How to explain a counterexample

Dale Miller

Inria Saclay &
LIX, École Polytechnique
Palaiseau, France

PHILMATH Seminar
14 December 2021
A modern issue with using proof assistants

In many modern proof assistants, when the user proposes a theorem to prove, the assistant often searches for a counterexample to that proposed theorem.

▶ Maybe the empty set was not considered properly or the existence of an even prime number was overlooked.

If the machine finds a counterexample, the natural question is: *How can the machine help the user understand what is wrong with their proposed theorem?*

▶ Revealing a term—such as the empty set or the prime number 2—might be sufficient.

▶ Sometimes much more sophistication is needed.
Useful automated tools

Systems that search for counterexamples to proposed theorems.

- Refute and Nitpick are available in Isabelle/HOL.
- QuickChick is implemented in Coq.

Property based testing systems are closely related.

\[
\forall L, K \in \text{list. } \{\text{true}\} \; K = \text{SortProg}(L) \; \{\text{sorted } K \land \text{perm } L \; K\}
\]

- Quickcheck: tests code with lots of (well selected) examples and checks various proscribed properties of them.
- Originally developed within the Haskell setting, similar tools have been built for ACL2, Agda, Isabelle, and PVS.

In these cases, the machine has a proof that the user is motivated to learn some aspects of it.
How can a user learn from a formal proof?

Print a LaTeX document? Naive for several reasons.

▶ The result could be very long and difficult to read.
▶ The line between computation (not needing explanation) and deduction (needing explanation) is often ad hoc.
▶ Different readers might need different levels of detail.
▶ The reader might need to consider only small parts of a proof.

Proof browsing? Seems less naive.

▶ The user chooses the parts of the proof of interest.
▶ If more details are needed, they can be selectively unfolded.

An *interaction* between the user and the proof seems best.
Two assumptions underlying this talk

1. The human user has proposed a conjecture and the machine has found a counterexample.
   - The explanation is provided to a *motivated* user and someone familiar with the proof assistant.

2. The theorems and conjectures will be based on items of computational interest instead of general mathematical interest.
   - This explains the kinds of examples I will be using.

We shall refer to the human as the *user* and the machine holding the proof as the *oracle*.

This project is only getting started.

- Many technical issues remain.
- Lifting this project to a broader setting is certainly of interest.
Proof evidence as sequent calculus proofs

Some proof assistants build natural deduction style proofs, often encoded as dependently typed $\lambda$-terms.

Proof evidence appears in many other formats: resolution refutations, Herbrand disjunctions, tableaux proofs, etc.

Most proof evidence can be presented as sequent calculus proofs.

For example, *Foundational Proof Certificates* can be used to translate a wide range of proof evidence into sequent calculus proofs [Chihani, M, and Renaud, 2017].
Sequents and the search for proofs

Assume that I have a several assumptions $H_1, \ldots, H_n$ written at the top of sheet of paper and one conclusion $B$ at the bottom of the proof.

In the middle of the sheet is blank space that needs to be filled with a proof.

This state of affairs is encoded in sequent calculus as:

$$\vdash$$

$$H_1, \ldots, H_n \vdash B$$

The sheet of paper is encoded as the sequent $H_1, \ldots, H_n \vdash B$ and the empty space corresponds to $\vdash$.

We usually read inference rules from conclusion to premises.
Sequent calculus inference rules

**Structural rules:** weakening and contraction

\[
\begin{align*}
\Gamma ⊢ Δ & \quad wL \\
& \quad \Gamma, B \vdash Δ \\
\Gamma ⊢ Δ & \quad wR \\
& \quad Γ, B, B \vdash Δ \\
\Gamma, B, B ⊢ Δ & \quad cL \\
& \quad Γ \vdash Δ, B, B \\
\Gamma \vdash Δ, B & \quad cR
\end{align*}
\]

**Identity rules:** initial and cut

\[
\begin{align*}
B & \vdash B \quad \text{initial} \\
Γ ⊢ Δ, B & \quad \text{cut} \\
& \quad B, Γ' ⊢ Δ' \\
& \quad Γ, Γ' ⊢ Δ, Δ'
\end{align*}
\]

**Introduction rules:** collections of left and right rules for every logical connective

We introduce these as we need them.
Cut-elimination and consistency

The central result in proof theory is *cut-elimination*: a sequent provable using the cut rule can be proved without the cut rule.

