The Story of ProofPower

Roger Bishop Jones

Date: 2010/01/25 13:11:02

Abstract

History and rationale of the development of ProofPower.

http://www.rbjones.com/rbjpub/pp/doc/t014.pdf

Id: t014.doc,v 1.4 2010/01/25 13:11:02 rbj Exp

p

Contents

1 Introduction

In 1990 the then International Computers Limited (ICL) began a collaborative R&D project of which its part was to be the implementation of the software which is now known as ProofPower. ProofPower is a proprietary open source software suite which provides support for the application of proof oriented formal methods to the development of information systems. ProofPower is now owned by Lemma1 Limited, and the software and supporting documentation may be downloaded from the `lemma-one.com` web site.

ProofPower is itself well described in its manuals and tutorials. Its use is illustrated by examples which come with the system or which have been published separately. The main emphasis of this material will therefore be on aspects of ProofPower which have been less well treated elsewhere, namely, its interrelated history and rationale.

1.1 Market Context

The origins of ProofPower lie principally in three market factors which came together towards the latter part of the 1980's.

ICL Enters Defence Market

The first of these was the entry of ICL into the defence marketplace, at the end of a delay imposed by the labour government when in 1967 it formed ICL from the remnants of the British commercial computing industry. The abstinence by ICL from operational defence computing was a time-limited *quid pro quo* for a period of protection which it was to enjoy in its core markets from competition by the (mainly defence) companies which had retained relevant technologies for military rather than commercial applications. ICL saw entry into operational defence related computing as a commercial opportunity and hoped to exploit its most innovative technologies such as the SIMD distributed array processor. Anticipating the need for considerable R&D to underpin its penetration of the defence industry ICL established a Defence Technology Centre which enjoyed for a while the latitude to invest in building expertise and capability without too close an eye on the bottom line.

Demand for Formal Methods in Security

The second significant factor was pressure from the special relationship between the UK and the USA in relation to the handling of classified intelligence data. Mindful of the threat to its national security which was posed by insecure information processing systems, the US government had set out guidelines for the development of secure systems which laid great emphasis on the use of mathematical methods in their design and implementation. Regulations promulgated on the certification of computer system for use in security critical applications mandated the use of *formal methods* where the highest levels of certification were sought. In order to continue in its privileged position in relation to access to sensitive information, the UK government would either have to buy its computer equipment from manufacturers who had completed the required US certification, or they would have to put in place their own comparable system of certification and encourage or contract the UK computer industry to develop systems for certification under that scheme. The UK was the first country to follow the US in this regard with its own national certification scheme, but this was to lead to a harmonised regime across Europe.

It was this which lead indirectly to the formation of a formal methods team at ICL Defence Systems motivated to reason formally about specifications in the Z language.

DTI IED

The third ingredient of the political pot-pourri was the desire of the Department of Trade and Industry to bring academic research closer to the needs of industry. Through its Information Engineering Directorate a programme of R&D through mixed industry and academic collaborative projects. The call for proposals arrived just when the formal methods team was beginning to understand the problem and the available technologies well enough to formulate a project which would address those problems by progress the technologies.

These are the political factors which made possible the FST project (IED 1563) under which ProofPower was first developed.

To understand the details of what the project produced and why and how it did so we need to look a little more closely into the technicalities.

1.2 Technical Context

While the establishment of this scheme was in progress the government was placing R&D contracts with UK industry relating to the development of secure systems. ICL defence systems won some of these contracts, and was made aware that this kind of work would have to be undertaken in a manner consistent with the emerging certification regime. A formal methods capability would therefore have to be established. THe first steps were taken in 1985.

Z had already been identified as the preferred formal specification language for work on government secure systems and so familiarisation with Z was an early priority, as was gaining experience in the proof technologies which seemed most promising for the kind of work expected. The two proof tools which were being evaluated by the formal methods team were NQTHM (otherwise known as the Boyer-Moore prover after its authors) and the Cambridge HOL system, both of which had also been used at RSRE Malvern for the formal treatment of digital hardware.

