

Visualization of Rule Bases - The Overall Structure ¹

Valentin Zacharias

(FZI Forschungszentrum Informatik, Germany
zacharias@fzi.de)

Abstract: In this paper we describe novel ideas and their prototypical implementation for the visualization of rule bases. In the creation of the visualization our approach considers not only the structure of a rule base but also records of its usage. We also discuss methods to address the challenges for visualization tools posed by rule bases that are large, created with high level knowledge acquisition tools or that contain low level rules that should remain hidden from the users.

Key Words: visualization, rule bases, knowledge visualization, visualization of knowledge structures, f-logic

Category: H.5.2, H.5

1 Introduction

The visualization of entire rule bases, independent of the answer to any particular query, is a problem that has so far been largely ignored by the research community. The goals for such a visualization are the same as for UML class diagrams and other overview representations of programs: to aid teaching, programming, debugging, validation and maintenance by facilitating a better understanding of a computer program by the programmers. The importance of such representations cannot easily be overstated and is probably higher than in object oriented programming; In object oriented programming the overall structure is explicitly created by the programmer with links between objects, the inheritance hierarchy and method calls; in rule bases the interaction between rules is only decided by the inference engine and usually never shown to the user. An overview representation of the entire rule bases is necessary to make this hidden structure visible. Before suitable visualizations for rule bases can be created, however, we need a definition of the *overall structure of the rule base* - doing this with respect to usage data is the main contribution of this paper.

This paper starts with a definition of rule bases. It then goes on to define the overall structure of a rule base, first the static structure and then the dynamic structure as a combination of all proof trees from uses of the rule base. In the end we shortly sketch some advanced problems, such as hiding language axioms or showing rule bases at different levels of abstraction.

¹ An extended version of this paper is available from <http://vzach.de/papers/visualizationLong.pdf>

2 Terminology - Rule Bases

A rule base is a finite set of rules. The rules are of the form:

$$p(A) \leftarrow q(B), \text{not } r(C, d)$$

The part to the left of the \leftarrow symbol is the *goal* or *head* of the rule, the part to the right the *body* or *subgoals*. The head of the rule consists of one *atom*, the body of a conjunction of *literals* (atoms that may be negated). The arguments of the predicates that make up head and body of the rule are called *terms*. Variables and constants as well as function symbols with terms as arguments are terms. A term without arguments is called a *ground term*. Variables are written with uppercase letters, ground terms with lowercase letters.

The meaning of a rule can be intuitively understood as *the statement in the head is true if the body can be proven* or even shorter as *if body then head*.

3 The static structure of rule bases - the rule graph

The core concept for the static structure of rule bases is the *depends-on* connection between rules. Such a link indicates that two rules seem to be able to work together. Consider the following rule base:

$$\text{father}(A) \leftarrow \text{male}(A), \text{parent}(A, B) \quad (1)$$

$$\text{parent}(X, Y) \leftarrow \text{child}(Y, X) \quad (2)$$

$$\text{employer}(X) \leftarrow \text{owns}(X, C), \text{employed_at}(A, C) \quad (3)$$

It can be seen that, with the right facts, rule (1) could work with the result of rule (2). The following fact base consisting of only two facts should serve as an example:

$$\text{male}(\text{mike}), \text{child}(\text{michelle}, \text{mike})$$

With these facts rule (2) could deduce that $\text{parent}(\text{mike}, \text{michelle})$ from $\text{child}(\text{michelle}, \text{mike})$ and rule (1) could then deduce that $\text{father}(\text{mike})$. In such a case we say that rule (1) *depends-on* rule (2). We say depends-on because the rule (1) could not have made the inference that Mike is a father without rule (2) being present; for this conclusion it depended on the other rule. For the depends-on connection we only consider the cases where a rule works directly on the conclusion of another rule, not the cases where there are intermediary rules. The depends-on connections are calculated on the rules without consideration for the actual facts that are available: a depends-on connection exists independently of the facts. A depends-on connection between a rule A and a rule B states that there exists a set of facts such that an inference of rule A directly depends on rule B being present.

