Linked Production Rules: controlling inference with knowledge

Paul Compton[1], Yang Sok Kim[2],Byeong Ho Kang[2]

[1] School of Computer Science and Engineering, The University of New South Wales
Sydney 2052, Australia
[2] School of Computing and Information Systems, The University of Tasmania
Hobart 7005, Australia

compton@cse.unsw.edu.au, yangsok.kim@utas.edu.au,
byeon.kang@utas.edu.au

Abstract. A key insight in artificial intelligence, which has been the foundation of expert systems and now business-rule systems, is that reasoning or inference can be separated from the domain knowledge being reasoned about. We suggest that the knowledge acquisition and maintenance problems that arise, might result from too great a separation of knowledge and inference. We propose Linked Production Rules, where each rule evaluated directs the next step of inference and the inference engine has no meta-heuristics or conflict resolution strategy. We suggest that this loses none of the power of conventional inference but may greatly improve knowledge acquisition and maintenance since various Ripple-Down Rule knowledge acquisition methods, which have had some success in facilitating knowledge maintenance can be described as specific instances of Linked Production Rules. Finally the Linked Production Rule approach suggests the possibility of a generalized Ripple-Down Rule method applicable to a wide range of problem types.
Keywords: inference engine, declarative knowledge, conflict-resolution, problem-solving methods, Ripple-Down Rules.

1 Introduction

A crucial insight in the history of artificial intelligence was that domain knowledge should not be embedded in procedural code, but that knowledge and reasoning can be separated. This insight resulted in the classic idea of a rule-based expert system, composed of a knowledge base, working memory and an inference engine. However, the inference engine itself contains knowledge embedded in procedural code: not only different reasoning strategies, but ad hoc conflict resolution strategies.

While there has been some research on specifying the knowledge used in controlling inference e.g. [1], it is probably fair to say that most business-rule systems, the latest incarnation of the expert system idea use the same sort of inference engine and conflict resolution strategies developed over 30 years ago for expert systems. The aim of this paper is to revisit the way in which domain knowledge and inference have

Preprint (with corrections)
Compton, P., Y. S. Kim and B. H. Kang (2014). Linked Production Rules: Controlling Inference with Knowledge. PKAW 2014 - Knowledge Management and Acquisition for Smart Systems and Services, Springer: 84-98.

been separated in expert systems, and now business rule systems, and question whether this is the best approach to building these systems. The discussion here is restricted to forward-chaining propositional production rule systems, as these are the dominant form of rule-based technology.

Typically [2] a production rule is described as having two components a condition part and an action list

```
if [condition] then [action-list]
```

This rule representation does not contain any information about which rule should be evaluated next as this is determined by the inference engine. In practice there may be other information specified with the rule such as salience but this only gives the rule a priority to be used in determining whether a rule should be evaluated. In contrast we propose an approach where each rule contains information that determines which rule will be evaluated next rather than this being determined by the inference engine using its conflict resolution heuristics.

In our proposal for **Linked Production Rules**, there are three types of component: [condition], [**case** action-list] and [**inference** action]. Inference actions do not change the case (working memory), but specify which rule is to be evaluated next and only a single rule can be specified. Case actions change the case in the conventional way. We use the term case rather than working memory to emphasise that rule-based systems process and add further facts to cases where a case is a group of data from the world (e.g. a partial patient record). The rule-based system reasons about the case and if possible adds to the case (e.g. a diagnosis or patient management plan).

In the Linked Production Rule approach inference actions are specified for every rule both for when the rule is satisfied and when it is evaluated but fails. This gives the following structure:

```
if [condition] then [case action-list],
                    [inference action]
else [inference action]
```

In this representation there is a single specific inference action for whether the rule fires or fails to fire.

The paper contains: a review of the problems with rule-based systems; a discussion of the simplicity of knowledge acquisition with Ripple-Down Rules and how this arises because they are instances of Linked Production Rules and finally an outline of how Linked Production Rules can be used for rule-based systems in general.

2 Issues with rule-based systems

After the initial euphoria over the possibilities with expert systems, it became clear that they were harder to build and maintain than hoped. XCON, one of the most successful early expert systems, was very expensive to maintain. It 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 [3]. XCON eventually had 6,500 rules with 50% changed every year – a major maintenance challenge [4].

It has been argued that the situated nature of knowledge is the underlying problem in knowledge acquisition (e.g. [5]). Clancey in particular argued that knowledge is not something in the head to be mined – a common metaphor – but is constructed in particular contexts. From experience maintaining an early medical expert system, it was argued that human expertise does not provide reasons why conclusion X is correct in any absolute sense, but why it is a better hypothesis than other conclusions in the context [6]. This parallels the qualification problem [7]: that it is impossible to enumerate every factor that might alter the outcome of an action.

