Errors And Anomalies In Rule Knowledge Bases

Roman Siminski, Alicja Wakulicz-Deja
Silesian University, Institute of Computer Science
41-200 Sosnowiec, 3 Zeromskiego St., Phone: +48-32-2 918 945 ext. 38
email: {siminski | wakulicz}@ us.edu.pl

ABSTRACT: Expert systems are being developed and used in a wide variety of disciplines throughout the word. Much attention has been paid to check the reliability of expert systems. Verification of knowledge bases has emerged as a significant problem in the development of expert systems. Although the basic verification concepts are shared by software engineering and knowledge engineering, verification methods of conventional software are not directly applicable to expert systems and the new, specific methods of verification are required. In this work we present background on knowledge bases verification, main problems and goals. We consider basic issues of verification and validation of knowledge bases, we explain basic types of errors in knowledge bases like redundancy, ambivalence, circularity. Next we describe the best known methods and tools for knowledge bases verification. Finally, we argue that knowledge bases verification can not be delayed until the final knowledge base realization and we suppose that verification should be performed incrementally and should be included into the development process. In this paper we introduce the main assumptions of verification approach implemented in the kbMaker system and we briefly describe the current state of work on this assistant tool for incremental knowledge base building and verification.

KEYWORDS: expert systems, knowledge bases, verification, validation.

1 INTRODUCTION

Expert systems are being developed and used in a wide variety of disciplines throughout the word. Much attention has been paid to check the reliability of expert systems, because safety is usually the main goal in some application (e.g. medicine or nuclear power-station). Reliability of the expert systems has become a key factor in knowledge engineering. Consequently issues concerning validation and verification of knowledge bases become more and more important. Verification and validation of knowledge bases has emerged as a significant problem in development of knowledge-based systems. Quality assurance of knowledge bases is more and more important as the field of expert systems evolves and matures.

Building complex knowledge bases requires encoding a large amount of domain knowledge. After acquiring this knowledge from domain experts, much of the effort in building a knowledge bases goes into verifying that the knowledge is encoded correctly. In general, verification and validation of any software system consist in checking that this system is fully operational and performs satisfactorily its intended function. Therefore, although the basic validation concepts are common for knowledge and software engineering. But we encounter difficulties if we try to apply classical definitions of verification and validation (from software engineering) to knowledge engineering. The tasks performed by the expert systems usually can not be correctly and completely specified. These tasks are usually ill-structured and no efficient algorithmic approach is known from them.
Therefore, although the basic validation concepts are common for expert systems and conventional software, validation concepts and methods of conventional software are not directly applicable.

In the next section of this work we present background on knowledge bases verification, main problems and goals. In the third section of this work we explain basic types of errors and anomalies in rule knowledge bases like redundancy, ambivalence, circularity. Next we describe the best known methods and tools for knowledge bases verification. In the last section we present current state of work on the prototype of an assistant tool for knowledge base building, validation and maintenance.

2 VERIFICATION OF KNOWLEDGE BASES

Verification, validation and testing are terms that have been used for several year in classical software engineering. The following definitions are given in (Adrin & Branstand & Cherniavsky 1982) for the field of software quality assurance:

Validation is the determination of the correctness of the final program or software produced from a development project with respect to the user needs and requirements. Validation is usually accomplished be verifying each stage of the software development life cycle.

Verification is the demonstration of consistency, completeness and correctness of the software at each stage and between each stage of the development life cycle.

Testing is the examination of the behavior of the program by executing the program on simple simple data sets.

A program is verified against its specification. Formal specifications are then the vital to validation and verification id classical software. Sometimes knowledge engineering is compared to software engineering, but here many differences can be founded. Perhaps the crucial characteristic of knowledge engineering that distinguish it in an essential way from software engineering is the impossibility to obtain a correct and complete formal specification of expert system.

Expert systems are problem solvers for specialized domains of competence in which effective problem solving normally requires human experts. Expert systems are often intended for ill-defined problems and they need to work with incomplete or uncertain knowledge. Expert systems have proven to be an effective technology for solving ill-defined problems but from a software engineering point of view - when the problem is ill-defined then the user requirements are ill-defined, it leads to difficulties in determining whether the system meets its requirements. In some situation expert systems provides only the hope for a good solution.

