-solving method could then be applied to the domain knowledge in the ontology e.g. (Musen 1998; Pérez and Benjamins 1999; Crubézy and Musen 2003). The motivation for ontologies was to develop re-useable and more formally structured domain knowledge and avoid the problems of ad-hoc encoding of domain knowledge in rules. However, with the emergence of research on the idea of a Semantic Web, the main focus of ontology research has been to develop the possibility of ontologies for the Semantic Web. For example, there are some 200,000 registered users of the Protégé, but end-users tend to use Protégé for developing ontologies, but not for developing systems (Musen 2013). There is a huge literature on this and a range of standards has been developed, but it is beyond the scope of this whitepaper to review this material.

The emphasis on the Semantic Web took emphasis away from using ontologies in production knowledge-based systems and the major problem that remains to be addressed is how to use rules with ontologies. Hitzler et al note that there has been a surprising divergence between ontology research closely related to logic and the use of production rules in business-rule systems and expert systems. They note:

Unfortunately, unlike with Description Logics and logic programming, we do not currently have a well established common semantic framework (i.e., modal theory) ready to hand to aid us with integration. (Hitzler and Parsia 2009)

In listing current and future research Eiter et al. similarly note:

In fact, there are many more kinds of rules out there which we need to integrate with ontologies as well; for example, production rules as available in traditional expert systems engines, that are based on an operational semantics; business rules, which are used in the context of business policies and whose semantics is not always clear (Eiter, Ianni et al. 2008)

And in 2010 Baumeister and Seipel noted:

Thus, conventional approaches for evaluating and maintaining ontologies need to be extended and revised in the light of rules, and new measures need to be defined to cover the implied aspects of rules and their combination with conceptual knowledge in the ontology. (Baumeister and Seipel 2010)

For a well described practical application for assisting with brain anatomy labelling that uses both rules and an ontology see (Golbreich, Dameron et al. 2005). The ontology provides structured knowledge about the main brain entities and properties, while rules are used to represent the interdependencies between properties.

They conclude:

Using rules with OWL (Web Ontology Language) ontology requires caution, in particular with an “integrated” language like SWRL (Semantic Web Rule Language). It is impossible to have at the same time, decidability, soundness, completeness, performance and expressivity. Therefore, the features expected from the application should be carefully evaluated, with regards to the properties and limitations of the method to be used.

In other words the same problem of dealing with the complexity of rules and the need to carefully debug rules that has been described in this white paper still occur when using an ontology.

There is no question of the value of the value of both ontologies and problem-solving methods, particularly for a large project, but they do not avoid the challenge of building and maintaining rules.

Machine learning

The purpose of this whitepaper was to consider why rules are difficult to add when, at first glance, they seem so easy to create. Our focus is those systems where rules are added by people, whether domain experts, business analysts or knowledge engineers. Knowledge acquisition by automatic derivation of rules by machine learning is outside this scope, but since it has always been considered as a way of avoiding the problems of people building rules, machine learning techniques will be discussed briefly.

Machine learning techniques automatically derive rules (or expertise in some form) automatically from suitable example data. There are now a wide range of machine learning techniques, decision tree techniques, neural networks, support vector machines and so on, but in terms of the discussion here, they are all concerned with deriving expertise from data.

The crucial issue is that these learning techniques can only be used **if appropriate data is available** i.e. *the training data include sufficient representatives of every data pattern that may occur and has been classified sufficiently accurately*. This is a major challenge: large databases with sufficient examples for each data pattern and which are accurately classified, are very rare. Apart from needing multiple examples of rare patterns there is a trade-off between classification accuracy and how fine-grained the classifications are. A simple example is the on-going [difficulty](#) in accurately coding hospital discharge summaries. In many health systems funding for hospitals depends on the discharge coding and government auditors check the accuracy of coding. In one study the actual discharge code matched the predicted code in only 39 out of 125 cases. This was perhaps a very specialised case: coding for patients undergoing anaesthetic video-assisted thoracoscopy, but this is precisely the point: the more subtle and fine-grained the classification the more difficult they are to make reliable. And of course the more fine-grained the classification the more cases are required to get sufficient examples for each data pattern, which in turn makes the task more difficult for human coders. Of course if one has an automatic classification system, these errors are avoided, but if one has an automatic classification system there is little point in developing a machine-learning classifier.

