Sequent Calculus Prover

How To Guide

CMU 15-316: Software Foundations of Security

Version 2.0 — January 28, 2026

Contents

1 Introduction

The **Sequent Calculus Prover** is an interactive proof assistant for constructing proofs in propositional logic, first-order logic, and dynamic logic. Users build proof trees by selecting formulas and applying inference rules step-by-step.

1.1 Features

- Propositional logic connectives: $\wedge$, $\vee$, $\rightarrow$, $\neg$, $\leftrightarrow$, $\top$, $\bot$
- First-order quantifiers: $\forall$, $\exists$
- Dynamic logic: box modality $[\alpha]P$ with assignments, tests, loops
- Structural rules: weakening, contraction, cut
- Custom user-defined rules with JSON persistence
- LaTeX export for proof documentation

2 Installation

1. Ensure Python 3.x is installed with Tkinter (included by default).
2. Clone or download the repository.
3. Run the application:

```
python sequentGen.py
```

No external packages are required.

3 Input Syntax

Sequents are entered in the format: `antecedent |- succedent`

3.1 Propositional Logic

Connective	Syntax Options	Example
And ($\wedge$)	`and`, `&`	`p and q`
Or ($\vee$)	`or`, `|`	`p or q`
Not ($\neg$)	`not`, `~`	`not p`
Implies ($\rightarrow$)	`implies`, `->`	`p -> q`
Iff ($\leftrightarrow$)	`iff`, `<->`	`p <-> q`
True ($\top$)	`true`, `top`	`|- true`
False ($\bot$)	`false`, `bot`, `bottom`	`false |- p`

3.2 Quantifiers

Quantifier	Syntax	Example
Universal ($\forall$)	`forall x. P`	`forall x. P(x) |- Q`
Existential ($\exists$)	`exists x. P`	`|- exists y. Q(y)`

3.3 Dynamic Logic

Construct	Syntax	Example
Assignment	`[x := e]P`	`[x := 5]Q |- R`
Test	`[?P]Q`	`[?guard]result |- S`
Skip	`[skip]P`	`[skip]Q |- Q`
Sequence	`[a; b]P`	`[x:=1; y:=2]P |- Q`
Choice	`[a ∪ b]P`	`[a U b]P |- Q`
Loop	`[a*]P`	`[loop*]P |- Q`
If-Then-Else	`[if P then a else b]Q`	(conditional program)
While Loop	`[while P do a]Q`	(iteration)
While w/ Invariant	`[while_{J} P do a]Q`	(with invariant J)
For Loop	`[for 0 <= i < n do a]Q`	(bounded iteration)

3.4 Comparison Operators

Operator	Syntax
Equals	`x = y`
Not Equals	`x != y`
Less Than	`x < y`
Less or Equal	`x <= y`
Greater Than	`x > y`
Greater or Equal	`x >= y`

4 Step-by-Step Workflow

Step 1: Enter a Sequent
Type your sequent in the input field at the top. Use `|-` to separate the antecedent (left) from the succedent (right).

Example: `p implies q, p |- q`

Step 2: Click "Start Proof"
The proof tree appears in the left pane, and the current sequent's formulas are displayed in the formula lists on the right.

Step 3: Select a Formula
Click on a formula in either the **Antecedent (LHS)** or **Succedent (RHS)** list to select it.

Step 4: Choose a Tab and Apply a Rule
Navigate to the appropriate tab and click the rule button:

- **Propositional** — ∧L/R, ∨L/R, →L/R, ¬L/R, ↔L/R, ⊥L/R, ⊤L/R
- **Quantifiers** — ∀L/R, ∃L/R
- **Dynamic Logic** — [:=]R, [?]L/R, [;]L/R, [skip]L/R, [∪]R, [∗]unfold, etc.
- **Structural** — Weakening, Contraction, Cut
- **Custom** — User-defined rules

Step 5: Close Branches
Use the **Identity (Axiom)** button when a formula appears on both sides. Use ⊥L when ⊥ is in the antecedent, or ⊤R when ⊤ is in the succedent.

Step 6: Repeat
Continue selecting formulas and applying rules until all branches are closed (marked with ✓).

5 Rule Reference

5.1 Propositional Rules

Rule	Type	Description
$\wedge$L	Unary	$\Gamma, A \wedge B \vdash \Delta \Longrightarrow \Gamma, A, B \vdash \Delta$
$\wedge$R	Binary	$\Gamma \vdash A \wedge B, \Delta \Longrightarrow (\Gamma \vdash A, \Delta)$ and $(\Gamma \vdash B, \Delta)$
$\vee$L	Binary	$\Gamma, A \vee B \vdash \Delta \Longrightarrow (\Gamma, A \vdash \Delta)$ and $(\Gamma, B \vdash \Delta)$
$\vee$R	Unary	$\Gamma \vdash A \vee B, \Delta \Longrightarrow \Gamma \vdash A, B, \Delta$
$\rightarrow$L	Binary	$\Gamma, A \rightarrow B \vdash \Delta \Longrightarrow (\Gamma \vdash A, \Delta)$ and $(\Gamma, B \vdash \Delta)$
$\rightarrow$R	Unary	$\Gamma \vdash A \rightarrow B, \Delta \Longrightarrow \Gamma, A \vdash B, \Delta$
$\neg$L	Unary	$\Gamma, \neg A \vdash \Delta \Longrightarrow \Gamma \vdash A, \Delta$
$\neg$R	Unary	$\Gamma \vdash \neg A, \Delta \Longrightarrow \Gamma, A \vdash \Delta$
$\leftrightarrow$L	Binary	$\Gamma, A \leftrightarrow B \vdash \Delta \Longrightarrow (\Gamma, A, B \vdash \Delta)$ and $(\Gamma \vdash A, B, \Delta)$
$\leftrightarrow$R	Binary	$\Gamma \vdash A \leftrightarrow B, \Delta \Longrightarrow (\Gamma, A \vdash B, \Delta)$ and $(\Gamma, B \vdash A, \Delta)$
$\top$L	Unary	$\Gamma, \top \vdash \Delta \Longrightarrow \Gamma \vdash \Delta$ (removes vacuous $\top$)
$\top$R	Closes	$\Gamma \vdash \top, \Delta$ closes (true is always provable)
$\bot$L	Closes	$\Gamma, \bot \vdash \Delta$ closes (false implies anything)
$\bot$R	Unary	$\Gamma \vdash \bot, \Delta \Longrightarrow \Gamma \vdash \Delta$ (removes $\bot$ from succedent)