Of these two the Cambridge HOL system seemed to offer best prospect of success in reasoning about specifications in Z, in default of any proof tool for Z itself.

The main factors in this assessment were:

- firstly that the language HOL (Higher Order Logic) supported by the Cambridge HOL system was much closer in logical expressiveness to Z than was the impoverished first order logic of NQTHM.
- secondly that the LCF paradigm, giving the user a powerful functional programming language for programming proofs and for other kinds of extension seemed to promise a greater flexibility of the tool for adaptation to tasks for which it had not been originally intended.

By contrast with work in HOL, for which the concrete syntax was entirely coded in ASCII, Z had a culture of pencil and paper specification using not only more customary logical symbols but also many other special mathematical symbols and also a special graphical layout in which formal specifications were presented in various kinds of boxes and other structures. It was also a novelty of the Z culture that specifications were presented embedded into into formal textual discourse. The relationship the formal and the informal material in computer documents (usually programs of course) was inverted, instead of adding annotations into the formal material, Z added formal materials into a proper English document.

Cambridge HOL needed no special facilities for document preparation, expecting HOL scripts to be treated just like a program. Specifications in Z however were intended for a readership who might admire a beautifully handwritten English document with interspersed mathematics, but would look askance at a program listing. An element of our initial tasks in developing ways of reasoning formally about Z was therefore to provide document preparation facilities.

The method adopted was to manually transcribe a specification from Z to HOL, maintaining as close a correspondence as possible between the original specification in Z and the translated specification in HOL.

Sun workstations came with a font editor and it proved straightforward to edit into a standard font a good selection of the logical and mathematical symbols used by Z. Cambridge HOL could be wrapped in a couple of UNIX filters which translated between these symbols and the corresponding ASCII sequences used by HOL.

As well as crudely effecting a partial reconciliation between the lexis of Z and HOL, some more fundamental issue were addressed at this early stage.

The Cambridge HOL user community attached value to the fact that HOL is the kind of logical system in which mathematics and its applications could be undertaken by conservative extension (it provides a *foundation* for mathematics). The introduction of new axioms was supported by the tool, but frowned upon by its users except in those very rare circumstances in which it was not technically avoidable.

The Z notation had no apparent concern for this matter, and the approved specification style freely admitted the introduction of "axiomatic specifications" which combined the introduction of new (so called) global variables (constants) with an axiom stating any desired property.

At ICL we were fully sympathetic to the HOL tradition harking back to *Principia Mathematica* but could not eschew the use of axiomatic specifications in Z. We therefore translated Z axiomatic specifications into a good but not absolutely faithful approximation in Z by a (behind the document) translation into a definition using the HOL choice function. When we later came to a substantial project in which formal verification would be critical a special version of Cambridge HOL was produced for that project in which, along with a small number of other security enhancements, a feature was introduced which make such conservative translations more fully faithful to the meaning of the original specification. An equivalent facility was later introduced to the main Cambridge HOL in response to our enquiries on this point.

So far ProofPower has not appeared on the scene, but we have seen some minutiae which eventually left their mark on its character.

The DTC formal methods team applied formal methods to the development of secure systems, using transcription into HOL for formal proofs for three years before the opportunity arose to proposed a substantial program of work on the development of tools to support this kind of work. This work included the formal aspects of a complete hardware development and manufacturing project which was undertaken so as to achieve certification at the highest possible levels of assurance, involving the development of a formal machine checked proof that the system met the formal statement of the critical security requirements.

Transcribing Z into HOL with care to ensure that meaning is preserved and then reasoning formally with the resulting specification, all the while wondering how the methods of transcription might be made more complete and systematic to the point that they could be implemented in an embedding is not a bad way to get a good understanding of the semantics of Z. We were therefore reasonably equipped when the opportunity to start developing formal methods tools in earnest to make some substantial strides forward.

Nevertheless, full and faithfully support for Z via embedding into HOL was not something we were confident could be done so as to yield a usable and productive proof environment.