Therefore, one of the main impediments to successful verification of expert system is the nature of expert systems themself. The second impediment are implementation methods and tools. ES system are usually implemented using declarative languages or specialized expert system shells. Rule-based languages are perhaps the most popular ones. ES behavior arises from cooperative interaction of rules in knowledge base interpreted by inference engine. For knowledge bases with hundreds or thousands of the rules numbers of possible inference paths is very high. In that cases knowledge engineer can not be totally aware that all possible rules interaction are legal and provides expected results.

Therefore, although the basic verification concepts are shared by software engineering and knowledge engineering, verification methods of conventional software are not directly applicable to ES and the new, specific methods of verification are required.

There are some definitions of those concepts. Some of those definitions differs only slightly, others seem totally contradictory. In (Hoppe & Meseguer 1993) we can found a proposal of common terminology for knowledge based systems. In our work we concern on knowledge base verification. Verification of knowledge base is the process of checking the rule bases for four types of anomaly: redundancy, ambivalence, circularity, and deficiency. These terms are defined in next chapter. We have take a specific formulation of anomalies for rule bases from (Preece & Batarekh & Shinghal, 1990) as the base of our work.

3 ERRORS AND ANOMALIES

In the title of our paper we make a distinction between errors and anomalies that may occur in knowledge bases. Errors will require further maintenance action to remove them. Anomalies may not necessarily represent problem in themselves, but rather symptoms of possible errors. Consequently a distinction if the errors and anomalies seems to be very delicate.

3.1 AN EXAMPLE KNOWLEDGE BASE

In following section we present the anomalies that can occur in rule knowledge bases.. To illustrate the presentation an example knowledge base is given. The knowledge base is used to choose aa implementation environment for hypothetical software application. We distinguish the following environment: Delphi, C++ Builder, Optima++, Visual Basic. An example rule base seems to be somewhat artificial and trivial, the numbers of errors displayed would never have occurred if, for example, some supported tools have been followed. We use knowledge representation language Sphinx. Sphinx is domain independent and is suitable for solving problems

which belongs to classes of computer decision support, classification, diagnostics and data analysis/interpretation (Michalik & Siminski 1998).

An example knowledge base:

```
knowledge base environments

facets
  ask     yes

  implementationEnvironment :
     val oneof
     { "Delphi", "C++ Builder", "Optima++", "Visual Basic" };

  applicationType :
     val oneof
     { "standard data base application", "system low-level application" };

  environmentType :
     val oneof
     { "RAD", "compiler-based" };

  preferredLanguage :
     val oneof
     { "Pascal", "C/C++", "Basic" };

  destinationSystem :
     val oneof
     { "Windows family", "Unix family" };

  implementationTime :
     val oneof
     { "short", "long" };

  userInterface :
     val oneof
     { "windowBased", "consoleBased" };

  professionalSkills :
     val oneof
     { "poor", "good" };

end; // facets

rules

   0001: implementationEnvironment = "Delphi" if
             applicationType = "standard data base application"
             & environmentType = "RAD"
             & preferredLanguage = "Pascal";

   0002: environmentType = "RAD" if
             implementationTime = "short"
             & userInterface = "windowBased";

   0003: environmentType = "compiler-based" if
             implementationTime = "long"
             & userInterface = "windowBased";

   0004: environmentType = "compiler-based" if
             implementationTime = "long"
             & userInterface = "consoleBased";

   0005: implementationEnvironment = "Delphi" if
             applicationType = "standard data base application"
             & preferredLanguage = "Pascal"
             & implementationTime = "short"
             & userInterface = "windowBased";

   0006: implementationEnvironment = "C++ Builder" if
             applicationType = "standard data base application"
             & environmentType = "RAD"
             & preferredLanguage = "C/C++";

   0007: implementationEnvironment = "Optima++" if
             applicationType = "standard data base application"
             & environmentType = "RAD"
             & preferredLanguage = "C/C++";
```

```
   0008: implementationEnvironment = "Visual Basic" if
             applicationType = "standard data base application"
             & environmentType = "RAD"
             & preferredLanguage = "Basic";

   0009: environmentType = "RAD" if
             implementationEnvironment = "Visual Basic"
             & implementationTime = "short";

   0010: implementationEnvironment = "Visual Basic" if
             applicationType = "standard data base application"
             & environmentType = "RAD"
             & preferredLanguage = "Basic";

   0011: not implementationEnvironment = "Optima++" if
             applicationType = "standard data base application"
             & environmentType = "RAD"
             & preferredLanguage = "C/C++";

   0012: environmentType = "RAD" if
             implementationTime = "short"
             & userInterface = "windowBased"
             & professionalSkills = "poor";

   0013: environmentType = "RAD" if
             implementationTime = "short"
             environmentType = "RAD";

end; // rules

control
  goal( ”environmentType = X” ); // Runs the backward chaining inference engine
end;

end; // knowledge base
```