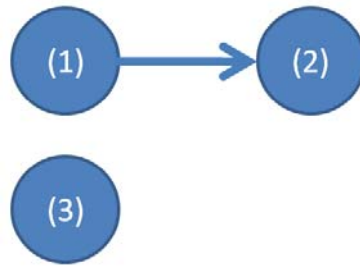


Figure 1: The static structure for the example rule base in section 3.

The rule (3) on the other hand, has no depends-on connection to any of the other rules - there exists no set of facts such that rule 3 could directly work on data inferred by one of the other two rules. Also note that depends on connections are not symmetric: rule (1) depends-on (2) but not the other way around.

A simple way to calculate an approximation, of the depends-on connection is the following: a depends-on connections exists from rule A to rule B, iff rule A has a body atom that can be unified with the head atom of B.

The static structure of the rule base then, is a directed graph $G_R = (R, A)$ with

- A set of vertices R , there is one vertice for each rule in the rule base.
- A set of ordered pairs A of vertices called arcs or arrows. An arc $e = (x, y)$ is considered to mean that the rule represented by the vertice x depends on the rule represented by y. Such an arc exists for each depends-on link between rules.

The rule graph for the simple rule base example in this section is shown in figure 1. It quickly conveys information such that $\{1,2\}$ and $\{3\}$ are independent parts of the rule base, that a deletion (2) will affect (1) and that someone interested in understanding rule (1) should also look at (2). For a rule base consisting only of three rules such a diagram is obviously not needed, but it can be seen how this kind of information can aid the creation, debugging, maintenance and reuse of larger rule bases.

4 Dynamic Structure

The static structure defined in the previous section represents the potential of rules to interact, without making assumptions/using information about the queries and facts a rule base is used with. The dynamic structure of a rule base complements this by combining the proofrees from all known queries to the rule

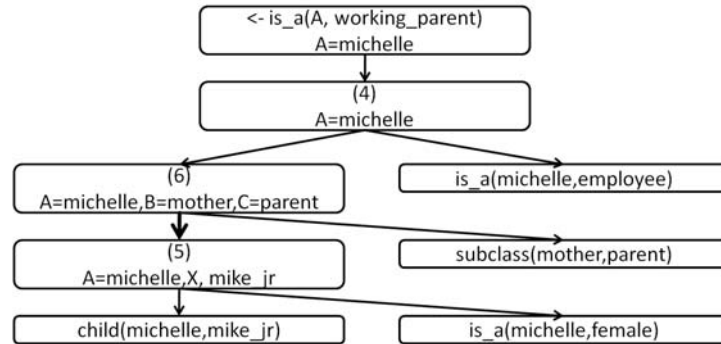


Figure 2: A proof tree for the example in section 4.

base into an overview picture. The resulting picture shows which rules and rule connections actually matter in the use of a rule base. When used to visualize the proof trees from the tests for a rule base it also gives a picture of test coverage.

Consider the following rule base. This rule base makes heavy use of a type hierarchy encoded in special predicates - a structure typical for rule bases created in F-logic [6].

$$is_a(A, working_parent) \leftarrow is_a(A, parent), is_a(A, employee) \quad (4)$$

$$is_a(A, mother) \leftarrow is_a(A, female), child(X, A) \quad (5)$$

$$is_a(A, C) \leftarrow is_a(A, B), subclass(B, C) \quad (6)$$

The static structure of this rule base is confusing (see figure 3) as almost everything depends on everything else; looking at the rule interactions in the actual use of the rule base can help here.

The structure that represents the interaction of rules and facts in the inferring of a result is called a proof tree. The root node of a proof tree is always the query, its variables bound to the result that has been returned. The children of this node are the rules that were directly needed to prove the query. The children of these rules are again the rules needed to prove them. Leafs of the proof tree are formed by the facts in the knowledge base. A bit more formally we can say that a proof tree is a directed graph $G_P = (V, A)$ with

- A set of vertices V . Each vertice can be understood as the application of one rule during the inference process. There is also (partial²) function $f_r : V \rightarrow R$ that gives the rule that was applied for a vertice in the proof tree.

