Planning Proof Content for Communicating Induction

Amanda M. Holland-Minkley
Department of Computer Science
Cornell University
hollandm@cs.cornell.edu

Abstract

We describe some of the complications involved in expressing the technique of induction when automatically generating textual versions of formal mathematical proofs produced by a theorem proving system, and describe our approach to this problem. The pervasiveness of induction within mathematical proofs makes its effective generation vital to readable proof texts. Our focus is on planning texts involving induction. Our efforts are driven by a corpus of human-produced proof texts, incorporating both regularities within this corpus and the formal structure of induction into coherent text plans.

1 Introduction

The goal of having natural language versions of formal, computer-generated mathematical texts has been driven by the increasing quantity of formal mathematics being produced by a variety of projects. The purposes for these collections varies over pedagogical purposes (Constable, 1996), proving sophisticated theorems (Cederquist et al., 1997), formalizing foundational theories (Huang et al., 1994), and using theorem proving to verify code and hardware (O'Leary et al., 1994; Liu et al., 1999). For all of these purposes, some of the individuals wishing to understand the proofs will not be familiar with the system used to produce the proofs and its specialized syntax. The domain of formal mathematics has a definite need for natural language versions of its objects. The necessity of automatic generation of these texts is clear not only from the large number of formal proofs being produced, but also from the technical expertise required to understand the proofs and transform them to natural language reliably. We focus on producing full, static proofs, such as would be found in textbooks or research publications. Though this prevents the degree of customization available from interactive systems (such as (Fiedler, 2001)), it allows the application of existing natural language search and summarization tools over the collected proof texts of a formal library.

One of the most pervasive and complex proof techniques common to almost every domain of mathematics is *proof by induction*. Induction is used in proofs from number theory to code verification. It is often the first sophisticated proof technique taught in introductory logic courses but is used in the most complicated proofs in both mathematics and computer science. An ability to express induction clearly is central to any effective tool for generating texts from formal proofs.

In this paper, we will lay out some of the complications involved in expressing induction in texts and our proposed solutions. Our focus will be on planning the texts to express this wide-ranging technique. Our examination of induction will be driven by a corpus of human-produced proof texts which employ induction, as well as a commitment to ensuring the validity of the formal proof within the informal proof text. We will describe an approach currently being used to generate texts from formal proofs and how this system is expanded to handle

induction. Finally, we will discuss how our observations about induction may apply to producing texts employing other sophisticated proof techniques such as diagonalization.

2 Background

2.1 Previous Work

Many of the systems actively in use producing formal proofs employ high-level strategies, or *tactics*, to encode proof strategies, guiding reasoning with proof heuristics (Huang et al., 1994; Gordon and Melham, 1993). This is in contrast to systems which produce and present their proofs via the low-level logical rules which establish the system's logical framework. There can be from 10's to 100's of low-level rules needed to perform the same inference as done by a single high-level rule. One can liken this distinction to the gap between high-level programming languages that people program with and low-level assembly languages with a fixed number of simple commands. In many tactic provers, users can define their own tactics or combine tactics, sometimes nondeterministically.

Choosing a representation involves a tradeoff between these two views of formal inferences, one which is limited in form but produces very large proofs with inference steps much smaller than people naturally think in, and one which is arbitrarily complex and heuristic but produces proofs at a more natural level of detail. Previous approaches have concluded that tactic proofs, because of their breadth of application and nondeterminism, can be unreliable as input to a generation system (Chester, 1976; Coscoy, 1997; Coscoy et al., 1995). Working only from a low-level proof, though, requires rebuilding the high-level structure of the proof from the bottom up. The planning process requires extensive mathematical knowledge to deduce how the small steps should be reassembled into natural inference blocks.

It is clear from this work, and our own observations, that a tactic proof alone is not sufficient input to a generation system (Huang, 1994). Our solution, though, is to use the tactic proof to guide the preliminary plan and then use the low level proof to supplement the plan with any missing details and verify its accuracy (Holland-Minkley et al., 1999). Because, in the theorem proving system we are working with,

```
⊢ ∀i:{8...}. ∃m,n:ℕ. 3 * m + 5 * n = i
|
BY D 0 THENA Auto
|
1. i:{8...}
⊢ ∃m,n:ℕ. 3 * m + 5 * n = i
|
BY NSubsetInd 1 THEN Auto
|\
| 1. i:ℤ
| 2. 0 < i
| 3. 8 = i
| ⊢ ∃m,n:ℕ. 3 * m + 5 * n = i
| |
| BY DTerm ⌈1⌉ 0 THENM DTerm ⌈1⌉ 0
     THEN Auto
| |
| DONE
 \
  1. i:ℤ
  2. 8 < i
  3. ∃m,n:ℕ. 3 * m + 5 * n = i - 1
  ⊢ ∃m,n:ℕ. 3 * m + 5 * n = i
  |
  BY D 3 THEN D 4
  |
  ⋮
```

Figure 1: Sample Nuprl proof, as presented in corpus collection study, proving the Stamps Theorem

it is required that tactic inferences be expandable to a sequence of primitive inferences, which are static within the system, we can direct our search of the low-level proof by anchoring it with reference points in the tactic proof.

