A Mizar Mode for HOL

John Harrison

Åbo Akademi University, Department of Computer Science
Lemminkäisenkatu 14a, 20520 Turku, Finland

Abstract. The HOL theorem prover is implemented in the LCF manner. All inference is ultimately reduced to a collection of very simple (forward) primitive inference rules, but by programming it is possible to build alternative means of proving theorems on top, while preserving security. Existing HOL proofs styles are, however, very different from those used in textbooks. Here we describe the addition of another style, inspired by Mizar. We believe the resulting system combines the secure extensibility and interactivity of HOL with Mizar's readability and lack of logical prescriptiveness. Part of our work involves adding new facilities to HOL for first order automation, since this allows HOL to be more flexible, as Mizar is, over the precise logical connection between steps.

1 HOL

The HOL theorem prover [13] is descended from Edinburgh LCF. While adding features of its own like stress on definitional extension as a reliable means of theory development, it remains true to the basic idea of the LCF project. This is to reduce all reasoning to a few simple primitive (usually forward) inference rules, but to allow a full programming language to automate higher level 'derived rules', broken down into these primitives. For example, HOL includes derived rules for linear arithmetic, tautologies and inductive definitions. Ordinary users can simply invoke them without understanding their implementation, but because they do ultimately decompose to simple primitives, can feel confident in their correctness. Should they need other, perhaps application-specific, proof procedures in the course of their work, they can write them using the same methodology.

This combination of reliability and flexibility is the outstanding feature of LCF systems, and there is usually not a serious loss of efficiency in derived rules [17]. Some of these derived rules may present the user with a quite different view of theorem proving from that implemented in the logical core. Even in the original LCF publication [14] we find the following:

> The emphasis of the present project has been on discovering how to exploit the flexibility of the metalanguage to organise and structure the performance of proofs. The separation of the logic from its metalanguage is a crucial feature of this; different methodologies for performing proofs in the logic correspond to different programming styles in the metalanguage. Since our current research concerns experiments with proof

methodologies – for example, forward proof versus goal-directed proof – it is essential that the system does not commit us to any fixed style.

To some extent, this theoretical flexibility is already a practical reality in HOL. In addition to the basic 'machine code' of forward primitive rules, there are several supported proof styles, all of which fit together smoothly:

- There are numerous more complicated forward proof rules, which can make the business of theorem proving much more palatable than it would be using the primitives. However, before each inference rule is applied, it's necessary to muster all the required hypotheses exactly, and either include their proofs verbatim, or bind them to names and use those. It's very hard to do proofs in this way unless the exact structure of the proof is already planned before starting to type.
- Backward, tactical proof was one of the most influential ideas in the LCF project. Most large HOL proofs are done in this way, perhaps because the required hypotheses appear naturally and determine the proof structure automatically. It also allows more convenient use of local assumptions and choosing variables. This flexibility is further increased if the tactic mechanism allows 'metavariables' whose instantiations can be delayed [32, 26].
- Equational reasoning is one of the most widely used parts of the HOL system, largely thanks to an elegant and flexible implementation [25]. Depth conversions and rewriting tools allow the convenient iterated instantiation and use of equations. There are also straightforward means of handling associative and commutative operators.
- Window inference [29] is a methodology for organizing localized proof efforts. Users may focus on a particular subterm or subformula and transform it, exploiting contextual information that comes from its position in the whole formula. For example, when transforming ψ into an equivalent formula ψ' in the expression $\phi \wedge \psi$, we may assume ϕ. Grundy [15] both mechanized window inference in HOL and generalized it to arbitrary preorder relations, such as implication and the refinement relation on programs.
- Prasetya [27] has written a package to support two features of textbook proofs: the use of a series of lemmas, and the use of iterated equations (we shall have more to say about this latter issue later).
- Specialized decision procedures for various particular domains such as linear arithmetic [6] are also available, as well as a number of derived definitional mechanisms [23, 24].

Most of these styles suffer from being rather low-level, making explicit too many details that are normally elided. More precisely, they are too *logically prescriptive*, demanding that even the most obvious steps be mediated by the exactly appropriate logical rule(s). This isn't just a problem because doing it is tedious. A beginner might well simply *not be able to drive the system well enough* to get it to do the requisite steps. For example, many HOL users find manipulation of assumptions difficult. Decision procedures, on the other hand,

are perhaps too high-level, compressing into one line substantial mathematical detail.

Whether too high-level or too low-level, all the proof styles suffer from one common failing: the proofs are expressed using complicated combinations of arcane higher order functions in a computer programming language. Though it's easy to guess what many of them do, a HOL script looks nothing like a textbook proof. Even HOL experts cannot really read a typical HOL proof without replaying it in a session. Annotating proofs with intermediate theorems, as done by Paul Jackson in Nuprl, certainly makes them more readable. However it also causes them to expand substantially, and gives a rather artificial separation between the proof instructions to the machine and the parts that are intended for human consumption.

2 The Mizar Proof Script Language

The Mizar theorem prover,[1] developed by a team in the Białystok branch of Warsaw University under the leadership of Andrzej Trybulec, is quite different from HOL in almost every respect. It was designed primarily for the formalization of mainstream mathematical proofs rather than for verification applications; it is based on Tarski-Grothendieck set theory rather than simple type theory, and the proof checker is built on entirely different lines. However we believe that its proof script language provides many interesting ideas and lessons. We do not claim it is ideal for all applications; standard HOL styles may for example be better in many verification tasks, or where complex decision procedures are to be used. But for its original purpose of proofs in pure mathematics, it has a lot to recommend it — the enormous amount of mathematics that has been proof checked in the Mizar system stands as a testament to that.

