Review of Fundamental Proof Methods in Computer Science: A Computer-Based Approach

SELMER BRINGSJORD
Department of Cognitive Science & Department of Computer Science
Rensselaer AI & Reasoning Lab
Rensselaer Polytechnic Institute
Troy, New York 12180, USA
(*e-mail:* `selmer.bringsjord@gmail.com`)

NAVEEN SUNDAR GOVINDARAJULU
Rensselaer AI & Reasoning Lab
Rensselaer Polytechnic Institute
Troy, New York 12180, USA
(*e-mail:* `naveensundarg@gmail.com`)

submitted 14 January 2020; revised 15 March 2020; accepted

Abstract

Fundamental Proof Methods in Computer Science: A Computer-Based Approach (= FPMICS) by Arkoudas and Musser (= A&M) is a marvelous book, one symbiotically united with the longstanding Athena environment, by our lights among the best mechanical proof assistants available today, especially when proof pedagogy is a requirement for such an assistant. We briefly analyze and summarize the FPMICS-Athena pair, including strengths and shortcomings, and compare the approach of A&M with that of others.

KEYWORDS: Fundamental Proof Methods in Computer Science: A Computer-Based Approach, Expressivity, FPMICS, Athena

1 Introduction

Fundamental Proof Methods in Computer Science: A Computer-Based Approach (= FPMICS) by Arkoudas and Musser (= A&M) is a positively marvelous book, one symbiotically united with the longstanding Athena environment, by our lights among the best mechanical proof assistants available at this time, especially when proof pedagogy is a requirement for such an assistant.[1] FPMICS is also, by any metric, lengthy: 941 pages, tip to toe. Nonetheless, in our opinion, for those determined to teach or learn how to produce substantive proofs in the 21st century and beyond, FPMICS is peerless, and gradually digesting it, with Athena at one's side along the way,

[1] Athena, and the novel formalisms that undergird it, also powers a paradigm for programming and computation, for formal-methods scientists and professional practitioners (e.g., see Arkoudas 2001). While we as a matter of fact find this paradigm attractive, it is outside the scope of the present review. We also regard it to be outside the review to defend our opinion that Athena, in no small part because of its deep linkage to natural deduction (which by definition is in accord with how humans in the business of proving things go about doing so), is the best mechanical proof assistant available to not just students, but professionals in automated reasoning.

$$\boxed{\begin{aligned}(a^{-1})^{-1} &= I\cdot(a^{-1})^{-1}\\ &= (a\cdot a^{-1})\cdot(a^{-1})^{-1}\\ &= a\cdot(a^{-1}\cdot(a^{-1})^{-1})\\ &= a\cdot I\\ &= a\end{aligned}}$$

Fig. 1. Simple, Informal, Ultimately Formally *In*valid Chaining Proof

will produce in the learner a deep understanding of proof in the context of both computer science and artificial intelligence (AI). Such deep understanding is central to at least the following areas in computer science: software verification, theory of computation, algorithms, programming languages; and the following in AI: automated reasoning (specifically automated deduction), and logic-based AI.[2] We are of the opinion that using the FPMICS-Athena pair is also ideal for teaching and learning the art of proof in cognate fields, e.g., extensional logic (as we point out below, FPMICS provides no coverage of intensional logic, e.g., modal logic) as traditionally taught in mathematics and philosophy. It is important for potential readers of FPMICS to understand that this book arises from an affirmation of suitably high standards for what makes for a knowably valid proof. These standards are very nicely conveyed by a deceptively simple example that launches the book in earnest — an example of an informal proof that happens to be, after refinement, ultimately valid, but in its initial form is certainly not *verifiably* valid: no machine proof checker, nor for that matter no sufficiently careful human checker, would accept the informal proof in question. In the example, what is to be proved is the algebraic identity (let's label it 'τ')

$$(a^{-1})^{-1} = a$$

from the set $\mathscr{I}$ of these three axioms:

- *Right-Identity*: $x \cdot I = x$
- *Left-Identity*: $I \cdot x = x$
- *Right-Inverse*: $x \cdot x^{-1} = I$

The informal proof is a chaining sequence shown in Figure 1. Is this a formally valid proof? That is, is each step correct on the basis of what we are supplied as starting material? More precisely and using customary notation in formal logic, computer science, AI, and formal philosophy, is it true that $\mathscr{I} \vdash \tau$? No. This will be revealed by any attempt to mechanically *verify* the proof formally, because associativity is absent from $\mathscr{I}$ but very much needed for the third step in the informal proof. The complete analysis and refinement of this informal proof by A&M points succinctly forward to much of what FPMICS covers.