Situated cognition explained why experts cannot readily justify conclusions out of context, but we suggest that the separation of knowledge and inference in rule-based systems has compounded the difficulties of acquiring, and particularly maintaining, knowledge. As suggested by the qualification problem, and as observed (e.g [8]), there always seem to be errors and omissions that need fixing; however, one does not readily know how changes in knowledge will interact with the inference engine and its conflict resolution strategies acting on top of domain knowledge and exerting procedural control over how the domain knowledge is used. "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"[9]. In a study of expert systems for industrial applications, in which less that half the apparently successful systems were actually functioning, Bachmann et al noted: "Parts of the problem solving strategy applied by experts in the field of configuration or machinery diagnosis are not represented in the knowledge base of the system but in its inference engine. This is totally ignored if highly specified ready-made shells are used". [10]

Various researchers have identified that expert systems addressed different kinds of tasks e.g. [11,12]. This led to comprehensive software engineering approaches where the nature of the problem, and hence the appropriate problem-solving method, were identified prior to acquiring specific domain knowledge (e.g. [13]). One would expect such methods, as with any systematic software engineering, to increase the likelihood that a successful system will result; however, it is unclear whether this reduces the maintenance problems due to the separation of knowledge and inference, or whether these remain because the knowledge embedded in the inference engine is not under the control of the domain expert. In a recent survey of developers of rule-based systems, using a range of tools and methods, the major problem identified was knowledge debugging i.e. dealing with the difficulties in making changes to a knowledge base [14]. It seems clear that these debugging problems are precisely because the developer often does not know how a change in the knowledge base will be handled by the inference engine without trial and error.

Our focus here is on rule-based systems because of their re-emergence as business rule systems. A central assumption behind business rule systems is again that de-

clarative knowledge in the form of rules is much more maintainable than procedural code in data base triggers [15] – the expert system insight revisited. A web search suggests that these systems are largely based on inference techniques deriving from work on RETE networks [16] and the conflict resolution strategies used in early expert systems. There is also interest in sequential inferencing for business rules, but aimed at improving performance for a given rule set rather than improving knowledge maintenance e.g. [17]. The general idea of a strict inference sequence is part of the OMG standard [2]; what we propose here is a particular way structuring the sequence.

One conflict resolution strategy, salience, relates to our proposal. Salience and the more formally defined Courteous Logic [1] allow the expert or knowledge engineer to specify the relative priority between pairs of rules. Such approaches are of increasing importance and Courteous Logic is part of RuleML [18]. These techniques are an option the expert or knowledge engineer may apply to some rules, to help resolve between which rule to fire if both are candidates. In contrast, the Linked Production Rule approach requires that the next rule to be evaluated is explicitly specified by the current rule being evaluated. Possibly this full specification of inference paths could be implemented with Courteous Logic, or other forms of labeled logic, but the starting point of these approaches is to determine between candidate rules which should be evaluated next, whereas with Linked Production Rules, each rule that is evaluated (whether true or false) specifies the next rule to be evaluated.

3 Ripple-Down Rules (RDR)

In this section we briefly outline some of the successes of Ripple-Down Rules (RDR). Since, as will be shown, RDR are an example of Linked Production Rules, the utility of RDR demonstrates or at least suggests the utility of Linked Production Rules. The two most common forms of RDR, single and multiple classification, will be presented as Linked Production Rule systems below.

3.1 RDR introduction

RDR were developed in response to the maintenance problems of expert systems [19]. The essential feature of RDR is that rules are added to deal with specific cases (generally while the system is already in operational use). That is, a case has been processed by the system with a specific inference sequence and has failed to give the correct output. New rules to give the correct output for the case are added into the inference sequence. This rule placement is automatic and outside the control of the expert (or knowledge engineer). Since there is no knowledge structuring by the expert or knowledge engineer and no requirement to understand the knowledge base as a whole, rules can be added by domain experts. Greater detail of these algorithms will be shown using the linked rule representation below.

The strict inference sequencing in RDR enables a second type of knowledge acquisition support. An expert may add a rule for the case that is overly specific, but they

can only introduce an error affecting the previous knowledge base by adding a rule that is too general, so that cases previously processed by the same sequence may now incorrectly fire the new rule. Such cases can be shown to the expert who either makes their rule more specific to exclude the cases, or accepts that the rule should apply to these cases. Suitable cases can be provided by saving the cases for which rules are added, known as 'cornerstone cases'.