This result is usually proved by permuting cut rules up into the proof until they disappear.

*Consistency* of the underlying logic follows immediately:
Assume that $\Xi_1$ is a proof of $\vdash B$ and $\Xi_2$ is a proof of $\vdash \neg B$ (equivalently, $B \vdash \bot$). Thus, we have the following proof.

\[
\Xi_1 \cdot \vdash B \quad B \vdash \bot \\
\hline
\vdash \bot \quad \text{cut}
\]

This empty sequent must also have a cut-free proof, but this is impossible.
A partial proof of the empty sequent

Assume that the user is convinced that sequent \( \vdash B \) is provable (in, say, first-order intuitionistic or classical arithmetic).

Also assume that the proof assistant (the oracle) has constructed a proof \( \Xi \) of the negation of \( B \), i.e., of the sequent \( B \vdash \cdot \).

We can write the \textit{partial} proof structure

\[
\begin{array}{c}
\cdot \vdash B \\
\Xi \\
B \vdash \cdot \\
\cdot \vdash \cdot
\end{array}
\]

\textit{cut}

Thus, there can be no proof of the left premise.

\textbf{Main design choice:} If the user is still convinced of the provability of \( B \), we take advantage of that state of mind and allow the user to continue building a proof of \( B \).
A conjunctive conjecture: the additive case

If $B$ is $B_1 \land B_2$, the user is convinced that $B_1$ and $B_2$ are provable. If the oracle’s proof uses the *additive* rule for conjunction, then it knows that one of these cases has a counterexample.

$$\begin{array}{c}
\Xi_i & \Xi_i \\
\vdash B_1 & \vdash B_2 & B_i \vdash \\
\vdash B_1 \land B_2 & B_1 \land B_2 \vdash & \text{cut.}
\end{array}$$

Permute the cut rule upward in this partial proof.

$$\begin{array}{c}
\Xi_i & \Xi_i \\
\vdash B_i & B_i \vdash \\
\vdash . & \text{cut}
\end{array}$$

Thus the oracle should instruct the user to try to prove $B_i$. The user is informed which of these two cases should be pursued to discover a problem in the formulation of the theorem.
A disjunctive conjecture

If $B$ is $B_1 \lor B_2$, the user is convinced that $B_1$ or $B_2$ is provable.

\[
\begin{align*}
\vdash B_i & \quad \Xi_1 \quad \Xi_2 \\
\vdash B_1 \lor B_2 & \\
\vdash \quad \vdash \quad \vdash \quad \vdash \quad \vdash \quad \vdash \\n\Xi_1 & \quad \Xi_2 \\
B_1 \vdash \quad B_2 \vdash \\
B_1 \lor B_2 \vdash \\
\vdash & \\
\text{cut.}
\end{align*}
\]

Permute the cut rule upward in this partial proof.

\[
\begin{align*}
\Xi_i & \\
\vdash B_i & \quad B_i \vdash \\
\vdash & \\
\text{cut}
\end{align*}
\]

Thus the oracle is prepared to respond to either case that the user wants to explore.
A universally quantified conjecture

If the $B$ is the universally quantified formula $\forall x. B'$, the interaction would provide an actual instance of that quantifier that would lead to a dead-end in the proof attempt.

\[
\begin{array}{c}
\therefore B'x \\
\therefore \forall x. B'x \\
\forall x. B'x \vdash \cdot
\end{array}
\]

\[
\begin{array}{c}
\therefore B't \\
B't \vdash \cdot
\end{array}
\]

\[
\begin{array}{c}
\forall x. B'x \vdash \cdot \\
\cdot \vdash \cdot \text{ cut}
\end{array}
\]

In this case, permuting the cut rule upwards causes the term $t$ to be substituted for the eigenvariable $x$, yielding

\[
\begin{array}{c}
\therefore B't \\
\cdot \vdash \cdot \text{ cut.}
\end{array}
\]

The user is asked to focus on one particular instance of the universal quantifier they believe should be true.
A conjunctive conjecture: the multiplicative case