2.1 Presenting natural deduction proofs

Systems of Natural Deduction seem to provide quite a direct rendering of typical mathematical reasoning, including common idioms such as reasoning from assumptions, performing case splits, etc. The actual format of natural deduction proofs as usually presented is, however, rather different from that of textbook proofs. The Mizar proof language improves things by associating deduction steps with English constructs that can be put together into a fairly conventional mathematical proof. For example:

- '`let` x `be` α`;` <*proof of* $\phi[x]$ >' is a proof of $\forall x : \alpha.\ \phi[x]$.
- '`assume` ϕ`;` <*proof of* ψ >' is a proof of $\phi \Rightarrow \psi$.
- '`take` a`;` <*proof of* $\phi[a]$ >' is a proof of $\exists x.\ \phi[x]$.

These and other similar constructs define the 'proof skeleton', i.e. the overall logical structure of the proof.

[1] See the Mizar Web page: '`http://web.cs.ualberta.ca:80/~piotr/Mizar/`'.

2.2 Stepping beyond natural deduction

Though natural deduction captures many mathematical idioms, it is not ideal for every application. For example, equality reasoning is usually done using certain obvious techniques like substitution and rewriting, rather than by explicitly stringing together axioms for the equivalence and congruence properties of equality via natural deduction rules. And at the formula level, it is sometimes more attractive to reason directly with logical equivalence rather than decompose it to two implications [37].[2] In fact in HOL there is already a great emphasis on the use of equivalence: it is just equality on booleans, so all the powerful equational proof techniques like rewriting are available to exploit it.

In fact, one often wants to make very simple logical steps that do not correspond to individual natural deduction rules. Notwithstanding their theoretical interest, natural deduction rules are not sacrosanct. For example passing from $A \vee B$ and $\neg A$ to B (an instance of resolution) is at least as 'natural' as natural deduction $\vee$-elimination, let alone the sequence of ND steps needed for the above inference. In general we might be completely uninterested in the exact series of inferences, e.g. we might wish to pass from $a = 0x$ to $a = 0$ using the theorem

$$\forall x\ y.\ xy = 0 \equiv x = 0 \vee y = 0$$

without writing out a full natural deduction derivation. All this suggests beefing up natural deduction with the ability to make rather simple 'obvious' jumps, and this is precisely what the Mizar system does. The user may write 'ϕ `by` A_1, ..., A_n', meaning that ϕ is considered an obvious consequence of the theorems $A_1, \ldots, A_n$ (these are either preproved theorems or labelled steps in the present deduction).

The body of a Mizar proof contains a list of steps justified with '`by`'; these are usually just formulas, with or without labels, but sometimes skeleton constructs also use '`by`' for their justification. For example '`consider` x `such that` $P[x]$' performs an $\exists$-elimination step; in subsequent steps, $P[x]$ may be assumed. However this requires justification for $\exists x.\ P[x]$, and enough theorems must be provided for this to be deduced. To avoid a proliferation of labels, the previous step may be implicitly assumed by prefixing a step with '`then`'.[3] Finally, certain formulas are prefixed with '`thus`' or '`hence`' (the latter equivalent to '`then thus`'); these are 'conclusions'. The set of conclusions collected in a list of steps should always be sufficient to justify the current objective or *thesis*. For example, if the thesis is $\phi \wedge \psi$, then one might have two conclusion steps containing ϕ and ψ; if the thesis is $\phi \equiv \psi$, the conclusions might be $\phi \Rightarrow \psi$ and $\psi \Rightarrow \phi$. To achieve a kind of bracketing of sets of conclusions, which could otherwise be ambiguous,

[2] Moreover, when doing exploratory interactive work, it is convenient that all equational steps are reversible, so one can feel confident that a provable subgoal is not being replaced by an unprovable one.

[3] The Mizar system makes formulas introduced using certain constructs like '`assume`' available by default without labelling; our version makes this optional, and always allows labelling.

an individual step can be justified not using '`by`', but rather by a whole nested proof enclosed between '`proof`' and '`end`'.

The thesis is tracked automatically by the system as the proof script is processed, starting from the initial goal. For example, if the current thesis is $\forall x : A.\, P[x]$, then after processing a step '`let x be A`' the thesis becomes $P[x]$. For one or two constructs, knowledge of the thesis is necessary; it cannot be constructed from the proof. For example '`take m`' followed by a proof of $m \leq m$ could be a proof of $\exists x.\, x \leq m$ or of $\exists x.\, x \leq x$, among others. Apart from providing the system with additional information in such cases, the thesis is a useful sanity check, since the skeleton structure should correspond to the thesis.[4] Moreover, it allows one to use the special word '`thesis`' rather than repeatedly quote the formula; this is convenient since if several case splits are performed, there will typically be many conclusions '`thus thesis`'. The system attempts to modify the thesis intelligently given a conclusion step. For example if the thesis is $\phi \equiv \psi$ and a conclusion step proves $\phi \Rightarrow \psi$, then the thesis becomes $\psi \Rightarrow \phi$. Mizar allows a nested proof within '`now ... end`', which unlike the nested proofs within '`proof ... end`', does not make the thesis known at the outset. In our HOL implementation we disallow this, since it does not fit very tidily with our reduction to tactics. It could easily be implemented, but would require the entire nested proof to be processed separately.

A few other mathematical idioms are admitted, in particular the kind of iterated equality reasoning whose usefulness we have already noted; our HOL version can handle other binary relations like numeric inequalities too, à la Grundy. In the HOL version, using three dots as the left-hand argument of a binary operator is a shorthand for the previous left-hand argument that was given explicitly, and there is also an implicit '`then`' to link the previous step. So for example one may write:

```
  a = b by Th1,Th2;
... = c by Th3;
... = d by Th4,Th5,Th6;
```

which serves as a proof of $a = d$. The reader familiar with the typical calculational style of proof [12] will see that the use of iterated equality in the Mizar language is almost identical to that, each step in the transitivity chain being justified by a hint, albeit of a rather uniform kind.