3.2 RDR experience

The first RDR system in routine use, using SCRDR inference (see below), was a 2000 rule system for chemical pathology [20]. Later pathology systems in routine use were based on MCRDR (see below). In a study of these systems, chemical pathologists from one laboratory added about 16,000 rules across 20 knowledge bases over a 29 month period, at an median speed of 77 secs per rule [21]. Since rules are only added for cases, 77 secs is the time to call up a case, add a rule and test the rule against other cases. The case has already been identified as requiring a new conclusion – a standard pathologist task. The median time of 77 secs covers the purely knowledge engineering task of selecting and verifying conditions for the rule, or rules against past cases. In a more recent larger study the median time to add a rule across 17 laboratories, 256 knowledge bases and about 56,000 rules was 78 secs [22]. About 46% of the rules were added after the systems had been in use for more than a year and the median time to add these rules was 91 secs. This on-going and very rapid rule acquisition strongly suggest that experts found it still easy to add rules more than a year after the system had been introduced. This contrasts with the comments that the whole knowledge base needs to be understood [9]. There is no directly comparable data on the time to add rules to a conventional knowledge base, but Zacharias' study implies the average time to add a rule is more like half an hour to five hours [14].

RDR are used commercially in other areas apart from medicine. For example Ivis [23] and Erudine [24] use technology based on RDR for marketing and software engineering respectively, while IBM offers data a cleansing product based on RDR [25]. The point of these references is to demonstrate that RDR, which we will now describe in terms of Linked Production Rules, is a commercially successful approach to building rule-based systems, and which as the results from pathology demonstrate, allows very rapid and simple rule addition throughout the life of the system.

3.3 RDR as Linked Production Rules

SCRDR inference. Table 1 shows an example of a single classification RDR (SCRDR) knowledge base as linked rules. The same SCRDR knowledge base is shown in Fig 1 using a more conventional RDR tree representation. SCRDR was the first RDR system proposed [19] and used in the first RDR pathology system[20]. An SCRDR knowledge base is a binary tree, but in practice such trees are very unbalanced since rules tend to be added as new rules or new corrections to a rule rather than deeper and deeper corrections.

Rule no	Case actions	Inf. action (true)	Inf. action (false)
1	assert conc 1	exit	to rule 2
2	assert conc 2	to rule 3	to rule 4
3	retract conc 2 assert 3	exit	to rule 6
4	assert conc 4	to rule 5	exit
5	retract conc 4 assert 5	exit	to rule 7
6	retract conc 2 assert 6	exit	exit
7	retract conc 4 assert 7	exit	exit

Table 1: an SCRDR knowledge base as Linked Production Rules. Inf. action is an abbreviation for inference action. The rule numbers indicate the order in which rules were added. Inference starts with rule 1 and terminates when an exit is reached. For illustration, the conclusions are identified by the number of the rule which asserts them. In practice some of the conclusions may be identical as different inference paths may give the same conclusion.

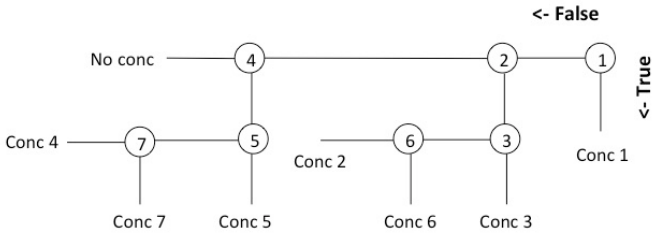

Figure 1: the same knowledge base as Table 1, but shown as a binary tree. The rule numbers are in circles. If a rule fires the right hand link is traversed, if it fails to fire the left hand link is traversed. The rule numbers indicate the order in which rules were added

Table 2 shows examples of extra rules being added to the knowledge base in Table 1. Rule 8 is added for a case where no conclusion was given. The false branch of rule 4, the former exit point for the case, links to rule 8. Rule 9 is added because rule 6 has given the wrong conclusion and so the rule 6 inference action for true, links to rule 9. Rule 10 is added since rule 4 gave the wrong conclusion, and no correction rule fired (i.e rules 5 & 7). The false branch inference action for rule 7, links to rule 10 giving the same sequence of 4 true, 5 & 7 false whenever 10 is evaluated.

4	assert conc 4	to rule 5	**to rule 8**
6	retract conc 2 assert 6	**to rule 9**	exit
7	retract conc 4 assert 7	exit	**to rule 10**
8	*assert conc 8*	*exit*	*exit*
9	*retract conc 6 assert 9*	*exit*	*exit*
10	*retract conc 4 assert 10*	*exit*	*exit*

Table 2: the three rules that have been added to Table 1 are shown, as well as the three rules whose inference action was changed to link to the new rules. New rules are in *italics* and changed actions are **bold**.

These changes to inference actions are not under the control of the expert, but automatically determined by the inference sequence for the case for which a rule is being added. The case action for a rule is not changed; only the inference action.