As an example of the challenges in producing training data, one of the earliest data sets in the [UC Irvine repository](#) of test data for machine learning, came from the original GARVAN-ES1 expert system referred to earlier (Horn, Compton et al. 1985). To ensure consistent classifications for the UC Irvine repository the Garvan data was not simply taken from patient archives, but run through the expert system.

Secondly a coarser set of 13 classifications was used to ensure sufficient examples for each class rather than the 60 classifications provided by GARVAN-ES1 (Quinlan, Compton et al. 1987).

One needs to distinguish here between data mining and machine learning, as the terms are often used interchangeably. Data mining techniques are concerned with discovering new patterns in the data, unknown to human experts, such as relationships and clusters, and are increasingly powerful and useful. E.g. collaborative filtering is a very powerful technique used by Amazon and others to suggest possible purchases. If you have bought the same book as others you may be interested in looking at their other purchases. We use the term “machine learning”, in this white paper, for techniques which learn how to assign classifications to new cases, given a set of previously classified training cases. That is, for problems where a sufficient number of training cases have already been accurately classified by humans or in some other way.

If appropriate data are available, certainly machine learning is highly useful and should be used, but appropriate training data is rare. If to develop a suitable data set one has to use a domain expert to classify the cases, the question arises of whether it is more efficient to use the expert to provide rules using RippleDown[®]. Firstly, the expert can construct a rule for single case, rather than having to patiently classify sufficient examples for the machine learning to be able to work. Secondly RippleDown ensures that the cases are classified consistently. This was demonstrated in a study using an early Ripple-Down Rule structure (Wang, Boland et al. 1996). So in the process of assisting the expert in consistently classifying cases to be used for machine learning RippleDown not only ensures that experts provide consistent classification, but at the same time accumulates the rules to automate the classifications.

We can note in passing that the use of fixed knowledge structures in Ripple-Down Rules, rather than the freely designed knowledge structures normally allowed, is similar to machine learning. In fact, a number of different machine learning algorithms have been based on one of the early Ripple-Down Rule structures. In practice the most widely used Ripple-Down Rule machine learner is RIDOR from the [WEKA machine-learning workbench](#), based on an algorithm developed by Gaines (Gaines and Compton 1995). A Google or Google Scholar search on “ridor” and “machine learning” provides thousands of links to machine learning research where RIDOR has been used. Relevant to knowledge acquisition from humans is a Ripple-Down Rule learner using “minimum description length methods” (i.e. where the amount of information used to encode the knowledge in the domain is kept as small as possible), where either the machine learner or a human can add a rule (Yoshida, Wada et al. 2004).

Knowledge elicitation

An important part of knowledge acquisition research, has been knowledge elicitation, that is helping and prompting experts to express knowledge about the domain.

Probably the most powerful knowledge elicitation techniques, and which come closest to the issues in debugging rules are based on Kelly's Personal Construct Psychology (Kelly 1955). The idea of Repertory Grids emerged from Kelly's psychology as a way of understanding a person's perceptions and viewpoints without prejudging a frame of reference. The core technique is that a person is asked to suggest three objects in the domain and are then asked in what way two of the objects are alike and different from the third. These axes of similarity and differentiation are known as constructs. Gradually a model of how the person sees the world, particularly relationships, emerges.