In summary, Mizar scripts admit a division into the 'proof skeleton', which uses the special keywords to set out the basic structure of the proof, and the individual steps within the proof, mostly using '`by`' and its relatives. There is thus an attractive combination of a clearly structured natural deduction proof together with flexibility over the individual inferences. Apart from simply making things easier, this might also appeal to the many mathematicians who are uninterested in, or actively dislike, logic and foundations.[5]

[4] This is decidable and in Mizar is checked by a separate pass.

[5] Probably its emphasis on practical usability rather than foundational questions is partly responsible for the amount of real mathematics done in Mizar.

2.3 Stepping beyond Mizar

We have found it useful to add a few other features to the language beyond those included in Mizar itself (conversely of course, we have left out some features of the Mizar language such as its special type coercing functions that don't have a natural HOL counterpart). These new features do not by any means exhaust the possibilities; on the contrary a careful study of existing mathematical textbooks and papers, concentrating more on their form than their content, might reveal many other useful additions.

First, we allow the idiom (when trying to prove X, say): '`suffices to show` X'', usually accompanied by a justification using '`by`'. This construct allows backward proof. Sometimes the steps are quite trivial, corresponding to typical mathematical steps like 'by [induction] we need only prove $P[0]$ and $\forall n.\, P[n] \Rightarrow P[n+1]$', or 'by [symmetry in m and n] we may assume that $m \leq n$'; but they could be more substantial. Mathematics books often contain an admixture of backward proof, and some suggest that the proportion could profitably be increased.

Second, we allow the use of arbitrary HOL rules in justifications. As well as just a list of theorems, the '`by`' command takes an optional identifier for a HOL inference rule. There is a standard default rule, of which more below, but users may use their own in special situations. By contrast Mizar has no facilities for extension with abbreviations for complicated proof idioms (e.g. repeated rewrites), not even a simple macro language as in PVS. So we can see that the traffic of ideas is not all in one direction: here we use HOL to address a weakness of Mizar.

3 Mizar Proofs in HOL

Our initial experiments involved taking a complete Mizar-style proof script and translating it to HOL primitive inferences. However in this way the Mizar proof style becomes decoupled from the others in HOL, whereas one of the attractions of the existing system is that say, forward and backward proof can be intermixed freely. In addition, we believe that another advantage HOL has over Mizar is that its style of interaction is less batch-oriented. In Mizar, the typical style of user interaction is an edit-compile cycle rather like the use of a programming language compiler, possibly processing a fairly large file each time, whereas with HOL one can try out proof steps, see their effect, and either press on or back up and try something else.

3.1 Mizar tactics

Therefore we now attempt to integrate Mizar-style proofs with HOL tactic proofs. If we think of the 'state' of a Mizar proof, that is, the current thesis and the list of facts derived and labelled so far, as the conclusion and hypotheses of a HOL goal, then there is a close relationship between most Mizar skeleton

constructs and certain HOL tactics. Roughly speaking, the relationship is as follows:[6]

Mizar construct	HOL tactic	(Reversed) ND rule
assume	DISCH_TAC	$\Rightarrow$ intro
let	X_GEN_TAC	$\forall$ intro
take	EXISTS_TAC	$\exists$ intro
consider	X_CHOOSE_TAC	$\exists$ elim
given	DISCH_THEN o X_CHOOSE_TAC	$\Rightarrow$ intro and $\exists$ elim
suffices to show	MATCH_MP_TAC	$\Rightarrow$ elim
set	ABBREV_TAC	abbreviation

For our purposes, it is desirable to extend HOL's tactic mechanism with the ability to label assumptions using chosen names. In this way we can associate assumptions in the goal with the appropriate Mizar labels. Such an extension is, we feel, desirable in any case. Manipulation of assumptions is a perennial problem in HOL, since both numbering and explicit term quotation can be sensitive to quite small changes in the proof, necessitating more sophisticated techniques [5]. The change to the HOL sources took only half an hour, mostly just inserting '`snd`' or '`map snd`' in various tactics. The types of goals and tactics change, but these are normally wrapped in aliases anyway when used at a higher level, so there seems little danger of proofs being broken by the change. Given this slight enhancement of the tactic mechanism, we are now quite close to an interpretation of Mizar steps as tactics. Note that the head of the assumption list is considered the 'previous step' in the Mizar sense, and is selected for '`then`' linkage. Unless it is labelled, the next step deletes it from the assumption list, so that just as with Mizar, the previous result only exists ephemerally.

We actually define special 'Mizar tactics' which are very similar to their HOL analogs in the above table. For example, '`MIZAR_ASSUME_TAC`' is just like HOL's '`DISCH_TAC`' except that it checks that the term being discharged is the same as the one given as an argument, and attaches any specified label to that new assumption. If we were attempting to emulate Mizar's ability to process proofs without an explicit thesis, then it would be necessary to make these tactics work even in the event of a mismatch with the thesis; all that matters is that the subsequent proof reconstruction works as intended.

In order to reduce the load of user type annotation, all the Mizar tactics accept preterms rather than terms.[7] The Mizar tactics then typecheck them in the context of variable typings in the current goal. However if there are variables in the goal with the same name but different types, these are excluded, since an arbitrary choice could leave the user stymied. In that case, some annotation may be needed.

[6] ABBREV_TAC is not part of the HOL standard tactic collection, but is much used by the present author; it can be found for example in the code for the reals library.

[7] Readers unfamiliar with HOL preterms can think of them as untyped syntax trees that become terms only after typechecking.

We define special constants that are expanded during Mizar's preterm to term translation. These are '`thesis`' of type '`:bool`', which is expanded to the current thesis, and '`...`', of polymorphic type, which is expanded to the left hand of the previous step. We provide one that Mizar itself does not: '`antecedant`' refers to the antecedant of the current goal. We quite like the idea of adding others, such as the first and second conjunct of a conjunction, and plan to experiment with this.