3.2 REDUNDANCY

An expression in a knowledge base is redundant iff for every permissible set of input data, the final hypotheses inferred are the same, regardless of presence or absence of the expression. The expression is typically a literal or rule. The most general case of this is a redundant rule, for example the rule 8 and 10 in our example are duplicated. The rule 5 is redundant because it is a logical consequence of rules 1 and 2. Another case of redundancy presents rules 2 and 12. The rule 12 is subsumed by the rule 2 (or that the second rule subsumes the first). Since the rule 2 is more general than the rule 12, the rule 12 is redundant. In an example knowledge base exists unused attributes and its values, they are usually symptoms of missing rules (see 3.5).

3.3 AMBIVALENCE

A knowledge base is *ambivalent* iff for some permissible set of input data, we can infer an impermissible set hypotheses. For example, rules 7 and 11 are ambivalent because this same set of true conditions lead to contradictory conclusions. If we consider rules 6, 7 we can see that that rules are inconsistent - this same set of true conditions lead to different conclusions.

3.4 CIRCULARITY

Circularity presents an urgent problem in backward chaining systems. A knowledge base has *circularity* iff it contains some set of rules such that a loop could occur when the rules are fired. We can distinguish between direct cycle, where the rule calls itself like rule 13 and indirect cycles, where several rules are invoked like in case of rules 8 and 9.

3.5 DEFICIENCY

A knowledge base is *deficient* iff there exist a permissible set of input data which a hypothesis that should be inferred is not inferred. Deficiency is typically due to missing knowledge. For example, consider the following set of fact:

```
implementationTime = "short"
userInterface = "consoleBased"
applicationType = "standard data base application"
preferredLanguage = "Visual Basic"
```

there are not any rules for this situation and no useful conclusion can be inferred. The unused attributes and values are usually a symptoms of deficiency, and therefore some knowledge is missing. In our example `destinationSystem` as an unused attribute, value `"system low-level application"` is unused value of attribute `applicationType`. We can refer to deficiency as a *incompleteness.*

4 THE BASIC VERIFICATION METHODS AND TOOLS

Many methods have been developed in order to obtain right knowledge bases, a variety of tools are described in references. This tools may be divided into two categories (Nouira & Fouet 1996) – domain independent tools, which try to detect anomalies which consist of an abuse or unusual of the knowledge representation scheme, and domain dependent tools use meta–knowledge from domain to verify of knowledge base. These range from static tools that check for structural errors, to incremental verifiers that include support for multilevel expert systems with uncertainty. In the following, we briefly describe the most relevant verification methods and tools. See also (Coenen & Bench–Capon 1993) and (Nazareth 1996) for further details.

4.1 DECISION TABLE APPROACH

Decision tables, widely used in conventional software specification to determine the effect of conditional statements, has been used to testing of set of rules for errors and anomalies. A number of variations on decision table approach have also be implemented. In next chapters we discuss some of the systems based on decision table approach.

The first well known verification tool was Rule Checker Program (RCP) (Suwa et al. 1982). It could be considered as the first verifier referenced in the literature. The RCP was developed as the part of the Stanford group (Buchanan 1984) perceived a need for a tool to identify incorrect or missing knowledge in knowledge bases. The RCP was developed for ONCOCIN expert system but the verification techniques developed in this tool were largely domain–independent.

The Expert System Checker (ESC) (Cragun & Steudel 1987) was developed as the refinement of the RCP. The ESC was designed to check rule bases according to the syntax of Level 5 Shell. ESC is a decision table based verification program which improving the performance of checking in the tables.

