# Verbalization of High-Level Formal Proofs

**Amanda M. Holland-Minkley**
Department of Computer Science
Cornell University
Ithaca, NY 14853
hollandm@cs.cornell.edu

**Regina Barzilay**
Department of Computer Science
Columbia University
New York, NY 10027
regina@cs.columbia.edu

**Robert L. Constable**
Department of Computer Science
Cornell University
Ithaca, NY 14853
rc@cs.cornell.edu

**Abstract**

We propose a new approach to text generation from formal proofs that exploits the high-level and interactive features of a tactic-style theorem prover. The design of our system is based on communication conventions identified in a corpus of texts. We show how to use dialogue with the theorem prover to obtain information that is required for communication but is not explicitly used in reasoning.

## Introduction

The problem of generating text from formal proofs has become more important as use of theorem provers in hardware and software verification creates a large body of formal mathematics (O'Leary *et al.* 1994; Rushby 1997; Gordon & Melham 1993). There is also independent interest in the subject of *formalized mathematics* for its own sake (Cederquist, Coquand, & Negri 1997; Jackson 1995). Some automated reasoning groups are putting their formal mathematics on the web but it cannot be searched in its current form by standard web tools. This kind of material may also play a role in mathematics education. In each case, there is interest in making the formal mathematics produced accessible to an audience that is untrained in the formalism's specialized language.

At least two of the theorem provers generating this large body of mathematics, Coq (Paulin-Mohring & Werner 1993) and Nuprl, are used to create formal proofs that are intended to be readable. However, even these readable formal proofs require training to understand. Consequently, the massive amounts of material that experts can create with these theorem provers remains less accessible than it should be. While natural language proofs would increase accessibility, the texts produced must be faithful to the essential reasoning behind these proofs.

In this paper we describe an approach to providing faithful natural language proofs where the high-level nature of the formalism is used to guide the generation process. The high-level proof representation is an accessible source of the information needed in content planning and sentence construction. We also demonstrate the usefulness of allowing the generation tool to query the prover in order to acquire this type of information.

The specific technical question that we consider here is how to generate text from tactic-style formal proofs created by the Nuprl system. A tactic can be thought of as a large inference step that is assembled from many small primitive ones. The Nuprl tactic system was designed to mimic human reasoning. A sample of a formal Nuprl proof is given in Figure 1. We will give evidence that tactic steps correspond approximately with human inference steps. This property of tactic-style proofs will make the structure of the proof easily accessible.

The tactic proof by itself does not necessarily contain all of the information needed to generate good proof text. Some proof content is captured within the general mathematical knowledge of the reasoner and is not included in specific proofs. In addition, there is a gap between the information that is needed by the reasoner to determine that an inference is correct and the information that is needed to communicate to a person why that inference is correct. We will show that if the theorem prover makes available the primitive proof that lies beneath the tactic proof and allows queries from the generation system, we can obtain this additional semantic information.

Based on these new methods, we generated natural texts from a collection of tactic-based proofs. We used the high-level proof structure for discourse planning and used the mathematics communication conventions that we identified from the corpus to determine sentence content in the context of the discourse. Furthermore, we began to identify knowledge needed for natural language communication that is not present in the formal proof and explored how to use the theorem prover's reasoning capabilities to obtain it. The texts we produced show that these methods are effective, at least in the chosen domain of elementary arithmetic.

## Previous Work

Previous work on the task of generating text from formal proofs has focused on translating from low-level

## Nuprl Proof

```
*T ndiff_ndiff
⊢ ∀a,b:ℤ.∀c:ℕ.(a -- b) -- c = a -- (b + c)
|
BY (UnivCD ...a) THEN Unfold ‘ndiff‘ 0
|
1. a: ℤ
2. b: ℤ
3. c: ℕ
⊢ imax(imax(a - b;0) - c;0) =
| imax(a - (b + c);0)
|
BY (ArithSimp 0 ...a)
|
⊢ imax(-c + imax(a + -b;0);0) =
| imax(a + -b + -c;0)
|
BY RWN 1 (LemmaC ‘add_com‘) 0 THEMM
|  RWH (LemmaC ‘imax_add_r‘) 0 THENA Auto
|
⊢ imax(imax((a + -b) + -c;0 + -c);0) =
| imax(a + -b + -c;0)
|
BY RWH (RevLemmaC ‘imax_assoc‘) 0 THENA Auto
|
⊢ imax((a + -b) + -c;imax(0 + -c;0)) =
| imax(a + -b + -c;0)
|
BY RWN 2 (UnfoldTopC ‘imax‘) 0
|  THEN SplitOnConclITE THENA Auto’
|\
| 4. 0 + -c ≤ 0
| ⊢ imax((a + -b) + -c;0) =
| imax(a + -b + -c;0)
| |
1 BY Auto
 \
  4. 0 < 0 + -c
  ⊢ imax((a + -b) + -c;0 + -c) =
  | imax(a + -b + -c;0)
  |
  BY Assert ⌈c = 0⌉ THENA Auto’
  |
  5. c = 0
  |
  BY HypSubst 5 0  THEN Auto’
```