1.3 The Proposal

It was a prime objective of ICL's part of the FST project to realise the best possible support for proof in Z by some kind of embedding into HOL. At the time that the proposal was prepared, however, there was considerable uncertainty about what could be achieved and how. The project proposal was therefore almost mute on this topic, and put forward as ICL's task the production of a proof assistant for the HOL logic engineered to standards appropriate for its use in commercial developments of software for certification at the highest levels of assurance. Additional declared desiderata were, improved usability of the tool and productivity in its intended uses.

These desiderata were to be realised by:

- Development following industrial product quality standard methods (including inspection of

detailed design).

- The application of formal methods to the design of the new tool. Formal specifications of the syntax and semantics of the HOL language were prepared, and of the "critical aspects" of the design of the proof assistant (viz. those which were pertinent to the risk of the tool accepting as a theorem some sentence which was not legitimately derivable in the relevant logical context).

2 The Development of ProofPower

Work began on a disposable prototype proof tool, which was called "ICL HOL" several months before the official start of the FST project at the beginning of 1990, so a working proof tool was available very early in the project. Though at the time of writing the project proposal no complete injection of Z into HOL was known, in the first few months of 1990 a mapping was devised and partially specified in Z which was thought to be semantically correct (up to a bit of debate about the semantics of undefinedness) and practicable. The code of the prototype was not used in the product standard development which followed rather more slowly, but the prototype was used as a base for prototype Z proof tool which was developed concurrently with the product version of the HOL tool which was called ProofPower.

3 Features of ProofPower

ProofPower is *primarily* a conservative re-engineering of 'classic' Cambridge HOL (HOL88) with support for Z added by semantic embedding. The innovations, such as they are, can be related to these two factors.

3.1 Engineering

Use of Standard ML Standard ML failed to draw in the entire ML community, but adopting standard ML for this development still looks like the right choice.

Formal Specification of Key Features Key features of the tool, notably the logic it supports and the logical kernel which ensures that the tool will only accept as theorems sentences which are derivable in that logic, and formally specified (in ProofPower-HOL). The logical kernel includes critical aspects of management of the theory hierarchy.

Systematic Software Development ProofPower has a detailed design, implementation and test documents for each software module. Since all these documents contain SML code which is either compiled into or executed to test ProofPower, (detailed design documents include signatures) there is little tendency for the detailed design to slip out of sync with the implementation.

Name Space Improvements One feature of LCF-like provers is that most of the tools used to implement the prover are accessible to the user through the metalanguage namespace and may be re-used in extending the capabilities of the system. This make for a rich and powerful environment for programming proofs, but also for a very large namespace in which the user may struggle to find the features he needs. It was a design objective of ProofPower to ameliorate this situation in modest ways.

The general policy was as follows:

(a) For low level features needed for efficient coding of new proof facilities the coverage of each kind of facility should be complete and systematic, so that the names of the various individual interfaces can often be obtained by following some obvious rule. Elementary examples of such conventions are the systematic use of "mk_" and "dest_" prefixes for syntactic constructors and destructors.

(b) User level proof facilities (e.g. tactics) should so far as possible make it unnecessary for the user to select a specific tool. An elementary example is *STRIP_TAC* which inspect the principle logical connective of the current goal and perform an appropriate proof step, making it less likely that the user will have to know the name of the low level tactic which deals with each individual connective. The tactic of the same name in ProofPower is considerably extended in power, addressing not only principle logical connective of a formula but also in many cases the relation of an atomic formula. It is context sensitive, applying more rules in richer logical contexts and, for example, in the case of membership predications will be able to apply in appropriate contexts a range of different rules for the various kinds of structure of which membership can be asserted (e.g. membership of an intersection will be transformed into a conjunction of membership assertions).

(c) documentation of the namespace in the reference manual is generated automatically from the detailed design documentation, and includes a KeyWord In Context index making it easy to discover all the names which contain a particular substring. Names are usually compounded from a sequence of elements related to the function of the name.

