A symbolic model of proof acquisition in ACT-R

Beckett Morris[1], Kerstin Haring[2]
[1]Student Contributor, University of Denver
[2]Faculty Advisor, Department of Computer Science, University of Denver

Abstract

Learning to construct mathematical proofs—formal arguments demonstrating the truth of a mathematical statement using logical deductions and previously established facts—is one of the most challenging skills in STEM education. This research aims to build the foundations for a symbolic cognitive model, using the ACT-R cognitive architecture and implementing in Python with the pyactr package, to explore how different proof strategies can be thought through with only symbols and rules. The model observes simple proofs, and its abilities are assessed based on its generalization capabilities, efficiency, and error patterns. By developing and analyzing such a model, this research provides new insights into how symbolic reasoning skills are acquired, offering a detailed case study of proof acquisition within a cognitive architecture.

Keywords: cognitive architecture, mathematical proof, symbolic learning, educational psychology

1 INTRODUCTION

Learning to construct mathematical proofs—formal arguments demonstrating the truth of a mathematical statement using logical deductions and previously established facts—is one of the most challenging yet crucial skills in STEM education. Students on the cusp of advanced mathematics and computer science can find the mental switch from applied to abstract reasoning difficult[1,2,3]. While the task of proving may appear simple, the mental gymnastics required to logically flow from statement to conclusion can be tricky, especially at the beginning of such study[2]. This reality raises the following questions: How can proofs be distilled down from the learner's perspective? How can proof acquisition be modeled within a cognitive architecture—a framework designed to model human cognition—, so as to gain insight into both the mind of the student and the optimal strategy of the instructor? This work begins prodding such questions.

In some sense, proofs can be viewed as purely symbolic tasks, made up of a set of symbols that can be manipulated with a set of rules. That is, proofs are often syntactic and procedural. This view translates well into the modeling of proof acquisition in a cognitive architecture. Cognitive architectures are computational implementations of a theory of human cognition. They define, within software, the structures for systems like memory and learning that align with their underlying theory. Unlike more targeted forms of artificial intelligence, cognitive architectures are designed to exhibit broad and flexible intelligence. Indeed, the goal of cognitive architectures is "to find the mechanisms and representations that specify a formal foundation for a unified theory of cognition"[4]. In particular, the cognitive architecture ACT-R attempts to best simulate human cognition by combining a declarative memory, which stores factual information like symbols and their definitions, with a procedural memory, which stores rules as if-then statements. ACT-R therefore provides the foundation to model the incremental acquisition of mathematical proof strategies from both a symbolic and cognitively plausible perspective.

There has been little prior exploration of the symbolic modeling of proof acquisition in cognitive architectures, ACT-R or otherwise. However, research regarding the pedagogy of proof, theorem provers, and other tasks posed in cognitive architectures provides some footing.

To start, the steep learning curve associated with mathematical proof is well-documented[2,3]. Though students may possess "a factual knowledge of a mathematical concept," the task of using the concept in a proof, or proving the concept itself, does not easily follow[2]. For instance, Knobelsdorf and Frede[3] emphasize a sense of confusion among undergraduates in their first proof course: Students "were confusing [emphasis added] premises with conclusions and frequently asked what was given and what they were supposed to prove" and "were using wrong mathematical nota-

tions and seemed to be uncertain or confused [emphasis added] about what information they already had and what they had to prove". Knobelsdorf and Frede[3] successfully diminished students' confusion by supporting students with the theorem prover Coq, which offered proof scaffolding and immediate feedback.

Coq, now rebranded as Rocq, is representative of a larger group of interactive theorem provers (ITPs), also known as proof assistants, which are capable of proving involved mathematical theorems and verifying complex software systems[5]. ITPs like Coq define type systems and proof languages capable of mechanizing mathematical reasoning with human guidance[6]. ITPs, and their automated theorem prover cousins that minimize the role of the human prover, testify to the possibility of formalizing mathematical proof structure and syntax, although they do so in a way that does not attempt to mimic the human learning process.

Prior experiments in cognitive architectures seek the cognitive realism that theorem provers do not, though none have examined the acquisition of mathematical proof strategies. For example, ACT-R has been used to model diverse domains such as driving behavior[7,8,9], pilot performance[10], and solving puzzles like the Tower of Hanoi[11].