## Definitions

```
*A ndiff     a -- b ==  imax(a - b;0)

*T imax_add_r
⊢ ∀a,b,c:ℤ.imax(a;b) + c = imax(a + c;b + c)

*T imax_assoc
⊢ ∀a,b,c:ℤ.imax(a;imax(b;c)) =
 imax(imax(a;b);c)
```

Figure 1: A sample Nuprl proof

Theorem: For integers $a$ and $b$ and natural number $c$, $(a - -b) - -c = a - -(b + c)$.

Consider that $a$ and $b$ are integers and $c$ is a natural number. Now, the original expression can be transformed to imax(imax($a - b; 0) - c; 0)$ =imax($a - (b + c); 0)$. From the add_com lemma, we conclude imax($-c+$imax($a + -b; 0); 0)$ =imax($a + -b + -c; 0)$. From the imax_assoc lemma, the goal becomes imax(imax($(a + -b) + -c; 0 + -c); 0)$ =imax($a + -b + -c; 0)$. There are 2 possible cases. The case $0 + -c \leq 0$ is trivial. Consider $0 < 0 + -c$. Now, the original expression can be transformed to imax($(a + -b) + -c; 0 + -c)$ =imax($a + -b + -c; 0)$. Equivalently, the original expression can be rewritten as imax($(a+-b)+-c)$ =imax($a+-b+-c; 0)$. This proves the theorem.

Figure 2: Automatically generated text; associated Nuprl proof and relevant definitions shown in Figure 1

formal languages, particularly from natural deduction style formal proofs *without tactics*. In some cases generation was based on a low-level proof even when a corresponding high-level proof was available (Coscoy, Kahn, & Théry 1995). The high-level proof was avoided because each reasoning step in a high-level proof could incorporate trial-and-error techniques or show facts that may not be necessary in the final proof. There was also concern that the set of available tactics varies between theorem proving systems and may change within a system as it is improved. Though the former is a concern for us, the latter does not seem to be a problem to us because we believe that a theorem proving system and its generation component must work together. Furthermore, it was conceded that the higher-level proof encompassed information about the reasoning process that was quite valuable for communication.

When generating text from a low-level formal proof, two major problems arise. The first problem is that if every low-level step is expressed, the text will be too verbose and contain many unnecessary or "obvious" steps. To give an idea of the magnitude of this problem, in the Nuprl system the low-level primitive proof underlying the example shown in Figure 1 has 674 steps in it. The EXPOUND system (Chester 1976) addresses this problem of verboseness by omitting certain specific low-level inference steps whenever they are encountered or indicating their presence while suppressing the details. The generation system for the Coq theorem prover takes the approach of generating text for every step but aggregating the text for multiple proofs steps into a single sentence when the steps are of the same logical form and occur adjacent in the proof (Coscoy 1997). In general, though, each low-level step produces a sentence in the output proof. In the PROVERB system (Huang 1994b), before generation is performed the input natural deduction proof is translated to an intermediate form that combines several low-level inference steps into one larger high-level step that is intended to reflect a human reasoning step. Generation then proceeds using this new representation. This is a time-consuming process that often requires heuristics to determine what information will be omitted.

The second major problem with low-level proofs is determining the order in which to express the steps of the proof (discourse planning). In a low-level proof format, proof steps can be listed in an almost arbitrary order so long as a step that establishes a precondition of another step in the proof occurs before that step in the proof order. Chester (1976) defines a partial order on steps of a natural deduction proof such that a graph can be drawn showing these dependencies between proof steps. From this graph a coherent linear ordering is given to the proof steps. PROVERB gets its proof step ordering by organizing its assertion-level steps into a dependency tree and then using a notion of local focus to determine a path through the tree (Huang 1994b). In both cases, the system has to create a tree structure in order to determine the text's discourse structure.