(d) the use by ProofPower of special logical and mathematical symbols extends to the names in the standard ML namespace (of which ProofPower effects a lexical variant with an extended character set).

The Theory Hierarchy For theorem proving in the context of large specifications some way of structuring the logical context is desirable, and in these tools this is done through a hierarchy of "theories", which in this context are a kind of hybrid between the logician's notion of a theory and the computer scientists of a module. In normal use a theory is the largest unit of conservative extension of the logical system, though in extremis extensions not know to be conservative will also be permitted and recorded as such. It is important for the preservation of logical coherence that when changes other than mere additions are made to the hierarchy that everything which logically depends upon the material changed is modified or invalidated at that point. In the Cambridge HOL system this was accomplished by deleting theories, and any change to a theory could be accomplished only by deleting that theory in its entirety and all its descendants.

Flexibility in this matter depends upon how much detailed information is held about the interdependecies. Holding more information permits greater flexibility, but complicates the data structures involved in theorem proving. When LCF was first developed computers were very slow, and one put up with an inflexible system.

The ProofPower theory hierarchy is designed with a greater degree of flexibility, falling short of the full flexibility which might flow from the fullest recording of dependencies. A definition or specification can be deleted from a theory without deleting the entire theory. It is however necessary to delete (and reload if required) all the definitions in the theory which took place subsequent to the one under modification (and those in theories lower in the hierarchy).

The Logical Kernel Cambridge HOL, as well as the inferences rule of HOL provides a back-door "mk_thm" which allows any sequent to be made into a theorem and thereby rendering the logic technically inconsistent. In a tool intended for use in developments subject to formal evaluation this is difficult to defend. It is of course no feat of engineering to omit this feature, the rationale for which still escapes me even though some have felt it worth defending. It is I believe now used in Cambridge HOL to check the validity of intermediate states of the sub-goal package.

The Sub-Goal Package LCF-like provers are implemented using an abstract data type to implement a type of theorems, the only constructors of which are the rules of the logic (let us assume for present purposes that axioms are rules without premises). In all the cases of interest here the underlying logic is in fact an asymmetric sequent calculus (in which the sequents have a list of assumptions and a single conclusion) permitting a forward (or Hilbert style) proof system which has some similarities with a natural deduction system (and can be made very similar with the benefit of derived rules).

The full convenience of "backward proof" is then realised using a "sub-goal package". The user starts a proof by passing to the sub-goal package the sequent which he wishes to prove (as a sequent rather than a theorem, since it will not be available as a theorem until after it has been proven). At each step in the proof which follows the sub-goal package presents to the use a single "current goal" and the user nominates to the subsoil package a "tactic" to be applied to the sub-goal. When a tactic is successfully applied to a sub-goal it breaks it into zero or more further sub-goals and supplies with it a rule which derives the sub-goal (as a theorem) from the list of theorems corresponding to the new sub-goals. If a tactic can solve a sub-goal, then its introduces no new sub-goals and offers a proof which requires no theorems to be supplied to it, and which can therefore be invoked by the sub-goal package to obtain the desired theorem. If the user persists in supplying relevant tactics to the sub-goal package until all sub-goals are in this way discharged, the sub-goal package will be able to construct a proof of the original goal by composing the proofs obtained from each of these steps. This proof will be a rule which when applied to the empty list of theorems will compute and return the desired theorem.

Tactics are not infallible, not only may they fail to offer a step in the desired proof at all, they may make an offer which they later fail to redeem. The failure of a tactic to deliver on its promise is a bug in the tactic, and these are rare. Tactical programming is often done by users, and it is very inconvenient to discover an error in a tactic only at the point of completion of a large proof which uses the tactic, not least because the diagnostics available at this point may not be good. This kind of thing did happen in the Cambridge HOL system which we were using before the development of *Product*.

The sub-goaling package design in ProofPower is immune to this problem. Instead of remembering a tree of sub-goals and proof functions the state of the sub-goal package is coded up as a theorem in which the assumptions are (codings of) the outstanding sub-goals, and the conclusion is a coding of the target goal (a constant is used which mimics the semantics of the sequent turnstile). The construction of this sub-goal package state theorem involves invocation of the proof function at the same time as the tactic is invoked so that its failure is detected immediately.