The same idea works as a brainstorming tool in a wide range of applications to help bring to light important, but perhaps hidden issues in a domain (Boose 1989). The factors that differentiate objects (or classifications) in the domain gradually emerge. Some Repertory Grid tools are linked to induction and representation and reasoning tools and can be used generate complete knowledge based systems (Gaines and Shaw 1993). A sophisticated public domain version of the approach is also available (Gaines and Shaw 2007 and <http://repgrid.com/>). Despite wide application and considerable success the main role of these tools seems to have been in brainstorming, in particular comprehensively elucidating the factors in a domain that need to be taken into account in building a rule-based system. They do not seem to have found major application in developing and debugging large production knowledge bases. The most likely reason is that these techniques work mainly on the conceptual level, identifying important concepts and relations, rather than addressing the issues in linking concepts to data and manipulating them via rules to produce practically useful output. Other techniques such as laddering and card sorting also assist in elucidating ideas but are simpler than Repertory Grids (Burton, Shadbolt et al. 1990).

Summary

Historical evidence as well as an analysis of what is involved in developing rules, shows that although it is technically very easy to create rules; it is a far more difficult task to ensure these rules give the correct answers when added to a knowledge base and it gets more and more difficult to add or change rules as the size of the knowledge base grows and/or as time passes and there is less connection to the original work of creating and structuring the knowledge base.

Secondly, various technologies that are in use or have been proposed to make it simpler to manage rules have been briefly reviewed. The literature indicates that although these all provide support in various ways, there still remains a fundamental challenge in building and maintaining rules.

For those interested in exploring in more detail the knowledge acquisition and engineering research outlined here, a useful starting point is a recent special issue of the International Journal of Human-Computer Studies on 25 years of knowledge acquisition research (Motta (ed) 2013).

Finally, we have not described RippleDown in this whitepaper, as it is described in other whitepapers on the PKS website, but we note that log data from a large number of knowledge bases, shows that the median time to add a rule across 57,626 rules was 78 secs (Compton, Peters et al. 2011). To date no other technology has been able to provide anything like this speed and ease of adding and maintaining rules.

References

- Baumeister, J. (2010). "Advanced empirical testing." *Knowledge-Based Systems* **24**: 83-94.
- Baumeister, J., M. Menge and F. Puppe (2008). "Visualization techniques for the evaluation of knowledge systems." *FLAIRS* **8**: 329-334.
- Baumeister, J. and D. Seipel (2010). "Anomalies in ontologies with rules." *Web Semantics: Science, Services and Agents on the World Wide Web* **8**: 55-68.
- Baumeister, J., D. Seipel and F. Puppe (2009). Agile development of rule systems. *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*. , IGI Publishing: 253-272.
- Bobrow, D., S. Mittal and M. Stefik (1986). "Expert systems: perils and promise." *Communications of the ACM* **29**(9): 880-894.
- Bonatti, P. A., D. Olmedilla and J. Peer (2006). *Advanced policy explanations on the Web*. Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006): 200-204.
- Boose, J. (1989). "A survey of knowledge acquisition techniques and tools." *Knowledge Acquisition* **1**: 3-37.
- Buchanan, B. (1986). "Expert systems: working systems and the research literature." *Expert Systems* **3**: 32-51.

Burton, A., N. Shadbolt, G. Rugg and A. Hedgecock (1990). "The efficacy of knowledge elicitation techniques: a comparison across domains and levels of expertise." *Knowledge Acquisition* **2**: 167-178.

Clancey, W. J. (1983). "The epistemology of a rule based system - framework for explanation." *Artificial Intelligence* **20**: 215-251.

Clancey, W. J. (1985). "Heuristic classification." *Artificial Intelligence* **27**: 289-350.

Clancey, W. J. (1997). *Situated Cognition: On Human Knowledge and Computer Representations (Learning in Doing - Social, Cognitive and Computational Perspectives)*, Cambridge University Press.

Compton, P., R. Horn, R. Quinlan and L. Lazarus (1988). "Maintaining an expert system." *Proc 4th Aust Conf on applications of expert systems*: 110-129.

Compton, P. and R. Jansen (1988). "Knowledge in context: A strategy for expert system maintenance." *Proc AI 88*: 283-297.

Compton, P. and R. Jansen (1990). "A philosophical basis for knowledge acquisition." *Knowledge acquisition* **2**: 241-257.