We find that by using high-level tactic-style formal proofs as input both of the above problems can be solved easily. First, a Nuprl proof is a tree, and it can be traversed in a depth-first manner to obtain an appropriate linear ordering on the proof steps. Also, we will show that each proof step is already of approximately the same size as a human reasoning step; the proof already resembles the assertion level proof that Huang claims is necessary for generating a good text version of a proof.

## The Nuprl Theorem Prover

Since 1983, the Nuprl proof development system (Constable *et al.* 1986; Constable 1997; Jackson 1995) has been used to help people interactively create formal proofs in a theory considered adequate as a foundation for all mathematics, including computational mathematics and programming. The system has been used to produce thousands of proofs. Most of these have been by-products of verifying that hardware and software systems meet formal specifications.

Nuprl was designed to support the kinds of reasoning seen in rigorous mathematical writings in nearly all mathematical fields, especially computer science. So its formal proofs tend to resemble nonformal but rigorous ones, and when users are creating them they are able to imitate what they would produce in less formal contexts.

The Nuprl logic has two separable components, an *assertion language* (sometimes called the logical theory) and a *proof language* (a deductive mechanism). The proof language includes a procedural language for building primitive proofs. The programs for building proofs are called tactics. They can be written as justifications in proofs. The resulting data structure is a *tactic tree*, a sequent proof that allow tactics as justifications (Allen *et al.* 1990). Tactic-style proofs can be represented as a proof tree where each node in the tree represents a current goal and the children of a node are the new subgoals remaining after application of a tactic. Some tactics are programs that assemble a large inference step from many small primitive ones, such as `SplitOnConclITE` in Figure 1. However, some tactics are programs that search for proofs using heuristics. We will see the impact of these search tactics later. The proofs that users see are in this high-level tactic proof language, as in Figure 1.

## Building and Evaluating the Corpus

In choosing to generate texts from tactic-style proofs, we act on the intuition that these proofs are closer to ones that humans create than are low-level proofs. By this, we mean that each tactic step corresponds approximately to an inference that a person would take when proving the same theorem. The steps are at the same level of granularity as human proof steps. The size of a proof step is an intuitive notion that indicates how much reasoning is happening within that step or how much closer to the goal that step takes the proof.

Among practicing mathematicians it is commonly accepted that there are a variety of general proof strategies to which specific steps in proofs correspond. There have been efforts to define and describe these types of proof steps in order to establish principles for creating proofs and writing them effectively (Solow 1982; deBruijn 1994; Constable, Knoblock, & Bates 1984). We pursue a parallel effort and define categories of proof steps that are not only part of the same strategy but are also communicated in the same manner. We call these categories Mathematics Communication Conventions (MCCs). If our claim that tactic steps approximate human proof steps holds, we can associate a single MCC with each step in a formal proof.

In order to verify our intuition that tactic proof steps closely resemble the proof steps which people would use to explain a proof, we followed the traditional generation methodology (McKeown 1985; Lester 1994) and built a corpus of texts generated by people from formal proofs produced by Nuprl. This corpus is based on fourteen formal proofs, all of which prove basic facts about integer arithmetic. We chose to use proofs from a library that was created with readability in mind. The proofs are all simple enough that the mathematics presented was straightforward for the participants to understand. Study participants were asked to read these proofs and then write translations of them into English. The participants ranged from being experts in the Nuprl system to having never used the system or seen its formal language before. However, all participants had at least a basic familiarity with formal languages in general.

A brief tutorial on how to read Nuprl proofs and a guide to the meaning of terms in the proof was available for novice participants to consult while performing the translations. The study was administered electronically on the web with the proofs given in their HTML format

| Type of mapping | # occurrences | % occurrences |
|---|---|---|
| 1-1 | 180 | 64% |
| 1-2 | 52 | 19% |
| 1-3 | 12 | 4% |
| 2-1 | 23 | 8% |
| 3-1, 4-1 | 8 | 3% |
| 1-0 | 4 | 1% |
| 0-1 | 1 | <1% |
| 1-1, 1-2, 2-1 | 255 | 91% |
| 1-1, case split, or analogy | 232 | 83% |

Figure 3: Number and percentage of occurrences of types of mappings from Nuprl proofs to English sentences in corpus; percentage taken out of 280 total mappings

and participants performed the tasks at their own pace. They were not required to translate all of the proofs. In all, there were nine participants in the study; the resulting corpus has 52 translated proofs in it.