2.2 Description of Nuprl proofs

Our generation process uses proofs produced, independently of any natural language generation scheme, by the Nuprl theorem proving system (Constable et al., 1986). The structure of Nuprl proofs and the tactics available to prove them were designed with an eye towards consistency with mental models of mathematics, while still maintaining the system's type-theoretic foundation. The formal proofs are structured in trees, where each node con-

tains the current goal to be proved, any current hypotheses, and the tactic to be applied to prove the goal; the directed edges represent the application of tactics to produce subgoals. Leaves store goals which are entirely established by the application of the corresponding tactics, and the root node contains a null hypothesis list and the proof's theorem as a goal. The beginning portion of a Nuprl proof is shown in figure 1.

Tactics make progress on proving the goals of a proof largely through heuristic approaches to searching for proofs or building justifications of specific forms. For example, the `Arith` tactic performs basic arithmetic simplifications while the `Decide x` tactic performs a case split over whether the expression given in parameter `x` is true or not. There are some standard Nuprl tactics, but users can define their own tactics; in general, tactics build proofs by combining many primitive proof rules.

Primitive proof rules are inherent to the Nuprl system and its logic. A tactic node can be decomposed into collection of primitive proof steps ordered in a proof tree of its own, rooted at the goal of the tactic node, and with its leaves being the goals of the tactic node's children. A single tactic node can expand to a primitive proof containing as many as 100 primitive proof nodes.

2.3 Planning from Tactic Proofs

We have observed in previous work (Holland-Minkley et al., 1999) based on evaluation of proof texts about elementary number theory that the structure of a tactic proof can be linearized into the structure of a content plan through a direct mapping of tactic proof nodes to sentences, ordering the nodes in the proof tree via depth first search. Furthermore, determining what content from each node should be communicated can be done by analyzing what proof technique was used in the current inference. The set of common inference types observed in our corpus comprise much the same set of basic inference techniques described by mathematicians as fundamental for creating and writing effective proofs (Solow, 1982; Constable et al., 1984), such as case analysis, lemma application, and arithmetic simplification. We call these inference techniques with their shared modes of expression *Mathematical Communication Conventions (MCCs)*.

Based on these observations, we have built a prototype system which has verified that plausible proof texts can be automatically generated using these techniques. Our system follows traditional generation system architecture (Reiter and Dale, 2000), with our primary focus being a planner built based on the regularities noted above. The lexical chooser was hand-coded in FUF and uses the SURGE grammar (Elhadad, 1993), generalizing the sentences from our corpus corresponding to each of the inference techniques identified.

3 Planning Induction

Having established the ability to build a content planner around mathematical communication conventions (MCCs) associated with common proof techniques, it was clear that, in extending the system to handle more complex proofs, the ability to process induction would be key. Given the system structure as presented above, we would like this extension to only involve detecting this new proof technique and determining its content and verbalization in isolation. However, unlike most other proof techniques which only impact the proof locally, applying induction can impact nodes in the proof tree remote from its initial application. We evaluate a corpus of proof texts to determine if a localized approach to generation can be used or if a more complex planning process, such as used in restructuring low-level proofs in other generation systems, will be necessary. 21 subjects with varying degrees of familiarity with the Nuprl system and formal representations were asked to write textual versions of four different formal proofs involving induction over integers. In total, 77 proof texts were collected.

3.1 Isolating Induction

Our first goal is to identify what effect, if any, a proof node corresponding to induction has on the content plan's structure. We need to determine if the distributed effects of the inductive process can be localized within the proof plan, allowing us to insert induction MCCs into our pre-existing system.

Just as case splitting mimics branching in a program, introducing non-linearity, induction mimics looping, introducing a more complex non-linearity. There is concern that our simple linearization pro-

cess will no longer hold, or that, given the looping nature of induction, it will be reflected throughout the entire subproof below it rather than only locally, thus requiring any generation system handling induction to process proof steps differently in the context of an inductive proof versus out of it.