5.2 Quantifier Rules

Rule	Type	Description
$\forall$L	Unary	Instantiate with user-provided term t: adds $P(t)$ to antecedent
$\forall$R	Unary	Uses fresh variable x': replaces $\forall x.P$ with $P(x')$
$\exists$L	Unary	Uses fresh variable x': replaces $\exists x.P$ with $P(x')$
$\exists$R	Unary	Instantiate with user-provided term t: adds $P(t)$ to succedent

5.3 Dynamic Logic Rules

Rule	Type	Description
[:=]R	Unary	$[x := e]Q \Longrightarrow x' = e \vdash Q[x'/x]$ (fresh x')
[?]L	Binary	$[?P]Q$ on left $\Longrightarrow (\vdash P)$ and $(Q \vdash)$
[?]R	Unary	$[?P]Q$ on right $\Longrightarrow P \vdash Q$
[;]L	Unary	$[\alpha;\beta]Q \Longrightarrow [\alpha][\beta]Q$ (on left)
[;]R	Unary	$[\alpha;\beta]Q \Longrightarrow [\alpha][\beta]Q$ (on right)
[skip]L	Unary	$[\text{skip}]Q \Longrightarrow Q$ (on left)
[skip]R	Unary	$[\text{skip}]Q \Longrightarrow Q$ (on right)
[∪]R	Binary	$[\alpha \cup \beta]Q \Longrightarrow [\alpha]Q$ and $[\beta]Q$
[∗]unfold	Binary	$[\alpha^*]Q \Longrightarrow Q$ (exit) and $[\alpha][\alpha^*]Q$ (iterate)
[if]R	Binary	$[\text{if } P \text{ then } \alpha \text{ else } \beta]Q \Longrightarrow (P, [\alpha]Q)$ and $(\neg P, [\beta]Q)$
[while]unfold	Binary	Unfolds one iteration of while loop
[while]inv	Ternary	Uses invariant J: (1) $\vdash J$, (2) $J \wedge P \vdash [\alpha]J$, (3) $J \wedge \neg P \vdash Q$
[for]R	Unary	Desugars to: $[i := 0; \text{while } (i < n) \text{ do } (\alpha; i := i+1)]Q$

5.4 Structural Rules

Rule	Type	Description
WL	Unary	Weakening Left: Add any formula to the antecedent
WR	Unary	Weakening Right: Add any formula to the succedent
CL	Unary	Contraction Left: Remove duplicate from antecedent
CR	Unary	Contraction Right: Remove duplicate from succedent
Cut	Binary	Introduce lemma C: proves $(\vdash C)$ and $(C \vdash)$

6 Custom Rules

Create your own inference rules via the **Custom** tab.

6.1 Creating a Custom Rule

1. Click **"+ Add Rule"** in the Custom tab.
2. Enter a **Rule Name** (e.g., "DoubleNeg").
3. Select the **Side**: LHS (antecedent) or RHS (succedent).
4. Choose the **Rule Type**:
   - **Unary** — One child branch

- **Binary** — Two child branches
- **Close** — Closes the branch immediately

5. Use **placeholders** in formula fields:

Placeholder	Meaning
`LEFT`	Left operand of binary formula
`RIGHT`	Right operand of binary formula
`INNER`	Inner formula of ¬
`FORMULA`	The entire selected formula

6. Click **"Save Rule"**.

6.2 Example: Double Negation Elimination

To create a rule that simplifies $\neg\neg P$ to P:

- **Name:** DoubleNegElim
- **Side:** RHS
- **Type:** Unary
- **Add to RHS:** `INNER` (extracts the inner $\neg P$, then you'd apply ¬R)

Custom rules are automatically saved to `custom_rules.json` and persist across sessions.

7 LaTeX Export

Click **"Export LaTeX"** to generate proof tree code using the `\infer` macro format:

```
\begin{rules}
\infer[\ms{id}]
  {p, q \vdash p}
  {}
\end{rules}
```

Use with the `bussproofs` or similar package in your LaTeX documents.

8 Reference Materials

The `Notes To Reference/` folder contains PDF lecture notes from CMU 15-316:

File	Content
`02-prop.pdf`	Propositional logic sequent calculus
`03-dynamiclogic.pdf`	Dynamic logic and box modality
`04-semantics.pdf`	Formal semantics
`05-safety.pdf`	Safety proof techniques
`06-memsafety.pdf`	Memory safety proofs

9 Troubleshooting

"Please select a formula"
Click on a formula in the LHS or RHS list before applying a rule.

"Rule already applied to this node"
Select a leaf node (no children) in the proof tree.

"This branch is already closed"
Move to a different open branch in the tree.

Identity rule doesn't work
Ensure the exact same formula appears on *both* sides of the sequent.

Custom rule not appearing
Check that `custom_rules.json` exists and is valid JSON.

Happy Proving!

For issues or contributions, see the project repository.