Our corpus afforded us the observations we needed to verify our intuition that Nuprl's proof steps closely approximate human reasoning steps in written proofs. The first hypothesis we wanted to check was that Nuprl proof steps and English text proof steps are approximately the same size. To verify this, we compared the English proofs collected in our study with the Nuprl proofs on which they were based. Specifically, we noted whether a single step in the Nuprl proof was reflected in the corresponding English proofs by a single sentence or multiple sentences, or whether it and one or more other Nuprl proof steps were described together in a single sentence.

In our corpus, there were a total of 352 English sentences, and 280 matchings between Nuprl proof steps and English sentences. The statistics we collected on the frequency with which mappings from Nuprl proof steps to English sentences were one-to-one or of another type are shown in Figure 3. We also consider how many of the mappings that are not one-to-one involve a case split or analogous reasoning occurring in the proof. Case splits often involve "set-up" sentences in English texts to indicate what parameter of splitting was used, or to show that a case was finished and the next is being considered. When cases are analogous in a proof, the Nuprl steps in the analogous case are omitted form the English proof.

In summary, we observed that 64% of the proof steps map to exactly one English sentence and that 91% of the sentences either map one Nuprl proof step to one or two English sentences or map two Nuprl proof steps to one English sentence. Furthermore, in looking at the mappings which are not one-to-one, 52% of these are due to a case split or an analogy is occurring in the proof (45% due to case splits, 7% due to analogy).

From these observations we conclude that Nuprl proof steps are approximately the same size as human inference steps. Most of the departures from the one-to-one mapping between Nuprl steps and English sentences are due to case splits in the proof. We conclude that it is practical for a generation system, using a Nuprl formal proof that was built with readability in mind as input, to plan what content to include in each sentence on a node-by-node basis. Given the ease with which case splits can be identified in a proof, it is straightforward to identify cases in which it might be suitable to generate multiple sentences.

Having verified that we can treat tactic steps as approximating human reasoning steps, we would like to define a set of MCCs such that all proof steps whose communication involves the same information presented in the same general structure are grouped together. We did this by first reading the English proofs and collecting together all of the sentences that had the same general content or role in the proof. Having done this, we looked at the Nuprl proof steps that corresponded to these sentences and observed that the reasoning being done was also similar. By using these observations we were able to define a set of ten MCCs (shown in Figure 4) that group together Nuprl proof steps that can be articulated by similar utterances communicating the same content.

In most cases, there was substantial similarity between all but a small number of the English sentences in an MCC. For some MCCs, the similarity among its corresponding sentences in the corpus was so strong that we were able to write a regular expression that generated every one of those English sentences. Therefore, once one identifies an MCC that covers a given proof step, one knows what type of sentence should be generated.

## Overview of the System

Using the information that we collected from the corpus, we have built a simple natural language generation system in order to determine if it is feasible to use the patterns we discerned as the basis for such a system and to identify areas where more sophisticated techniques would be required. Our system is based on the traditional language generation system architecture (McKeown 1985; Hovy 1988). A typical language generator consists of two main components: a content planner, which selects the information from the knowledge base that should be included in the generated text, and a linguistic component, which maps concepts to words and builds an English sentence from them. The linguistic component follows the standard structure:

- a lexical chooser, which determines the syntactic structure of the sentence and the words to realize each semantic role; and
- a sentence generator, here FUF, which builds a syntactic tree, chooses closed class words, and linearizes the tree as a sentence.