The CHECK (Nguyen et al. 1985) is an automated rule verifier that is also based on ideas about decision tables. The CHECK was developed at Lockheed Corporation, as a verification facility for their expert system shell, the Lockheed Expert System (LES). The Check was developed from RCP system, expanding and redefining the range of anomalies checked for. LES requires the knowledge engineer to partition rules into sets, thus CHECK operates on a partitioned rule base. CHECK detect most anomalies by building three cache tables from rule base. Those tables can be used to detect anomalies such as redundancy, conflict, circular inference chains (using acyclic detection graph algorithm). As witch RCP, the fundamental limitations of CHECK are the narrow range of anomalies checked for, also the use of CHECK is limited to systems build using LES shell. The CHECK system is a precursor of the EVA and ART. Rule Checker which was tailored the rule language of the widely–used Automated Reasoning Tool (ART) instead of the LES.

4.2 PETRI-NET BASED APPROACH

Petri-nets are abstract formal representation of information flow often used to illustrate data flow in conventional systems. The concepts of Petri-nets has also been applied to verification and validation of knowledge bases.

Knowledge base is usually modeled as a Petri-net with literals represented by places, and rules represented as transitions. The Petri–net is then transformed to an incidence matrix using transitions as rows and places as columns. Detection is performed in incremental fashion, by multiplying the incidence matrix with a vector representation of the new rules. Completeness is verified by multiplying the incidence matrix with a condition test matrix (Agarwal&Tanniru 1992), (Nazareth 1993), (Pipard 1988).

4.3 GRAPH-BASED APPROACH

This approach advocated the use graphs as a model of rules base or inference paths. Early graph-based approach advocated the use on inference graphs, with rules as individual nodes, and shared literals as the basis for edges between nodes. Rules were represented in predicate calculus without uncertainty. Detection was performed through an exhaustive traversal of the inference graphs. This approach was employed in the verifier ARC (Nguyen 1987).

Digraph–based approach was devised for detecting structural errors in a rule–based system. It models the rule set using directed graphs, with literals and rules as nodes, and associations between them as arcs. In addition, the use of domain knowledge to close semantic gaps is introduced. Verification extends to errors involving redundancy, conflict, circularity, deadends and unreachable goals. A formal model is proposed, and a set of propositions involving these errors are presented and rigorously proved. Detection is performed through reachability analysis of paths in the digraph. The actual implementation employs a matrix notation, and inspection of a trace matrix. The use of a matrix representation facilitates portability to multiple platforms (Nazareth 1991).

Algebraic graph transformation approach concentrates primarily on redundancy and subsumption of structural knowledge in a rule base. It employs hypergraphs to achieve this, representing antecedent and consequent proposition symbols as nodes and rules as hyperarcs. This approach involves an extremely precise and formal treatment of the subject, including the development of a formal notation for hypergraphs, and the use of a graph grammar and language. Propositions for knowledge verification are formulated and rigorously proved. The exercise is also extended to rule sets involving uncertainty. The notion of semantic knowledge to aid verification is also discussed, as well as a set of possible normal forms for knowledge bases (Valentine 1994).

4.4 OTHER METHODS AND TOOLS

The EVA (Stachowitz et al 1987b) and COVER (Preece 1990) (Preece & Batarekh &. Shinghal 1990) are the systems which perform verification of rule bases translated into the generic representation before – this makes those systems independent of any specific shell. Those systems are don't use any specific methods, a set of verification methods are used.

Many practitioners consider that knowledge bases can only be checked at the clausal level – rule base can be translated into clausal form using logical equivalencies. One of the best known examples of a knowledge base verification system that utilizes normal form is MELODIA (Charles 1991) (Coenen & Bench–Capon 1993).

Another approach utilizes the KB–Reducer program which was inspired by the assumption–based truth maintenance system (ATMS) developed by de Kleer (de Kleer 1986), and finds the set of assumptions which must be hold for each knowledge base hypothesis to be true.

5 SUMMARIZATION OF VERIFICATION METHODS AND TOOLS

The vast majority of current work in the verification of rule knowledge bases concerns the verification on some properties of the base separately considered, most of the validation approaches assume to work on fully implemented base. Verification is rather at the end of the knowledge base development process, where the consequence of any modification are not foreseeable.

