GRAY-BOX PROVING IN THEOREMA

Wolfgang Windsteiger

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University Linz (JKU)

Seminar Formal Methods and Automated Reasoning — June 11, 2024

JOHANNES KEPLER
UNIVERSITY LINZ

INTRODUCTION

- A few words on Theorema for “newcomers” . . .
- Default style: White-Box Prover
  - Every single logical step is reflected in a node in the proof tree.
  - Every single node in the proof tree results in an explanation of the proof step.
- Human style: Simple steps are not explained in too much (?) detail.
- Still, not black-box without explanation.

EXAMPLE

Definition: Let p be a partial order on A and $s \in A$. We call s the smallest element in A w.r.t. p iff $p[s,x]$ for all $x \in A$.

Written in Theorema language:

$$\underset{\substack{s,p,A \\ po[p,A] \wedge s \in A}}{\forall} smallest[s,A,p] :\Longleftrightarrow \underset{x \in A}{\forall} p[s,x]$$

If we need to prove $smallest[1,\mathbb{N},div]$, we would create 3 subgoals:

$$po[div,\mathbb{N}] \qquad 1 \in \mathbb{N} \qquad \underset{x \in \mathbb{N}}{\forall} div[1,x]$$

Proof: In order to prove $smallest[1,\mathbb{N},div]$ we have to show:

1. $po[div,\mathbb{N}]$: ...
2. $1 \in \mathbb{N}$: ...
3. $\underset{x \in \mathbb{N}}{\forall} div[1,x]$: ...

WHAT WE ARE AIMING AT ...

- Identify $po[p, A]$ and $s \in A$ as side-conditions in the definition.
- When expanding $smallest[1, \mathbb{N}, div]$ check side-conditions "silently".
- Result: only 1 subgoal, namely $\underset{x \in \mathbb{N}}{\forall} div[1, x]$.
- Proof: In order to prove $smallest[1, \mathbb{N}, div]$, due to ..., we have to show $\underset{x \in \mathbb{N}}{\forall} div[1, x]$.
- The "silent check" should be efficient and cover simple cases.

EFFICIENT CHECKING OF (SIDE) CONDITIONS

We propose an efficient mechanism that allows to prove statements that can be easily derived from a given knowledge base. The intended use of this mechanism within the Theorema system is in places where we need to

- quickly verify the truth of simple statements (e.g., atomic formulas without quantifiers) and
- only need rough informations about the logical reasoning behind the scenes.

We consider a statement U easily derivable (from K) if

- $U \in K$ or
- $(\underset{x}{\forall} S \Rightarrow T) \in K$ with a substitution σ s.t. $T\sigma = U$ and $S\sigma$ is easily derivable.

A SIMPLE RECURSIVE ALGORITHM FOR CHECKING

Input:
- the formula to be checked,
- the knowledge base, and
- a list of formulas already used in the derivation so far.

Output:
- boolean value indicating whether the formula is easily derivable and
- a list of all formulas needed in the entire derivation.

Base cases:

$$\texttt{quickCheck[f\_,\{\_\_\_,f\_,\_\_\_\},U\_]} := \texttt{\{True,Union[U,\{f\}]\}}$$

$$\texttt{quickCheck[f\_,K\_,U\_]} := \texttt{\{False,\{\}\}}$$

Recursion (roughly): for every (quantified) implication $f \equiv \underset{x}{\forall} S \Rightarrow T$:

$$\texttt{quickCheck[T}^{*}\texttt{,K\_,U\_]} := \texttt{quickCheck[S,K,Union[U,\{f\}]]}$$

A NOTE ON IMPLEMENTATION

1. We use rule-based programming style in Mathematica, i.e. instead of nested if-then-else clauses we have individual cases implemented by separate functions that differ by parameter patterns $\rightsquigarrow$ can easily be modified dynamically.
2. Instead of recursion like

$$\mathtt{quickCheck[T}^*\mathtt{,K\_,U\_]} := \mathtt{quickCheck[S,K,Union[U,\{f\}]]}$$

   we implement recursion as

$$\mathtt{quickCheck[T}^*\mathtt{,K\_,U\_]} := \mathtt{Module[}v, \mathit{body}\ \mathtt{/;}\ \ldots\mathtt{qCQ[S,K,U]}\ldots\mathtt{]}$$

   where qCQ is just a wrapper around quickCheck that, instead of returning $\{b, U\}$, returns only the boolean value b and stores the used formulas U in a global variable, from which they can be retrieved later. This allows the quickCheck-mechanism to be used inside boolean conditions directly.