**Statement of Goal** — communicates the theorem to be proved — *"Show for* $x, y : Z.|x| = |y|$ *iff* $x = \pm y$*."*
**Variable Declaration** — communicates the names of any variables introduced in the proof and possibly their types — *"Consider any two integers a and b."*
**Case Statement** — communicates that proof will proceed by argument by cases — *"It is either the case that* $a * b = 0$ *or it isn't."*
**Case Consideration** — introduces one of the cases in an argument by cases, possibly stating the assumption made in that case — *"In the first case, assume that* $0 + -c$ *is less than or equal to* $0$*."*
**Trivial Case Consideration** — introduces a case which can be proved trivially in an argument by cases — *"If* $a * b \neq 0$ *the result follows immediately."*
**Contradiction** — communicates that the conclusion is proved by virtue of a contradiction being reached — *"We conclude that* $a = 0$ *and* $b = 0$*, which contradicts our initial assumptions."*
**Analogy** — indicates that two or more cases from a case statement are proved by the same reasoning — *"The second case is symmetric."*
**Inference Step** — communicates that the reasoning proceeds by an inference on the conclusion or one of the hypotheses, a generic type of reasoning not covered by one of the previous categories — *"By the lemma, either* $a = 0$ *or* $b = 0$*."*
**Transformation** — communicates a chain of reasoning steps, usually from the inference step category — *"By lemmas minus_imin and add_com this reduces to imax*$(-a, -b) + -c =$*imax*$(-(a + c), -(b + c))$*."*
**Trivial Step** — an inference step which is either omitted entirely or where "math" or "obviousness" are given as justification — *"It is trivial to see that both sides are equal."*
**Statement of Conclusion** — communicates that the conclusion has been proved, possibly with a restatement of that conclusion — *"Therefore it must be that* $a * b$ *is not zero."*

Figure 4: Mathematics Communication Conventions, with sample sentences from corpus

The lexical chooser is implemented in FUF and represents the lexicon as a grammar following (Elhadad 1993). The main work in this part was building the *domain vocabulary*, which maps concepts in the mathematics domain to words according to the context. The FUF sentence generator, which uses SURGE as its English grammar, was unchanged.

The major effort in the development of the system was the construction of the *content planner*. The content planner determines what information from the Nuprl proof tree should be included in the proof text. Each node in the proof tree represents one step of reasoning. A proof node contains the list of assumptions that are currently active, the conclusion that node is trying to draw, what tactic is applied at that step and the set of children of that proof node. It also contains information that is relevant only for the theorem prover, such as well-formedness information for each expression and information needed to transform proofs into executable code.

However, as we have noted, there is information that is needed for communication that is not stored in the proof tree, either because it is part of the background mathematics knowledge assumed in the system or because it was determined by the proof agent in a manner that did not produce a suitable proof object. This is the information that we must determine by appealing to the theorem prover. Adding this capability to the generation system results in turning the communication between the content planner and the Nuprl system into a dialogue, rather than one-directional communication as is standard in natural language generation systems.

In this section we will show our solutions to two problems that arose while constructing the system. One problem was made easy to solve by having high-level proofs. The other problem was made more complicated because of Nuprl's rich type system. We will point out specific places where dialogue with the theorem prover can help the generation process and describe the type of querying that might take place.

The first of these two problems is how to map the tree structure of Nuprl proofs to a linear structure that approximates human-written proofs. As we described before, the step size and ordering of proof steps are very similar between Nuprl proofs and the human-written versions in our corpus. Taking advantage of this, we traverse the tree in a depth-first manner and translating each proof node to a single sentence (or two sentences, at the start of a case consideration). The major issue to address, then, is how to determine which MCC from Figure 4 covers the step, since the Nuprl proof does not include this information. The MCC will completely define what information should be selected from the Nuprl node for that reasoning step. We discuss MCC identification further below.

The second problem we address on the content planner level is what representation of an object in the proof is most suitable for the current context. Since Nuprl is system with rich type hierarchy, the type of a variable may be a subtype of many other types in the system. As a simple example, a variable $x$ that is a natural number can be thought of as an integer with the restriction on it that $x \geq 0$. We found from our corpus and mathematical literature that the choice of which type to use is highly determined by the context. So given our variable $x \in \mathbb{N}$ we might choose to refer to it as a natural number or as an integer in the text, depending on how it is being used and its surrounding context. This issue will be mentioned again later.

A sample of the output that our system generates is shown in Figure 2. It should be noted that, for technical reasons, the generation system is as of yet unable to import the Nuprl formula display forms and lemma names and these need to be added by hand, though the generation system does indicate where this information should be given. We use the lemma names as they are given in the Nuprl system in order to be consistent with the library structure that already exists and with which the users are familiar.

## MCC Identification

We have developed a set of rules derived from the corpus that identify the MCC of Nuprl reference steps. For each of an MCC's occurrences in the corpus, we analyzed the node structure, its level in the tree, and the node's neighbors in the tree. We found that in some cases a node's local information completely determines its MCC. For example, every node that has more than one child falls in the **case statement** MCC, or one of the two subtypes of this MCC: **contradiction** and **analogy**. Other MCCs require looking at which tactic is applied. For example, to determine if a lemma application has occurred it must be checked whether one of a set of lemma-application tactics were used.