MCRDR Inference. Multiple classification RDR (MCRDR) was introduced to allow multiple conclusions for a case, e.g multiple diagnoses [26] and is widely used in pathology [22]. MCRDR has an n-ary tree structure; i.e. at the top level all rules are evaluated and if a rule fires all its correction rules are evaluated and so on; however, since rules are always evaluated one-by-one, the order can be made explicit. MCRDR papers do not specify the order of sibling rule evaluation but of course a specific order was implemented by the person programming the MCRDR system. Since the whole sequence must be specified with Linked Production Rules, we specify that older siblings should be evaluated before newer - which we believe has been the common actual practice. The difference here is that the sequence is specified in the knowledge base, rather than being part of the inference engine implementation.

Rule no	Case actions	Inf. action (true)	Inf. action (false)
1	assert conc 1	to rule 2	to rule 2
2	assert conc 2	to rule 3	to rule 4
3	retract conc 2 assert 3	to rule 4	to rule 6
4	assert conc 4	to rule 5	exit
5	retract conc 4 assert 5	to rule 7	to rule 7
6	retract conc 2 assert 6	to rule 4	to rule 4
7	retract conc 4 assert 7	exit	exit

Table 3: an MCRDR knowledge base as Linked Production Rules; same formatting as Table 1.

4	assert conc 4	to rule 5	**to rule 8**
7	retract conc 4 assert 7	**to rule 8**	**to rule 8**
8	*assert conc 8*	*exit*	*exit*

Table 4a: addition of rule 8 giving an extra conclusion

4	assert conc 4	to rule 5	**to rule 8**
6	retract conc 2 assert 6	**to rule 9**	**to rule 9**
7	retract conc 4 assert 7	**to rule 8**	**to rule 8**
8	*assert conc 8*	*exit*	*exit*
9	*retract conc 6 assert 9*	*to rule 4*	*to rule 4*

Table 4b: addition of rule 8 and then rule 9. Rule 9 is a correction to rule 6

4	assert conc 4	to rule 5	**to rule 8**
6	retract conc 2 assert 6	**to rule 9**	**to rule 9**
7	retract conc 4 assert 7	**to rule 10**	**to rule 10**
8	*assert conc 8*	*exit*	*exit*
9	*retract conc 6 assert 9*	*to rule 4*	*to rule 4*
10	*retract conc 4 assert 10*	*to rule 8*	*to rule 8*

Table 4c: addition of rules 8, 9 and 10. Rule 10 is a further correction to rule 4.

Table 3 shows an MCRDR knowledge base as Linked Production Rules with a parent before child and older child before new child sequence. The difference from the SCRDR KB in table 1 is that inference passes to a sibling rule rather than exiting after a conclusion is decided, resulting in potentially many conclusions. Tables 4a-c show how further MCRDR corrections are made; they are shown cumulatively, with one extra rule being added in each table.

The SCRDR and MCRDR Linked Production Rule representations in Tables 1 and 3 provide a complete description of SCRDR and MCRDR inference and how new rules are added. Because with RDR the rules are at least implicitly linked, RDR knowledge acquisition results in a new rule or rules being automatically linked into the inference sequence, as shown in the examples above. This allows for very simple rule addition, with the rule then being checked using the case differentiation support normally provided by RDR. That is, if any previous case can fire the new rule, then the expert needs to select further features from the current case to be added to the rule to distinguish the cases, or alternatively the expert may decide the conclusion from the new rule should apply to the previous case. The normal practice with RDR is to store and check the cases which have prompted the addition of a rule, so-called cornerstone cases. For SCRDR only the case for the parent rule needs to be considered, while for MCRDR the case for the parent rule, and the cases from sibling rules to the new rule need to be considered. A rule added at the top level of an MCRDR tree requires that all cases be checked, but in practice the expert only considers 2 or 3 cases before the rule is sufficiently precise to exclude all previous cases.

It is beyond the scope of this paper to consider other RDR methods and the range of RDR research (reviewed by [27]), but all RDR methods implicitly have a fixed inference path and a fixed way of adding new or correction rules into the inference path, and also check new rules against some sort of selection of past cases.

4 Production rule systems in general

Colomb and Chung have proved that any propositional production rule expert system is equivalent to and can be converted into a decision table [28]. Although this conversion is a simple in theory, it can produce unmanageably large decision tables. Colomb's solution, inter alia, is to use a set of cases known to be processed by the expert system to remove unused rows [29]. If as we suggest here the system is built as a decision table, with rows (rules) added only to deal with specific cases, the explosion in the number of rows will not arise.

