Requirements for Successful Verification in Practice

S. Spreeuwenberg, R. Gerrits

LibRT
Postbus 90359
1006 BJ AMSTERDAM, The Netherlands
info@LibRT.com

Abstract

Many large scale companies use knowledge-based systems (KBS) to support their decision making processes. The quality of the decisions made depend on the quality of the underlying knowledge. It has been stated many times that verification techniques can be used to improve decision making and the quality of the knowledge rules in a knowledge based system. Furthermore, verification is seen as one of the key issues in system certification. After a short introduction to the current state of the art of knowledge verification this paper describes a verification technique used in a commercial development environment for knowledge intensive applications: VALENS. We will describe the experiences with VALENS in some recently finished experiments. Based on these results and an overview of the literature we will discuss the discrepancies between verification in practice and verification in theoretical / scientific situations. This leads us to an overview of the requirements for successful verification in practice. Obeying these requirements will increase the return on investment for knowledge based systems.

Introduction

Verification establishes the logical correctness of a KB i.e. the rules in a KB are checked to see if they are logical consistent, non-circular, complete, not redundant and not obsolete (the taxonomy of anomalies from A. Preece [4] is followed except that the term contradiction is used instead of ambivalance). Verification should not be confused with validation techniques, as stated by Gonzales[18] in an excellent overview of the controversy between scientists in defining these terms.
Validation tries to establish the correctness of a system with respect to its use in a particular domain and environment. In short the software community agrees that validation is interpreted as "building the right product", verification as "building the product right". It has been argued that the latter is a pre-requisite and sub-task of the former (Laurent[5]).
Until recently commercial development environments did not offer verification techniques despite the fact that the scientific world has stated the importance and offered solutions for this issue. In short they stated that verification techniques are important when:
KB components are embedded within safety critical or business critical applications (Ed P. Andert Jr, [1]).
When people without a background in system programming or system analysis define and maintain the knowledge in a KBS, the support of a V&V tool helps them to cope with the complexity. (Spreeuwenberg [2])
In all the main phases of the knowledge engineering life cycle, V&V is an important aspect when it comes to delivering a high quality KBS. (Anca Vermesan [3])
It has been concluded that "a uniform set of definitions should encourage developers to begin to think seriously about the need to perform formal V&V on their intelligent systems, and will also provide the foundation for researchers to develop tools that will be usable by others" (Gonzales and Barr, 2000).
In this article we will describe the implementation of verification techniques in a commercial development environment for knowledge based systems. The result of this work is implemented in a 'general' verification component called VALENS. This component is 'general' in the sense that you can integrate it in a development environment or case tool. Once we implemented this tool we have found some more reasons that make V&V a commodity in the mainstream software development industry. After discussion of the state of the art of verification research, the VALENS tool and our experiences with VALENS, we will transform these findings into requirements for applying verification techniques. Our final goal is to improve the quality of knowledge based systems and optimally support experts by formalizing their knowledge.

Overview of verification research

In the beginning of the '90s, different universities devoted much attention to V&V of KBS. There were some tools developed to verify rule bases of which Preece [6] has given an overview and comparison. An even more extensive overview comes from Plant [7] who lists 35 V&V tools built in the period 1985–1995. Most of the systems where developed at a university and it is hard to find out what the current status of those systems is.

Verification tools

The verification tools can be compared on a number of criteria. We have compared some of the widely known systems on the following criteria:

- The anomalies that are detected by the tool
- The language that is supported by the tool
- The focus and behavior of the tool in the analyses or development phase of a system