² The proof tree also contains nodes that represent the “firing” of the query and the usage of facts

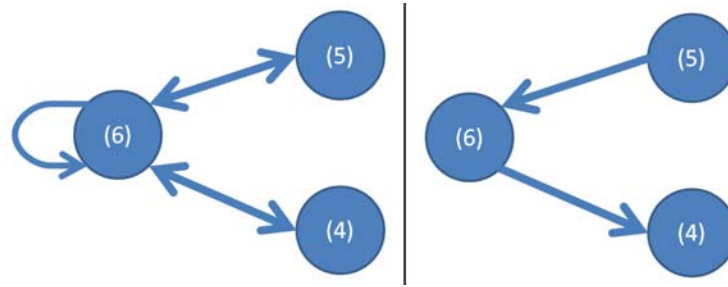


Figure 3: The static structure (to the left) and the dynamic structure (right) of the example in section 4. In this example the dynamic structure is very similar to a proof tree - because the usage data consists only of this one proof tree; this is not the case generally.

- A set of arcs A . An arc $e = (x, y)$ is considered to mean that the application of a rule represented by vertex x depended on the rule firing represented by y .

An example proof tree is created by the query $is_a(A, working_parent)$ posed against the rule base above and the following facts:

```
subclass(mother, parent), is_a(michelle, female)
is_a(michelle, employee), child(michelle, mike_jr)
```

A graphical representation of this proof tree is shown in figure 2. It shows the query on top with the variable A bound to michelle. This result of the query directly depended on the firing of rule (4) that in turn depended on rule (6) and the fact $is_a(michelle, employee)$ etc. Proof trees are well known tools to represent the inference process that lead to a particular result. Our novel approach however, combines all known proof trees for a rule base to create an overview representation of the entire rule base, independent of any particular query or result. To do this we define usage data U as the multiset of all known proof trees for a rule base: $U = \{G_{P,1}, G_{P,2}, \dots, G_{P,i}\}$ and the function f_r defined earlier. The proof trees could be created by tests of the rule base or be collected during the actual use of the system. We can then add a weight to each arc/rule in the rule graph (see section 3) based on how often this arc/rule appears in the proof trees in the usage data. Graphical representations of the rule graph can then hide unused depends-on links and highlight important rules and links.

The dynamic structure of the example in this section is shown in figure 4. Compared to the static structure it gives a much better picture of the interactions between these rules as they would happen for real world data.

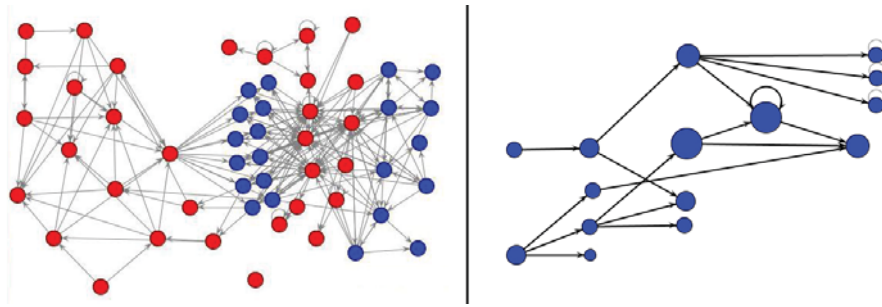


Figure 4: *Visualizations of a small working rule base. The left picture shows the unaltered static structure, the right picture shows the structure after rule nodes have been scaled based on their use, low level axioms have been hidden and some rules representing high level entities have been joined.*

Displaying the dynamic structure in this way is not the only possible way to use the usage data for a rule base. The following section will shortly discuss how it can also aid transformations of the rule graph to hide certain rules or to display the rule graph at different levels of abstraction.

5 Hiding and Joining Rules

The techniques presented so far work well enough for showing the structure of toy rule base. Even small real life rule bases, however, pose additional challenges: they are too large to show every rule, very general rules/axiom like rules (similar to rule (6) in the previous section) confuse the picture and sometimes graphical editors are used that create multiple rules for one high level entity known to the user. For instance figure 4 shows the static structure of a small but functional rule base - the structure is much too confusing to be of any use.