It is difficult to find real problems that have an available set of knowledge that can simply be implemented correctly at once without need for further review and modification. The knowledge engineering process iterates between development knowledge base after consultation with the experts and reviewing the existing knowledge base and its performance. Proponents of producing a knowledge base following the traditional top-down life cycle are rare. As an alternative to the traditional waterfall model, the spiral life cycle model allows an iteration of requirements, design knowledge base definition, prototype and test (Bohem 1988).

The incremental, evolutionary or experimental life cycle paradigm is based on ideas concerned with rapid prototyping as implemented for the development of some traditional software systems (Coenen & Bench–Capon 1993). Several authors have made proposals to include verification during early stages of knowledge base development, but this does not appear to have become a common practice.

6 THE kbMaker SYSTEM

We argue that knowledge bases verification can not be delayed until the final base realization. There is to high risk that errors will be found to late which may be very expensive to correct. We suppose that verification should be included into the development process and that different types of verification should take place in different phases of development process. Verification has to be performed incrementally and can not be delayed until knowledge base will complete.

Verification should incorporates analysis of static and dynamic properties of knowledge base. Main goal of static analysis is demonstrating that a rule base is free from typical anomalies as redundancy, contradiction, subsumption etc. Dynamic analysis is the testing a knowledge base through its execution using run time subsystem that utilizes some inference engine algorithms and techniques.

This strategy is the main approach of kbMaker project. The kbMaker is the part of the CAKE project (AITECH Katowice). CAKE is the name of the project and the software tool for interactive, incremental construction, validation and refinement of the PC-Shell's (Michalik & Siminski 1998) knowledge bases.

The kbMaker is implemented in C++ and works under the control of Windows 3.x/95. In current version, system provides only a subset of syntactic and structural validation functions. System detects:

1. redundant and subsumed rules,
2. contradictory and ambivalent rules,
3. unused attributes and its values,
4. missing rules for decision attributes (attributes for a goal statement).

7 CONCLUSIONS

We have briefly present a main goal of kbMaker system that allows to create and to validate progressively knowledge contained in the knowledge bases of the PC–Shell system. Present version of the kbMaker system is only a prototype, in current version, system provides a static verification methods. Further research and development on this topic will first be aimed at providing also analysis of dynamic properties of knowledge base and development „on-the-fly" verification techniques.

ACKNOWLEDGEMENTS

This work was supported by the Committee for Scientific Research, Warsaw, Poland, Grant No. 8T11C 005 12 and AITECH, Artificial Intelligence Laboratory, Katowice, Poland.

REFERENCES

Adrion W.R., Branstad M.A., Cherniavsky J.C., 1982, „Validation, verification and testing of computer software", ACM Computing Surveys, June, 14(2) pp. 159-192.

Agarwal R., Tanniru M., 1992, „A petri-net based approach for verifying the integrity of production systems", International Journal of Man-Machine Studies, 36 (3), pp. 447–468.

Bahill A.T., 1991, „Verifing and Validating Person Computer-Based Expert Systems", Prentice Hall, Englewood Cliff, NJ, 1991.

Bohem B.W., 1988, „A Spiral Model of Software development and Enhacement", IEEE Computer, Vol 21, No 5, pp. 61-72

Buchanan B.G., Schortliffe E.H., 1984, „Human engineering of medical expert systems", In B.G. Buchanan and E.H. Shortliffe, editors, Rule–Based Expert Systems: the Mycin Experiment of the Stanford Heuristic Programming Project, chapter 32, pp. 599–612, Addison Wesley, Reading MA.

Charles E., 1991, „Checking Knowledge Bases for Inconsistencies and Other Anomalies", Workshop notes from the Ninth National Conference in Artificial Intelligence, AAAI–91, Knowledge–Based system Verification, Validation and Testing, Anaheim CA.

Coenen F. Bench–Capon T., 1993, „Maintenance of Knowledge–Based Systems”, Academic Press Inc. San Diego.

Cragen B.J. Steudel H.J., 1987, „A Decision–Table–based Processor for Checking Completeness and consistency in Rule–Based Expert Systems”, International Journal of Man–Machines Studies, vol. 26, pp. 633–648.

de Kleer J., 1986, „An assumption based TMS”, Artificial Intelligence (Netherlands), 28(2), pp. 127–162.

Ginsberg A. 1988, „Knowledge–base reduction: A New Approach to Checking Knowledge Bases for Inconsistency and Redundancy”, Proceedings of 7th National Conference on Artificial Intelligence AAAI 88, vol. 2, pp. 585–589.

