# Sequoia: A Playground for Logicians
## (System Description)

Giselle Reis(✉), Zan Naeem, and Mohammed Hashim

Carnegie Mellon University, Doha, Qatar
giselle@cmu.edu, {znaeem,mqh}@andrew.cmu.edu

**Abstract.** Sequent calculus is a pervasive technique for studying logics and their properties due to the regularity of rules, proofs, and meta-property proofs across logics. However, even simple proofs can be large, and writing them by hand is often messy. Moreover, the combinatorial nature of the calculus makes it easy for humans to make mistakes or miss cases. Sequoia aims to alleviate these problems. Sequoia is a web-based application for specifying sequent calculi and performing basic reasoning about them. The goal is to be a user-friendly program, where logicians can specify and "play" with their calculi. For that purpose, we provide an intuitive interface where inference rules can be input in LaTeX and are immediately rendered with the corresponding symbols. Users can then build proof trees in a streamlined and minimal-effort way, in whichever calculus they defined. In addition to that, we provide checks for some of the most important meta-theoretical properties, such as weakening admissibility and identity expansion, given that they proceed by the usual structural induction. In this sense, the logician is only left with the tricky and most interesting cases of each analysis.

**Keywords:** Sequent calculus · Meta-properties · Web-based app

## 1 Introduction

Proof (and derivation) trees are a central structure for sequent calculus. They are used to check validity of formulas and sequents, as well as for checking meta-properties of the calculus (such as rule permutability, invertibility, and cut-elimination). In the latter, trees are generally *schematic*, using context variables to represent a family of trees with that shape.

When checking the validity of sequents, the required proof trees can become quite large (both in depth and breadth) to be written on paper. At the same time, the proofs of meta-properties may involve several (often small) proof trees to cover all the cases. Many times they are slight variations of each other, or largely the same across logics. Building and then verifying these proof trees is often a tedious task. Several tiresome issues arise: symbols can easily be misplaced, look unclear, or be confused by mistake; trees may have to be adjusted or re-sized to fit the writing space; and the proofs themselves may not look as elegant as their


N. Peltier and V. Sofronie-Stokkermans (Eds.): IJCAR 2020, LNAI 12167, pp. 480–488, 2020.
https://doi.org/10.1007/978-3-030-51054-1_32

typeset counterpart. However, a glance towards creating proof trees in a digital environment shows a separate set of issues. Currently there are very few tools for creating proof trees intuitively. The most common method is to write the proofs in LaTeX, or a program that produces such proofs. Even then, the process can easily become too long and cumbersome.

Sequoia is a web application that makes sequent calculus tree building, whether schematic or not, simple and intuitive. Sequoia is aimed at students and academics who find the traditional methods too cumbersome, and provides a user-friendly means to create multiple calculi specified from user-defined symbols and inference rules, as well as a way to correctly build proof trees with their defined calculi. We use a sound and complete algorithm that computes all valid applications of a rule to a sequent. In addition, Sequoia features meta-theoretical property checking for weakening admissibility, identity expansion, and permutability. It provides all the straightforward cases needed for the complete proofs of these properties, alleviating the user from the monotonous part, and allowing them to focus on the interesting cases. All these features are ready to be used now, and we are still improving Sequoia by adding more properties to check and more proof tree building tools.

Sequoia can be accessed at: https://logic.qatar.cmu.edu/sequoia/, and the source code is available at https://github.com/meta-logic/sequoia.

## 2 System Description

Sequoia consists of a front-end built in JavaScript and HTML, and a back-end built in MongoDB and Standard ML New Jersey. The application runs in a Node.js environment. The front-end is responsible for displaying the user defined rules and symbols, rendering the interactive proof tree, and presenting possible proof tree transformations for property testing, among other things. Aside from running the server, the back-end stores the user's defined calculi (including rules and symbols), computes all the possible valid proof trees when a rule is applied to a tree sequent, and constructs all possible tree derivations for a particular meta-property. The following sections will progressively describe the design of our system by first providing the basic representations for the datatypes in SML, then describing the schematic tree building, and explaining our approach to automating the meta-property tests.