We selectively hide and join rules to deal with these problems. Figure 4 shows how this techniques work together. The right side shows the same rule base as the left side after usage data has been used to scale the rules nodes, axioms have been hidden and some rule sets (reflecting rules automatically created from one high level entity) have been joined. These transformations have uncovered the relatively simple, layered and ordered structure of this diagnostic rule base.

5.1 Hiding Rules

For certain very general, axiom like rules it makes sense to completely hide them from the view of the user. This is particular important for F-logic rule bases that

add a number of axioms to the rule base that are transparent to the user. Again the usage data is used to ensure that only relevant links are displayed.

Usage data in the form of proofrees can help to better deal with this problem because it contains *paths* through the rule base, not only arcs between two rules. When hiding rules this enables to retain only the actual paths through the hidden node(s). A detailed description of this can be found in the extended version of this paper available from <http://vzach.de/papers/visualizationLong.pdf>.

5.2 Joining Rules

There are a number of cases where it makes sense to join a number of rules for display purposes:

- Some engineering environments create more than one rule for a high level entity edited by the user. Here the visualization should only shows the high level entity without losing information.
- In a large rule base there is often some hierarchical structure on top of the rules (like rule packages or different files defining rules). The system can join all rules in each package and thereby give a high level view of the rule base, allowing to *zoom in* at one package, to show the rules in a package.
- In large rule bases without auxiliary structure clustering algorithms could be used to identify and the jointly display rule clusters, again allowing to expand the clusters.

Joining of rules is done by replacing a number of nodes in the rule graph with one new node. The detailed algorithm for this has to be omitted for brevity but can also be found in the extended version of this paper.

6 Related Work

To our best knowledge no approach that uses usage data in the form of proofrees for the visualization of rule bases exists. In fact we are not aware of any approach that uses the runtime rule interactions to create visualizations of entire rule bases.

There is a large number of approaches that visualize the inference process that lead to a single result (e.g. [2]) and some that show the static structure of rule bases [7, 4] but none that uses runtime rule interactions to create visualizations of entire rule bases. The approaches that show the static structure of rule bases also do not consider the challenges posed by large rule bases, high level editors and the hiding of rules.

Further visualizations of rules exist within the data mining community [1, 3, 5, 8, 9]. These approaches visualize association rules in order to facilitate the

analysis of these rules and the extraction of the most important information from them. They also face the problem of a large set of rules from which a few interesting ones need to be selected. These approaches, however, are mostly concerned with the problems posed by the statistical nature of association rules. The data used for the creation of the visualization, and its goal is very different.

7 Conclusion

A visualization of an entire rule base is an important tool for the creation, maintenance and reuse of rule bases. A rule base can be visualized based on its static structure or based on the actual rule interactions that happen during queries to the rule base. The joining and hiding of rules can be used to deal with the challenges posed by large rule bases and those that contain low level rules that should remain hidden from the user's view. The data about the actual rule interaction is necessary to join and hide rules while keeping the overall picture intact.

Acknowledgements

The author thanks Imen Borgi for doing a large part of the implementation reported on in this paper.

References

1. D.Bruzzese and P.Buono. Combining visual techniques for association rules exploration. In *Proceedings of the working conference on advanced visual interfaces*, pages 381 – 384, Gallipoli, Italy, 2004.
2. M. Eisenstadt and M. Brayshaw. The transparent prolog machine (tpm): an execution model and graphical debugger for logic programming. *Journal of Logic Programming*, 5(4):277–342, 1988.
3. Usama Fayyad, Georges Grinstein, and Andreas Wierse. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann, 2001.
4. C.A.McK. Grant. *Software Visualization in Prolog*. PhD thesis, University of Zurich, 2001.
5. Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 1st edition, 2000.
6. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.
7. J.W. Lewis. An effective graphics user interface for rules and inference mechanisms. In *Conference on Human Factors in Computing Systems Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 139 – 143, Boston, Massachusetts, United States, 1983.
8. M.Burch, S.Diehl, and P.Weiger. Visual data mining in software archives. In *Proceedings of the 2005 ACM symposium on Software visualization*, pages 37 – 46, 2005.
9. M.Kreuseler and H.Schumann. A flexible approach for visual data mining. In *IEEE Transactions on Visualization and Computer Graphics*, volume 8, pages 39 – 51, Germany, 2002.