3.2 Case splitting constructs

It is necessary to deal with the constructs that can split a goal into several subgoals, namely nested subproofs and '`per cases`'. Nested proofs are dealt with simply by using HOL's standard '`SUBGOAL_THEN`' tactic, which sets up two subgoals: the lemma itself, and the original goal with the lemma as an extra assumption. However Mizar's case-splitting construct requires more care. Like '`DISJ_CASES_TAC`', HOL's case-splitting tactic, it performs a natural deduction $\vee$-elimination step. It is used as follows:

```
per cases by <justification>
  suppose X1
  ...
  ...
  hence thesis

  ...

  suppose Xn
  ...
  ...
  hence thesis
end
```

The justification is supposed to be able to prove $X1 \vee \ldots \vee Xn$. Now, as compared with HOL's case-splitting construct, the above does not make explicit at the start how many subgoals will be generated, nor what the eventual disjunctive theorem to justify is. Therefore a direct translation into HOL's corresponding constructs would require processing of the complete construct, and as we've already said, we are keen to allow expansion of every stage of the proof interactively. Accordingly we proceed as follows.[8]

The '`per cases`' is translated into a HOL tactic that simply yields the same subgoal, but with a justification phase which, on receiving a theorem with an additional assumption, tries to prove and so discharge it using the stated justification. Then each '`suppose X`' causes a split into two subgoals, one with assumption '`X`', one identical to the original. The assumption is that the second subgoal

[8] Note that some of the tactics mentioned below are 'invalid' in the LCF sense, i.e. they may not be able to reconstruct the goal from the subgoals.

will in fact produce a theorem with some additional assumption; the justification stage of this tactic performs a HOL '`DISJ_CASES`' step, i.e. $\vee$-elimination. Finally, the '`end`' construct simply proves the goal under an assumption of falsity; this is trivially disposed of by the eventual disjunction justification.

The above scheme works rather well, and allows a direct step-by-step exploration of the Mizar proof using HOL's subgoal package. Note that because of the way '`per cases`' is dealt with, a string of Mizar tactics has itself no structure, and needs to be applied repeatedly to the head of a current list of goals. Interactively, this is done by the goal stack anyway. However to compose a sequence of Mizar tactics, one must not use '`THEN`', which applies its second argument tactic to all subgoals. Nevertheless it is easy to define a variant of '`THEN`' that will package up a sequence of Mizar tactics into a single tactic.

3.3 Parsing

Writing proofs directly using the Mizar tactics is not a big improvement on the readability of standard HOL tactics, even though the structures into which they are organized may be more natural. Instead of that we have a special parser for Mizar texts. Within this, we still use the HOL notation for terms, rather than Mizar's more readable but more verbose alternative. This could be changed if desired. There are actually one or two syntactic ambiguities arising from lumping arbitrary HOL terms together with Mizar keywords. For example '`let`' could either introduce a Mizar step, or be the start of a HOL term; likewise '`L:A`' could either be a labelled term '`A`' or a term '`L`' with type '`A`' These could be cleared up without difficulty if they become troublesome.

In fact, we usually install as the default quotation parser a function that parses Mizar steps and reduces them to a tactic. The only problem with this is that we want to be able to refer not only to labels in the existing derivation, but also to pre-proved theorems. These theorems are all bound to ML identifiers, so to use them inside quotations, it is necessary to use antiquotation, i.e. precede each name by a carat. Since Slind's system for antiquotation [31] is polymorphic, this presents no problems in principle. However since our experiments are being conducted in a version of HOL without antiquotation,[9] we have adopted the temporary solution of pushing all required external lemmas onto the assumption list with appropriate labels. Similarly, we use a global list binding inference rules to names, which allows various inference rules to be used in the same quotation.

4 Proof support

We have dealt with parsing the proof, but have only discussed the processing of the skeleton constructs into HOL inferences. It remains to translate the individual 'obvious' proof steps in the same way. Since we allow essentially arbitrary

[9] This version of HOL is implemented in CAML Light. See [19] for the starting point of this work.

ML code in a '`by`' statement, any HOL rules can be used. However the existing HOL rules are not really capable of emulating Mizar's recognition of logically obvious steps. To maximize the benefits, something of similar scope is required. Mizar incorporates an automatic theorem prover for first order logic which, while not very powerful, has evolved over time to become extremely quick at checking 'obvious' inferences. For Mizar, speed is essential, since as we have already said, the system is normally used in batch mode. For us, high speed is less important, and is unlikely to be achievable anyway. So we can afford to err on the side of making the checker more powerful. We don't attempt to emulate Mizar's own prover, but start from scratch using fairly standard techniques for automated theorem proving. We provide two alternative provers, one based on tableaux, the other on model elimination. Before giving more details, we will make some general comments and look at how formulas are normalized for input to the provers.

Only first order logic?

Mizar's logic is almost first order, but it supports free second order variables, making axiom schemas much more civilized to deal with. (This logic is often facetiously referred to as 1.01^{th}-order logic.) The automated theorem prover that '`by`' invokes is only for first order reasoning. When one wants to instantiate a second order variable, e.g. in induction or set comprehension schemes, a separate command '`from`' is invoked, together with an explicit instantiation for that variable.

In HOL, although higher order *features* are constantly used, many of the *proofs* are 'essentially first order'. We reduce higher order to first order logic in a well-known mechanical way: introduce a single binary function symbol a to represent 'application', and translate HOL's $f\ x$ into $a(f, x)$, etc.[10] Then it is often the case that when a theorem is provable in higher order logic, the corresponding first order assertion is also provable.