Compton, P., L. Peters, T. Lavers and Y.-S. Kim (2011). Experience with long-term knowledge acquisition. *Proceedings of the sixth international conference on Knowledge capture*. Banff, Alberta, Canada, ACM: 49-56. (also a white paper at <http://www.pks.com.au/technology/white-papers>)

Craw, S. (1996). "Refinement complements verification and validation." *International Journal of Human Computer Studies* **44**: 245-256.

Crubézy, M. and M. A. Musen (2003). Ontologies in support of problem solving. *Handbook on ontologies*, Springer: 321–341.

Date, C. J. (2000). *What, not how: the business rules approach to application development*. Reading, Mass., Addison Wesley.

Davis, R. and J. King (1975). "An overview of production systems." *Stanford Artificial Intelligence Laboratory Memo AIM-271*.

Eiter, T., G. Ianni, T. Krennwallner and A. Polleres (2008). Rules and ontologies for the semantic web. *Reasoning Web*: 1-53.

Forgy, C. (1982). "Rete: A fast algorithm for the many pattern/many object pattern match problem." *Artificial intelligence* **19**: 17-37.

Forgy, C. (1989). "Static and run-time characteristics of OPS5 production systems." *Journal of Parallel and Distributed Computing* **7**: 65-95.

Forgy, C. and J. McDermott (1977). "OPS, a domain-independent production system language." *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* **77**(1): 933-939.

Fox, M. (1990). "AI and expert system myths, legends, and facts." *IEEE EXPERT* **Feb**: 8-20.

Gaines, B. and M. Shaw (1993). "Knowledge acquisition tools based on personal construct psychology." *The Knowledge Engineering Review* **8**(1): 49-85.

Gaines, B. and P. Compton (1995). "Induction of Ripple-Down Rules Applied to Modeling Large Databases." *Journal of Intelligent Information Systems* **5**(3): 211-228.

Gaines, B. and M. Shaw (2007). WebGrid Evolution through Four Generations 1994-2007. *TR-2007-WG* University of Calgary Technical Report.

Golbreich, C., O. Dameron, O. Bierlaire and B. Gibaud (2005). *What reasoning support for Ontology and Rules? The brain anatomy case study*. Workshop "Protégé With Rules, Madrid, Spain.

Hicks, R. (2007). "The no inference engine theory--Performing conflict resolution during development." *Decision Support Systems* **43**: 435-444.

Hitzler, P. and B. Parsia (2009). Ontologies and rules. *Handbook on Ontologies*, Springer: 111-132.

Horn, K., P. J. Compton, L. Lazarus and J. R. Quinlan (1985). "An expert system for the interpretation of thyroid assays in a clinical laboratory." *Australian Computer Journal* **17**(1): 7-11.

Jacob, R. and J. Froscher (1990). "A software engineering methodology for rule-based systems." *IEEE Transactions on Knowledge and Data Engineering* **2**(2): 173-189.

Kelly, G. (1955). *The Psychology of Personal Constructs*. New York, Norton.

Lenat, D. B., M. Prakash and M. Shepherd (1985). "CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks." *AI magazine* **6**(4): 65-85.

McCarthy, J. (1977). *Epistemological problems of artificial intelligence*. International Joint Conference on Artificial Intelligence: 1038-1044.

McCarthy..., J. (1959). "Programs with common sense." *McCarthy, John (1959). Programs with Common Sense, Proceedings of the Teddington Conference on the Mechanization of Thought Processes, London: Her Majesty's Stationery Office.*

Merritt, D. (2004). "Best Practices for Rule-Based Application Development." *Microsoft Architecture Journal* **1**: 1-14.

Motta, E (ed) (2013). special issue: "25 years of Knowledge Acquisition." *Int. J. Human-Computer Studies*. 71(2) :131-216

Musen, M. A. (1998). "Modern architectures for intelligent systems: reusable ontologies and problem-solving methods." *Proceedings of the AMIA Symposium*: 46-52.

Musen, M. A. (2013). "The knowledge acquisition workshops: A remarkable convergence of ideas." *International Journal of Human-Computer Studies*. 71(2): 195-199.