Fortunately, we find that induction requires a more complex, but still compatible, set of MCCs than other proof techniques examined to date. Hand analysis of the corpus identified those sentences expressing content not predicted by the regularities of the previous corpus, as encoded by our MCCs. In all of the 77 proofs analyzed, the unpredicted sentences communicated content about the inductive aspect of the proof. Furthermore, for all of the proof steps which did not involve an inductive inference or the usage of an inductive hypothesis, the same MCCs as previously identified were adequate to represent the corresponding sentences. We conclude, then, not only that the alterations in the text were due to induction, but also that placing other types of inference in the context of an inductive proof did not change their treatment. This allows induction to be incorporated into the larger generation structure as-is.

3.2 Induction Communication Content

Following this observation, we identify induction MCCs by collecting all sentences in the corpus which communicate the same inductive content. We find the same tight correlation among sentences about induction as was found between sentences about the application of other proof techniques, allowing matching sentences to be generalized to MCCs.

Induction is communicated in three ways: the invocation of induction, the set up of any base cases and step (or inductive) cases, and the application of the inductive hypothesis within a step case, defining the set of induction MCCs: InductionStatement, BaseCase, StepCase, and IHInvocation (see figure 2). 70% of the proofs in the corpus exhibited all of these MCCs, with IHInvocation being omitted most often; 94% of the proofs exhibited both InductionStatement and Base/StepCase MCCs. The question of when IHInvocation should be used will be considered in more detail below.

Given these MCC's, the formal proof can be processed with the existing content planner, adjusted only to (1) detect the application of induction and (2) check whether a hypothesis being applied is the marked induction hypothesis. As the content planner currently must check the type of inference and find all invoked hypotheses, these alterations are easily incorporated into the existing system. Upon detection of induction, the InductionStatement MCC is added to the plan in progress, and the subcases are processed in a depth-first manner (as with case splitting), each introduced by the BaseCase or StepCase MCC. All other nodes in the proof can be planned as before, with the single exception of those proof nodes within a step case which invoke the induction hypothesis, which will substitute the IHInvocation MCC for a general HypInvocation MCC.

InductionStatement{**InductionValue**}
"We proceed by induction on i."

BaseCase{**InductionValue, BaseAssumption**}
"Base Case: $i = 8$"

StepCase{**InductionValue, StepBasis, InductionHypothesis**}
"Assume $8 < i$ and there are m,n such that $3 * m + 5 * n = i - 1$.

IHInvocation{**InductionValue, InductionHypothesis, Instantiations, Conclusion**}
"[...] since by the induction hypothesis there are m and n such that $3 * m + 5 * n = i - 1$."

Figure 2: The MCCs used for induction inferences as a function of the proof content they require from the Nuprl proof, with example verbalizations

Our final problem is to determine which proof node content the planner needs to pass on for realization. Analyzing the corpus texts, we observe two styles of texts used to express induction, which we label *formal induction* and *informal induction*. Formal induction presents induction in a regular, stylized manner, such as is taught in logic courses or is shown in introductory textbooks. The InductionStatement verbalization clearly states that induction is being used, and identifies the quantity over which induction is being performed. BaseCase's are always identified as such, as is the StepCase, which includes a statement of the induction hypothesis. Fi-

Formal induction:	Informal induction:
Proceed by induction on n. For the base case, n = 0, fib(n) = 1 and fib(n+1) = 1, and the GCD of 1 and 1 is 1. For the inductive case, assume that the GCD of fib(n-1) and fib(n) is 1. ... Now, by the induction hypothesis, the GCD of fib(n) and fib(n) + fib(n-1) is also 1, by "gcd_p_shift". ...	Proceed by induction on n. For n = 0, the GCD of fib(n) and fib(n+1) is 1. Now consider n>0 and assume the result is true for n-1. ... Note that since the GCD of fib(n) and fib(n-1) is 1, the GCD of fib(n) and fib(n) + fib(n-1) is also 1, by "gcd_p_shift". ...

Figure 3: Sample texts showing the same inductive reasoning using both the formal and informal proof text styles

nally, IHInvocation always states that the hypothesis being applied is the induction hypothesis.

In contrast, informal induction is abbreviated. The InductionStatement remains the same as in formal induction. However, the BaseCase and StepCase are no longer necessarily labeled as such. Most significantly, the InductionHypothesis is not explicitly stated in the StepCase. Rather, it is implied by the quantity over which induction is performed, and then stated later in the IHInvocation. Examples of both styles of induction are shown in figure 3.2.