Despite an intense focus on tangential topics, from the educational psychology of mathematical proof to theorem provers and other ACT-R experiments, no investigation into the cognitive modeling of proof acquisition exists—an investigation with broad implications for learners, teachers, and the fields of education, psychology, and mathematics. The present work lays the foundation for such an investigation. To follow, I discuss the topic of mathematical proof, the ACT-R cognitive architecture, and an overview of the development of a model within ACT-R to simulate the acquisition of proof strategies. I then evaluate the results of the model in its current, primitive state. Finally, I suggest steps for further developments in this area.

2 BACKGROUND

2.1 The Pedagogy of Mathematical Proof Productions

Proving in mathematics is a difficult skill, involving "a complex interaction between rigorous and intuitive thought"[2]. In pedagogical research, proof techniques are split into procedural proof productions, syntactic proof productions, and semantic proof productions. Procedural proof productions involve the rote application of a procedure. This repetitive, algorithmic approach requires little engagement with the proof content on the part of the prover. Syntactic proof productions involve manipulating definitions and symbols. As such, syntactic proofs are often dubbed as simply "symbol pushing." While these types of proofs can be involved, they tend to require more effort to keep track of syntactic details than to massage an abstract representation of the concept at hand. Moreover, the number of proofs that can be constructed procedurally and syntactically are limited, and their convincing power can be weak[2].

Semantic proof productions are a more dynamic form of proof, and their use has been shown to distinguish high-level mathematicians from undergraduate students[1,2]. Semantic proof productions require the prover to first intuit why a statement is true before harnessing such intuition to formalize a proof. This necessitates spontaneity, instinct, and an ability to translate between the informal and the formal—qualities that less experienced mathematicians may not yet possess[1]. For this reason, mathematics education typically places an early emphasis on procedural and syntactic proof productions. Procedural and syntactic productions familiarize students with foundational reasoning patterns, build skills in proof writing, and provide a basis for more sophisticated mathematics, including a natural shift toward semantic proof productions.

Given that the cognitive model to come concerns learners' initial acquisition of proof strategies, it is constructed to focus on procedural and syntactic proof productions and not on semantic ones. However, it is important to recognize that educational psychology research reveals what Weber deems a "pedagogical problem"[1]. That is, "There is a danger that if undergraduates only write" procedural and syntactic proofs "and do not reflect on their proofs and proving processes, then the act of proving may not be effective at promoting understanding"[1]. In sum, the procedural and syntactic proof productions examined within the present cognitive model serve as sound introductory methods into proof crafting, though it should be noted that semantic productions play a critical part in reaching mathematical maturity.

2.2 Cognitive Architectures and ACT-R

Cognitive architectures emerged as part of research into artificial general intelligence in the 1950s, seeking to model the human mind and thereby step closer toward a computational replica of human-level intelligence[12]. Cognitive architectures chase cognitive realism, which distinguishes them from other intelligent systems like theorem provers and large language models.

Not all cognitive architectures agree upon the mechanisms that most accurately model human intelligence, however. Cognitive architectures divide into three categories: symbolic architectures, emergent architectures, and hybrid architectures. Symbolic architectures represent cognitive processes as manipulations applied to symbols. The mechanistic rigidity of symbolic systems makes them well-suited to planning and reason-

ing tasks but limits their ability to adapt to a changing environment[12]. Emergent architectures provide the adaptability that symbolic systems lack via a network of parallel models where information propagates through signals, akin to neural networks[12]. These systems are easier to design, but they require (expensive) training and can lack transparency[12]. Hybrid architectures seek the middle ground by merging elements from both symbolic and emergent architectures, though "there are no restrictions on how the hybridization is done and many possibilities have been explored"[12]. According to Kotseruba and Tsotsos's review, hybrid architectures are the most abundant[12].