Proofs that cannot be done in the first order reduction are those that require the *instantiation* of higher order variables, i.e. the invention of lambda-abstractions. For example, when trying to prove $\forall n.\, n + 0 = n$ by induction, the induction theorem needs to be specialized to the relation $\lambda n.\, n + 0 = n$, or equivalently, to the set $\{n \mid n + 0 = n\}$. There are techniques, mostly based on higher order unification [20], for finding higher order instantiations automatically — for example the TPS system [1] works in this way. Alternatively, it's possible to write down the combinator axioms in first order logic, so that in principle, lambda-abstractions (in combinator form) can be discovered using standard first-order

[10] Constants (necessarily nullary) and variables can translated directly into first order logic constants and variables, and the logical connectives (at least when used in the standard way, using no higher order tricks) can be directly translated. Actually we optimize the above somewhat by using function and predicate symbols directly provided they are always used consistently in a first order way. This is in fact usually the case.

proof search.[11] This was first proposed by Robinson [28], but his system appears to us unsound: since it does not respect types, the Russell paradox could apparently be derived quite easily by applying a fixpoint combinator to the negation operation. Dowek [11] gives a precise treatment, discussing how the type system can be used too.

We elected to accept the fact that certain higher order proofs cannot be found automatically (after all, other HOL rules can be used in '`by`' if necessary). Note however that if the appropriate term is already bound to some function, then just throwing in the definition is enough; the lambda-term is then expressible in a first order way. Effectively this function works like the appropriate combinatory expression. For example, given the theorems:

$$\forall x.\, x \in Ins(s, y) \equiv x \in s \vee x = y$$

and

$$\forall x.\, x \in Del(s, y) \equiv x \in s \wedge x \neq y$$

as well as the defining property of the empty set, then our first order provers are quite capable of deducing the right instantiations to prove:

$$\forall s.\, s = \emptyset \vee \exists x, t.\, s = Del(t, x) \wedge x \notin t$$

Moreover there is no difficulty with *using* lambda-abstractions; we transform each term $P[\lambda x.\, t[x]]$ into $\forall f.\, (\forall x.\, f(x) = t[x]) \Rightarrow P[f]$ automatically.

Another flaw in our system is that we do not preserve type information when translating to first order logic, which may lead to expansion of the search space with type-incorrect unifications, or could even result in proofs that fail when translated back to HOL inferences. In practice, however, this works surprisingly well in the domains we have tried. Much of the reasoning involves one or two types, which are implicitly encoded anyway in most formulas (note that we treat different instances of polymorphic constants, such as equality, as distinct). Providing better higher order and type-correct automation is an interesting research project. In any case, superior theorem-proving tools developed later may easily be hooked into our Mizar system.

Rather than work directly on the HOL term representation, our provers first translate into their own internal representation of first order logic, which is used during proof search. When a proof is found, it's then translated back into HOL. Systems like Isabelle which feature unification of metavariables in the tactic mechanism, can implement these rules very easily and directly. By contrast our approach looks a bit artificial. However it keeps the proof search fast, and this is the speed-critical part of the automated provers. The eventual proof is usually short and can be translated into HOL very quickly. Such an approach has already been used to implement provers rather similar to ours in HOL [21]. We elected

[11] This idea also lies behind the popularity in the first order ATP community of the finite NBG axiomatization of set theory [7] — in exactly the same way a finite set of building blocks replaces an infinite comprehension schema.

to start from scratch rather than use their work, to make it easier for us to experiment with different ideas for proof automation, e.g. the incorporation of equality.

Preprocessing

When the system needs to prove that ϕ follows from assumptions $\psi_1, \ldots, \psi_n$, it begins, as does Mizar and as do most automated theorem provers, by forming the conjunction $\neg\phi \wedge \psi_1 \wedge \ldots \wedge \psi_n$ and attempting to refute it. The first stage is to convert it to negation normal form (i.e. a form where negations are applied only to atoms) and Skolemize it. Skolemization is done by a one-way process, specializing universal variables and introducing ε-terms for existential variables [4]. For example if the initial formula ψ is $\forall x\ y.\ \exists z.\ \phi[x, z]$, then we proceed through $\psi \vdash \exists z.\ \phi[x, z]$ to $\psi \vdash \phi[x, \varepsilon z.\ \phi[x, z]]$, introduce the local assignment $f = \lambda x.\ \varepsilon z.\ \phi[x, z]$ (it can easily be eliminated after refutation), and so get $\psi, f = \lambda x.\ \varepsilon z.\ \phi[x, z] \vdash \phi[x, f(x)]$. The preprocessing phase attempts to split formulas up into separate units as much as possible — in order to refute $\phi \vee \psi$, the disjuncts can be refuted separately. Conjunction is distributed over disjunction in an attempt to maximize this splitting (though this is disabled after a limit is reached, otherwise large tautologies lead to an exponential number of subtasks). Moreover, the expansion of bi-implications as either

$$(p \equiv q) \rightarrow (p \wedge q) \vee (\neg p \wedge \neg q)$$
$$\neg(p \equiv q) \rightarrow (p \wedge \neg q) \vee (\neg p \wedge q)$$

or

$$(p \equiv q) \rightarrow (p \vee \neg q) \wedge (\neg p \vee q)$$
$$\neg(p \equiv q) \rightarrow (p \vee q) \wedge (\neg p \vee \neg q)$$

is chosen to maximize splittability, and thereafter (i.e. after passing a universal quantifier) is chosen to keep the conjunctive normal form short, since one of our provers below uses CNF. (We do not use sophisticated 'definitional' techniques [9], which can give refutation equivalent CNF by introducing variables for all subexpressions, conjoining their definitions and forming the CNF of that.)

Splitting is most useful for proving equivalences: they are decomposed into two implications for the main prover to handle. In some contrived examples the improvement can be dramatic. For example, 'Andrews' Challenge':

$$((\exists x.\ \forall y.\ Px \equiv Py) \equiv ((\exists x.\ Qx) \equiv (\forall y.\ Qy)))$$
$$\equiv ((\exists x.\ \forall y.\ Qx \equiv Qy) \equiv ((\exists x.\ Px) \equiv (\forall y.\ Py)))$$

gets split into 32 independent subgoals, each of which is fairly easy. The problem as a whole, however, is a real challenge for CNF-based systems like the model elimination prover we describe below. On the other hand, our tableaux prover does this kind of splitting as part of the proof process anyway, so the gains from splitting are marginal.