Since the order of inference for the rows (rules) in a decision table is not defined, there is no loss of generality in specifying a particular order, e.g. from the first to the latest row or rule added. If a new rule is required for a case it is added at the end. Table 5 shows an example where rule 3 has given the wrong conclusion. It is retracted by rule 3.1 and a rule with the same body, but giving the new conclusion, is added at the bottom. The expert simply provides the rule to give the new conclusion, and

this is used for both. As with RDR, (cornerstone) cases are used to check if cases from earlier rules fire rule 6, and the rule is made more specific if necessary.

Although the structure in Table 5 is one of many possible, it has a particular feature, that if an expert adds a rule to retract a conclusion this is automatically linked to the rule whose conclusion is to be retracted. I.e. rules that retract conclusions are not added as standard rules as normally allowed, but always linked to the rule being corrected. Such rules are *composite rules* [30] and have also been described as censored production rules for use in data mining [31]. The difference between this approach and the RDR systems described above is that rules now only have one level of correction, and these corrections do not provide a conclusion to replace a previous conclusion, but simply stop the conclusion being given, with a new rule used to add the new conclusion. In fact in the first experiments with MCRDR using simulated experts, this particular structure was used with the correction rules called stopping rule [32].

Rule no	Case actions	Inf. action (true)	Inf. action (false)
1	assert conc 1	to rule 2	to rule 2
2	assert conc 2	to rule 3	to rule 3
3	assert conc 3	**to rule 3.1**	to rule 4
3.1(6)	*retract conc 3*	*to rule 4*	*to rule 4*
4	assert conc 4	to rule 5	to rule 4
5	assert conc 5	**to rule 6**	**to rule 6**
6	*assert conc 6*	*exit*	*exit*

Table 5: a decision table where rule 3 gives the wrong conclusion and should be replaced by the rule 6 conclusion.

The inference considered so far has been simple one-pass inference. Intermediate conclusions, to reduce the amount of knowledge required, are widely used in expert systems [12]. The use of intermediates generally requires that inference is repeated over the whole knowledge base, which of course results in rules that use the raw data to provide some intermediate conclusion, being evaluated before rules that need these intermediate conclusions to reach further conclusions – perhaps further intermediate conclusions. Since all rules are repeatedly considered for evaluation, a RETE network is often used to improve efficiency [16], with inference finally stopping when no further conclusions are made.

Rule no	Case actions	Inf. action (true)	Inf. action (false)
1	assert conc 1	to rule 1	to rule 2
2	assert conc 2	to rule 1	to rule 3
3	assert conc 3	to rule 1	to rule 4
4	assert conc 4	to rule 1	to rule 5
5	assert conc 5	to rule 1	exit

Table 6: a flat rule structure with repeat inference

We use the following to provide structure inference: knowledge acquisition or maintenance takes place because a case is identified as needing a new or corrected conclusion. This happens only after inference on the case is complete; i.e. all inference cycles have been completed. If a new rule is added at the bottom, like rule 6 in Table 5, this should be processed after the previous repeat inference for the case is complete so rule 6 is reached after the same inference sequence giving the same wrong conclusion. If this applies to all rules as they are added we end up with the structure shown in Table 6, where after each rule satisfied, inference returns to the first rule. Table 7: shows an example of where a correction rule is added for a repeat inference. Two correction rules are added to illustrate the links.

Rule no	Case actions	Inf. action (true)	Inf. action (false)
1	assert conc 1	to rule 1	to rule 2
2	assert conc 2	to rule 1	to rule 3
3	assert conc 3	**to rule 3.1**	to rule 4
3.1(6)	*retract conc 3*	*to rule 4*	*to rule 3.2*
3.2(7)	*retract conc 3*	*to rule 4*	*to rule 1*
4	assert conc 4	to rule 1	to rule 5
5	assert conc 5	to rule 1	to rule 6
6	*assert conc 6*	to rule 1	to rule 7
7	*assert conc 7*	to rule 1	*exit*

Table 7: a flat rule structure with repeat inference with two correction rules to retract the conclusion of rule 3, when the conclusions of rule 6 or 7 should be given instead.

To stop endless repeat inference the inference engine does not re-evaluate any rule that has already been satisfied (even if its conclusion has been retracted by one of its refinements). When inference is directed to such a rule the inference action of the false branch is followed. For problems such as configuration, where a solution has multiple components, with dependencies between them, but only one value for each component, we further constrain inference. If a component is already part of the case (from the original data or added by a previous rule) any further rule assigning a value for that component is missed (i.e. its false branch is followed). If components in the solution need to be added to or changed, they are fixed in the order in which they are inferred. That is, the first component is fixed and the case is rerun and the next error fixed – perhaps newly introduced by the first correction, and the case is rerun again and so on. Because of the way the rules are linked, no earlier conclusion for a case can be altered by a later change, so this approach guarantees that the maximum number of changes to fix a case (i.e. the number of rules added) is no more than the number of components that make up the overall conclusion for the case. Each fix, or addition to the inference sequence, should take the same minimal time as with RDR.