ACT-R, which stands for "Adaptive Control of Thought—Rational," is one such hybrid architecture. ACT-R is the result of over 40 years of computational and empirical research and was developed and is maintained by members of Carnegie Mellon University's Department of Psychology[13]. The ACT-R framework centers around the integration of two memory modules: procedural memory and declarative memory. Procedural memory holds skill-based knowledge encoded as if-then pairs, while declarative memory contains factual knowledge implemented as chunks, i.e. slot-value pairs. In addition to these two memory modules, ACT-R includes an imaginal module, which holds temporary internal representations, a goal module, a motor module, and a visual module[13]. With the exception of the perceptual memory module, which is not explicitly accessible, ACT-R accesses its modules via dedicated buffers, each capable of holding one representation at a time[14]. At any given moment, the content of these buffers "represent the state of ACT-R at that moment"[14].

To progress from state to state, ACT-R relies on pattern matching and production execution. The pattern matcher first searches for production rules held in procedural memory that match the current state of the buffers[14]. Only one matching production can fire at a time. If multiple production rules match, conflict resolution takes place. All matching production rules are ordered based on their expected utility, which measures the likelihood and associated costs of completing the current goal after executing the production in question[15]. The production rule with the highest utility is not always the rule chosen, however. ACT-R adds noise to explore more of the solution space, ultimately better approximating human cognition. Once a production rule has been selected, ACT-R fires the chosen rule in production execution.

Another critical process, called production compilation, may occur during production execution, as well. Production compilation involves the creation of new or the combination of existing production rules. ACT-R's production system, which monitors the sequence of production rules, enacts production compilation in scenarios where the model repeatedly fires the same sequence of rules. In essence, production compilation reflects psychology's Hebb's law: "Neurons that fire together wire together." Through the process of pattern matching, production selection, and production execution—including production compilation—, cognition in ACT-R "unfolds as a succession of production firings"[14].

It is important to distinguish between the cognitive architecture and the model that implements it. Cognitive architectures "supply a general theory of cognition that is independent of particular phenomena, to which the modeler adds representations to perform a specific task. The architecture is then simulated on a computer to produce behavior" through the model[4]. The architecture's results are therefore reliant upon the model, which is reliant upon a given knowledge base and any necessary inputs. The model to come, for instance, is given several worked training proofs and definitions to learn from.

3 METHODS

3.1 Model Setup

Before implementation, I first selected training proof tasks and outlined the model's design. I chose training proofs with the goal of testing a range of reasoning strategies and utilizing a range of concepts, all at a level suitable for an undergraduate student in an introductory proof course. The training proofs are summarized in Table 1.

After selecting five training proof tasks, I developed annotated steps for each proof. For example, the annotated proof for the fifth induction proof task is as follows:

1. Base case: $n = 1, 1 = \frac{1(1+1)}{2}$.
2. Inductive hypothesis: Assume true for $n = k$.
3. Show for $n = k + 1 : \frac{k(k+1)}{2} + (k + 1)$.
4. Combine using common denominator: $\frac{k(k+1)+2(k+1)}{2} = \frac{(k+1)(k+2)}{2}$.
5. Therefore holds for $n = k + 1 \rightarrow$ holds for all n.

The next setup step surrounded the model's design. I began by defining the model's goal structures and declarative memory chunks, including the chunk types and slots. Goal structures, or goal chunks, indicate the current problem-solving state and are held within ACT-R's goal buffer. Each goal structure represents an individual task or subgoal, guiding the model's behavior by specifying its current objective and strategy. The model's general goal structure is given in Figure 1.

In terms of declarative memory, I identified a need for three chunk types: definitions to encode formal definitions of a concept, rewrites to represent algebraic identities, and lemmas to store general and learned results. The slots for these chunks are shown in Figure 2.

Training proof task	Strategy	Key concepts
If n is even, then n^2 is even.	Direct	Definition of even, algebraic substitution
$\sqrt{2}$ is irrational.	Contradiction	Rational numbers, properties of divisibility
If n^2 is odd, then n is odd.	Contrapositive	Definition of odd, logical equivalence
Every integer is either even or odd.	Cases	Modular arithmetic, mutual exclusivity
$1 + 2 + ... + n = \frac{n(n+1)}{2}$	Induction	Base case, inductive step, hypothesis application

Table 1 Selected training proof tasks.

```
(isa goal
     type        <type>          ; kind of task (e.g., prove, apply-rule, etc.)
     target      <statement>     ; main statement to prove
     method      <proof-method>  ; approach (e.g., direct, contrapositive, etc.)
     context     <givens>        ; known assumptions or givens, will evolve dynamically
     status      <state>)        ; current state (e.g., in-progress, completed, etc.)
```

Figure 1. General goal structure.