A tableaux prover

Our first automatic prover is a simple tableaux prover, which is essentially a copy of leanTAP [4]. It is extremely simple, but quite fast for moderately simple tasks. The idea of tableau provers is simply to perform backward search for a cut-free sequent proof, discovering variable instantiations by (first order) unification with Prolog-like backtracking. Beyond the limitations on search space already imposed by the underlying sequent calculus, the formulas are processed in a strictly round-robin manner. This means that universal assumptions can be instantiated n times, but only after all others have been tried at least $n-1$ times. The Mizar notion of an 'obvious' inference [30], is that universal formulas are only instantiated *once*. So what we do is quite similar, but we just have a *bias* against re-using formulas, rather than a strict prohibition. Just as with Mizar, one can force multiple use of an assumption by listing it several times in the '`by`' statement. Though from one point of view an artificial hack, this has some resemblance to a mathematical proof where one says for example 'using transitivity twice we get...'. Indeed, the equality-free part of Mizar is not unlike a tableau prover: it reduces the problem to disjunctive normal form (like the splitting of a tableau into separate branches) and then successively instantiates universal formulas until a refutation can be reached by unification with the negation of another formula [36].

The main extension over leanTAP is a simple system for equality handling, which is necessary for many mathematical proofs. (Mizar includes its own equality-handling techniques.) Simply throwing in equality axioms is too inefficient given such undirected usage of assumptions. But dealing with equality in tableau provers is a hot research topic, especially since the key question of simultaneous rigid E-unification has recently been proved undecidable [10]. We chose a rather ad hoc method which nevertheless works quite well in practice. When a literal $P(s_1,\ldots,s_n)$ is processed given a complementary literal $\neg P(t_1,\ldots,t_n)$, we do not merely attempt to unify each (s_i,t_i) pair, but take each inequation $s_i \neq t_i$ and add it to the tableau branch, resulting in n new branches. And each time an inequation is the currently processed formula and there are at least some equations on the relevant branch, the equality-handling rules kick in. These simply search for a proof in equational logic, but cut down on redundancy by imposing strict canonicality requirements on the proof, e.g. that transitivity is applied after all congruence rules and is always chained right-associated, and that symmetry is only applied to axioms or assumptions.

A model elimination prover

As a more heavyweight and powerful alternative to the tableaux prover, we also developed a model elimination (MESON) prover, based on the Prolog Technology Theorem Prover [33]. Such systems work by reducing to clausal form and then further to a set of pseudo-Horn clauses that can be used for Prolog-style backward search. The default search mode is one of our own invention — see [18] for more details and a comparison with other techniques. The MESON prover is

slower than tableaux for simple problems, because of the greater overhead of preprocessing into clauses. But on bigger examples, it usually outperforms tableaux. In particular it has a measure of goal-direction, which makes it practical in problems where large numbers of assumptions (even hundreds) are involved. These would almost certainly fail using tableaux, at least based on such a simple-minded round robin instantiation strategy. In this prover we deal with equality simply by throwing in all the equality axioms; though not dazzlingly efficient, it turns out to be satisfactory in most cases because of MESON's goal-direction. It is not necessary to include congruence axioms for Skolem functions [22].[12]

Because it is more powerful, we usually use MESON, together with the equality axioms, as the default prover. It seems quite a good choice for filling in obvious steps, the criticism being if anything that it is too powerful. (Actually, Tarver [35] also discusses using MESON in a supporting capacity within an interactive prover.) To avoid long delays where a theorem isn't actually provable (e.g. because the user has not supplied all the required assumptions), we place quite a strict limit on the number of inferences performed internally during search. However this isn't nearly as quick as the Mizar prover at detecting impossible goals.

5 Examples

We will now give a couple of examples of proofs in our Mizar format. Both of these just take a few seconds to process. The first is a rather cute predicate calculus fact due to Łoś. In fact, MESON is capable of proving this completely automatically, so a 1-step Mizar proof '`thus thesis`' is sufficient. However the proof search takes rather a long time, and in any case it's more illuminating to see the reasoning involved. The thesis to be established is:

$$\begin{array}{l}(\forall x\ y\ z.\ P(x,y) \wedge P(y,z) \Rightarrow P(x,z)) \wedge \\ (\forall x\ y\ z.\ Q(x,y) \wedge Q(y,z) \Rightarrow Q(x,z)) \wedge \\ (\forall x\ y.\ Q(x,y) \Rightarrow Q(y,x)) \wedge \\ (\forall x\ y.\ P(x,y) \vee Q(x,y)) \\ \Rightarrow (\forall x\ y.\ P(x,y)) \vee (\forall x\ y.\ Q(x,y))\end{array}$$

And the Mizar proof, verbatim, is as follows. Note that no type annotations are needed, as they are all derivable from the initial thesis (which gives all the first order variables type '`A`'). The default quotation parser '`X`' is set to generate Mizar tactics from the script; hence to set up the goal we locally reassert its usual definition. To emphasize the high level of interactivity, and the complete integration with the tactic mechanism, we show how the proofs can actually be entered in a HOL session, single-stepped through using the standard tactic expansion function '`e`'.

```
let X = parse_term in
```

