Image Cover Sheet

CLASSIFICATION	SYSTEM NUMBER
UNCLASSIFIED	497168

TITLE

PROOF LOGGING FINAL REPORT

System Number:

Patron Number:

Requester:

Notes:

DSIS Use only:

Deliver to: FF

Proof Logging Final Report

TR-95-5471-05

Sentot Kromodimoeljo
Bill Pase

Release date: JUNE 1995

ORA Canada
267 Richmond Road, Suite 100
Ottawa, Ontario K1Z 6X3
CANADA

DSS Contract No. W2207-2-AF08/01-SV

Contents

1 Introduction

This is the Final Report for Proof Logs in NEVER as they appear in EVES, version 2.4.2. It describes the design of proof logging, and the difficulties encountered during the contract "Proof Logging Tool Development" (W2207-2-AF08/01-SV). For a technical description of Proof logs, the reader is referred to [1], while a user level description is given in [2]. Proof Logging is related to another effort, called Proof Checking, which will be described separately.

This report is divided into four sections. The section on design describes the issues that arose during the design of Proof Logging and the trade-offs involved. The section on difficulties describes the areas that caused the most problems. The section on lessons learned describes a number of things that the completion of Proof Logging taught us. The section on future work describes aspects of proof logging that remain to be completed.

2 The Design

This section describes a number of the design decisions and the issues that are involved in proof logging. The technical aspects of the design are described in [1]. The main consideration for the design of proof logging is that it support both proof checking and proof browsing. Since the goal is to formalise logging and checking, the design ought to ease this task whenever possible.

Traditionally, proofs are described using a sequent notation. However, NEVER treats a proof as a sequence of transformations of an expression to `(true)`. Each transformation preserves equivalence, thus the sequence preserves equivalence, and thus the expression is shown to be equivalent to `(true)`. In NEVER a conjecture to be proven is stated as a single expression, and throughout the proof, it is this single expression that is being transformed. The transformations are performed on the concrete syntax, every single transformation on the concrete syntax is logged.

Many proofs in NEVER require several commands to complete. Each of the commands take a single input expression, and produce a single output expression. The output expression is equivalent to the input and results from the transformations that the command performed. Every command is responsible for generating a proof log of the transformations. If the initial expression is successfully transformed to `(true)`, then the expression is a theorem and the sequence of transformations performed by all of the commands constitutes the proof log for the proof of the theorem. This approach permits the generation of a proof log for each command to be developed independently.

An additional feature of this approach is that the proof log for a command need not describe how the command actually performed the proof, only that the proof log transform the input expression to the output expression. This observation is used to generate proof logs that omit the details of how a proof was performed by the command. This is particularly important for the logging of proofs performed by the simplifier. Simplifier proofs do not correspond to the simple model of transforming an expression.

This flexibility causes problems for proof browsing, since the proof and the proof log are not required to closely mirror one another. This is solved by the addition of markers, or comments, in the proof log that provide extra information to the proof browser to aid in the reconstruction of the proof.

A number of proofs are performed by NEVER that are hidden from the user. For example, the proofs that occur at the time of a declaration, and the presimplification of proof obligations. These proofs need to be logged so they can ultimately be checked.

A major issue is the choice of the transformations, or inference rules. For convenience of implementation there are a large number, each corresponding to a transformation performed by the various commands. However, for showing the soundness of the system, it would be preferable if there were a small number of inference rules. Currently NEVER utilises a large number of inference rules, but a much smaller set has been identified from which the remaining rules can be derived.

3 Difficult Areas

The addition of proofs logs to NEVER did present several difficulties. Though they did not prevent the completion of the effort, their solution is worth noting. The three main problems were: syntactic abbreviations, induction, and the simplifier. This section discusses each of these difficult areas.

Syntactic abbreviations are commonly used in EVES that map to constructs in the NEVER theorem prover. For example, n-ary functions are represented by their binary versions, the boolean connectives are represented as "if" expressions, and defined constants are represented by their values. These abbreviations are expanded immediately upon input, and may be replaced upon output. Other efforts for logging in theorem provers have tended to ignore this issue and have not considered abbreviations (and other input/output conversions) a part of the proof. However, errors have been found in this area of NEVER, and so we were reluctant to ignore the issue.

The solution is to log the abbreviations as if they were a part of the proof. In fact, we consider them to be part of the proof. It turns out that logging the abbreviations required a substantial amount of effort. Furthermore, the additions to the proof log added considerably to the number of inferences that are recorded. Another problem is that the abbreviations can be expanded prior to any proof command, but we associate proof logs with proof commands. This requires that proof logs be associated with declarations and the "try" command.

The second area of difficulty is induction. The NEVER model for proof logging is to record a sequence of equivalence preserving transformations. This does not correspond to the usual descriptions of induction. This was solved by logging induction as the use of a single strong induction principle, plus the case analysis and instantiations of the hypothesis. This approach allows induction to be represented as a sequence of simple equivalence preserving transformations.

The last difficulties are presented by the simplifier. The simplifier performs deductions by means of decision procedures. These procedures do not function by performing equivalence preserving transformation and thus do not fit in with the NEVER model for proof logging. However, since the system is required to log a sequence of inferences that transform the input expression to the output expression, not necessarily the sequence it performed, the requirement upon the simplifier is that it produce a log for this transformation. Thus, the difficulty becomes one of producing a proof log each time the simplifier succeeds in obtaining a proof.