Generally, more analysis than this is required. A step falls in the **contradiction** MCC if after a case split one of the two cases is proved by obtaining false as a hypothesis. For us, this is not information that is necessarily available in the tactic-tree representation of the proof. However, Nuprl also stores a second, low-level representation of the proof: the primitive proof tree. This representation is not generally accessible by Nuprl users. Contradictions can be identified by retrieving the primitive proof tree associated with a leaf of the tactic-proof tree and checking this piece of the primitive proof tree for an occurrence of false as a hypothesis. Because our planning is initiated at the tactic-level, we only have to examine part of the primitive proof tree, thus saving time.

All of the cases described above can be accurately identified by examining features of the Nuprl output. A more difficult MCC to identify is **trivial case**. In general, this is a highly subjective judgement that humans do not agree on. We do not attempt to answer the most general question here of what does and does not qualify as being a trivial reasoning step; this is a significant and difficult open area of research. Instead, we say that a reasoning step is trivial if it follows from the Nuprl tactic `Auto`. This tactic applies simple automated proof search; it is an example of a heuristic. In practice, Nuprl users invoke this tactic to finish off lines of reasoning when the result has become obvious. Because we are dealing with Nuprl proofs designed to be readable we make the justifiable assumption that `Auto` will have been invoked by the Nuprl user only in cases they deemed obvious themselves.

One of the MCCs whose identification requires a more semantic understanding of the proof tree is **analogy**. This occurs when the reasoning used to prove each of the subproofs of a case split is similar, or analogous. What it means for two cases to be analogous is not well defined; again, this is a subjective notion. The naive approach of comparing cases by checking if the same tactics were used in each reasoning step can result in wrong predictions. In some instances a tactic, such as `Auto`, may have a variety of primitive proof trees it can generate. It cannot be determined from the tactic level tree which of the alternatives has been generated. This means that two instances of the same tactic do not need to result in the same reasoning, hence leading to non-analogous cases. Therefore, solely syntactic features of the subtrees are not sufficient to find similarity between trees, and the whole problem requires a solution on the semantic level.

We have developed a method by which Nuprl can be used to solve this problem. Given two trees, the `Analogy` tactic, which runs in the Nuprl system, identifies whether the reasoning method used in the first tree can be "re-run" on the second tree to prove its result. Tactic application of this type is expensive, since `Analogy` tries to repeat the whole proof, and therefore it is unreasonable to invoke this tactic for every proof step. It is necessary for the content planner to examine the syntactic context and only apply the tactic if it is likely that the cases are analogous.

According to an analysis of Nuprl proofs, we find that there is reason to suspect that the `Analogy` tactic will apply if, for each of the first few steps in the two tactic trees, the set of tactics applied are the same and, when one of those tactics is a lemma or definition application, the same argument is given to the tactic in both cases. By varying how many proof steps need to match before running the `Analogy` tactic, the number of times that the tactic is run and the precision with which the analogy MCC is detected would vary. The fewer steps that are compared, the more often the `Analogy` tactic is run and the more instances of analogy would be detected. This is because our technique for saying that two tactic trees are not similar enough occasionally discards trees that would be judged analogous. However, by being conservative in this way we only end up omitting a simplification to the proof; we never add any inaccuracy. When we do claim that two proof trees followed by analogous reasoning, we are certain that this was the case because the line of reasoning from one proof tree was explicitly re-run on the other proof tree. Determining this requires a dialogue between Nuprl and the generation system.

## Variable Type Representation

Consider a proof that introduces three variables $x$, $y$ and $z$, where $x$ is a natural number and $y$ and $z$ are integers. According to our corpus, people often prefer to introduce these variables in the text by saying "Let x, y and z be integers where x $\geq$ 0." Also, if one knows that a natural number variable is introduced in order to be used in a lemma that takes integers as input, it is clearer to state initially that the variable is an integer. In order to solve this problem, one must know about the subtype relations in the system. If one has a static hierarchy of types it is straightforward to check for any such relations. However, Nuprl does not store a type hierarchy and because its type system is dynamic and constantly growing it is not possible to build such a hierarchy outside of the system. This makes the problem of describing the type of a variable more difficult when using output from Nuprl.