[2] A presentation of logic-based AI, a form of AI for which FPMICS-Athena is a perfect match, is given, e.g., in (Bringsjord 2008). In the opinion of the first author of the present review, computer science itself should be identified with abstract formal logic and automated reasoning (for a defense see (Bringsjord 2019)); and if this position is affirmed (even to just an appreciable degree), FPMICS-Athena might well then be the vehicles to use when introducing computer science itself to students.

2 Brutal Encapsulation of FIPMICS

As to what FPMICS does cover, here's an encapsulation, inevitably rather brutal given the length of the book.

As we've already noted, A&M immediately draw the reader in with the equality-chaining example referred to above; and with the analysis of that specimen completed and the entire road ahead inspired, then provide in Chapter 2 a self-contained, lucid introduction to Athena — one suitable not only for students, but professionals intending to explore subsequent, advanced use of this cutting-edge proof assistant. Part II of the book now commences, and consists, first, in a return to equality chaining, replete with coverage of more advanced techniques, including automated conjecture testing. A&M then move within Part II to a remarkably efficient and innovative proof-oriented coverage of zero-order logic (= $\mathscr{L}_0$) in Chapter 4, including a hands-on summary of SAT solving (a still-vibrant area of AI and automated reasoning today), and a final section in which is given an elegant implementation of a method that takes an $\mathscr{L}_0$ sentence φ to a proof if φ is a theorem, and fails otherwise. Chapter 5, an absolutely key part of the FPMICS-Athena pair, covers $\mathscr{L}_1$ (= first-order logic) masterfully, and its penultimate section, "Additional exercises," ends with an exercise that, pedagogically, we find to be a perfect gauge as to whether the student of proof methods is really and truly ready to move forward from this point to more challenging material, in which robust formal proofs, and what it takes to obtain them, are obtained. This exercise (a clever one from Suppes 1951) is given at almost exactly the half-way point of the book, and consists in the challenge of proving eight theorems about the logic of a naïve scale used to measure the relative mass of two objects x and y. Chapter 5 ends with "5.8 Chapter notes," a rather unassuming, less-than-two-page section that we nonetheless regard to crucial to an understanding of FPMICS, because there the authors explicitly affirm the centrality of relatively inexpressive *many-sorted* $\mathscr{L}_1$ to their paradigm, and the basic rationale of their having explicitly and unabashedly placed this logic in this special position, to the exclusion, e.g., of category theory and higher-order logic/type theory. (We return to the the issue of expressivity momentarily.) The last chapter in Part II, 6, extends and refines the chaining style of proof, which is then followed for the remainder of the volume.

Part III of FPMICS, comprising Chapters 7–10, is devoted to proofs involving key discrete structures, including natural numbers, integers, sets (finite only, sensibly), and maps; the latter, a standard abstract data type, will be known to some of our readers as *dictionaries* or *associative arrays*. Part IV of the book is composed of two chapters. The first of this pair, Chapter 17, provides a lucid, elegant correctness proof for a humble compiler. Chapter 18, which features **While**, a simplified imperative programming language, is devoted to representing and deductively reasoning about programming languages. For those hoping the FPMICS-Athena combination will provide a portal to software verification, Part IV will not disappoint. FPMICS ends with two appendices, the first (A) a streamlined account of the syntax and semantics of Athena's core constructs, and the second (B) a remarkably economical but informative presentation of logic programming and Prolog (which includes the best intuitive but accurate encapsulation of least Herbrand models we have seen). Appendix B ends with coverage of an Athena implementation of a Prolog interpreter for definite clauses.