In some versions of Cambridge HOL the desired checking of the proof function takes place at the point at which it is produced using "mk_thm" to obtain the necessary premises. This method however does depend upon this hole in the abstract data type, which might undermine trust in the proof tool.

3.2 Z and Other Embedded Languages

The idea of a semantic embedding of one language into some other language is to treat the first language as if it were syntactic sugar for expressions in the other. The idea is that some capability in respect of the second language is thereby transferred to the first. In this case the primary capability of interest is proof construction and checking.

In the case of embedding the language Z in HOL though in some deep logical sense the languages are closely related, in the superficial matters such as syntax they are worlds apart. A semantically correct embedding for the whole of Z into HOL would be relatively complex, and in all aspects of functionality a proof tool would have to be customised to Z in order to achieve reasonable levels of usability and productivity.

The advantages are few but substantial. An important advantage is that by this means proof capability can be realised where the semantics are known but the proof rules are not known. Soundness of inference is guaranteed by the soundness of the proof system for the target language provided that the embedding is semantically correct. A second advantage is that sharing between the languages of that most precious and costly item, theorems, is maximised. Thus a theory of integer or real numbers developed for HOL will be substantially re-usable in Z, together with domain specific proof automation such as a linear arithmetic prover.

Z was not the only other language for which support might be needed, though it was thought to be the most important (in terms of prospective business), so a generic tool was desirable. A syntactically generic approach might have been adopted, but we decided instead to aim for genericism via embedding.

In this section, as well as describing some of the key features of the support for Z in ProofPower the effect on the underlying core implementation of HOL will be picked out, not only specifically of the need to do Z, but also of the perceived need for a degree of genericism.

document preparation It is the Z style of literate formal specification which has determined that ProofPower is oriented around processing of formal texts extracted from LaTeX documents. The document processing facilities are based around a program called "sieve" which processes documents according to the instructions in a sieve steering file watching for tags in the document which introduce the many different kinds of formal material which may be included in such a document. The same kinds of documents may be processed in different ways for different purpose, for example to yield pure LaTeX for printing, or scripts suitable for loading the formal material into the ProofPower proof tool for processing to enter specifications into the theory hierarchy, to generate and check formal proofs and store the resulting theorems. It is hard now to see how we could have managed without this kind of machinery, but it remains the case that the initial impetus to this manner of working came from Z, and that the academic versions of HOL still work directly from ASCII text files and do not involve themselves in these matters.

term quotations and pretty printing The dialogue with the proof tool takes place through the interactive metalanguage ML. Cambridge HOL had special mechanisms to make invocation of an object language parser straightforward in the form of special quotations marks for this purpose. Metalanguage quoting in object language terms is also supported, allowing the insertion into an object language term of an expression of the metalanguage of type *TERM*.

In ProofPower this kind of facility is extended in two main ways.

Firstly the character set is extended so that the dialogue includes the most commonly used logical and mathematical symbols. This is done by coding up the special characters into strings of characters

which are acceptable in standard ML.

The effect is that not only the quoted object language terms may contain these special characters, but also the ML names, so that ML names may be chosen which directly relate to the symbols in the language, e.g. $\Rightarrow\_elim$, $\forall\_intro$.

The object language quotation facilities in ProofPower include not only the primary object language HOL and embedded ML, but also the Z language, and designed to allow other languages to be added. Full multilingual mixed language parsing and pretty printing is supported. That means that HOL and Z can be mixed together in a single term quotation, fragments of HOL being included inside Z or vice-versa. Of course there are constraints on what is allowed and this is largely controlled by the type system and the injection used by the embedding of Z in HOL of the types of Z into those in HOL. A HOL term quoted inside a Z term must have a type which is in the image of the injection and its position in the surrounding Z term or predicate must be consistent with that type. To make pretty printing of a term possible constants are associated with languages, and this information control the selection of pretty printers for the different parts of a term. Mixed language terms are not normally encountered in Z proofs, the proof facilities are smart enough to keep the proof in Z. However, tactical programming will frequently involve programming transformations which provide an inference within one language using transformations passing through terms which do not belong to that language (the primitive rules of the language will not generally stay in the image of a language embedding).