The simplifier performs proofs involving equality, integers, frules and grules, and instantiation of quantified variables. Equality is logged as the corresponding sequence of equality substitutions. Integer reasoning is logged as the use of the appropriate integer axioms. The application of frules and grules are logged similar to rewrite rules, i.e., the application of an axiom with a set of instantiations. The logging for the instantiation of quantified variables is not yet completed (this is discussed later in the report). The addition of proof logging for the simplifier has substantially increased the size of proof logs.

4 Lessons Learned

This section discusses the lessons that were learned during the development and implementation of proof logging in NEVER. The lessons were the size of proof logs, the advantages of annotating the source code, and the value of type coercion.

Early in the proof logging effort, there was concern that the proof logs would be excessively large. This would have meant they needed to be stored on disk, not in memory. This would have made checking, and especially browsing less convenient for the user, and more difficult to implement. It was even suggested that proof logs would be so large that they would have to be checked concurrently with their generation, so they would not have to be stored at all. Had this turned out to be true, the design would have been greatly complicated. Fortunately, these fears were unfounded. Proof logs are quite reasonable in size, and can be stored in memory as part of the prover database. This has greatly eased the development of a prototype checker and browser, both of which were instrumental in testing the design and implementation of proof logs.

An interesting side-effect of the implementation of proof logging was the errors that were discovered in the NEVER theorem prover. For a piece of code to generate a proof log, it was required that the effect of the code on the expression being transformed be fully described. The generation of proof logs can be thought of as annotations for the theorem prover. The exercise of developing these annotations uncovered errors in the code. These were found when it was not possible to write down the inferences that a particular piece of code was performing. During the implementation a number of these errors were discovered, and corrected. These errors included missing side conditions, out of domain problems, and faulty boundary conditions. Although these errors could have resulted in unsound proofs, no unsound proofs occurred in the test suite.

Previous versions of EVES (prior to 2.4.2) did not perform type coercion. This was particularly

an issue for the arithmetic operators, which were not guaranteed to return a value of type `(int)`. Proofs involving these operators would have to show, or state as a hypothesis, that the value was an `(int)`. An alternative is to coerce the result to `(int)` regardless of the parameters. This can ease the proof burden both to the user and to the theorem prover. It turns out that an approach based on type coercion makes proof logging of integer reasoning simpler. Because of this, the theorem prover has been changed to coerce types. For example, `(+ x y)` is of type `(int)` and is equivalent to `(+ (ord x) (ord y))`, where `ord` acts as a function that coerces to `(int)`.

5 Future Work

This section discusses future work directed towards improving proof logging in NEVER. Two aspects are discussed, the possibility of a reduced number of inference rules, and the logging of automatic instantiations in the simplifier.

For the logging of proofs there is an issue of the choice of the inference rules that are chosen. For convenience, a large set is used in the current implementation. This has the advantages of greater correspondence to the proof and a shorter proof log. However, a smaller set would ease the soundness proofs for the inference rules and the correctness of the proof checker. Since the correctness of the proof checker is expected to be difficult, a set of inference rules that reduce the effort would be advantageous. A reduced set of rules would require that additional annotations be inserted into the log for the proof browser.

The simplifier component of the theorem prover is able to perform automatic instantiation of quantified variables. In the current release, version 2.4.2, these instantiations are not entered into the proof log. They may either be assumed correct or automatic instantiations can be disabled. Proof logging ought to be completed for automatic instantiations, or the feature should be removed from future versions of the theorem prover. A first approach would be to investigate the importance of automatic instantiations in existing EVES proofs as exemplified by the test suite. Based upon this analysis it needs to be determined whether to log the existing automatic instantiations, or to replace it with a simpler scheme.

6 Conclusion

This Final Report for Proof Logs in NEVER has described interesting aspects of the contract. These included an outline of the design, a discussion of the difficulties, the lessons learned, and the future work. This effort is to be followed by the Proof Checking contract, which is devoted to the development of a proof checker for the proofs logs that can be generated by EVES version 2.4.2. The technical details of proof logging can be found in [1].

The remaining issues for proof logging are the choice of inference rules and logging of automatic instantiations. For the first, a small set of inference rules have been designed that would be easier to prove sound and for which a proof checker would be easier to develop and proof correct. It would be valuable to convert the proof logging to generate the inferences in the small set. For the second, either the logging of automatic instantiations must be implemented, or the automatic instantiation feature will have to be removed from NEVER.

References

[1] S. Kromodimoeljo, B. Pase, Proof Logs, TR-95-5471-03, ORA Canada, June 1995.

[2] S. Kromodimoeljo, B. Pase, Proof Logging User Manual, TR-95-5471-04, ORA Canada, June 1995.

NO. OF COPIES NOMBRE DE COPIES	COPY NO. COPIE N°	INFORMATION SCIENTIST'S INITIALS INITIALES DE L'AGENT D'INFORMATION SCIENTIFIQUE
1	1	
AQUISITION ROUTE FOURNI PAR ▶	DSACCIS 3 via CRAD DRP	
DATE ▶	16 Apr 96	
DSIS ACCESSION NO. NUMÉRO DSIS ▶		

DND 1158 (6-87)

National Defence / Défense nationale

Symicon 497168

PLEASE RETURN THIS DOCUMENT TO THE FOLLOWING ADDRESS:

DIRECTOR
SCIENTIFIC INFORMATION SERVICES
NATIONAL DEFENCE
HEADQUARTERS
OTTAWA, ONT. - CANADA K1A 0K2

PRIÈRE DE RETOURNER CE DOCUMENT À L'ADRESSE SUIVANTE:

DIRECTEUR
SERVICES D'INFORMATION SCIENTIFIQUES
QUARTIER GÉNÉRAL
DE LA DÉFENSE NATIONALE
OTTAWA, ONT. - CANADA K1A 0K2