3 Expressivity and the A&M Paradigm

Returning as promised to the question of expressivity, extensional formal logics, as is well-known, are routinely cast in a continuum, with their location in this continuum determined by how expressive they are. For instance, $\mathscr{L}_0$ has no trouble expressing the sentence "Six is greater than five" (e.g., $G(\bar{6},\bar{5})$, where use of the bar simply indicates that the symbol beneath it is one stipulated in the formal language to denote some entity outside this language), but cannot express this sentence: "There are at least two numbers greater than 5." For this we need quantifiers, and they are not available in $\mathscr{L}_0$. They *are* available, however, in $\mathscr{L}_1$; for in this logic we can capture the English sentence in question with for instance this formula:

$$\varphi := \exists n \exists m [n \neq m \wedge G(n,\bar{5}) \wedge G(m,\bar{5})].$$

As we have conveyed above, FPMICS in general, and Athena in particular, offers all that's needed to represent and deductively reason over formulae like φ. But consider the following sentence, which even very young students can see is true: "There are at least two natural numbers greater than five, and they share at least one property with the number zero." This sentence can't be expressed in $\mathscr{L}_1$; but it *can* be easily captured in $\mathscr{L}_2$, second-order logic, like this:

$$\psi := \exists n \exists m [n \neq m \wedge G(n,\bar{5}) \wedge G(m,\bar{5}) \wedge \exists X(X(n) \wedge X(m) \wedge X(0))]$$

Unfortunately, ψ is beyond the reach of FPMICS-Athena, and we do find this objectionable. As a rather speculative guess, it may be that A&M are impressed with, and indeed driven by, Lindström's Theorems, which famously show that $\mathscr{L}_1$ does have a certain "maximality," since for logics more expressive than it, certain meta-theorems fail, e.g., the Compactness Theorem. Nonetheless, there is a deep two-part problem that we think must be faced head on in the next edition of FPMICS: (i) FPMICS-Athena is guided by a style of proof (viz., natural deduction) that relevant humans (e.g., mathematicians writing their proofs informally for journals) abide by, yet (ii) we now know that mathematics itself is basically a *second*-order enterprise (e.g., see (Shapiro 1991; Simpson 2010)). When we consider specifically AI, we in fact see that some recent research, for instance the recent verification of the reasoning in Gödel's modal argument for God's existence (Benzmüller and Paleo 2014), not only requires $\mathscr{L}_3$, but in fact requires intensional operators as well — and such operators are not covered in FPMICS.

4 Competitors

Competitors to FPMICS-Athena can be broadly divided into two kinds: (1) non-type-theory-based systems, such as ACL2; and (2) type-theory-based systems, such as Isabelle or Coq. We briefly give an overview of ACL2, Coq, and Isabelle; and compare these systems with FMPICS-Athena. Because of space limitations, we leave aside a number of other systems, such as PVS (Owre et al. 1996) and Mizar (Grabowski et al. 2010).

4.1 Non-Type-Theory-Based ACL2

Among all other competitors, the core logic of ACL2 (Kaufmann and Strother Moore 1996) is the closest to Athena's logic. ACL2 has two components: (a) a purely functional programming language $\mathscr{P}$ that is a subset of Common Lisp; and (b) a first-order axiomatization $\Gamma_{\mathscr{P}}$ capturing some aspects of $\mathscr{P}$. Just like Athena, ACL2 has a purely first-order system as its representational core and employs a version of Lisp for its programming language (Scheme in the case of Athena). The similarity ends here, however. Users of ACL2 build models of a hardware or software system s by writing a program σ_s that models the system. Users then can ask ACL2's theorem prover to prove various properties about the system s. Any program σ_s written in ACL2 extends the formalization to $\Gamma_{\mathscr{P}} \cup \Delta(\sigma_s)$, where $\Delta(\sigma_s)$ is a set of automatically constructed formulae that model the program σ_s. This enables users to prove one or more properties ϕ_s of s using ACL2's theorem prover. In contrast, in the case of Athena, the user manually constructs and asserts both $\Gamma_{\mathscr{P}}$ and $\Delta(\sigma_s)$ before proving any property ϕ_s.

While ACL2 is an industrial-strength theorem proving system that has been used to prove non-trivial properties about real-world computational systems (hardware and software), unlike Athena, ACL2 does not come with a formal proof system for its proof-computation component. ACL2 employs a host of decision procedures that simplify and speed-up automated proof search. While the decision procedures output human-readable artifacts, the proof-search system and its outputs do not have a precise semantics, in stark contrast with Athena. The chief focus of ACL2 is proof discovery; in the case of FPMICS-Athena it is proof presentation and checking.