If the oracle's proof uses the *multiplicative* rule for conjunction, then it knows only that both conjunctions cannot be proved.

\[
\begin{align*}
\cdot \vdash B_1 & \quad \cdot \vdash B_2 \\
\cdot \vdash B_1 \land B_2 & \\
\cdot \vdash B_1 \land B_2 & \vdash \cdot \\
\cdot \vdash \cdot & \equiv \\
B_1, B_2 \vdash \cdot & \equiv \\
B_1, B_2 \vdash \cdot & \equiv \\
B_1, B_2 \vdash \cdot & \vdash \cdot \\
\end{align*}
\]

Permute the cut rule upward in this partial proof.

\[
\begin{align*}
\cdot \vdash B_1 & \quad \cdot \vdash B_2 \\
\cdot \vdash B_1 \land B_2 & \\
\cdot \vdash B_1 \land B_2 & \vdash \cdot \\
\cdot \vdash \cdot & \equiv \\
B_1, B_2 \vdash \cdot & \equiv \\
B_1, B_2 \vdash \cdot & \vdash \cdot \\
\end{align*}
\]

The oracle can claim that if one of \( B_1 \) or \( B_2 \) can be proved then the other one cannot be proved.

The user should attempt to prove the easier or these two first.
Three disciplines: Game theory, ludics, proof theory

- There seems to be a strong connection here between dialogue games for proofs [Hintikka, Lorenzen, etc]. In the interaction between the user and the oracle, the oracle has a winning strategy that is derived from its a formal proof.

- We use cut-elimination on non-proof objects: they necessarily have open premises. Such objects have been called paraproofs. This observation suggests connections also with Ludics [Girard 2001].

- Proof theory, especially, the theory of focused proof systems, can be used to extend these examples.
Focused Proof Systems

Andreoli gave a focused proof system for linear logic in 1991.

Focusing is ambiguous when applied to classical and intuitionistic logics. Liang & M [2009, 2011] have described a general framework for obtaining focused proof systems for those two logics.

Proofs are constructed using phases of inference rules: the invertible (negative) phase and the non-invertible (positive) phase.

These two phases can be related to the moves in a two players game. A precise connection between the cut-free proofs in MALL and winning strategies is given in [Delande, M, & Saurin, 2010].
Synthetic inference rules

Focused proof systems can be used to build *synthetic inference rules*. Cut-elimination automatically holds for such synthetic inference rules [Marin, M, Pimentel, Volpe 2020].

Consider defining a path in graph with adjacency given by \( \text{adj}(\cdot, \cdot) \).

\[
\forall x \ [\text{path}(x, x)] \\
\forall x, y, z \ [\text{adj}(x, y) \land \text{path}(y, z) \supset \text{path}(x, z)]
\]

\[
\therefore \text{path}(x, x) \quad \therefore \text{adj}(x, y) \quad \therefore \text{path}(y, z) \\
\therefore \text{path}(x, z)
\]

These right-rules are justified using focusing within, say, Gentzen’s LK or LJ proof systems.

To provide left-rules, we move beyond logic towards arithmetic.
Unfolding fixed points

The predicate $\text{path}(\cdot, \cdot)$ can be defined as a fixed point using techniques described by Schroeder-Heister (definitional reflection) [1993] and Girard [1992]. In that setting, there is no least or greatest fixed points: this is arithmetic without induction.

When the underlying graph is a finite DAG (directed acyclic graph), the least and greatest fixed points coincide.

If we make equality and $\text{adj}(\cdot, \cdot)$ into side conditions, we have the following right and left introduction rules for $\text{path}(\cdot, \cdot)$.

\[
\begin{align*}
\cdot \vdash \text{path}(x, x) & \quad \cdot \vdash \text{path}(y, z) \\
\cdot \vdash \text{path}(x, z) & \quad \text{provided } \text{adj}(x, y) \\
\{ \text{path}(y, z) \vdash \cdot \ | \ \text{adj}(x, y) \} & \quad \text{provided } x \neq z
\end{align*}
\]
Path or no path in a DAG

Assume that the oracle and user agree on equality of nodes and adjacency in the graph.