[12] Thanks to Geoff Sutcliffe for pointing out this piece of ATP folklore.

```
g `(!x y z. P x y /\ P y z ==> P x z) /\
   (!x y z. Q x y /\ Q y z ==> Q x z) /\
   (!x y. Q x y ==> Q y x) /\
   (!(x:A) y. P x y \/ Q x y)
   ==> (!x y. P x y) \/ (!x y. Q x y)`;;

e `assume L: antecedant`;;
e `Ptrans: !x y z. P x y /\ P y z ==> P x z by L`;;
e `Qtrans: !x y z. Q x y /\ Q y z ==> Q x z by L`;;
e `Qsym: !x y. Q x y ==> Q y x by L`;;
e `PorQ: !x y. P x y \/ Q x y by L`;;
e `per cases`;;
e `  suppose !x y. P x y`;;
e `  hence thesis`;;

e `  suppose ?x y. ~P x y`;;
e `  then consider a,b such that L1: ~P a b`;;
e `  then L2: Q a b by PorQ`;;
e `  per cases`;;
e `    suppose !x. Q a x`;;
e `    hence thesis by Qtrans,Qsym`;;

e `    suppose ?x. ~Q a x`;;
e `    then consider c such that L3: ~Q a c`;;
e `    then L4: P a c by PorQ`;;
e `    per cases by PorQ`;;
e `      suppose P c b`;;
e `      then P a b by Ptrans,L4`;;
e `      hence thesis by L1`;;

e `      suppose Q c b`;;
e `      then Q a c by Qtrans,Qsym,L2`;;
e `      hence thesis by L3`;;
e `    end`;;
e `  end`;;
e `end`;;
```

Our second example is the fact that a group where the group operation is idempotent must in fact be Abelian.

$$
\begin{array}{l}
(\forall x.\, xx = 1) \wedge \\
(\forall x\ y\ z.\, x(yz) = (xy)z) \wedge \\
(\forall x.\, 1x = x) \wedge \\
(\forall x.\, x1 = x) \\
\Rightarrow \forall a\ b.\, ab = ba
\end{array}
$$

In the HOL version we use the symbol '`#`' for the group operation, after declaring it infix. This time we show the steps all folded together in a single quotation.

```
let X = parse_term in
g `(!x:A. x # x = i) /\
   (!x y z. x # (y # z) = (x # y) # z) /\
   (!x. i # x = x) /\
   (!x. x # i = x)
   ==> !a b. a # b = b # a`;;

e `assume L: antecedant;
   Idemp: !x. x # x = i by L;
   Assoc: !x y z. x # (y # z) = (x # y) # z by L;
   Ident: !x. i # x = x by L;
   Ident': !x. x # i = x by L;
   let a,b be A;
   (a # b) # (b # a) = a # (b # b) # a by Assoc;
                ... = a # i # a by Idemp;
                ... = a # a by Ident;
                ... = i by Idemp;
   then (a # b) = (a # b) # (a # b) # (b # a) by Ident';
            ... = ((a # b) # (a # b)) # (b # a) by Assoc;
            ... = i # (b # a) by Idemp;
   hence thesis by Ident`;;
```

Conclusions

We have shown how another proof style can be added to the HOL system. The resulting system can be argued to combine the best features of HOL's and Mizar's theorem-proving technology. As the examples show, one can produce quite readable proof scripts and have HOL manage the internal decomposition to primitive inferences automatically. In fact, our work fully bears out the remark that 'transforming proofs that are capable of being validated with MIZAR's basic checker into formal natural deduction proofs would be straightforward' [36]. This is another indication of the flexibility and potential of the LCF approach.

We address two weaknesses of HOL: the unreadability of its tactic scripts, and its logical prescriptiveness. At the same time we provide a version of Mizar's proof language which is more interactive and allows secure extensibility. For one computer theorem prover to take ideas from others is in the spirit of the QED project [2], though we do not link actual systems as that project envisages. Experimentation with various proof styles, and experience with other systems generally, would be valuable. Probably there is no unique best style for all application areas, which makes it all the more attractive to allow the intermixing of different styles as we do here.

As well as the initial ease of construction and readability, an important consideration for formal proofs is their maintainability and modifiability [8]. It is interesting to enquire whether Mizar proofs are likely to be better in this respect. Since they are more readable and less sensitive to the precise choreographing of logical steps, they seem better; on the other hand they involve more extensive

quotation of terms, and so could break more easily if these terms change. However by being explicit rather than implicit, they may be easier to change simply by a semi-automatic editing process.

Future work should probably focus on more powerful automation, integrating the type system and higher order instantiations in a more elegant way. For example, we could implement some of Andrews's techniques as HOL derived rules. It would also be worth experimenting with additions to the proof language. Another interesting idea is automatic proof presentation. For example, some readers might find the 'obvious' steps to be unobvious, but it would be possible to record the proof that the machine finds, and incorporate it into the proof script. Perhaps the processed script could be organized into a hypertext format to allow different readers to browse it at different levels of detail [16]. This could be integrated with more general work on producing a readable summary of machine proofs.

Finally, the theorems that get shipped to the automated prover might provide an interesting set of test cases for automated theorem proving. They have the merit of being realistic problems that arise in real proofs, whereas, for example, Andrews's challenge and its ilk are specifically developed with a view to providing problems for current technology. If they are too easy, then one could arrange for intermediate steps in the Mizar proof to be automatically excised until the proofs reach some given level of difficulty.

Acknowledgements

This work was inspired by the Mizar system; I'm very grateful to Andrzej Trybulec and others in Białystok who helped me to understand the system. I owe a great deal indirectly to Bob Boyer, whose energy and enthusiasm for the QED project has helped to bring users of different proof systems together. Comments on an earlier presentation from Ralph Back, Philipp Heuberger and Jockum von Wright were extremely helpful. I have also profited from discussions with Donald Syme. The automated theorem proving part of the work was inspired by Larry Paulson's implementation of MESON for Isabelle [26]. My work was very generously funded by the European Commission under the Human Capital and Mobility Programme.

References

1. P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A theorem proving system for classical type theory. Research report 94-166, Department of Mathematics, Carnegie-Mellon University, 1994.
2. Anonymous. The QED Manifesto. In A. Bundy, editor, *12th International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Computer Science*, pages 238–251, Nancy, France, 1994. Springer-Verlag.
3. M. Archer, J. J. Joyce, K. N. Levitt, and P. J. Windley, editors. *Proceedings of the 1991 International Workshop on the HOL theorem proving system and its Applications*, University of California at Davis, Davis CA, USA, 1991. IEEE Computer Society Press.