### 2.1 Datatypes

Currently, Sequoia supports sequent calculi with multiple contexts on the left and right. We restrict the rules in a calculus to operate on one connective at a time. We also require that rules have no restrictions on their contexts, such as:

$$\frac{\Box\Gamma \vdash A}{\Gamma, \Box\Gamma \vdash \Box A}$$

Note that such rules can often be rewritten using multiple contexts.

Our datatypes are defined as:

$$\begin{array}{rl}
\textbf{Formula} & F ::= p, q, ... \mid A, B, ... \mid \bullet_1(F_1, ..., F_{r_1}), \bullet_2(F_1, ..., F_{r_2}), ... \\
\textbf{Context} & \Gamma/\Delta ::= \Theta_1, ..., \Theta_k, F_1, ...F_l \\
\textbf{Sequent} & S ::= \Gamma_1 \mathbin{/\!\!/_1} ... \mathbin{/\!\!/_{n-1}} \Gamma_n \vdash \Delta_1 \mathbin{/\!\!/'_1} ... \mathbin{/\!\!/'_{m-1}} \Delta_m \\
\textbf{Rule} & R ::= (\text{rule name}, S_i, [S_1, ...S_z]) \\
\textbf{Tree} & T ::= (\text{rule name}, S_i, [T_1, ...T_y])
\end{array}$$

Where $p, q, ...$ are atom variables, $A, B, ...$ are formula variables, $\bullet_1, \bullet_2, ...$ are connectives with arities $r_1, r_2, ...$ respectively, $\Theta_1, \Theta_2, ...$ are context variables, $/\!\!/$ is a context separator and $\vdash$ is a sequent sign. Context separators are symbols used to separate different contexts on either the left or right sides the sequent. For example, in the focused system for linear logic [3], three symbols (";", "⇓", "⇑") are used to separate different parts of the right context. Note that all the mentioned symbols must be declared by the user and cannot contain superscripts.

Proof trees are recursive structures made up of a sequent, the rule name (if any) and a set of proof trees above the sequent; rules consist of a conclusion sequent and a set of premise sequents.

### 2.2 Core Operations

Rule application is the most important operation of Sequoia. It relies on three core operations: unification, substitution, and variable renaming. Rule application is a function applied to a rule and a sequent. We will use the following rule and sequent as our running example (assuming all the symbols have been declared by the user):

$$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \wedge B} \wedge_r \qquad \Delta, F \vee G, H \vdash X \wedge Y$$

**Unification.** The first step for rule application is obtaining the valid unifiers between the sequent and the conclusion sequent of the rule. The reason we are not using pattern matching is because we may have to substitute context variables in the sequent as well as in the inference rule. For example, if the context is $\Gamma, r, p \wedge q$ and the conclusion of the rule is $\Gamma_1, \Gamma_2, A \wedge B$, then we need: one of $\Gamma_i$ substituted by a context variable and $r$, and $\Gamma$ substituted by two new context variables. Unification of sequents and contexts is defined as:

$$\frac{\vdash = \vdash' \quad \Gamma_1 \mathbin{/\!\!/_1} ...\Gamma_n \doteq \Gamma'_1 \mathbin{/\!\!/'_1} ...\Gamma'_n \mid \Sigma_1 \quad (\Delta_1;_1 ...\Delta_k)\sigma \doteq (\Delta'_1;'_1 ...\Delta'_k)\sigma \mid \Sigma_2 \quad \forall \sigma \in \Sigma_1}{\Gamma_1 \mathbin{/\!\!/_1} ...\Gamma_n \vdash \Delta_1 \mathbin{/\!\!/_1} ...\Delta_k \doteq \Gamma'_1 \mathbin{/\!\!/'_1} ...\Gamma'_n \vdash' \Delta'_1 \mathbin{/\!\!/'_1} ...\Delta'_k \mid \Sigma_1 \circ \Sigma_2}\ \mathsf{seq}$$

$$\frac{}{\dot{=} \mid \{\}}\ \mathsf{ctx}_0 \qquad \frac{/\!\!/_1 = /\!\!/'_1 \quad \Gamma_1 \doteq \Gamma'_1 \mid \Sigma_1 \quad (\Gamma_2 \mathbin{/\!\!/_2} ...\Gamma_n)\sigma \doteq (\Gamma'_2 \mathbin{/\!\!/'_2} ...\Gamma'_n)\sigma \mid \Sigma_2 \quad \forall \sigma \in \Sigma_1}{\Gamma_1 \mathbin{/\!\!/_1} \Gamma_2 \mathbin{/\!\!/_2} ...\Gamma_n \doteq \Gamma'_1 \mathbin{/\!\!/'_1} \Gamma'_2 \mathbin{/\!\!/'_2} ...\Gamma'_n \mid \Sigma_1 \circ \Sigma_2}\ \mathsf{ctx}$$