Assume that $a, w, b_1, \ldots, b_n$ ($n \geq 0$) are all distinct nodes and that $a$ is adjacent to exactly $b_1, \ldots, b_n$.

\[
\begin{align*}
\vdash & \text{path}(b_j, w) & \text{path}(b_1, w) \vdash \cdot & \vdots
\end{align*}
\]
\[
\begin{align*}
\vdash & \text{path}(a, w) & \text{path}(a, w) \vdash \cdot & \text{cut}
\end{align*}
\]

\[
\begin{align*}
\vdash & \text{path}(b_j, w) & \text{path}(b_j, w) \vdash \cdot & \text{cut}
\end{align*}
\]

If the oracle has a proof of $\text{path}(a, w) \vdash \cdot$, it seems to have no useful information to give to the user.
“You can’t prove a negative”

The meaning of this questionable expression might be rephrased:

*You claim that there is no treasure in this maze. Since I don’t trust you, I will conduct my own search.*

In the finite DAG situation, this means that interacting with the oracle provides no information to convince the skeptic.
A collection of invertible rules (called a negative or asynchronous phase) contains no useful proof information beyond the direct computation of its premises from its conclusion.

E.g. the left-introduction rule for $\text{path}(\cdot, \cdot)$ is invertible.

A collection of non-invertible rules (called a positive or synchronous phase) contains useful proof information that an oracle can communicate.

E.g. the right-introduction rule for $\text{path}(\cdot, \cdot)$ is not invertible.
Least and greatest fixed points

The proof theory of “generic fixed points” has been extended to include least fixed points (induction) and greatest fixed points (co-induction) within intuitionistic and linear logics [Baelde, McDowell, M, Momigliano, Tiu 2000-2012].

These extensions yield Heyting arithmetic and “linearized” arithmetic.

\[
\frac{\Gamma, St \vdash C \quad BSx \vdash Sx}{\Gamma, \mu Bt \vdash C} \quad \text{Induction}
\]

- \(\mu Bt\) is the least fixed point of the predicate operator \(B\) applied then to term \(t\).
- \(S\) is the invariant of this rule.
- While this rule breaks the subformula property, cut-elimination results can still be proved.
No-path example

One way to prove that there is no path from $a$ to $b$ is to find a collection $C$ of nodes such that

\[
\begin{align*}
\cdot &\vdash a \in C \\
\times &\in C \land \text{adj}(x, y) \vdash y \in C \\
b &\in C \vdash \cdot \text{path}(a, b) \\
\end{align*}
\]

The skeptical user can attempt a proof.

\[
\begin{align*}
\vdots \\
\cdot &\vdash \text{path}(a_2, b) \\
\cdot &\vdash \text{path}(a_1, b) \\
\cdot &\vdash \text{path}(a, b)
\end{align*}
\]

Continuing in this way, we have \(\{a, a_1, a_2, \ldots\} \subseteq C\). But there is no information to guide the skeptic user.
Generalizations: Simulation and Bisimulation

In the study of concurrent processes, simulation and bisimulation are defined using greatest fixed points.

\[
\text{sim } P \ Q := \forall A, P' \ [P \xrightarrow{A} P' \supset \exists Q' [Q \xrightarrow{A} Q' \land \text{sim } P' \ Q']] 
\]

This is a bipole, flipping from negative and positive polarity.

The game that arises from examining the winning strategies associated to focused proofs of this formula match exactly Stirling’s games for simulation.

When the transition system (the \( \rightarrow \cdot \cdot \rightarrow \cdot \) relation) is a finite DAG, then the interaction between the user and oracle can proceed as expected.

\[
\cdot \vdash \text{sim } P \ Q \quad \text{sim } P \ Q \vdash \cdot \\
\cdot \vdash \cdot 
\]
Further directions

- What if the oracle also has a partial proof? Maybe that has value if it has enough proof evidence to convince the user.
- More generally, if someone proposes to pay anyone for a proof of $B$, there should also be a value for a proof of $\neg B$.
- Possible implementations
  - Abella: a small proof system that does not yet have a fixed notion of proof-as-a-value: it only has proof scripts.
  - Coq: with the addition of a plugin that implements $\lambda$Prolog, rather sophisticated interactions should be natural to write.


Thank you for your attention

Art by Nadia Miller