It is beyond the scope of this paper to provide detailed examples of Linked Production Rules for other problem types; however, the approach outlined is essentially a problem solving method that builds a solution for a case one component at a time, regardless of whether it is a classification or a construction problem. Errors in partic-

ular components are corrected by knowledge acquisition rather than by reasoning as in problem solving methods such as Propose and Revise.

A final question is computational efficiency because inference is repeated each time a conclusion is reached. Considered very simplistically: if there are *n* components to a solution and these are inferred randomly through the m rules, on average m x n/2 rules will need to be evaluated, with m x n as the worst case. If we consider a conventional system in the same simplistic way, all rules are evaluated on each pass for the conflict resolution to determine which satisfied rule will be acted on. The means at least m x n rules will be evaluated. Of course a RETE network or other techniques can be used to speed the process – but such techniques, which essentially maintain an index of rules that have fired and rules that could fire, should be able to be applied to Linked Production Rules, suggesting a general Linked Production Rule system should be at least as efficient as a conventional system.

4.1 Ontologies and Linked Production Rules

A key aspect of the system described is that inference can only change a case by adding some fact or conclusion to it. Inference cannot retract a fact or conclusion that has already been reached, not can it change data provided initially. Putting this generally: whatever the status of the case (or working memory), when a rule fires it only adds to the case, it does not change or retract what is already part of this case whether a rule conclusion or part of the data supplied to the system. If an incorrect conclusion is made this is fixed by having an exception rule, which prevents that conclusion being added to the case in the first place. If the original data includes an error, again the system does not change this, although of course it may output a conclusion that the data should be re-checked. That is, it is not the responsibility of a rule to correct a prior error, whether made by a rule or in the original data; the source of the error should be corrected so that the error is not made in the first place, and this can be easily done through on-going rule addition as with Ripple Down Rules.

We have described initially a system where a conclusion is a simple Boolean, so that once a conclusion is asserted it can't be retracted, but then for configuration we have assumed a number of parameters each of which can take a single value, so that once a value for a particular parameter has been assigned no other rule can assign a different value for that parameter. If one were to assume a taxonomy was specified, then once a diagnostic conclusion was reached about a type of leaf disease, for example, another rule should not make a conclusion about a type of liver disease, because leaf disease implies the data is about a plant, while liver disease implies that data is about an animal. This is a far fetched example but we can use it to generalize the constraint that a case can only be added to, not changed: I.e. a conclusion cannot be asserted which has ontological implications that conflict with the ontological implications of a conclusion that has already been asserted or of the original data. This relates to term-subsumption systems such as KL-ONE [33], some of which include Ripple-Down Rules [34]. In such systems one can reason both by inheritance and by rules, whereas here we propose that the ontology simply provides constraints to ensure that rules can only add to a case not change it. Although the broad principle of not allowing rules to fire which would alter the case, either explicitly or implicitly via

an ontology, seems reasonable, we have not explored how this would work with more complex ontologies.

5 Discussion

The Linked Production Rule system proposed for production rules in general, is not necessarily the only general linked rule system possible. Our aim has been simply to demonstrate that such a system is possible. Since the rule acquisition task of adding a rule into a sequence is essentially the same as for RDR, each change should take no more than a minute or two. This contrasts greatly with comments about expert systems in general [9].

Conversely there seems to be no advantage in the various conventional conflict resolution strategies over Linked Production Rules. Linked Production Rules with a strict rule ordering, replace position-in-text rule ordering and salience etc. In particular salience, or the more recent Courteous Logic, is an ad hoc ordering or prioritising imposed by the knowledge engineer for some rules. Linked Production Rules impose a total ordering, but determined by the order in which rules are added and corrected. In particular Linked Production Rules replaces the conflict resolution strategy of selecting the more specific rule. The exception rules used in Linked Production Rules (e.g. 3.1 & 3.2 in Table 7) require the parent rule also to be satisfied, but once the conclusion is retracted, replacement rules such as 6 & 7 do not need to be more specific. Strategies such as recency ordering and data ordering are also embedded in the linked rule approach – or rather they are explicit, rather than some arbitrary function of the inference engine.

Our approach has similarities with work on verification e.g [35,36] and refinement knowledge acquisition techniques [37], as these techniques are based on identifying inference paths; however, this is done after the knowledge is assembled, whereas we propose to build all rules with a linked structure from the outset.