Note that the symbols between contexts and the sequent sign must match for the unification to succeed, and that contexts are ordered. The unification of individual contexts is done through multiset unification using constraints [6].

Suppose that we want to apply the rule $\wedge_r$ to the sequent in the example above. Then, we need to get the unifiers between $\Gamma_1, \Gamma_2 \vdash A \wedge B$ ($\wedge_r$ conclusion) and $\Delta, F \vee G, H \vdash X \wedge Y$. This will produce a number of valid unifiers and a constraint theory alongside each unifier, one of them being:

$$\sigma = \{A \to X,\ B \to Y,\ \Gamma_1 \to [\Gamma_1', H],\ \Gamma_2 \to [\Gamma_2', F \vee G]\} \qquad (\Delta = \Gamma_1', \Gamma_2')$$

Constraint theories are used to maintain consistency between a conclusion and its premises in a proof tree. Its importance is discussed later in Sect. 2.3.

**Substitution.** Once unification is done, every valid unifier represents one possible way of applying the inference rule to the sequent. The premises are determined by applying the resulting unifier to the rule's premises. For example, the unifier above can be applied to the premises of $\wedge_r$, resulting in:

$$(\Gamma_1 \vdash A)\sigma = \Gamma_1', H \vdash X \qquad (\Gamma_2 \vdash Y)\sigma = \Gamma_2', F \vee G \vdash Y$$

Which is a correct set of premises when applying $\wedge_r$ to $\Delta, F \vee G, H \vdash X \wedge Y$, given the constraint $\Delta = \Gamma_1', \Gamma_2'$:

$$\frac{(\Gamma_1 \vdash A)\sigma \quad (\Gamma_2 \vdash B)\sigma}{(\Gamma_1, \Gamma_2 \vdash A \wedge B)\sigma}\ \wedge_r \quad = \quad \frac{\Gamma_1', H \vdash X \quad \Gamma_2', F \vee G \vdash Y}{\Delta, F \vee G, H \vdash X \wedge Y}\ \wedge_r$$

**Variable Renaming.** When applying a rule, we can assume that the sequent and the conclusion sequent of the rule have different variable names because of the symbol restrictions in the symbols tables. However, after applying a rule some problems might arise. For example, applying the $\wedge_r$ rule on $\Gamma \vdash (a \wedge b) \wedge c$ would yield the premises $\Gamma_1 \vdash a \wedge b$ and $\Gamma_2 \vdash c$ and the constraint $\Gamma = \Gamma_1, \Gamma_2$. However, applying $\wedge_r$ again on $\Gamma_1 \vdash a \wedge b$ would cause problems in unification as $\Gamma_1$ is used in both the rule and the sequent. To avoid this problem, all context variables are renamed after unification. To rename a context variable, we simply add a fresh superscript to the name of the variable, or update it if the name has one already.

### 2.3 Functionalities

The key features of Sequoia are that it allows the user to build ground and schematic proof trees, and to automate the process of testing for certain meta-properties. Currently, Sequoia is able to check rule permutability, weakening admissibility (for each context), and identity expansion.

**Tree Building.** Sequoia's tree building relies entirely on rule application. Given a tree, a constraint list, and a set of rules, a selected rule can be applied to an open sequent in the tree to produce a new tree and constraints with the appropriate updates. To do this, we first compute all possible unifiers of an open sequent and a rule. Then for each unifier, the empty premise set of the open sequent is replaced by the new premises obtained as explained above. The unifier is applied to each sequent in the tree, including the open sequent, and the constraint list is updated with the unifier's accompanying constraint theory. The constraint list is bound to the tree and accounts for the context variables changing at different levels in the tree as a result of multiple rule applications. The user can undo rule applications, as well as export the proof tree to LaTeX.

