

DERIVING SAFETY CASES FROM AUTOMATICALLY CONSTRUCTED PROOFS

Nurlida Basir*, Ewen Denney[†] and Bernd Fischer*

* ECS, University of Southampton, Southampton, SO17 1BJ, UK
(nb206r, b.fischer)@ecs.soton.ac.uk

[†] SGT, NASA Ames Research Center Mountain View, CA 94035, USA
Ewen.W.Denney@nasa.gov

Keywords: formal proofs, safety case, natural deduction, automated theorem provers.

Abstract

Formal proofs provide detailed justification for the validity of claims and are widely used in formal software development methods. However, they are often complex and difficult to understand, because the formalism in which they are constructed and encoded is usually machine-oriented, and they may also be based on assumptions that are not justified. This causes concerns about the trustworthiness of using formal proofs as arguments in safety-critical applications. Here, we present an approach to develop safety cases that correspond to formal proofs found by automated theorem provers and reveal the underlying argumentation structure and top-level assumptions. We concentrate on natural deduction style proofs, which are closer to human reasoning than resolution proofs, and show how to construct the safety cases by covering the natural deduction proof tree with corresponding safety case fragments. We also abstract away logical book-keeping steps, which reduces the size of the constructed safety cases. We show how the approach can be applied to the proofs found by the Muscadet prover.

1 Introduction

Demonstrating the safety of large and complex software-intensive systems requires marshalling large amounts of diverse information, including models, code, specifications, mathematical equations and formulas, and tables of engineering constants. Obviously tools supported by automated analyses are needed to tackle this problem. For the highest assurance levels, these tools need to produce a *traceable safety argument* that shows in particular where the code and the argument itself depend on any external assumptions.

However, many tools commonly applied to ensure software safety rely on techniques such as static analysis [20] or model checking [7] that do not produce enough usable evidence (i.e., justification for the validity of their claims) and can thus not provide any further insights or arguments. In contrast, in formal software safety certification [8], as in other formal software development methods [3, 5], formal proofs are available as evidence. However, these proofs are typically constructed

by automated theorem provers (ATPs) based on machine-oriented calculi such as resolution [21]. They are thus often too complex and too difficult to understand, because the formalisms spell out too many low-level details. Moreover, the proofs may still be based on assumptions that are not valid, or may contain steps that are not justified. As consequence, concerns remain about using these proofs as *arguments* rather than just *evidence* in safety-critical applications. In this paper we address these concerns by systematically constructing safety cases that correspond to formal proofs found by ATPs and explicitly highlight the use of assumptions.

The approach presented here combines abstraction and visualization to reveal and present the proof's underlying argumentation structure and top-level assumptions. We work with natural deduction (ND) style proofs, which are goal-directed and thus closer to human reasoning than resolution proofs, and we show how the approach can be applied to the proofs found by the Muscadet ATP [19]. We explain how to construct the safety cases by covering the ND proof tree with corresponding safety case fragments. The argument is built in the same top-down way as the proof: it starts with the original theorem to be proved as the top goal and follows the deductive reasoning into subgoals, using the applied inference rules as strategies to derive the goals. However, we abstract away the ATP's book-keeping steps, which reduce the size of the constructed safety cases. The safety cases thus provide a "structured reading guide" for the proofs that allows users to understand the claims without having to understand all the technical details of the formal proof machinery. This paper is a continuation of our previous work to construct safety cases from information collected during the formal verification of the code [4], but here we concentrate on the certification components, i.e., the domain theory and the ATP used to support the software safety assurance process.

2 Formal Software Safety Certification

Our work is set in the context of formal software safety certification [8], where we use formal source code analysis techniques based on program logics to show that the program does not violate certain conditions during its execution. A *safety property* is an exact characterization of these conditions, based on the operational semantics of the programming language. Each safety property thus describes a class of haz-

ards. In our framework, the safety property is enforced by a *safety policy*, i.e., a set of verification rules that derived from initial set of safety requirements that formally represent the specific hazards identified by a safety engineer, and derive a number of logical proof obligations. Showing the safety of a program is thus reduced to formally showing the validity of these proof obligations: a program is considered safe wrt. a given safety property if proofs for the corresponding safety proof obligations can be found. Formally, this amounts to showing

$$D \cup A \models P \Rightarrow C \quad (1)$$

for each obligation, i.e., the formalization of the underlying *domain theory* D and a set of *formal certification assumptions* A entail a conjecture, which consists of a set of *preconditions* P that have to imply the *safety condition* C . The domain theory formalizes the extra-logical operations that occur in the obligations; it includes arithmetic functions and relations, programming language operations such as array indexing as well as application-specific operations such as matrix inversion. Assumptions typically specify global properties required by the component (e.g., the physical units of the input signals), while preconditions and safety conditions refer to properties at intermediate locations in the code.