stripping and proof contexts There is here both another application of the obvious principle that to provide generic multi-language support via semantic embedding one must make most features of the system potentially customisable to particular languages, or, more generally to particular contexts, which include both a position in the theory hierarchy and a language associated with that context, and also an example of a specific generalisation of a feature of HOL88 which arose from need to support Z.

When prototyping of support for Z on the first prototype ICL HOL reached the point of attempting goal oriented proofs using the goal package it was necessary to make the behaviour of *strip_tac* sensitive to the language so that it could handle correctly the Z universal quantifier. As soon as the possibility of making *strip_tac* context sensitive is considered the question what it can beneficially be used for becomes open. ICL were not involved in the early development of the LCF system of which *strip_tac* appears to be a highly used historical remnant. *strip_tac* looks like it is a somewhat incomplete attempt to provide a tactic which knows the basic natural deduction rules for the predicate calculus and automatically applies the rule relevant to the current goal, provided that can be done without extra information (such as an existential witness). This picture makes more sense in terms of the original LCF logic, in which there were fewer logical connectives than there are in HOL. If this idea is thought through it becomes apparent that a natural generalisation of *strip_tac* can be produced which is complete in respect of the propositional calculus (when repeated).

Z is based on set theory, and the natural systematic approach to proof in Z is to characterise each of the set-valued constructs in the Z language extensionally. Set theory used in this way fits well into a natural deduction framework. The simplest example is the handling of intersection. A goal which is a membership assertion of which the right hand expression is an intersection can be transformed into a conjunction of membership statements, and this behaviour is a natural extension to the capabilities of *strip_tac*. In general, if the semantics of the set valued Z terms is given extensionally as an equivalence statement in which an assertion of membership in the construct is said to be equivalent to some formula in which that construct does not occur, then these equivalence claims provide extensions to the stripping behaviour, or to the default behaviour of rewriting facilities. The effect of systematic adoption of these methods is to automate the transformation of quite elaborate expressions in Z's set theory into predicate calculus in which the set theoretic vocabulary has been

eliminated.

4 DAZ and CLAWZ

4.1 The Changing Marketplace

The initial development of ProofPower was motivated by the apparent demand for formal verification against specifications in Z at the extremes of high assurance in secure computing, and by the need for tools to support that process. The development supplied tools which enabled ICL to complete the design and verification of secure systems, and also gave skills to the formal methods team in the development of proof tools in a context in which external contracts for such development were in prospect. By the time that the FST project was complete, the context had changed. The first setback had been that expected open tenders for development of proof tools for use in secure systems development failed to materialise. The developments did take place, but were undertaken under existing formal methods consultancy contracts. The more serious setback was a complete volte-face in government policy on the development of secure systems. The dominant trend in military computing procurement was towards "COTS", Customised Off The Shelf procurements, rather than the more traditional and more expensive development of bespoke systems. For this or other reasons the expected stream of government contracts for the development of highly secure systems dried up.

At the same time the honeymoon period for the Defence Technology Centre had ended, the formal methods team had to make profits.

At the same time as the prospects for formal methods in secure computing were faltering, the application of formal methods to safety critical military systems was being underpinned by Defence Standard 00-55.

The Royal Signals and Radar Establishment at Malvern had pioneered research for the Ministry of Defence in formal methods, and (inter alia) had developed a "compliance notation" which permitted refinement of specifications in Z into programs in a safe subset of Ada. As the FST project came to an end, RSRE, by then part of the Defence Research Agency, put out an open tender for a tool to support the use of their compliance notation in the development of safety critical systems. This would open an alternative marketplace to ProofPower and the ICL High Assurance Team if the contract could be secured and the compliance tool built on ProofPower.