**Proof Transformations.** In some cases (such as checking for permutability), we need to decide whether a tree $\mathcal{T}_1$ with end sequent $S$ can be transformed into another tree $\mathcal{T}_2$ with the same end sequent. For that, we assume that $\mathcal{T}_1$ is a closed tree, i.e., each premise $S_{1,i}$ in $\mathcal{T}_1$ has a proof $\mathcal{D}_{1,i}$. Checking if $\mathcal{T}_1$ can be transformed into $\mathcal{T}_2$, amounts to checking that each open premise $S_{2,j}$ in $\mathcal{T}_2$ can be proved using some $\mathcal{D}_{1,i}$. The proof $\mathcal{D}_{1,i}$ for $S_{1,i}$ can be used to prove $S_{2,j}$ if: (1) the two sequents are the same (modulo context variables), or (2) if weakening is admissible in some contexts, that $S_{1,i}$ can be obtained by weakening $S_{2,j}$. If the proof can be used, we add constraints specifying which multiset of context variables in $S_{2,j}$ is equal to the multiset of context variables in $S_{1,i}$. Given this set of constraints and the ones obtained from unification when applying the rules, which are equalities between multisets, we try to find an AC1[1] unifier [1, section 10.3] such that it does not map context variables of $\mathcal{T}_1$ to empty or a multiset which contains more than one copy of each context. This approach to proof transformations has its limitations, since we do not take into consideration cases that succeed because of a rule's invertibility or the use of cut rules. Thus, the check is always sound, but not complete. The user needs to check by hand the cases that Sequoia cannot infer.

**Permutability.** Given two rules $R_1$ and $R_2$, the initial rules, and the weakening properties of the calculus, we say that $R_1$ *permutes up* $R_2$ if a proof tree $\mathcal{T}$ ending with the rule $R_2$ applied over $R_1$ can be transformed into a proof tree $\mathcal{T}'$ ending with $R_1$ applied over $R_2$. Sequoia performs this check by first generating all derivations $\mathcal{T}$ where $R_2$ is applied over $R_1$, and all derivations $\mathcal{T}'$ where $R_1$ is applied over $R_2$. Then, for each tree $\mathcal{T}$, we try to find a tree $\mathcal{T}'$ such that $\mathcal{T}$ can be transformed into $\mathcal{T}'$.

**Weakening Admissibility.** The admissibility of weakening for a calculus is checked for each context separately. Given a context $\Gamma$ in a sequent $S$, the theorem states: if a sequent $S[\Gamma]$ is provable, then so is $S[\Gamma, F]$. The usual proof

[1] Associative, commutative, with neutral element (the properties of multiset union).

proceeds by structural induction on the derivation of $S[\Gamma]$. Sequoia is able to check all "trivial" cases, i.e. the ones that require only the inductive hypothesis.

**Identity Expansion.** Identity expansion is the property that all identity rules can be applied on atoms. The usual proof proceeds by induction on the formula structure. Let $\bullet(F_1, ..., F_n)$ be a formula with main connective $\bullet$, $[\bullet_1], ..., [\bullet_k]$ be the rules for decomposing $\bullet$, and $[id]$ one identity rule. Sequoia checks if a proof ending with $[id]$ on $\bullet(F_1, ..., F_n)$ can be transformed into a proof using some of the rules $[\bullet_1], ..., [\bullet_k]$ and $[id]$ only on $F_1, ..., F_n$. This is done by applying left and right pairs of rules and trying to close the proof.

Once again, this check is sound, but not complete. For example, take the LJ calculus for intuitionistic logic with the following rules for conjunction left:

$$\frac{\Gamma, A_1 \vdash C}{\Gamma, A_1 \wedge A_2 \vdash C} \wedge_r^1 \quad \frac{\Gamma, A_2 \vdash C}{\Gamma, A_1 \wedge A_2 \vdash C} \wedge_r^2$$

Sequoia is not able to infer identity expansion because it will not apply contraction arbitrarily. Instead, if the following rule is used:

$$\frac{\Gamma, A_1, A_2 \vdash C}{\Gamma, A_1 \wedge A_2 \vdash C} \wedge_r$$

Then Sequoia succeeds in showing identity expansion for the case of $\wedge$.