Our solution is to start another dialogue with Nuprl to determine whether two types are in a subtype relation with each other. We have designed a tactic for Nuprl which allows us to check this. This tactic allows us to determine for any two variables whether one is a subtype of the other and, if it is, what additional conditions hold for the variable with the more restrictive type that do not hold for the other variable. With this information we can create alterative references for variables that make our text more concise.

This problem becomes much more serious in more complicated domains such as algebra and automata theory where the type hierarchy will have many levels. In these domains, this is an important problem to solve. Because we solve this problem by appealing to the theorem prover, which has a deeper understanding of the type system than our content planner can, we will be able to apply our solution to types of any complexity.

## Discussion

In this paper, we have presented a simple system which uses the structure of a tactic-style formal proof and the Mathematics Communication Conventions defined for such proofs to guide the generation process. We implemented and tested our generation system using a pre-existing Nuprl library about integer mathematics. We ran our generator on 37 proofs and obtained an accurate and readable version of the proof in each case. An example of our output is in Figure 2. More examples of our output can be seen on the web at `www.cs.cornell.edu/home/hollandm/res3_99.html`.

We found that the structure and content of high-level tactic-style proofs offers a useful starting knowledge representation for use in generation. This representation already contains much of the information needed for the generation process. We have been able to identify some of the additional knowledge that is not necessary for the reasoning process, and hence not included in the formal proof, but that is critical for communication. We have shown how this information can be extracted from the proof. As McAllester and Givan (1992) noted, this information affects the ease of understanding mathematical arguments. We intend to explore the question of what additional information is needed for understanding further.

We identified that the advantages obtained by using a tactic-style theorem prover are balanced by some disadvantages not faced when using low-level input proofs. The dynamic nature of the Nuprl type theory led to difficulties in identifying what type reference to use for variables. In addition, because tactics are high-level abstractions of reasoning there is occasionally low-level information that is lost and has to be obtained through the methods we have described here. In some instances the best solution will be to alter the Nuprl system to add the information to the formal proof.

We also observed that sometimes the communication knowledge we need requires mathematical information that is not present in the proof. We suggest that the solution to this problem is to directly use the reasoning capabilities of the theorem prover. This requires an adjustment to the standard generation architecture to allow for interaction between the theorem prover and the generation system. In this way, the content planner does not have to make semantic decisions about the proof. These semantic issues are passed off to the theorem prover, which is better suited to handle them, and heuristics are avoided in resolving these questions. Avoiding heuristics allows our solutions to be fairly domain independent and should be able to apply them as we extend the system.

Our accomplishments here have also helped us verify that the Nuprl developers were successful in their goal to create a theorem prover that reflects the reasoning processes of people.

## Future Work

This work has shown that high-level formal proofs are suitable and even have special advantages as input to a generation system. Having located some of the places where the information in high-level formal proofs is not sufficient for the generation task, and identified the potential for using dialogue between the generation system and the theorem prover to acquire this information, we want to incorporate this dialogue into our system and continue to explore its use in supplying communication information.

One problem that we will focus on is refining our notion of trivial proof to more closely approximate the notion of trivial used by people when they write proofs. We would also like a method for identifying analogous cases in proofs which relies more on determining whether the differences between the cases would seem significant to a reader or not. We believe that solving these problems will require adding a *user model* to our generation system, in order to have a specific type of reader against which our judgements of what type of reasoning is obvious or significant can be compared.

We also want to approach the problem of determining the best form for expressions when logically equivalent options are available; for example, chosing between the statement that "A implies that B implies C" and "A and B implies C". This will require the ability to determine which form is most useful to the reader given subsequent uses of the expression (McAllester & Givan 1992). It will also require the ability to create alternate forms of expressions in general cases, something we would be able to do by querying the theorem prover.

Additionally, we want to verify that our approach affords easy extension to generation from proofs in other domains of mathematics. Specifically, we want to focus on Nuprl's growing library of automata theory. This library is used in a joint project with the Ensemble system for verifying code (Hickey, Lynch, & Renesse 1999). The text generated from these proofs could be used as code explanations.

## Acknowledgments

We would like to thank Yael Dahan-Netzer and Michael Elhadad for their assistance with FUF/SURGE and Michael Elhadad, Lillian Lee, Kathy McKeown, James Shaw, and the anonymous referees for their comments on this paper. We would also like to thank the participants in our study for their part in generating our corpus. This work was funded by NSF grant DGE-9454149 and DARPA grant F30602-95-1-0047.