I accordingly defined the declarative memory chunks needed for the training proofs. These included definition chunks for the definition of even, odd, and rational; a rewrite chunk for the square of a product; and lemma chunks stating that "if an integer n is even, then n^2 is even" and its converse. Further chunks should ideally be added as the model learns new patterns and generalizations.

```
(isa def
     name             <definition-name>
     definition       <definition>)

(isa rewrite
     expression       <initial-expression>
     result           <rewritten-expression>)

(isa lemma
     name             <lemma-name>
     assumption       <assumptions>
     conclusion       <conclusions>)
```

Figure 2. Chunk types and slots for definition (def), rewrite, and lemma declarative memory chunks.

The final model setup step involved developing the production rules for the model's procedural memory. These if-then style rules drive the model's problem-solving behavior. As previously mentioned, ACT-R is capable of creating new or combining existing production rules through production compilation. To begin with, however, I initialized the model's procedural memory with a set of tailored production rules emerging from the annotated training proofs; each logical step in the training proofs became an explicit production rule.

3.2 Initial implementation

After determining the model's training tasks and initial goal chunks, declarative memory, and procedural memory, I followed Dotlačil and Brasoveanu's (2018) pyactr workflow to showcase how such a model could be implemented.

1. **Create chunks**
   The first task in Dotlačil and Brasoveanu's (2018) workflow requires creating chunks. I did so for both goal and declarative memory chunks. I defined the chunk types with pyactr's chunktype method and then used the chunkstring method to define specific chunks, following the types and slots outlined previously.
2. **Create model**
   Dotlačil and Brasoveanu's (2018) second task involves creating the model and its modules. This meant first instantiating an instance of pyactr's ACTRModel class. Several subtasks follow:
   - 2.1 **Store chunks in declarative memory**
     I then added the declarative memory chunks defined in step one to the newly instantiated model's declarative memory using pyactr's decmem.add method.
   - 2.2 **Create extra modules and buffers**
     This step was not applicable, as I am only using the memory modules and goal buffer.
   - 2.3 **Create production rules**
     To create production rules, I used pyactr's productionstring method and defined production rules following each of the five annotated training proof tasks.
   - 2.4 **Create environment process**
     An environment process, or simulation, is required to run the model[16]. I created an environment process with pyactr's ACTRModel class's simulation method.
3. **Run the model with parameters of interest**
   In the case of the present model, the parameters of interest included the five training proof goals, added to the goal buffer. I completed 20 trials of each training proof, recording the step count and elapsed time for each trial. It is important to note that I repeated the 20 trials in the same session

to avoid disturbances caused by ACT-R's native accumulation of symbolic learning.

4. **Read off behavioral data from the simulation** Dotlačil and Brasoveanu's final workflow step involves gathering behavioral data from the simulation[17]. In this model's case, the behavioral data included the step counts and elapsed run times for each of the training proof tasks. Results from the simulations are discussed in the following section.

4 RESULTS

I propose three pillars upon which to evaluate the model both now and with future progressions. The first pillar concerns the model's generalization capabilities: Can the model solve novel but structurally similar proofs using learned strategies? The second concerns the model's efficiency: Does the number of cognitive steps or total time required to complete a proof decrease with training? The third concerns the model's error patterns: What kinds of mistakes occur, and how do those mistakes change over time? Given that the model outlined above serves as a starting point, it scores poorly across all three pillars.

4.1 Generalization capabilities

In its present "setup state," the model's generalization capabilities are poor. Production rules are hyper-specific to each training proof. The only general production rule, in fact, is that of clearing the goal buffer once a goal's status has been marked as complete. Additionally, declarative chunks are added to the retrieval buffer manually. These characteristics result in an inflexible model.