The different elements of these proof obligations have different origins, and thus different levels of trustworthiness, and a safety case should reflect this. The premises and the safety condition are inferred from the program by a trusted software component implementing the safety policy, and their construction can already be explained in a safety case [4]. In contrast, both the domain theory and the assumptions are manually constructed artifacts that require particular care. The main hazard that we address in the safety cases here, by making explicit the use of hypotheses, is the unintended introduction of logical inconsistencies that can be exploited by the ATPs to construct logically correct but vacuous proofs.

The necessary analysis is hampered by the fact that the prover does not work on the obligations in the form as given in (1) but uses the form

$$\models \Lambda (D \cup A) \wedge P \Rightarrow C \quad (2)$$

which is logically equivalent but blurs the distinction between domain theory, assumptions, and preconditions, and lumps them all together as logical premises. However, proofs typically use only a subset of these premises as hypotheses, and the safety case should make explicit those that are actually used. In particular, it needs to highlight the use of assumptions. These have been formulated in isolation by the safety engineer and may not necessarily be justified, and the possibility of a logical inconsistency with the domain theory is substantially higher. Moreover, fragments of the domain theory and the assumptions may be used in different contexts, so the safety case must reflect which of them are available at each context. By elucidating the reasoning behind the certification process and drawing attention to potential certification problems, there is less of a need to trust the certification tools, and in particular, the manually constructed artifacts.

3 Converting ND Proofs into Safety Cases

3.1 Natural Deduction

Natural deduction [13, 16] is a form of proof that attempts to provide a foundational yet intuitive system to construct formal proofs. It consists of a collection of proof rules that manipulate logical formulas and transform premises into conclusions. A conjecture is proven from a set of assumptions if a repeated application of the rules can establish it as conclusion. The proof rules can be divided into basic rules, derived rules (which can be seen as proof “macros” that group together multiple inference steps) and replacement rules (which are derived rules for equivalence and equality handling). Here, we focus on some of the basic rules; a full exposition of natural deduction can be found in the literature [16].

Natural deduction uses two sets of rules for each logical connective or quantifier ($\wedge, \vee, \dots, \forall, \dots$), where one *introduces* the symbol, while the other *eliminates* it. In the introduction rules, the connective or quantifier is used as the top-level operator symbol of the unique conclusion, while it occurs in the elimination rules in the same role in one of the premises.

3.2 Conversion Process

Natural deduction proofs are simply trees that start with the conjecture to be proven as root, and have given axioms or assumed hypotheses at each leaf. Each non-leaf node is recursively justified by the proofs that start with its children as new conjectures. The edges between a node and all of its children correspond to the inference rule applied in this proof step. The proof tree structure is thus a representation of the underlying argumentation structure. We can use this interpretation to present the proofs as *safety cases* [17], which are structured arguments as well and represent the linkage between evidence (i.e., the deductive reasoning of the proofs from the assumptions to the derived conclusions) and claims (i.e., the original theorem to be proved). The general idea of the conversion from ND proofs to safety cases is thus fairly straightforward. We consider the conclusion as a goal to be met and the premise(s) as a subgoal(s); we further consider the applied inference rule as the strategy that shows *how* the conclusion is met. For each inference rule, we define a safety case template that represents the same argumentation. The underlying similarity of proofs and safety cases has already been indicated in [17] but as far as we know, this idea has never been fully explored or even been applied to machine-generated proofs.

The conversion we present here preserves the inferences and formulas of the original proof, but avoids overloading the constructed arguments with trivial proof steps. We identify semantically related or repeated identical inferences that can be abstracted away in order to construct a more concise argument. In the following we describe the safety case templates for the base rules of the calculus. We use the Goal Structuring Notation [17] to explicitly represent the logical flow of the proof’s argumentation structure.