The first criterion is formed by the anomalies that are detected by the tool. Some tools do not detect anomalies in a chain of logic, for example the Rule Checker Program (RCP) [8] and CHECK [9]. Others like RCP, CHECK and EVA [10] do not detect missing rules and unused literals. VALENS is complete with respect to the anomalies defined by Preece [4].
Another criterion is the language that is supported by the tool. Most verification and validation systems, which verify a knowledge base, cope with a restricted language, for example first order predicate logic (Nouira and Fouet [12]) or formal specification language (van Harmelen [11]) as opposed to the rich language of a (fourth generation) programming environment. There are also tools which have their own internal language defined and which, manually or automatically, translate diverse languages to the internal language. EVA is an example of a system with its own internal language and provides a set of translation programs that translate the rule languages of some expert system tools (for example, ART, OPS5 and LES) to an internal canonical form, based on predicate calculus. PROLOGA [13] works the other way around, it allows a user to create and verify decision trees and then generate code in diverse programming languages (for example, Aion, Delphi and C++). COVER and VALENS work in the programming language they where developed with, which is respectively Prolog and Aion (see next paragraph).
The last criterion for comparison of verification tools is their respective behavior in the analysis and development phase of a system. The work of Nouira and Fouet [11] concentrates on the analysis phase of a system but results in a valid and executable knowledge base. The work of van Harmelen [12] also concentrates on the analysis phase and validates formal specification language. The idea is that the formal specification has to be translated to a programming language to get an executable program. VALENS can be used by a developer after or during construction of a KB or can be integrated in a tool that allows users to write their own business rules. The output of the tool is a document in which all invalid rules (combinations) detected are reported.

Recent developments

What happened with the described verification tools? Some of them still have a research status and are used to explore new research domains. For example, the COVER tool of Preece is evolved in the COVERAGE tool for verifying rule bases in a multi agent architecture [14]. And the PROLOGA tool [13] is extended with intertabular verification [15]. But perhaps the 'boost' for V&V tools failed to occur because the promise of KBS failed in commercial environments. Another factor might be that not only business environments but also university research is driven by 'hypes' like 'knowledge mining', 'knowledge management' and 'intelligent agents' which follow each other in such tempo that there is no time to pick the fruit of planted trees. A third reason can be found in the fact that the discrepancy between theory and practice is rather large in this field. In this article we will gather some evidence for this thesis.
The prospects for V&V tools is currently changing as the traditional "inference engine" market becomes a "business rule management" market. The business rules management approach to knowledge based systems hold that the business community should maintain the rules of the business instead of a programmer from an IT department. Verification is becoming more important in the light of this approach because the business user's often lack knowledge about logic to write valid rules. Recent evidence of this change is seen in the incorporation of verification techniques into different business rule management tools.
In the next section a description of the VALENS tool is given. The description focuses on the aspects of the tool that are important to understand the results for verification requirements in practice.

Application description

VALENS (VALid ENgineering Support) is a verification component that can be used by a developer after or during construction of a KB or can be integrated in a (case)tool that allows users to write their own business rules. The input of the verification component is a set of rules and the output of the verification component is a set of invalid rules (combinations). The input / output of the component is specified in XML.
The VALENS component is integrated in a tool for the Aion development environment.

V&V in AION

VALENS tool is an add-on (additional installable feature) of Aion9 (short: Aion). Aion is a widely used commercial development environment for KBS and intelligent components. Some characteristics are:

– The inference engine supports rule and decision table processing in a backward, forward chaining or recursive forward chaining mode.
– The programming language is object-oriented.
– Meta-programming features enable a programmer to obtain information about the state of the inference engine.

– The Callable Object Building System (COBS) feature allows one to automate all the functions a developer can use in Aion.

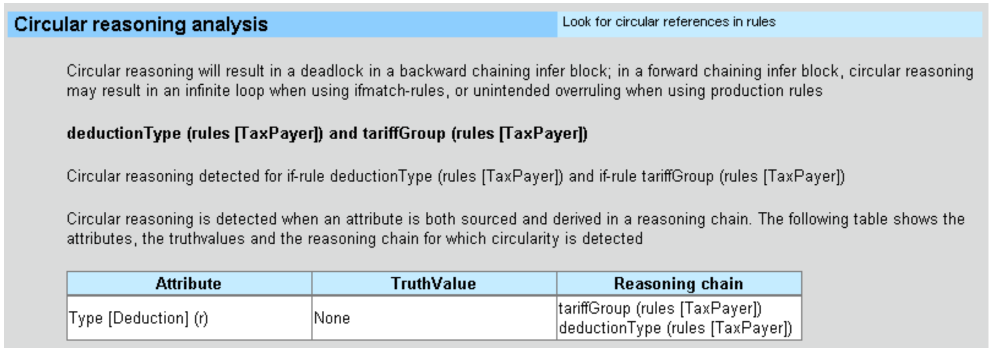
Several customers of Aion have expressed the need for verification techniques to be better able to maintain their large knowledge bases, which, in some case, contain thousands of rules.