4.2 Efficiency

The model shows no efficiency improvements on any of the five training proofs in terms of step counts, which would ideally decrease with training as the model learns to "chunk" productions together in production compilation. However, as exhibited in Figure 3, the elapsed times for four of the five training proofs show downward trends, as measured across the 20 trials. The contradiction proof was the outlier, resulting in an upward trend. Notably, the contradiction training proof has the highest elapsed time generally, as well. A potential reason for this proof's increasing elapsed time trend could be its retrieval-heavy nature, as the contradiction proof requires the most retrievals out of the five training proofs (Figure 4). Another important observation regarding the model's efficiency is the significant noise in elapsed time across trials for all five training proof tasks (Figure 3). This noise likely results from ACT-R's native subsymbolic characteristics that seek to simulate human-like variability.

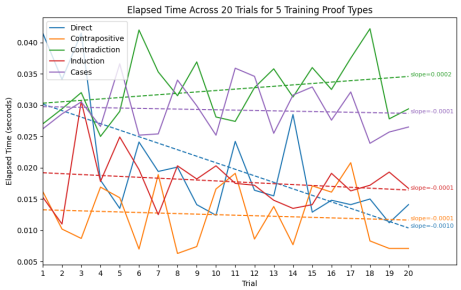

Figure 3. Trends in elapsed times for each training proof.

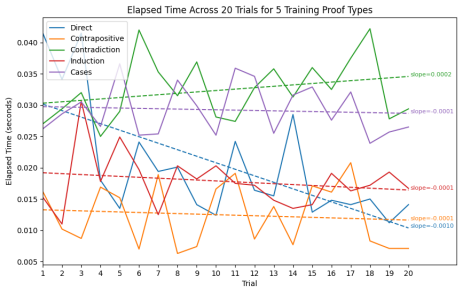

Figure 4. Training proof retrieval counts.

4.3 Error patterns

No errors currently occur, which does not imply that the model is perfect. Rather, since all given proofs are worked examples with explicit production rules and predetermined chunks, the model does not yet have the workings or opportunities to commit errors during the learning process.

5 DISCUSSION

This research outlining a symbolic ACT-R architecture aims to delineate how symbolic reasoning skills, as demonstrated in mathematical proof productions, are acquired and can be modeled within a cognitive architecture. At this stage, the model has been trained and run on five worked example proofs: a direct proof, proof by contradiction, by contrapositive, a case analysis proof, and an induction proof. Beyond completing the worked examples, the model establishes a scaffold for encoding syntactic and procedural proof productions within ACT-R's architecture. This work represents the first unified demonstration that ACT-R can be configured to execute complete mathematical proofs across multiple proof genres. The implementation also clarifies the cognitive operations and representational commitments required to symbolically step through formal

arguments, providing the design and foundation for future, more autonomous models. It further shows how proof states can be treated as dynamically evolving cognitive configurations, making explicit the sequence of attentional shifts, goal creations, and memory updates that a learner might make themselves. By articulating such mechanisms, the model sheds light upon the minute symbolic processes that shape the arguments of even basic proofs.

Though the current model demonstrates progress in this new area of modeling proof acquisition in ACT-R, it exists in an extremely primitive state that is not meant to complete this line of research. Several limitations exist. First, the production rules are not generalizable but instead "hard-coded" to walk the model through the training proofs. Second, the retrieval of chunks from declarative memory occurs explicitly as opposed to automatically. While these are characteristics of training, they do not support the model's self-driven acquisition of proof techniques through trial and error.

Future research should overcome such limitations by addressing the generalization of production rules, implementing automatic retrieval from declarative memory, and running the model on novel but nearby proofs, thereby building off of the present work to achieve active learning. To further probe the symbolic acquisition of proof strategies in these ways, I suggest the development of a progressively less constrained model by relaxing the explicit guidance provided by worked examples. This might involve: (1) replacing fixed, proof-specific productions with parameterized rule schemas that can be instantiated in multiple contexts; (2) enabling ACT-R's utility learning and chunk-strength mechanisms so that the model can select among competing steps rather than following a single predetermined path; and (3) introducing impasse-driven subgoaling to enable the system to reformulate its reasoning when progress stalls. Any of these pathways would enhance the current model with more autonomy to acquire and refine proof strategies.