3.3 Conjunctions

The rules for conjunction introduction and elimination shown in Figure 1 directly represent the intuitive interpretation of conjunctions: if A is true and B is true, then evidently $A \wedge B$ is true as well (\wedge -i), and if $A \wedge B$ is true, then both A and B must be as well (\wedge -e1 resp. \wedge -e2). The hypotheses that are available to show $A \wedge B$ true are also available to show A (resp. B) true as well. Similarly in the case of \wedge -e1 (resp. \wedge -e2) where the available hypotheses to show each conjunct true are also available to show $A \wedge B$ true.

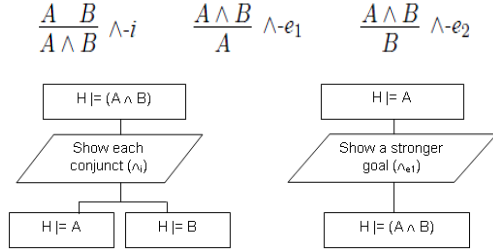


Figure 1: Safety Case Templates for \wedge -Rules.

The \wedge -rules can be directly converted into safety cases. In the case of \wedge -introduction, the satisfaction of the conclusion (i.e., goal of the safety case) is implied by the satisfaction of the two premises (i.e., subgoals of the safety case) based on the strategy of \wedge -introduction rule. For the \wedge -elimination rule, the strategy shows a logically stronger goal we can conclude A (resp. B) if we have a proof of $A \wedge B$. Figure 1 shows the safety case fragments for the conjunction rules.

3.4 Disjunctions

A disjunction can be introduced as long as one of the disjuncts is already known i.e., if A (resp. B) is true, then evidently $A \vee B$ is true as well. In the safety case, a goal $A \vee B$ is constructed, which is justified by the subgoal A (resp. B) via the strategy (\vee -i1) (resp. \vee -i2). The hypotheses that are available to show $A \vee B$ true are also available to show subgoal A (resp. B) true.

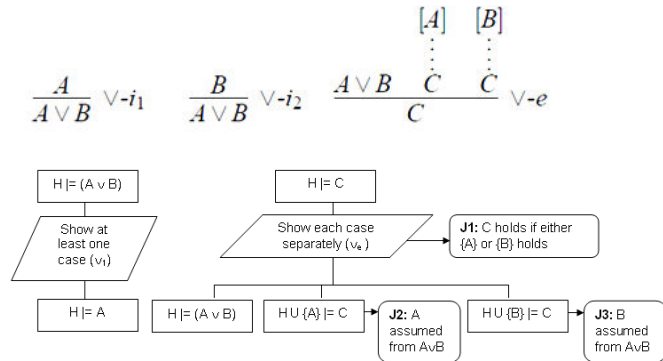


Figure 2: Safety Case Templates for \vee -Rules.

In contrast, in disjunction elimination, we only know that $A \vee B$ holds, but not which of A or B is true, so that we need to reason by cases to conclude C from $A \vee B$, i.e., separately con-

sider each of the two cases for the disjunction to be true. In the first case we thus assume A together with the available hypotheses and try to derive C , in the second case we assume B together with the available hypotheses and try to derive C . If both cases succeed, we can conclude C . The safety case fragment makes this argument explicit, and, in particular, explicitly justifies the use of the respective assumptions in the two cases. Figure 2 shows the safety case fragments for the disjunction rules.

3.5 Implications

The implication elimination follows the standard pattern but in the introduction rule we again temporarily assume A as hypothesis together with the list of other available hypotheses, rather than deriving a proof for it. We then proceed to derive B , and discharge the hypothesis by the introduction of the implication. The hypothesis A can be used in the proof of B , but the conclusion $A \Rightarrow B$ no longer depends on the hypothesis A after B has been proved.

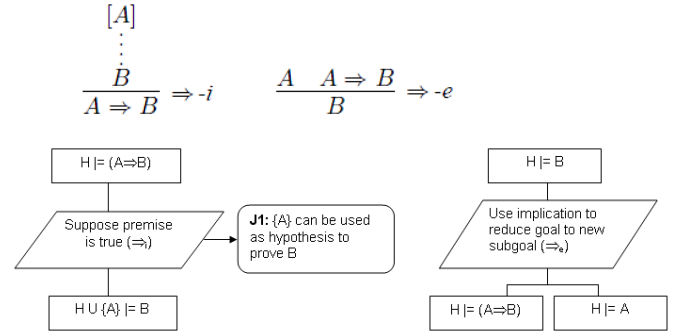


Figure 3: Safety Case Templates for \Rightarrow -Rules.