## 3 Usage

Sequoia was made with the goal in mind that a calculus construction and tree building tool should have a nice design and an intuitive interface for students and academics. All input is compiled in LaTeX, as it is a familiar typesetting language with a vast access to symbols.

**Symbols Table.** Before creating rules and building trees, users must declare the symbols to be used and their types. A symbols table consists of the symbol (input in LaTeX) and its type (chosen from a drop-down menu). Symbols can be updated by changing their type or simply deleted. There are two symbols tables: a rule one (symbols used for the rules in a calculus) and an end-sequent one (symbols used on the end-sequent of a proof tree). The following restrictions apply to both tables: the same symbol cannot be assigned different types (per calculus), and symbols cannot contain superscripts. Moreover, the symbols used for context variables and formulas in rules and end-sequents must be disjoint.

**Calculus Specification.** The homepage displays a user's defined calculi, a form to create new calculi, and buttons to add sample calculi that are available by default. Each calculus will have a card. Clicking on a calculus card will direct the user to the main page for that calculus, which contains all its rules and the rule symbols table. "Add Rule" directs the user to the rule creation page,

where they see the input fields for a rule: name, main connective, conclusion, and premise(s). There is also a drop-down menu for indicating on which side of the sequent the rule operates (left, right, or none). After filling in the information, clicking on "Preview" will show a rendering of the rule compiled in LaTeX and the rule symbols table from the calculus page. When all symbols in the premises and conclusion are in the table, the new rules can be added to the calculus.

**Tree Building.** Clicking on "Proof Tree" (upper right corner) takes the user to a page where they can build proof trees using the calculus rules they have defined (listed on the right). After entering an end-sequent and clicking on "Preview", they will see a rendering of the sequent in LaTeX and the end-sequent symbols table. Once all the context variable and formula symbols are declared in the table, the user can begin building the tree. The constraint list (initially empty) is shown on the left. To build on a proof tree, the user selects a leaf premise in the tree and a rule to apply on it. If the rule is not applicable, the user will be informed. Otherwise, the user is prompted with a selection of all the possible premise sets that result from applying the rule. Selecting a premise set renders the appropriate tree with these premises and the constraint list is updated with the associated constraint. In case the selected rule is *cut*, the user is prompted for the substitution to be used. They must type the cut-formula variable (used in the rule), and the cut-formula to be used for that variable.

**Properties Testing.** The properties page allows the user to test certain meta-properties for a sequent calculus system. Currently, the implemented meta-properties are: weakening admissibility, identity expansion, and permutability.

By clicking on "Weakening Admissibility" or "Identity Expansion", the user is presented with several cards representing contexts or connectives, respectively. Clicking on a card will show all its proof tree transformations for that property. By clicking on "Permutability", the user is shown all rules and must select two to perform the check. After clicking on "Permute Rules", Sequoia shows all the successful and failed proof transformations for permutability between them.

## 4 Related Work

There are several other tools for constructing and visualizing proofs. We will focus on the ones that are interactive and offer support for sequent calculi.

Closest to our approach is Carnap.io [5], a web-based tool built using proofJS and Haskell. Carnap supports different deductive systems and allows users to add their logic by implementing it in Haskell with the help of Carnap's type classes. Proof tree construction is done by typing the proof, while Carnap checks each step. The sequent calculus calculator [4] has an interface similar to Sequoia, and also allows the user to build proof trees in four different logics. The user needs to instantiate the rules before applying them. Axolotl [2] is a Java applet and mobile app for constructing proofs. It can handle proofs in sequent calculus,

natural deduction, or Hilbert systems. Sequent calculus rules are displayed on a one-dimensional notation, and proof goals may not contain context variables.

Both Carnap.io and the sequent calculus calculator require manual input from the user when building proofs, which are checked. Sequoia instead computes all possibilities for a *correct* rule application, and prompts the user to choose one. This is less tedious for experienced users, and more user friendly for those not versed in sequent calculus. This is also the approach used in Axolotl.

Concerning different logics, while Carnap.io allows the user to add more calculi, this requires expertise with Haskell and type classes, which most undergraduate students lack. We believe that the approach of inputting sequent calculus via LaTeX will be more appealing for those users. The other systems only work on a pre-determined set of calculi.