O'Leary, D. (1991). Design, development and validation of expert systems: a survey of developers. *Verification and Test of Knowledge-Based Systems*. A. M and L. J-P, John Wiley: 3-21.

OMG (2009). "Production Rule Representation (PRR)." *Object Management Group* <http://www.omg.org/spec/PRR/1.0>.

Pérez, A. G. and V. R. Benjamins (1999). "Overview of knowledge sharing and reuse components: Ontologies and problem-solving methods." *IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden*.

Polit, S. (1984). "R1 and Beyond: AI Technology Transfer at Digital Equipment Corporation." *AI Magazine* **5**(4): 76-78.

Preece, A. (1994). "Validation of knowledge-based systems: The state-of-the-art in north america." *Study of Artificial Intelligence, Cognitive Science and Applied Epistemology* **11**.

Preece, A. (2001). "Evaluating verification and validation methods in knowledge engineering." *University of Aberdeen*.

Preece, A. D. and R. Shinghal (1994). "Foundation and application of knowledge base verification." *International journal of Intelligent Systems* **9**(8): 683-701.

Puppe, F. (1993). *Systematic introduction to expert systems: Knowledge representations and problem-solving methods*. Berlin, Springer-Verlag.

Quinlan, J. R., P. J. Compton, K. A. Horn and L. Lazarus (1987). Inductive knowledge acquisition: A case study. *Applications of Expert Systems*. London, Addison Wesley: 159-173.

Richards, D. and P. Compton (1997). Combining Formal Concept Analysis and Ripple Down Rules to Support Reuse. *Software Engineering and Knowledge Engineering SEKE'97*. Berlin, Springer Verlag: 177-184.

Schreiber, G., H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. Van de Velde and B. Wielinga (1999). *Knowledge Engineering and Management: The CommonKADS Methodology*. Cambridge Mass., MIT Press.

Schreiber, G., B. Wielinga, R. d. Hoog, H. Akkermans and W. v. d. Velde (1994). "CommonKADS: A comprehensive methodology for KBS development." *IEEE Expert* 9(6): 28-37.

Soloway, E., J. Bachant and K. Jensen (1987). *Assessing the maintainability of XCON-in-RIME: coping with the problems of a VERY large rule base*. Proceedings of AAAI 87, Seattle, Morgan-Kaufman: 824-829.

Speel, P., A. T. Schreiber, W. Van Joolingen, G. van Heijst and G. Beijer (2001). Conceptual models for knowledge-based systems. *Encyclopedia of Computer Science and Technology*. A. Kent and J. G. Williams, Marcel Dekker. 44.

Sviokla, J. (1990). "An examination of the impact of expert systems on the firm: the case of XCON." *MIS Quarterly* 14(2): 127-140.

Vincent, P. (2008). "OCEB White Paper on Business Rules, Decisions, and PRR." (version 1.1): 1-13.

Wang, J. C., M. Boland, W. Graco and H. He (1996). Use of ripple-down rules for classifying medical general practitioner practice profiles repetition. *Proceedings of Pacific Knowledge Acquisition Workshop PKAW'96*, Coogee, Australia: 333-345.

Winograd, T. and F. Flores (1987). *Understanding computers and cognition*. Reading, MA, Addison Wesley.

Yoshida, T., T. Wada, H. Motoda and T. Washio (2004). "Adaptive Ripple Down Rules method based on minimum description length principle." *Intelligent Data Analysis* 8(3): 239.

Zacharias, V. (2008). *Development and Verification of Rule Based Systems—A Survey of Developers*. RuleML 2008, Orlando, Springer-Verlag: 6-16.

Zacharias, V. (2009). The Debugging of Rule Bases. *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*, IGI Global: 302-325.

Zacharias, V. and A. Abecker (2007). On Modern Debugging For Rule-Based Systems. *Software Engineering and Knowledge Engineering (SEKE 2007)*. Boston, Knowledge Systems Institute Graduate School: 349-353.