4.2 Type-Theory-based Systems

There are a number of systems that fall in this category. The two most widely used systems in this space are Coq and Isabelle; we briefly compare them with FPMICS. In these systems, very broadly, proof checking reduces to type checking. This reduction has its own set of advantages and disadvantages. Type-theory-based systems increase expressivity (since, unlike Athena, they permit quantification not only over individual variables, but in addition over function, and higher-order variables), but, as A&M point out, drawbacks include having to face an undecidable unification problem.

4.2.1 Coq

Coq is a proof assistant introduced in 1989 (Coquand et al. 1989; Bertot and Castéran 2004). It has been used extensively in both computing and mathematical domains, and is based on the calculus of inductive constructions, which extends the calculus of constructions, the most expressive calculus in the λ-cube for classifying λ-calculi (Barendregt 1991). Each vertex in the λ-cube denotes whether terms and types are allowed to depend on types or terms. In the calculus of constructions, types and terms can depend on types and terms. This is a major difference from Athena's untyped system, which may be easier to use given the current popularity of dynamically typed programming systems. Another notable difference between Coq's logic versus that of multi-sorted $\mathscr{L}_1$ at the heart of Athena-FPMICS is that Coq's does not have the law of excluded middle, as Coq is based on intuitionistic logic. Coq, like ACL2, accepts only primitive-recursive functions. FPMICS-Athena lets us work with the whole set of Turing-computable or general recursive functions.

Unlike FPMICS-Athena, Coq has two separate languages: (i) one for describing functions and structures, *Gallina*, and (ii) one for controlling the proof process, *Ltac*. Notably, from either Coq functions written in Gallina or proofs constructed in Ltac, Coq has the ability to extract programs in OCaml, Haskell, and Scheme.

Coq's separation into two languages has led to a formalization gap in Coq not present in FPMICS. While Coq first originated two decades ago, only recently has the semantics for Ltac been formalized. There still remain major challenges and rough edges in formalizing the Ltac language (Jedynak 2013).

4.2.2 Isabelle/HOL

Isabelle is a general framework for interactive theorem proving and can be used to construct proof assistants for a variety of logics (Paulson 1988). Of the many specializations of Isabelle to various logics, the most widely used is Isabelle/HOL, a version of Isabelle specialized for higher-order logic. Both Isabelle's core logic and Isabelle/HOL overall are closer to FPMICS than that of Coq; we thus suspect it will be easier for users to transition from Isabelle to Athena (or *vice versa*) than it would be to move from Coq to Athena, or the reverse direction. Also, like Coq and ACL2, Isabelle requires termination proofs for programs that are entered into the system. Meeting this requirement, as the programs in question grow in complexity, becomes increasingly difficult (Bove et al. 2016).

Similar to Coq, Isabelle has two separate languages: (i) an inner language for building programs and structures; and (ii) an outer language (Isar) for constructing proofs. This has led to a formalization gap similar to the aforementioned one that plagues Coq.

4.3 Extra Cognitive Burden of Proof Assistants

Type-theory-based systems suffer from a major drawback when compared with FPMICS. In order to work with proofs in a particular domain, a user must not only grasp concepts in the domain but also has to digest a large amount of type theory and other relevant concepts that is not needed for a paper-and-pencil proof. In stark contrast, and this marks a major plus for it, FPMICS comes closest to mirroring cognitively such a proof, while also providing soundness and completeness guarantees based upon its formal semantics.

5 Is Anything Missing?

Despite the vastness of FPMICS, it nonetheless misses some important topics; accordingly, the work in our opinion can and should be extended, in either a subsequent edition, or by sedulous, autodidactic readers who develop sufficient mastery of the present edition and Athena, and proceed from there. We now briefly mention two such missing topics.