In the safety case fragment (see Figure 3), we use a justification to record the use of the hypothesis A , and thus to make sure that the introduced hypotheses are tracked properly.

3.6 Universal Quantifiers

The natural deduction calculus can also be used for proofs in predicate logic. The proof rules focus on the replacement of the bound variables with objects and vice versa. For example, in the elimination rule for universal quantifiers, we can conclude the validity of the formula for any chosen domain element t_x .

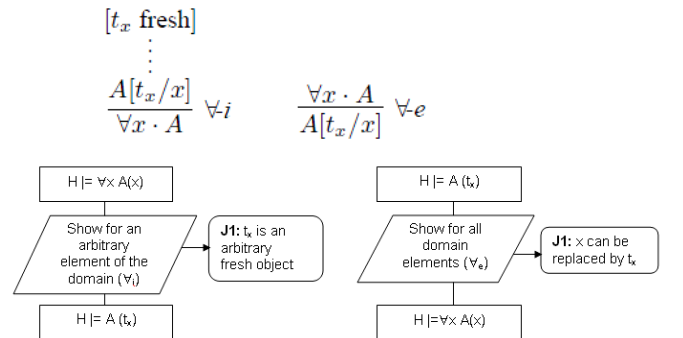


Figure 4: Safety Case Templates for \forall -Rules.

In the introduction rule, however, we need to show it for an arbitrary but fresh object t_x (that is, a domain element which does not appear elsewhere in H , A , or the domain theory and assumptions). If we can derive a proof of A , where x is replaced by the object t_x , we can then discharge this assumption by introduction of the quantifier. The safety case fragments (see Figure 4) record this replacement as justification. The hypotheses available for the subgoals in the \forall -rules are the same as those in the original goals.

4 Safety Case Generation Process

To automatically construct the ND proof safety case, we integrate the Muscadet [19] theorem prover with Adelard's ASCE tool [1]. We convert the proofs that are generated by the Muscadet prover into an XML format (i.e., PROOF-XML in Figure 5). The XML file contains all the relevant information that is required for the automatic safety case construction. Subsequently, an XSLT program is used to transform the proofs into another XML (i.e., file GSN-XML in Figure 5) logically representing a safety case, by applying the templates that have been defined in Section 3. The file format was designed so that the derived safety cases can be easily be adapted to different tools or applications. Finally, to present the resulting safety case graphically, we use a Java program to layout the logical information which involved some mathematical calculations in positioning the argument and to convert it into the standard Adelard ASCE file format. Figure 5 summarizes the safety case generation process.

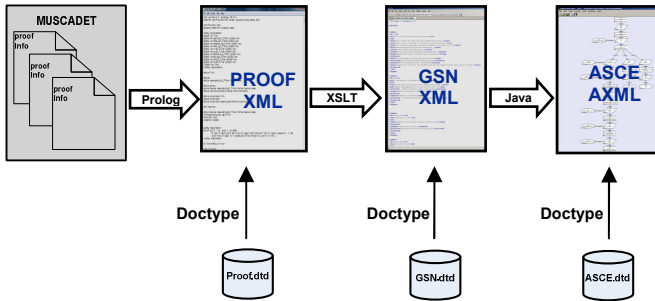


Figure 5: Safety Case Generation Process.

5 Hypothesis Handling

An automated prover typically treats the domain theory D and the certification assumptions A as premises and tries to derive $\wedge (D \cup A) \wedge P \Rightarrow C$ from an empty set of hypotheses. As the proof tree grows, these premises will be turned into hypotheses, using the \Rightarrow -introduction rule (see Figure 3). In all other rules, the hypotheses are simply inherited from the goal to the subgoals. However, not all premises will actually be used as hypotheses in the proof, and the safety case should highlight those that are actually used. This is particularly important for the certification assumptions. We can achieve this by modifying the template for the \Rightarrow -introduction (see Figure 6). We distinguish between the hypotheses that are actually used in the proof of the conclusion (i.e., A_1, \dots, A_k) and those that are vacuously discharged by the \Rightarrow -introduction (i.e., A_{k+1}, \dots, A_n).

We can thus use two different justifications to mark this distinction. Note that this is only a simplification of the presentation and does not change the structure of the underlying proof, nor the validity of the original goal. It is thus different from using a relevant implication [2] under which $A \Rightarrow B$ is only valid if the hypothesis A is actually used.