Different from all the aforementioned systems, Sequoia allows users to build *schematic* proofs, using context variables. The reason for including this feature is that, most of the time, logicians "play" with proofs using schemas as opposed to concrete formulas since they are trying to see patterns or investigate proof transformations regardless of concrete terms.

Tatu [7] and Quati [8] are web-based tools that allow users to check for certain meta-properties of sequent calculus systems. Tatu allows the user to check for identity expansion and cut admissibility, while Quati allows the user to check if rules permute over each other and shows the proof tree transformations rendered from LaTeX. To use those tools, users have to define their sequent calculus system in linear logic with subexponentials, a non-trivial task that cannot be easily automated. Another approach for checking meta-properties is using rewrite logic. In [9] the authors used Maude to automate the checking of permutability, admissibility and invertibility of rules. Although there is no user interface, the technique seems powerful and could be used in Sequoia for other checks.

Sequoia improves on Tatu and Quati by facilitating considerably the input of systems. The main difference between Sequoia and the tool based on Maude is that Sequoia displays the proof transformations that were found, thus showing the user how they work and increasing the trust in the system.

## 5 Future Work

There are a number of features and improvements we plan to add to Sequoia.

We have recently finished the implementation of two new features: checking cut admissibility (using Gentzen-style proofs), and supporting rules with context restrictions (such as the one mentioned in Sect. 2.1). These will be added to the website soon. The next meta-property we would like to add support for is rule invertibility. It should not be hard to check the simplest cases, which use a short derivation with cut. Most, if not all, of the operations needed are already implemented. We also plan to add support for first-order systems, but this will be more challenging, since it requires changes to some of the core operations. It will also result in more prompts to the user.

To improve usability, we are investigating the possibility of inputting sequent calculi or proofs by taking pictures of hand-written objects. We believe this

feature will make the system much more appealing, specially to undergrads who do their work by hand, and need to type it in LaTeX afterwards. A simpler addition that increases usability is allowing that rules be reused between calculi.

Concerning the meta-property proofs, we want to give the user the ability to export (incomplete) proofs to LaTeX. Given a stable framework for formalizing meta-properties, one could also think of exporting these proofs into partial proof scripts to be completed by the user.

## References

1. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, New York (1998)
2. Cerna, D.M., Kiesel, R.P., Dzhiganskaya, A.: A mobile application for self-guided study of formal reasoning. In: Quaresma, P., Neuper, W., Marcos, J. (eds.) Proceedings of the 8th International Workshop on Theorem Proving Components for Educational Software. Electronic Proceedings in Theoretical Computer Science, vol. 313, pp. 35–53. Open Publishing Association (2020). https://doi.org/10.4204/EPTCS.313.3
3. Di Cosmo, R., Miller, D.: Linear logic. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University (2019)
4. Hauser, F.: Sequent Calculus Calculator. https://seqcalc.io/. Accessed Jan 2020
5. Leach-Krouse, G.: Carnap: an open framework for formal reasoning in the browser. In: Quaresma, P., Neuper, W. (eds.) Proceedings 6th International Workshop on Theorem proving components for Educational software. Electronic Proceedings in Theoretical Computer Science, vol. 267, pp. 70–88. Open Publishing Association (2018). https://doi.org/10.4204/EPTCS.267.5
6. Naeem, Z., Reis, G.: Unification of multisets with multiple labelled multiset variables. In: 33rd International Workshop on Unification (UNIF) (2019)
7. Nigam, V., Pimentel, E., Reis, G.: An extended framework for specifying and reasoning about proof systems. J. Logic Comput. **26**(2), 539–576 (2016). https://doi.org/10.1093/logcom/exu029
8. Nigam, V., Reis, G., Lima, L.: Quati: an automated tool for proving permutation lemmas. In: 7th International Joint Conference on Automated Reasoning (IJCAR 2014), pp. 255–261 (2014). https://doi.org/10.1007/978-3-319-08587-6_18
9. Olarte, C., Pimentel, E., Rocha, C.: Proving structural properties of sequent systems in rewriting logic. In: Rusu, V. (ed.) WRLA 2018. LNCS, vol. 11152, pp. 115–135. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99840-4_7