5.1 Diagram-Infused Proofs

As is well-known, Euclid famously made much use of diagrams to prove things. Since at least that time, plenty of proofs have exploited visual content, and in fact today, students of the areas that FPMICS is intended for will find it exceedingly difficult to dodge requests for proofs that employ visual information in the course of their instruction. A simple example is a standard proof of the uncountability of $\mathscr{P}(\mathbb{N})$ using naïve set theory that renders diagonalization as a visual affair, by for instance depicting an infinite array, isolating the diagonal d running from northwest to southeast in that array, and then asking the student to consider a sequence $\bar{d}$ derived from the original sequence by diagonalization, so that $\bar{d}$ is guaranteed to be different than every row in the array (an elegant example is found in the following book, which by the way also proves the insolubility of the halting problems diagrammatically: Boolos et al. 2003). What is the relationship of the A&M paradigm to proof methods of today that make use of such visual content? As FPMICS makes clear, *deductions*, one of the two fundamental syntactic categories in Athena, are purely textual. (The second of these two categories is *expressions*, which represent computations (i.e., programs), and this category too is bereft of visual representations.) Interestingly enough, Arkoudas himself has elsewhere presented a system (Vivid) in which diagram-infused proofs can be expressed, and systematically evaluated (Arkoudas and Bringsjord 2009). We submit that given the centrality of visual/diagrammatic content in proofs, especially in proofs in the education space FPMICS is intended to address, A&M may wish in the short term to provide some additional functionality in Athena. Longer term, FPMICS itself should by our lights provide relevant coverage, in the next edition.

5.2 Graph-based Proofs

We have noted that the FIPMICS-Athena pair is purely textual in nature, in that proofs contain no pictorial content. But even in the case where such content is absent from proofs, proofs *themselves* need not be textual, and indeed if one examines the notebooks of formal scientists one routinely finds use of graphical notation, with for instance informal arrows drawn from one element to another. We point out, specifically, that proofs can be viewed as graphs, not as sequences of textual strings, which is what they are in FPMICS-Athena. After all, so-called *proof nets* have long been featured in connection with linear logic (Girard 1987), and indeed all deductive and non-deductive reasoning has been conceived of as creation and management of hypergraphs (Bringsjord et al. 2019). For concreteness, see in Figure 2 that a full formal proof of the informal proof of Figure 1, including the background axioms in question, is given in this hypergraphical natural-deduction paradigm (the bottom-most node being the theorem in question in A&M's example).

6 Conclusion

Despite our concerns, A&M have written a landmark book in the evolution of educational content and technology for advancing human understanding of the intertwined domains of formal proof and computation. Such understanding is necessary for the verification of computational systems, including those of the AI variety that increasingly will have the potential to harm us, or allow us to harm ourselves when employing them. But FPMICS and Athena, when studied, will pay dividends in many additional ways. In fact, this dyad is so good — so thorough, so extensive, and so timely — that a bold recommendation seems rational to issue, and we do so now: While all specialists in software verification certainly ought to study FPMICS-Athena, professionals who take rigorous reasoning seriously in *any* of the formal sciences would be well-served by obtaining FPMICS, and learning Athena, without delay.

Acknowledgement

We are deeply grateful for wise feedback and guidance from Alberto Pettorossi, and for support provided by AFOSR that has enabled us to explore many cutting-edge automated-reasoning systems in AI today.

References

ARKOUDAS, K. 2001. Certified Computation. Tech. Rep. AIM-2001-007, MIT AI Lab, www.ai.mit.edu. April. 'AIM' abbreviates 'AI Memo.'.

ARKOUDAS, K. AND BRINGSJORD, S. 2009. Vivid: An AI Framework for Heterogeneous Problem Solving. *Artificial Intelligence 173,* 15, 1367–1405.

BARENDREGT, H. 1991. Introduction to Generalized Type Systems. *Journal of Functional Programming 1,* 2, 125–154.

BENZMÜLLER, C. AND PALEO, B. W. 2014. Automating Gödel's Ontological Proof of God's Existence with Higher-order Automated Theorem Provers. In *Proceedings of the European Conference on Artificial Intelligence 2014 (ECAI 2014)*, T. Schaub, G. Friedrich, and B. O'Sullivan, Eds. IOS Press, Amsterdam, The Netherlands, 93–98.

BERTOT, Y. AND CASTÉRAN, P. 2004. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions.*

BOOLOS, G. S., BURGESS, J. P., AND JEFFREY, R. C. 2003. *Computability and Logic (Fourth Edition)*. Cambridge University Press, Cambridge, UK.

BOVE, A., KRAUSS, A., AND SOZEAU, M. 2016. Partiality and Recursion in Interactive Theorem Provers–An Overview. *Mathematical Structures In Computer Science 26,* 1, 38–88.

BRINGSJORD, S. 2008. The Logicist Manifesto: At Long Last Let Logic-Based AI Become a Field Unto Itself. *Journal of Applied Logic 6,* 4, 502–525.