The VALENS tool

The V&V application consists of three components: a user interface, the verification engine and a reporting component.

The user opens the KB and after starting VALENS selects the rule sets within that KB that need to be verified. When there are potential 'invalid rules' detected during the verification process, the KB is started in a forward chaining mode to test the thesis. We than capture the results of the inference engine for analysing whether a thesis is satisfied, and to catch the chain of logic that has caused a thesis to be satisfied.

Invalid rules are reported in a HTML document. Each fault is classified and explained as shown in figure 2, which shows the result-report for circular rules.


Circular reasoning analysis — Look for circular references in rules

Circular reasoning will result in a deadlock in a backward chaining infer block; in a forward chaining infer block, circular reasoning may result in an infinite loop when using ifmatch-rules, or unintended overruling when using production rules

deductionType (rules [TaxPayer]) and tariffGroup (rules [TaxPayer])

Circular reasoning detected for if-rule deductionType (rules [TaxPayer]) and if-rule tariffGroup (rules [TaxPayer])

Circular reasoning is detected when an attribute is both sourced and derived in a reasoning chain. The following table shows the attributes, the truthvalues and the reasoning chain for which circularity is detected

Attribute	TruthValue	Reasoning chain
Type [Deduction] (r)	None	tariffGroup (rules [TaxPayer]) deductionType (rules [TaxPayer])

Figure 2. Result report of VALENS

The result report shows a general explanation of the anomaly and the conflict that is detected. A conflict is defined in [16] as a minimal set of rules, eventually associated to an input fact set, that is a sufficient condition to prove an anomaly. Besides this information the report shows also the rule chain (the set of rules that caused the anomaly to occur) when applicable for the anomaly.

Verification algorithm

The verification algorithm that VALENS uses performs three main steps:

1. Construction of meta model

In this step all rule constructs, necessary to reason about the rules in the KB are instantiated. This step is performed on a "when needed" basis to reduce performance overhead.

2. Select potential anomalies

Potential anomalies are selected with the use of heuristics. These heuristics where designed as meta rules but are implemented as procedures due to performance considerations.

3. Proof anomalies

The theses (potentially invalid rules) are proved by running the rules to be tested in a forward chaining mode, while providing them with the right truth-values (input). We call this process proof-by-processing.

Benefits of the proof-by-processing algorithm used in VALENS compared to formal methods is that we are able to cope with procedural logic (function calls) in the rules and with rules in an object oriented environment.

For a more detailed description of the proof-by-processing algorithm used in VALENS to detect anomalies the user is referred to Spreeuwenberg [2].

Experience with VALENS

In practice, VALENS proves not only to ensure a valid (i.e. verified) knowledge base, but also the validity of its documented functional specifications, along with good communication with the domain experts and good use of knowledge engineering principles.

Experience with insurance company

Postbank Nederland BV became interested in the promise of a V&V tool for their Aion assessment KB. In a two months pilot project VALENS was evaluated in a real business situation.

We got the first version of the customer's KB to verify when the developing team of the Postbank had finished the rule base and the testing phase was at hand. Though VALENS can be applied earlier in the application development lifecycle, it was perfect timing: there would be a parallel verification and testing phase so the results of both processes could be compared.

VALENS did not detect any real errors in the KB. Though this might look disappointing, the testing phase neither did reveal any error that could have been detected by verification. VALENS did find many redundant and obsolete constructs in the KB. Some of these constructs were intentional, others were not, but everyone was impressed with the fact that VALENS was able to highlight these 'points of interest'.

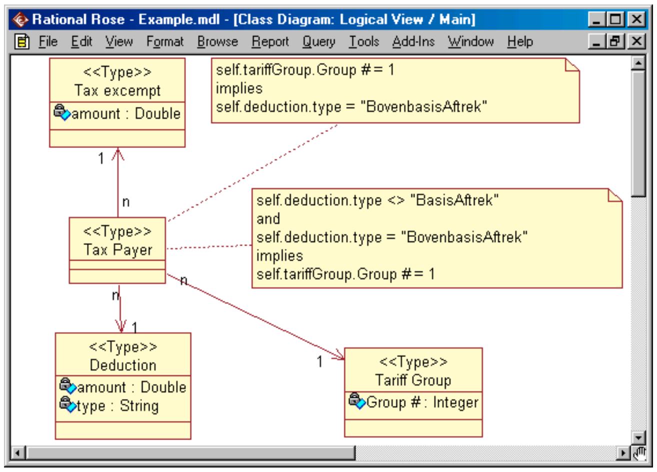
VALENS proved to be of good use in maintaining the integrity of the functional specifications of the KB and the realized (and revised!) KB.