The early assumptions with expert systems were that you simply added knowledge and built a system. Of course it turned out to be much more difficult than this, and one of the major insights was that in fact there were many different problem types [38] which needed to be thought about in different ways and for which different problem solving methods were appropriate. This sort of analysis started with Clancey's recognition that many expert systems used a method that he called heuristic-classification [12] and led on to careful descriptions of families of problem-solving methods [39] and methods for managing the overall process of building a knowledge-based system, including selecting the problem-solving method [13]. If with Linked Production Rules, the domain expert can build up a set of rules case by case, to provide the solution for the case and the process of adding and correcting rules (by adding retraction or correction rules) itself determines the inference path through the knowledge base, what is the problem-solving method?

One of course might say that linked-production rules are a problem-solving method, particularly when including heuristics such as:

- Errors are corrected by adding exception to clauses to rules, preventing the error.
- Intermediate conclusions can be used as in heuristic classification
- Inference returns to the first rule after each conclusion is asserted
- Once a particular conclusion is added it cannot be deleted or its value changed
- New conclusions cannot be added which conflict with the ontological implications of previous conclusions.

However, these heuristics are not really about reasoning or inference, but about managing knowledge acquisition, and reflecting the knowledge acquisition that has occurred. Returning inference to the first rule after a conclusion is asserted ensures that inference follows the same path for any case for which the expert has added rules. Not allowing conclusions to be changed by inference is a way of forcing changes to be made by the domain expert adding specific knowledge.

Traditionally problem-solving methods have been ways of inferring over some knowledge. In contrast Linked Production Rules as a problem-solving method, are a way of managing knowledge acquisition, so that the sequence of knowledge acquisition provides the sequence of inference. Although this may make it easier to add knowledge to a knowledge-based system, one must remember that many of the difficulties identified by methods like CommonKADS remain:

- One must develop a clear understanding of the problem to be addressed,
- and develop an appropriate representation and terminology for the domain,
- and issues of interfacing to other information system as well as appropriate user interfaces are critical if a system is to have any chance of success.

6 Conclusion

We have proposed Linked Production Rules as a replacement for general production rule systems and their conflict resolution strategies. Nothing seems to be lost in this approach, but knowledge acquisition and maintenance should be greatly facilitated. Although the approach closely links domain knowledge and inference, the expert is only required to provide domain knowledge; the inference actions providing the links are added automatically. The expert or knowledge engineer can completely ignore this structure, rather than trying to control the hidden decisions of inference heuristics.

7 Acknowledgement

This research was supported by an Australian Research Council Discovery Grant

8 References

1. Grosof, B.: Prioritized conflict handling for logic programs. In: Logic Programming: Proceedings of the 1997 International Symposium 1997, pp. 197–211
2. OMG: Production Rule Representation (PRR). Object Management Group http://www.omg.org/spec/PRR/1.0 (2009).

3. Polit, S.: R1 and Beyond: AI Technology Transfer at Digital Equipment Corporation. AI Magazine **5**(4), 76-78 (1984).
4. Soloway, E., Bachant, J., Jensen, K.: Assessing the maintainability of XCON-in-RIME: coping with the problems of a VERY large rule base. In: Proceedings of AAAI 87, Seattle 1987, pp. 824-829. Morgan-Kauffman
5. Clancey, W.J.: Situated Cognition: On Human Knowledge and Computer Representations (Learning in Doing - Social, Cognitive and Computational Perspectives). Cambridge University Press, (1997)
6. Compton, P., Jansen, R.: A philosophical basis for knowledge acquisition. Knowledge acquisition **2**, 241-257 (1990).
7. McCarthy, J.: Epistemological problems of artificial intelligence. In: International Joint Conference on Artificial Intelligence, Jan 1 1977, pp. 1038-1044
8. Compton, P., Horn, R., Quinlan, R., Lazarus, L.: Maintaining an expert system. In: Quinlan, J.R. (ed.) Applications of Expert Systems, vol. 2. pp. 366-385. Addison Wesley, London (1989)
9. Jacob, R., Froscher, J.: A software engineering methodology for rule-based systems. IEEE Transactions on Knowledge and Data Engineering **2**(2), 173-189 (1990).
10. Bachmann, R., Malsch, T., Ziegler, S.: Success and failure of expert systems in. different fields of industrial application. GWAI-92: Advances in Artificial Intelligence, 77-86 (1993).
11. Stefik, M., Aikins, J., Balzer, R., Benoit, J., Birnbaum, L., Hayes-Roth, F., Sacerdoti, E.: The organization of expert systems, a tutorial* 1. Artificial intelligence **18**(2), 135-173 (1982).
12. Clancey, W.J.: Heuristic classification. Artificial Intelligence **27**, 289-350 (1985).
13. Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., Van de Velde, W., Wielinga, B.: Knowledge Engineering and Management: The CommonKADS Methodology. MIT Press, Cambridge Mass. (1999)
14. Zacharias, V.: Development and Verification of Rule Based Systems—A Survey of Developers. In: Bassiliades, N., Governatori, G., Paschke, A. (eds.) RuleML 2008, Orlando, Jan 1 2008, pp. 6-16. Springer-Verlag
15. Date, C.J.: What, not how: the business rules approach to application development. Addison Wesley, Reading, Mass. (2000)
16. Forgy, C.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial intelligence **19**, 17-37 (1982).
17. Oracle: Linear Inferencing: High Performance Processing. An Oracle White Paper **February** (2009).
18. Grosof, B.: Representing e-commerce rules via situated courteous logic programs in RuleMl. Electronic Commerce Research and Applications **3**, 2-20 (2004).
19. Compton, P., Jansen, R.: Knowledge in context: A strategy for expert system maintenance. In: Barter, C.J., Brooks, M.J. (eds.) AI'88 (Proceedings of the 1988 Australian Artificial Intelligence Conference) 1989. Lecture notes in artifial intelligence, pp. 292-306 Springer-Verlag
20. Edwards, G., Compton, P., Malor, R., Srinivasan, A., Lazarus, L.: PEIRS: a pathologist maintained expert system for the interpretation of chemical pathology reports. Pathology **25**, 27-34 (1993).