## References

Allen, S. F.; Constable, R. L.; Howe, D. J.; and Aitken, W. 1990. The semantics of reflected proof. In *Proceedings of the Fifth Symposium on Logic in Computer Science*, 95–197. IEEE.

Cederquist, J.; Coquand, T.; and Negri, S. 1997. The Hahn-Banach theorem in type theory. In Sambin, G., and Smith, J., eds., *Proceedings of Twenty-Five Years of Constructive Type Theory*. Oxford University Press. To appear.

Chester, D. 1976. The translation of formal proofs into English. *Artificial Intelligence* 7:261–278.

Constable, R.; Allen, S.; Bromley, H.; Cleaveland, W.; Cremer, J.; Harper, R.; Howe, D.; Knoblock, T.; Mendler, N.; Panangaden, P.; Sasaki, J.; and Smith, S. 1986. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall.

Constable, R. L.; Knoblock, T.; and Bates, J. 1984. Writing programs that construct proofs. *J. Automated Reasoning* 1(3):285–326.

Constable, R. L. 1997. The structure of Nuprl's type theory. In Schwichtenberg, H., ed., *Logic of Computation*. Springer. 123–156.

Coscoy, Y.; Kahn, G.; and Théry, L. 1995. Extracting text from proofs. *Lecture Notes in Computer Science* 902:109–123.

Coscoy, Y. 1997. A natural language explanation for formal proofs. In Retoré, C., ed., *Proceedings of the 1st International Conference on Logical Aspects of Computational Linguistics (LACL-96)*, volume 1328 of *LNAI*, 149–167. Berlin: Springer.

deBruijn, N. G. 1994. The mathematical vernacular, a language for mathematics with typed sets. In Nederpelt, R. P.; Geuvers, J. H.; and Vrijer, R. C. D., eds., *Selected Papers in Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*. Amsterdam: Elsevier. 865–935.

Elhadad, M. 1993. FUF: the Universal Unifier. User Manual Version 5.2. Technical Report CUCS-038-91, University of Columbia.

Gordon, M., and Melham, T. 1993. *Introduction to HOL: A Theorem Proving Environment for Higher-Oder Logic*. University Press, Cambridge.

Hickey, J.; Lynch, N.; and Renesse, R. V. 1999. Specifications and proofs for ensemble layers. In *Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer.

Hovy, E. H. 1988. Planning coherent multisentential text. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*. Buffalo, N.Y.: Association for Computational Linguistics.

Huang, X. 1994a. *Human Oriented Proof Presentation: A Reconstructive Approach*. Ph.D. Dissertation, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany.

Huang, X. 1994b. PROVERB: A system explaining machine-found proofs. In Ram, A., and Eiselt, K., eds., *Proceedings of 16th Annual Conference of the Cognitive Science Society*, 427–432. Atlanta, USA: Lawrence Erlbaum Associates.

Jackson, P. B. 1995. *Enhancing the Nuprl Proof Development System and Applying it to Computational Abstract Algebra*. Ph.D. Dissertation, Cornell University, Ithaca, NY.

Lester, J. 1994. *Generating Natural Language Explanations from Large-Scale Knowledge Bases*. Ph.D. Dissertation, Department of Computer Science, University of Texas at Austin, Austin, TX.

McAllester, D. A., and Givan, R. 1992. Natural language syntax and first-order inference. *Artificial Intelligence* 56(1):1–20.

McKeown, K. R. 1985. *Text Generation: using Discource Strategies and Focus constraints to Generate Natural Language Text*. Cambridge, England: Cambridge University Press.

O'Leary, J.; Leeser, M.; Hickey, J.; and Aagaard, M. 1994. Non-restoring integer square root: A case study in design by principled optimization. In *International Conference on Theorem Proving & Circuit Design*.

Paulin-Mohring, C., and Werner, B. 1993. Synthesis of ML programs in the system Coq. *Journal of Symbolic Computations* 15:607–640.

Rushby, J. 1997. Systematic formal verification for fault-tolerant time-triggered algorithms. In Meadows, C., and Sanders, W., eds., *Dependable Computing for Critical Applications: 6*. Garmisch-Partenkirchen, Germany: IEEE Computer Society. 191–210.

Solow, D. 1982. *How to Read and Do Proofs: An Introduction to Mathematical Thought Process*. John Wiley & Sons.