Experience with legislation

VALENS is used to verify legal knowledge modeled with the POWER method. This method is developed as part of the POWER research program that aims to develop a method and supporting tools for the whole chain of processes from legislation drafting to executing the law by government employees. The goal of the POWER program is to improve legislation quality by the use of formal methods and verification techniques. The flexible nature of the VALENS verification component, the completeness and accuracy of the verification algorithms, and the possibilities for integration of VALENS in a modeling workbench has resulted in the decision to integrate VALENS in the POWER program [17]

We had the opportunity to model the (concept version of the) new Dutch income tax law. Since almost nothing of the old law on income remains intact, we were asked to look for anomalies.

The power method translates legislation into UML/OCL models.



```
self.tariffGroup.Group # = 1
implies
self.deduction.type = "BovenbasisAftrek"
```

```
self.deduction.type <> "BasisAftrek"
and
self.deduction.type = "BovenbasisAftrek"
implies
self.tariffGroup.Group # = 1
```

Figure 5 POWER model

The above POWER model is a translation of two articles of the dutch income tax law. The first article specifies the deduction type based on the tariff group a tax payer is assigned. The second article specifies how a tariff group is assigned to a tax payer.

The resulting OCL statements are generated into a rule-based environment as follows:

```
rule deductionType
ifrule current._tariffGroup.GroupNr = 1
then
  current._deduction.type = "BovenBasisAftrek"
end

rule tariffGroup
ifrule current._deduction.type <> "BasisAftrek"
and current._deduction.type = "BovenBasisAftrek"
then
  current._tariffGroup.GroupNr = 1
end
```

VALENS will detect circularity and present this in an HTML report from which an extract is shown in Figure 2.

We found more then 150 anomalies that were not detected by the knowledge groups before. The anomalies were reported to the drafters and repaired. The effectiveness of the feedback process depends heavily on representation. Therefore we have conducted some research on law-representations that promote the communication between legislative- and IT-experts (by means of a cognitive ergonomic study).

Other experiences

Experiences with knowledge bases in the US have forced us to make guidelines for the creation of 'verifiable' knowledge bases. These guidelines are in fact well known and standard knowledge engineering principles like:

– Separate user interface logic from business rules
– Separate control logic from business rules
– Separate data retrieval and data availability from business rules

All these guidelines assure that the business rules, to be verified, are specified in a declarative manner.

In a rich, object oriented, 3th or 4th generation programming environment you can easily violate the above principles if you are not aware of them. Although VALENS is able to cope with a limited amount of procedural logic, violating these principles not only undermines the maintainability of the application but also undermines the verifiability of the application.

In general VALENS is able to cope with functions in rules when the function can be replaced by its contents without violating the Aion rule syntax.

Example:

```
If theApplicant.GetAge > 25
Then theApplication.SetApproved(true)
```

In this example two functions are used. If the function calls are replaced by their contents the rule could look like this:

```
If currentDate - theApplicant.BirthDate > 25
Then theApplication.approved = true
```

If the function GetAge is specified using procedural control statements like "loop" or "while", the rule cannot be verified because the Aion rule syntax does not allow these statements in rules.

Requirements for successful verification

Given our experiences with verification in business situations we have concluded that there are some requirements for successfully applying verification technology in practice.

Programming languages

Knowledge based systems that are used in business environments are written in modern programming languages that support, in general, a richer language than propositional- and even predicate logic. A verification technology should, therefore, be able to cope with the use of functions in rules, the use of relations between objects by means of pointers and inheritance.

Declarative programming

Unfortunately the third and fourth generation programming languages that are used in business environments enable a programmer to mix the declarative manner of rule based programming with procedural code. Verification technology can only be applied to declarative specifications, which also improve the maintainability of the system. Therefore we need to state the requirement that some knowledge engineering guidelines have been followed in the construction phase of the knowledge base.