4. B. Beckert and J. Posegga. leanTAP: Lean, tableau-based deduction. *Journal of Automated Reasoning*, 15:339–358, 1995. Available on the Web from `ftp://sonja.ira.uka.de/pub/posegga/LeanTaP.ps.Z`.
5. P. E. Black and P. J. Windley. Automatically synthesized term denotation predicates: A proof aid. In P. J. Windley, T. Schubert, and J. Alves-Foss, editors, *Higher Order Logic Theorem Proving and Its Applications: Proceedings of the 8th International Workshop*, volume 971 of *Lecture Notes in Computer Science*, pages 46–57, Aspen Grove, Utah, 1995. Springer-Verlag.
6. R. J. Boulton. Efficiency in a fully-expansive theorem prover. Technical Report 337, University of Cambridge Computer Laboratory, New Museums Site, Pembroke Street, Cambridge, CB2 3QG, UK, 1993. Author's PhD thesis.
7. R. S. Boyer, E. Lusk, W. McCune, R. Overbeek, M. Stickel, and L. Wos. Set theory in first order logic: Clauses for Goedel's axioms. *Journal of Automated Reasoning*, 2:287–327, 1986.
8. P. Curzon. Tracking design changes with formal machine-checked proof. *The Computer Journal*, 38:91–100, 1995.
9. T. B. de la Tour. Minimizing the number of clauses by renaming. In Stickel [34], pages 558–572.
10. A. Degtyarev and A. Voronkov. Simultaneous rigid E-unification is undecidable. Technical report 105, Computing Science Department, Uppsala University, Box 311, S-751 05 Uppsala, Sweden, 1995. Also available on the Web as `ftp://ftp.csd.uu.se/pub/papers/reports/0105.ps.gz`.
11. G. Dowek. Collections, sets and types. Technical report 2708, INRIA Roquencourt, 1995.
12. A. J. M. van Gasteren. *On the shape of mathematical arguments*, volume 445 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990. Foreword by E. W. Dijkstra.
13. M. J. C. Gordon and T. F. Melham. *Introduction to HOL: a theorem proving environment for higher order logic*. Cambridge University Press, 1993.
14. M. J. C. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.
15. J. Grundy. Window inference in the HOL system. In Archer et al. [3], pages 177–189.
16. J. Grundy. A browsable format for proof presentation. In C. Gefwert, P. Orponen, and J. Seppänen, editors, *Proceedings of the Finnish Artificial Intelligence Society Symposium: Logic, Mathematics and the Computer*, volume 14 of *Suomen Tekoälyseuran julkaisuja*, pages 171–178. Finnish Artificial Intelligence Society, 1996.
17. J. Harrison. Metatheory and reflection in theorem proving: A survey and critique. Technical Report CRC-053, SRI Cambridge, Millers Yard, Cambridge, UK, 1995. Available on the Web as `http://www.cl.cam.ac.uk/users/jrh/papers/reflect.dvi.gz`.
18. J. Harrison. Optimizing proof search in model elimination. To appear in the proceedings of the 13th International Conference on Automated Deduction (CADE 13), Springer Lecture Notes in Computer Science, 1996.
19. J. Harrison and K. Slind. A reference version of HOL. Presented in poster session of 1994 HOL Users Meeting and only published in participants' supplementary proceedings. Available on the Web from `http://www.dcs.glasgow.ac.uk/~hug94/sproc.html`, 1994.

20. G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
21. R. Kumar, T. Kropf, and K. Schneider. Integrating a first-order automatic prover in the HOL environment. In Archer et al. [3], pages 170–176.
22. W. McCune. Equality in automated deduction. In T. Dietterich and W. Swartout, editors, *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 246–252, Boston, MA, 1990. MIT Press.
23. T. F. Melham. Automating recursive type definitions in higher order logic. In G. Birtwistle and P. A. Subrahmanyam, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*, pages 341–386. Springer-Verlag, 1989.
24. T. F. Melham. A package for inductive relation definitions in HOL. In Archer et al. [3], pages 350–357.
25. L. C. Paulson. A higher-order implementation of rewriting. *Science of Computer Programming*, 3:119–149, 1983.
26. L. C. Paulson. *Isabelle: a generic theorem prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994. With contributions by Tobias Nipkow.
27. I. S. W. B. Prasetya. On the style of mechanical proving. In J. J. Joyce and C. Seger, editors, *Proceedings of the 1993 International Workshop on the HOL theorem proving system and its applications*, volume 780 of *Lecture Notes in Computer Science*, pages 475–488, UBC, Vancouver, Canada, 1993. Springer-Verlag.
28. J. A. Robinson. A note on mechanizing higher order logic. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 123–133. Edinburgh University Press, 1969.
29. P. J. Robinson and J. Staples. Formalizing a hierarchical structure of practical mathematical reasoning. *Journal of Logic and Computation*, 3:47–61, 1993.
30. P. Rudnicki. Obvious inferences. *Journal of Automated Reasoning*, 3:383–393, 1987.
31. K. Slind. Object language embedding in standard ml of new jersey. Technical Report 91-454-38, University of Calgary Computer Science Department, 2500 University Drive N. W., Calgary, Alberta, Canada, TN2 1N4, 1991. Also appeared in Proceedings of 2nd ML Workshop.
32. S. Sokolowski. A note on tactics in LCF. Technical Report CSR-140-83, University of Edinburgh, Department of Computer Science, 1983.
33. M. E. Stickel. A Prolog Technology Theorem Prover: Implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.
34. M. E. Stickel, editor. *10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Computer Science*, Kaiserslautern, Federal Republic of Germany, 1990. Springer-Verlag.
35. M. Tarver. An examination of the Prolog Technology Theorem-Prover. In Stickel [34], pages 322–335.
36. A. Trybulec and H. A. Blair. Computer aided reasoning. In R. Parikh, editor, *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 406–412, Brooklyn, 1985. Springer-Verlag.
37. J. G. Wiltink. A deficiency of natural deduction. *Information Processing Letters*, 25:233–234, 1987.