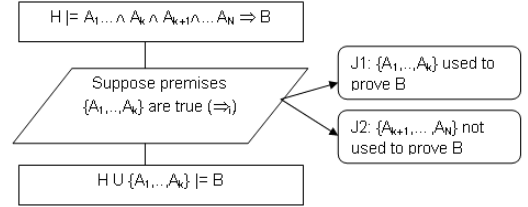


Figure 6: Hypotheses Handling in \Rightarrow -Introduction Rule.

In order to minimize the number of hypotheses tracked by the safety case, we need to analyze the proof tree from the leafs up, and propagate the used hypotheses towards the root. By revealing only the used hypotheses as assumptions, the validity of their use in deriving the proof can be checked more easily. In our work, we also highlight the use of the external certification assumptions that have been formulated in isolation by the safety engineer. For example in Figure 7, the use of the hypothesis `has_unit(tpfp_float_7_0e_minus_1, ang_vel)`, meaning that a particular floating point variable represents an angular velocity, which has been specified as external assumption, is tracked properly in the safety case, and the validity of its use in deriving the proofs can be checked easily.

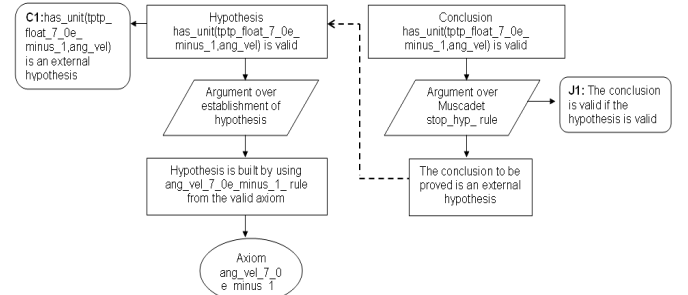


Figure 7: External Hypothesis.

6 Application to Muscadet

We illustrate our approach by converting proofs created by the AutoCert certification tool [11], which takes a set of requirements, and a domain theory consisting of logical axioms and so-called annotation schemas. These are used to infer logical annotations and construct proof tasks which are sent to the ATP in order to create proofs that the code complies with the requirements.

For these experiments, we used the Muscadet [19] theorem prover during the formal certification of the initialization safety of a component of an attitude control system. Muscadet is based on natural deduction, but to improve performance, it implements a variety of derived rules in addition to the basic rules of the calculus. This includes rules for dedicated equal-

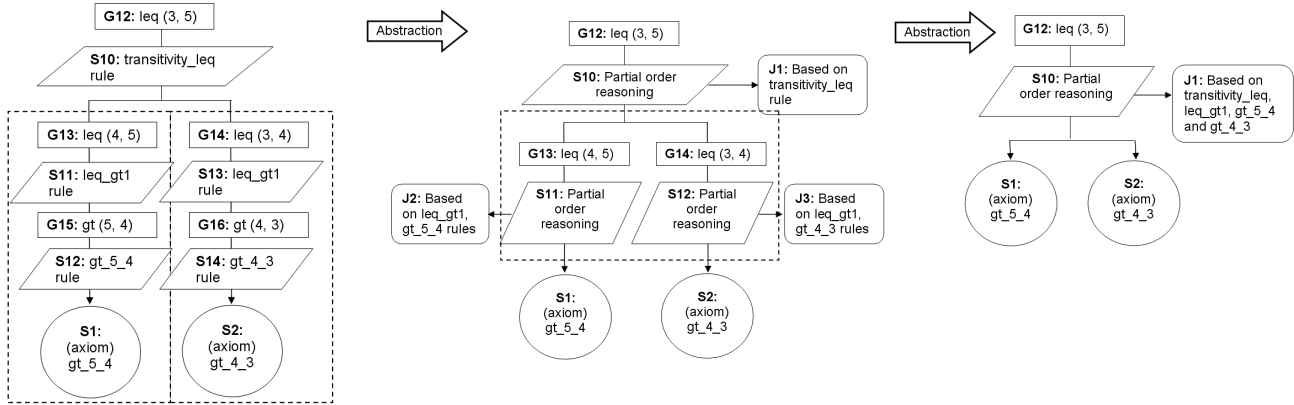


Figure 9: Abstraction of Safety Cases

Acknowledgements: The first author is funded by the Malaysian Government, IPTA Academic Training Scheme. This material is based upon work supported by NASA under awards NCC2-1426 and NNA07BB97C.