BRINGSJORD, S. 2019. Computer Science as Immaterial Formal Logic. *Philosophy & Technology*.

BRINGSJORD, S., GOVINDARAJULU, N. S., AND TAYLOR, J. 2019. *Logic: A Modern Approach: Beginning Deductive Logic via HyperSlate*$^{\text{TM}}$ *and HyperGrader*$^{\text{TM}}$. Motalen, Troy, NY. This is an e-book edition of January 15 2019. The book is accompanied by access to the HyperSlate$^{\text{TM}}$ and HyperGrader$^{\text{TM}}$ software systems.

COQUAND, T., HUET, G., ET AL. 1989. The Coq Proof Assistant.

GIRARD, J. 1987. Linear Logic. *Theoretical Computer Science 50,* 1, 1–102.

GRABOWSKI, A., KORNILOWICZ, A., AND NAUMOWICZ, A. 2010. Mizar in a Nutshell. *Journal of Formalized Reasoning 3,* 2, 153–245.

JEDYNAK, W. 2013. Operational Semantics of Ltac. Ph.D. thesis, Master'S Thesis, Institute Of Computer Science, University Of Wroc Law.

KAUFMANN, M. AND STROTHER MOORE, J. 1996. ACL2: An Industrial Strength Version of Nqthm. In *Proceedings of 11th Annual Conference on Computer Assurance. COMPASS '96*. 23–34.

OWRE, S., RAJAN, S., RUSHBY, J. M., SHANKAR, N., AND SRIVAS, M. 1996. PVS: Combining Specification, Proof Checking, and Model Checking. In *International Conference on Computer Aided Verification*. Springer, 411–414.

PAULSON, L. C. 1988. Isabelle: The Next Seven Hundred Theorem Provers. In *International Conference on Automated Deduction*. Springer, 772–773.

SHAPIRO, S. 1991. *Foundations Without Foundationalism: A Case for Second-Order Logic*. Oxford University Press, Oxford, UK.

SIMPSON, S. 2010. *Subsystems of Second Order Arithmetic*. Cambridge University Press, Cambridge, UK. This is the 2nd edition.

SUPPES, P. 1951. A Set of Independent Axioms for Extensive Qualities. *Portugaliae Mathematica 10,* 4, 163–172.