The present work offers an initial account of how symbolic mathematical reasoning skills can be represented and acquired in a cognitive architecture. By distilling proofs into goal structures, memory operations, and incremental inferences available in ACT-R, the model demonstrates how formal arguments can be reconstructed from the learner's point of view rather than that of a trained logician or cognitively implausible intelligent system. The model further exhibits how proof acquisition can be modeled as a sequence of cognitive commitments, strategic choices, and representational constraints, providing insight into the developing mind of the student and the instructional interventions most likely to support them. For example, the model's progress highlights that choosing the appropriate strategy is pivotal, and perhaps more difficult than executing the strategy itself. The method slot is included in the goal chunks deliberately, after all. Knowing what needs to be proven is of little use if the learner does not understand how to execute the proof. While a statement can be proven via multiple techniques, success depends upon a well-chosen strategy. The model's progress also demonstrates that definitions and intermediate claims operate as essential retrieval cues and that impasses might signal missing definitions or productions rather than failures in understanding. Although preliminary, this work establishes the foundations needed to study proof acquisition at a level of cognitive realism, opening a pathway toward models that can autonomously improve their strategies and ultimately illuminate the psychological mechanisms that underlie formal mathematical reasoning.

6 ACKNOWLEDGEMENTS

This research was supported by a Student Signature Work grant provided by the University of Denver. I would also like to thank Dr. Kerstin Haring for her unwavering support and enthusiasm.

7 EDITOR'S NOTES

This article was peer-reviewed.

REFERENCES

[1] Weber, K. A framework for describing the processes that undergraduates use to construct proofs. *International Group for the Psychology of Mathematics Education* **28** (2004).

[2] Weber, K. & Alcock, L. Semantic and syntactic proof productions. *Educational Studies in Mathematics* **56**, 209–234 (2004).

[3] Knobelsdorf, M., Frede, C., Böhne, S. & Kreitz, C. Theorem provers as a learning tool in theory of computation. *Proceedings of the 2017 ACM Conference on International Computing Education Research* 83–92 (2017).

[4] Taatgen, N. & Anderson, J. R. The past, present, and future of cognitive architectures. *Topics in Cognitive Science* **2**, 693–704 (2010).

[5] Marić, F. A survey of interactive theorem proving. *ResearchGate* (2015).

[6] Rocq. Welcome to a world of rocq (2025).

[7] Salvucci, D. D. Modeling driver behavior in a cognitive architecture. *Human Factors: The Journal of the Human Factors and Ergonomics Society* **48**, 362–380 (2006).

[8] Haring, K. S., Ragni, M. & Konieczny, L. A cognitive model of drivers attention. *Proceedings of the*

11th International Conference on Cognitive Modeling (2021).

[9] Haring, K. S., Watanabe, K., Ragni, M. & Konieczny, L. The use of act-r to develop an attention model for simple driving tasks. *Journal of Psychology Research* **3** (2013).

[10] Xu, R., Cao, S., Kearns, S. K., Niechwiej-Szwedo, E. & Irving, E. Computational cognitive modeling of pilot performance in pre-flight and take-off procedures. *Journal of Aviation/Aerospace Education & Research* **33**, 4–23 (2024).

[11] Gunzelmann, G. & Anderson, J. R. An act-r model of the evolution of strategy use and problem difficulty. *Proceedings of the 2001 Fourth International Conference on Cognitive Modeling* 170–181 (2001).

[12] Kotseruba, I. & Tsotsos, J. K. A review of 40 years of cognitive architecture research: Core cognitive abilities and practical applications (2018).

[13] Ball, J. T. Advantages of act-r over prolog for natural language analysis. *Proceedings of the 22nd Annual Conference on Behavior Representation in Modeling and Simulation* 1–8 (2013).

[14] University, C. M. Act-r. about. .

[15] Rudin, K. & Duek, D. Subsymbolic performance of act-r (2015).

[16] Brasoveanu, A. Introduction to (python) act-r. semantics seminar: Computing dynamic meanings (2015).

[17] Dotlačil, J. & Brasoveanu, A. Computing dynamic meanings: Building integrated competence-performance theories for semantics. day 1, part 2: pyactr tutorial. *European Summer School in Logic, Language, and Information* (2018).