References

- [1] Adelard ASCE home page, <http://www.adelard.com> (2007).
- [2] A. R. Anderson, N. Belnap. "Entailment: the logic of relevance and necessity", Princeton University Press. (1975).
- [3] J. Barnes. "High Integrity Software: The SPARK Approach to Safety and Security", Addison-Wesley. (2003).
- [4] N. Basir, E. Denney, B. Fischer. "Constructing a Safety Case for Automatically Generated Code from Formal Program Verification Information", In: *Proc. SAFECOMP 08*, LNCS 5219, pp. 249-262 (2008).
- [5] J.C. Bicarregui, J.S. Fitzgerald, P.A. Lindsay, R. Moore, B. Ritchie. "Proof in VDM: A Practitioner's Guide", *FACIT series*, Springer (1993).
- [6] D. Chester. "The Translation of Formal Proofs into English", *Artificial Intelligence* 7(3), pp. 261-278 (1976).
- [7] E.M. Clarke, J.M. Wing. "Formal methods: state of the art and future directions", *ACM Computing Surveys* 28(4), pp. 626-643 (1996).
- [8] E. Denney, B. Fischer. "Correctness of source-level safety policies", In: *Proc. FM 2003: Formal Methods*, LNCS 2805, pp. 894-913 (2003).
- [9] E. Denney, B. Fischer. "Software certification and software certificate management systems (Position paper)", In: *Proc. ASE Workshop on Software Certificate Management Systems*, ACM, pp.1-5 (2005).
- [10] E. Denney, J. Power, Konstantinos Tourlas. "Hiproofs: A hierarchical notion of proof tree", *ENTCS*, 155, pp.341-359 (2006).
- [11] E. Denney, S. Trac, "A Software Safety Certification Tool for Automatically Generated Guidance, Navigation and Control Code". In *IEEE Aerospace Conference. Big Sky, MT*, (2008).
- [12] M. Garnacho, M. Prin. "Convincing Proofs for Program Certification", In: *Proceedings Intl. Workshop of SafeCert, ENTCS*, (2008).
- [13] G. Gentzen. "Investigations into Logical Deductions", The Collected Papers of Gerhard Gentzen, M.E.Szabo (ed), North-Holland, Amsterdam. pp. 68-131 (1969).
- [14] X. Huang: Translating Machine-Generated Resolution Proofs into ND-Proofs at the Assertion Level. *Proc. 4th Pacific Rim International Conference on Artificial Intelligence*, LNCS 1114, pp. 399-410 (1996).
- [15] X. Huang, A. Fiedler. "Proof verbalization as an application of NLG", In *Proc 15th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, pp. 965-970 (1997).
- [16] M. Huth, M. Ryan. "Logic in Computer Science Modelling and Reasoning about Systems", 2nd Edition, Cambridge University Press (2004).
- [17] T. P. Kelly. "Arguing safety a systematic approach to managing safety cases", PhD Thesis, University of York (1998).
- [18] W. McCune, O. Shumsky-Matlin. "Ivy: A Preprocessor and Proof Checker for First-Order Logic", In M. Kaufmann, P. Manolios, and J. Strother Moore, editors, *Computer-Aided Reasoning: ACL2 Case Studies*, Advances in Formal Methods (4), Kluwer Academic Publishers, pp. 265-282 (2000).
- [19] D. Pastre. "MUSCADET 2.3: A Knowledge-Based Theorem Prover Based on Natural Deduction", In: *Proc Intl. Joint Conference on Automated Reasoning*, LNCS 2083, pp. 685-689 (2001).
- [20] PolySpace Technologies, <http://www.polyspace.com>, 2007.
- [21] J.A. Robinson. "A machine-oriented logic based on the resolution principle" *JACM* 12(1), pp. 23-41 (1965).
- [22] S. Trac, Y. Puzis, G. Sutcliffe. "An interactive derivation viewer", In: *Proc. Intl. Workshop User Interfaces for Theorem Provers*, ENTCS 174, Elsevier, pp.109-123 (2007).
- [23] G. Sutcliffe, D. Belfiore. "Semantic Derivation Verification", In: *Proc. 18th Florida Artificial Intelligence Research Symposium*, pp.641-646 (2005).