Communication of the results

Successful knowledge representation requires the communication of verification results in the terminology of, and in an understandable format to, the domain expert. So far the communication of the results have always been in the same format as the knowledge representation format. This is not a good representation when the people who need to solve the anomalies are domain experts (and not programmers or logicians).
In the case of the POWER method we can use the tractability features of this methodology to communicate the results in terms of the original law texts. This helps but is not sufficient; we also have to find a (visual) representation that reduces the complexity when an anomaly only occurs in a reasoning chain. Defining this representation requires some more research.

Conclusion

Until now knowledge verification has been a scientific research subject that was rarely practiced on real life knowledge based applications. When you start using and integrating verification techniques in a commercial development environment for knowledge based systems you experience that you need to meet the following three requirements to be successful:
The verification technique can cope with language constructions common in 3th and 4th generation programming languages.
The knowledge bases have been constructed without violation of some basic knowledge engineering principles.
The results of the verification process are communicated in terms of the domain so that business experts can repair the anomalies in the source of the knowledge.
We feel that these requirements are not only applicable for the development of verification systems but also for the development of validation systems. We are planning to extend our verification technology with validation technology and we think the success of this extension is guaranteed if we obey the requirements outlined in this article.

References

[1] Ed P. Andert Jr., 1992, Automated Knowledge Base Validation, AAAI Workshop on Verification and Validation of Expert Systems (July 1992)

[2] S. Spreeuwenberg, R. Gerrits, 1999, A Knowledge Based Tool to Validate and Verify an Aion Knowledge Base, Validation and Verification of Knowledge Based Systems, Theory, Tools and Practice, 67 – 78, ISBN 0-7923-8645-0

[3] A. Vermesan, Jarle Sjøvaag, Per Martinsen and Keith Bell, 1999,Verification and Validation in Support for Software Certification Methods, Validation and Verification of Knowledge Based Systems, Theory, Tools and Practice, 67 – 78, ISBN 0-7923-8645-0

[4] A. Preece, Shingal, 1994, Foundation and Application of Knowledge Base Verification, International Journal of Intelligent Systems, 9, 683 – 701

[5] J.P Laurent, 1992, Proposals for a valid terminology in KBS Validation. ECAI 92. John Wiley & Sons, Ltd., 1992

[6] A. Preece, 1991, Methods for Verifying Expert System Knowledge Bases.

[7] Robert T. Plant, 1995, Tools for Validation & Verification of Knowledge-Based Systems 1985 – 1995 References, *Internet Source*

[8] M. Suwa, A.C. Scott, E.H. Shortliffe, 1982, An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System, AI Magazine, Vol. 3, Nr. 4

[9] W.A. Perkins, T.J. Laffey, D. Pecora, T.Nguyen, 1989, Knowledge Base Verification, Topics in Expert System Design, 353 – 376

[10] C.L. Chang, J.B. Combs, R.A. Stacowits, 1990, A Report on the Expert Systems Validation Associate (EVA), Expert Systems with Applications, Vol. 1, Nr. 3, 217 – 230

[11] F.V.Harmelen, 1995, Structure Preserving Specification Languages for Knowledge Based Systems, International Journal of Human Computer Studies, Vol. 44, 187-212

[12] Rym Nouira, Jean-Marc Fouet, 1996, A Knowledge Based Tool for the Incremental Construction, Validation and Refinement of Large Knowledge Bases, Workshop Proceedings ECAI96

[13] J. Vanthienen, 1991, Knowledge Acquisition and Validation Using a Decision Table Engineering Workbench, World Congress of Expert Systems, 1861 – 1868

[14] N. Lamb, A. Preece, Downloaded: 01-05-2000, Verification of Multi-Agent Knowledge-Based Systems, *Internet Source*

[15] J. Vanthienen, C. Mues, G. Wets, 1997, Inter-Tabular Verification in an Interactive Environment, Proceedings Eurovav 97, 155 – 165

[16] N. den Haan, Automated Legal Reasoning, University of Amsterdam, Amsterdam, 1996 (diss)

[17] S. Spreeuwenberg, T. v. Engers, R. Gerrits, The role of verification of legal knowledge in improving the quality of legal decision-making, JURIX 2001.

[18] A.J. Gonzales, V. Barr, Validation and verification of intelligent systems, Journal of Experimental and Theoretical AI, Oct. 2000.