Ginsberg A., 1987,. „A New Approach to Checking Knowledge Bases for Inconsistency and Redundancy”, Proceedings of 3rd Annual Conference Expert Systems in Government, IEEE, pp. 102–111.

Harmelen F. 1996, „Applying rule-base anomalies to KADS inference structures”, Available on http://sunsite.ust.hk/dblp/db/indices/a-tree/Harmelen:Frank_van.html.

Hoppe T., Meseguer P., 1993, „VVT Terminology: A Proposal”, IEEE Expert, pp. 48–55, June.

Mengshoel O.J., Delab S., 1993, „Knowledge Validation: Principles and Practice”, IEEE EXPERT, pp. 62–68, June.

Michalik K., Siminski R., 1998, „The Hybrid Architecture Of The AI Software Package Sphinx”, Proceedings of Colloquia in Artificial Intelligence CAI’98, 28–30.10.98 £ódŸ, Poland, pp. 210-219.

Nazareth D.L, 1996, „Knowledge-Based Systems Verification Validation and Testing Web Page”, http://www.uwm.edu/People/derek/kbsvvt.

Nazareth, D.L. Kennedy, M.H., 1991, „Verification of Rule-Based Knowledge using Directed Graphs”, Knowledge Acquisition, 3 (3), pp. 339–360.

Nazareth, D.L., 1993, „Investigating the applicability of petri nets for rule-based system verification”, IEEE Transactions on Knowledge and Data Engineering, 4 (3), pp.402–415, June.

Nguyen, T.A., 1987, „Verifying Consistency of Production Systems”, Proceedings of the Third IEEE Conference on Artificial Intelligence Applications, Kissimmee, FL, pp. 4–8, February.

Nguyen, T.A., Perkins, W.A., Laffey, T.J., and Pecora, D., 1985, „Checking an Expert System's Knowledge Base for Consistency and Completeness”, Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA,vol 1, pp. 375–378, August.

Nouira R., Fouet J.M., 1996, „A Knowledge Based Tool for the Incremental Construction, Validation and Refinement of Large Knowledge Bases”, Preliminary Proceedings of Workshop on Validation, Verification and Refinement of KBS, August.

O’Keefe R.M., Balci O., Smith E.P., 1987, „Validating Expert Systems Performance”, IEEE Expert, Vol. 2, No. 4, pp. 81–90, Winter.

Pipard E., 1988, „Detection d’Incoherences et d”Incompletitudes dans les Bases de Regles, Le System INDE”, Proceeding AVIGNON, pp. 15–33.

Preece A.D. Batarekh A. Shinghal R., 1990, „Verifing Rule–Based Systems”, available on apreece@csd.abdn.ac.uk.

Preece A.D., 1994, „Foundation and Application of Knowledge Base Verification”, International Journal of Intelligent Systems, 9 pp. 683–701.

Preece A.D., 1996, „Validating Dynamic Properties of Rule–Based Systems”, International Journal of Human-Computer Studies, 44, pp. 146–169.

Preece, A.D., 1990, „Towards a Methodology for Evaluating Expert System”, Expert Systems, 7 (4), pp. 215–223.

Puuronen S., 1987, „A tabular rule–checking method”, Proceedings of 7th International Workshop on Expert Systems and their Applications (Avington, May 13–15 1987), pp. 257–268.

Smith J., Kandel A., 1993, „Verification and Validation of Rule–Based Expert Systems” CRC Press.

Stachowitz R.A., Combs J.B., 1987a, „Validation of Expert Systems”, Proceedings of The 20th Annual Hawai International Conference on System Science, pp. 686–695.

Stachowitz, R.A., Chang, C., Stock, T, and Combs, J., 1987b, „Building Validation Tools for Knowledge-Based System”, First Annual Workshop on Space Operations Automation and Robotics (SOAR 87), Houston, TX, pp. 209–216, August.

Suwa M., Scott A.C., Shortliffe E.H., 1982, „An approach to verifying completeness and Consistency in Rule-based Expert Systems”, AI Magazine, 3 (4), pp. 16–21.

Valiente G.A., 1994, „Knowledge Base Verification using Algebraic Transformations”, PhD dissertation, Informatics, Universitat de Les Illes Balears, July.