Across the corpus, 48% of the texts used formal induction, and 40% used informal induction. The remaining 12% of the proofs account for other combinations of these four MCCs, many of them in proofs with major inaccuracies. Considering individuals' backgrounds, we found a correlation between the style used and the familiarity of the individuals with formal proofs. Only subjects with Nuprl expertise showed a preference for informal induction, using this style in 69% of their proofs, whereas the novice subjects used both formal and informal induction equally, each 43% of the time. Given that even our "novice" subjects reported high expertise in mathematics and moderate familiarity with formal languages, we conjecture that the informal style is more natural if one is more confident with the proof content, and the formal style helps lend structure and anchor points to complex proofs which may be difficult to understand. Unfortunately, anecdotal experience suggests that truly novice subjects cannot understand a formal mathematical proof sufficiently to involve them in a similar corpus collection study.

4 Future Work

Induction is the first proof technique we have worked with in which multiple styles of expression occurred in corpus texts. With simpler proof techniques, there appears to be great consensus as to the norms for expressing their content. Work is underway on extending our system to incorporate the formal style of induction, but our results suggest that for some potential readers, proofs employing the informal induction style would be preferred. We hope to be able to use different MCCs to express these differences, such as FormalInductionStatement versus InformalInductionStatement, and adjust the planner to account for the current user model as the formal proof tree is traversed and planned, avoiding having to entirely recode the planning and lexical choice components for each model.

We would also like to use our experience in handling a complex proof technique to process more new techniques Proof by diagonalization offers many of the same challenges as induction, potentially combining information from many points in the proof, while still being easily identifiable formally. As a significant challenge, we would also like to be able to extend the system to identify and communicate proof by analogy, wherein one branch of a case split has its proof truncated and replaced by a citation that the reasoning is analogous to a previously communicated branch. This technique presents not just a significant challenge in choosing how to communicate what aspects correspond, but identifying what degree of similarity, to a reader, would appear analogous is unclear.

5 Conclusions

Through our analysis, we have shown how the proof technique of induction, central to any mathematical text generation system, can be isolated within a proof and incorporated into a proof plan without requiring major changes to the handling of other proof techniques. The expansion to handle induction not only significantly improves the text range of our system, but indicates an ease of expansion to other techniques within the mathematics domain.

Acknowledgements

I would like to thank Robert Constable, Lori Lorigo, and Regina Barzilay for their assistance with this work, and the study participants for their time and effort in producing this corpus. I would also like to thank the anonymous referees for their comments on this paper. This work was funded by ONR Grant #N00014-01-1-0765.

References

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

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

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

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

Robert L. Constable. 1996. Creating and evaluating interactive formal courseware for mathematics and computing. In M. Iskander, M. Gonzalez, G. Engel, C. Rushforth, M. Yoder, R. Grow, and C. Durney, editors, *Frontiers in Education*, Salt Lake City, Utah, November. IEEE.

Yann Coscoy, Gilles Kahn, and Laurent Théry. 1995. Extracting text from proofs. *Lecture Notes in Computer Science*, 902:109–123.

Yann Coscoy. 1997. A natural language explanation for formal proofs. In Christian Retoré, editor, *Proceedings of the 1st International Conference on Logical Aspects of Computational Linguistics (LACL-96)*, volume 1328 of *LNAI*, pages 149–167, Berlin, September23–25 . Springer.

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

Armin Fiedler. 2001. *P.rex*: An interactive proof explainer. In Rejeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning — 1st International Joint Conference, IJCAR 2001*, number 2083 in LNAI, pages 416–420, Siena, Italy. Springer Verlag.

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

Amanda M. Holland-Minkley, Regina Barzilay, and Robert L. Constable. 1999. Verbalization of high-level formal proofs. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*. AAAI, July.

Xiaorong Huang, Manfred Kerber, Michael Kohlhase, Erica Melis, Dan Nesmith, Jörn Richts, and Jörg Siekmann. 1994. Ω-MKRP: A proof development environment. In Alan Bundy, editor, *Proceedings of 12th International Conference on Automated Deduction*, pages 788–792. Springer-Verlag.

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

Xiaoming Liu, Christoph Kreitz, Robbert van Renesse, Jason Hickey, Mark Hayden, Kenneth Birman, and Robert Constable. 1999. Building reliable, high-performance communication systems from components. In *17^{th} ACM Symposium on Operating Systems Principles (SOSP'99)*, volume 34 of *Operating Systems Review*, pages 80–92.

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

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems*. Cambridge University Press.

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