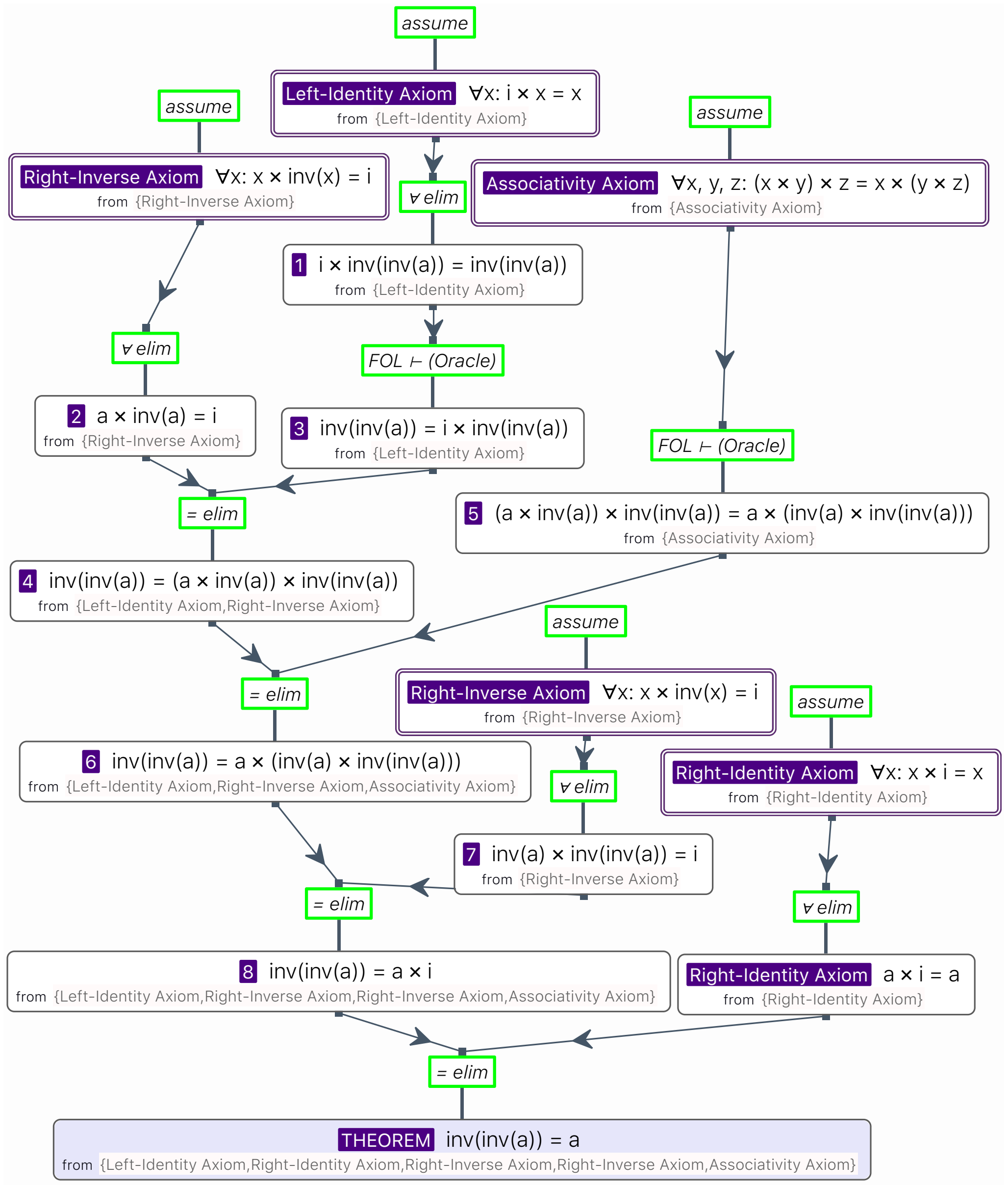
assume
Left-Identity Axiom ∀x: i × x = x
from {Left-Identity Axiom}
assume
Right-Inverse Axiom ∀x: x × inv(x) = i
from {Right-Inverse Axiom}
∀ elim
assume
Associativity Axiom ∀x, y, z: (x × y) × z = x × (y × z)
from {Associativity Axiom}
1 i × inv(inv(a)) = inv(inv(a))
from {Left-Identity Axiom}
∀ elim
FOL ⊢ (Oracle)
2 a × inv(a) = i
from {Right-Inverse Axiom}
3 inv(inv(a)) = i × inv(inv(a))
from {Left-Identity Axiom}
FOL ⊢ (Oracle)
= elim
5 (a × inv(a)) × inv(inv(a)) = a × (inv(a) × inv(inv(a)))
from {Associativity Axiom}
4 inv(inv(a)) = (a × inv(a)) × inv(inv(a))
from {Left-Identity Axiom,Right-Inverse Axiom}
assume
= elim
Right-Inverse Axiom ∀x: x × inv(x) = i
from {Right-Inverse Axiom}
assume
6 inv(inv(a)) = a × (inv(a) × inv(inv(a)))
from {Left-Identity Axiom,Right-Inverse Axiom,Associativity Axiom}
∀ elim
Right-Identity Axiom ∀x: x × i = x
from {Right-Identity Axiom}
7 inv(a) × inv(inv(a)) = i
from {Right-Inverse Axiom}
= elim
∀ elim
8 inv(inv(a)) = a × i
from {Left-Identity Axiom,Right-Inverse Axiom,Right-Inverse Axiom,Associativity Axiom}
Right-Identity Axiom a × i = a
from {Right-Identity Axiom}
= elim
THEOREM inv(inv(a)) = a
from {Left-Identity Axiom,Right-Identity Axiom,Right-Inverse Axiom,Right-Inverse Axiom,Associativity Axiom}

Fig. 2. Formally Valid Chaining Proof as HyperGraph. In hypergraphical natural deduction, each node n carries a set $\Gamma(n)$ of labels (which denote nodes). This information is shown as ‘from $\{\dots\}$’ in the figure above. If there is an arrow from node u to node v through an inference node, then u is a parent of v. For a node n, $\Gamma(n)$ can be distinct from the parent nodes of n or the leaf nodes of the tree rooted at n, the reason being that $\Gamma(n)$ tracks *undischarged* nodes only.