21. Compton, P., Peters, L., Edwards, G., Lavers, T.G.: Experience with Ripple-Down Rules. Knowledge-Based System Journal **19**(5), 356-362 (2006).
22. Compton, P., Peters, L., Lavers, T., Kim, Y.-S.: Experience with long-term knowledge acquisition. Paper presented at the Proceedings of the sixth International Conference on Knowledge Capture, KCAP 2011, Banff, Alberta, Canada,
23. Sarraf, Q., Ellis, G.: Business Rules in Retail: The Tesco.com Story. Business Rules Journal **7**(6), http://www.BRCommunity.com/a2006/n2014.html (2006).
24. Benham, A., Read, H., Sutherland, I.: Network Attack Analysis and the Behaviour Engine. In: Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on 2013, pp. 106-113. IEEE
25. Dani, M.N., Faruquie, T.A., Garg, R., Kothari, G., Mohania, M.K., Prasad, K.H., Subramaniam, L.V., Swamy, V.N.: Knowledge Acquisition Method for Improving Data Quality in Services Engagements. In: IEEE International Conference on Services Computer (SCC), Miami 2010, pp. 346-353. IEEE
26. Kang, B.H., Compton, P.: Knowledge acquisition in context : the multiple classification problem. In: Proceedings of the 2nd Pacific Rim International Conference on Artificial Intelligence, Seoul 1992, pp. 847-853
27. Richards, D.: Two decades of Ripple Down Rules research. The Knowledge Engineering Review **24**(2), 159–184 (2009). doi:10.1017/S0269888909000241
28. Colomb, R., Chung, C.: Strategies for building propositional expert systems. International Journal of Intelligent Systems **10**(3), 295-328 (1995).
29. Colomb, R.M.: Representation of Propositional Expert Systems as Partial Functions. Artificial Intelligence **109**, 187-209 (1999).
30. Crawford, E., Kay, J., McCreath, E.: IEMS - the intelligent mail sorter. In: Sammut, C., Hoffmann, A. (eds.) Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002), Syndey 2002, pp. 83-90. Morgan Kaufmann
31. Kim, Y., Compton, P., Kang, B.: Ripple-Down Rules with Censored Production Rules Paper presented at the Knowledge Management and Acquisition for Intelligent Systems - 12th Pacific Rim Knowledge Acquisition Workshop, PKAW 2012, Kuching,
32. Kang: Multiple Classification Ripple Down Rules (PhD thesis). UNSW, (1996)
33. Brachman, R.J., Schmolze, J.G.: An Overview of the KL-ONE Knowledge Representation System*. Cognitive Science **9**(2), 171-216 (1985).
34. Gaines, B.R.: Integrating Rules in Term Subsumption Knowledge Representation Servers. In: AAAI 1991, pp. 458-463
35. Nguyen, T., Perkins, W., Laffey, T., Pecora, D.: Checking an expert systems knowledge base for consistency and completeness. In: Proceedings of the 9th international joint conference on Artificial intelligence 1985, pp. 374–378
36. Preece, A.D., Shinghal, R., Batarekh, A.: Principles and practice in verifying rule-based systems. The Knowledge Engineering Review **7**(2), 115 - 141 (1992).
37. Craw, S.: Refinement complements verification and validation. International Journal of Human Computer Studies **44**, 245-256 (1996).
38. Chandrasekaran, B.: Towards a taxonomy of problem solving types. AI Magazine **Winter/Spring**, 9-17 (1983).
39. Puppe, F.: Systematic introduction to expert systems: Knowledge representations and problem-solving methods. Springer-Verlag, Berlin (1993)