THE NON-ATOMIC CASE

If S or T are propositional formulas we proceed as follows:

- If $f \equiv \underset{x}{\forall}(S \Rightarrow T_1 \wedge \ldots \wedge T_n)$: Since f is equivalent to the conjunction of the individual $\underset{x}{\forall}(S \Rightarrow T_i)$ we generate individual quickCheck-cases for each T_i.
- If $f \equiv \underset{x}{\forall}(S_1 \vee \ldots \vee S_n \Rightarrow T)$: Since f is equivalent to the conjunction of the individual $\underset{x}{\forall}(S_i \Rightarrow T)$ we generate individual quickCheck-cases for each S_i.
- If $f \equiv \underset{x}{\forall}(S_1 \wedge \ldots \wedge S_n \Rightarrow T)$ then backchaining must branch to all the S_i and it delivers `True` only if all individual branches succeed. In the implementation this is reflected by calling `quichCheck` with a list of formulas as first parameter. Details next slide!
- If $f \equiv \underset{x}{\forall}(S \Leftrightarrow T_1 \wedge \ldots \wedge T_n)$ with atomic S then it is processed as if it was an implication. Same for dual case where T is atomic and S is a conjunction.
- If $f \equiv S :\Leftrightarrow T_1 \wedge \ldots \wedge T_n$ then we treat it like an implication. Note that in this case S is always atomic.

BRANCHING WITH FREE VARIABLES

If T does not contain some of the variables ($free(T) = z$ and $y = x \setminus z$)

$$\underset{x}{\forall}(S_1 \wedge S_2 \wedge \cdots \wedge S_n \Rightarrow T) \quad \equiv \quad \underset{z}{\forall}\Big((\underset{y}{\exists} S_1 \wedge \cdots \wedge S_n) \Rightarrow T\Big),$$

i.e., in this case we cannot simply branch and check the S_i independently.

Try to find S_j and σ s.t.

- $free(S_j) = y$ and
- $S_j\sigma \in K$ and
- $\mathtt{qCQ}[S_k\sigma, K, U]$ for all $k \neq j$.

Finding S_j and σ is done in the same recursive pattern as above such that all possibilities are traversed.

APPLICATION 1: KNOWLEDGE EXPANSION

If $\underset{x}{\forall}(S \Rightarrow T) \in K$ and an instance $S\sigma \in K$ then $K := K \cup \{T\sigma\}$.

Instead of computing σ: generate rule `S* :> T` and apply it to all formulas in K.

If the pattern `S*` matches, we found an instance of S, and the rule generates the respective instance of T. (Use pattern matching of Mathematica!)

Simple Example.

$$\underset{x}{\forall}(4|x \Rightarrow even(x)) \quad \rightsquigarrow \quad \texttt{4|x\_ :> even[x].}$$

- Suppose we have $4|20$ in our knowledge base.
- Pattern $4|x\_$ matches $4|20$,
- rule application produces “new knowledge” $even(20)$.
- Corresponds to inferring $even(20)$ from the given knowledge.

APPLICATION 1: KNOWLEDGE EXPANSION

Quite often in mathematics, we have

$$\underset{x}{\forall}(S_1 \wedge S_2 \wedge \cdots \wedge S_n \Rightarrow T),$$

Equivalent formulation as “nested implication”

$$\underset{x}{\forall}\big(S_1 \Rightarrow (S_2 \Rightarrow \ldots \Rightarrow S_n \Rightarrow T)\big)$$

would result in a cumbersome step-by-step inference until finally deriving T.

Alternatively, for any choice $1 \leq i \leq n$, another alternative equivalent formulation is

$$\underset{x}{\forall}\big((S_1 \wedge \cdots \wedge S_{i-1} \wedge S_{i+1} \wedge \cdots \wedge S_n) \Rightarrow (S_i \Rightarrow T)\big),$$

View $S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_n$ as side-conditions, under which we derive T from S_i!

$$\mathtt{S}_i^* \; \mathtt{/;} \; \mathtt{qCQ[\{S}_1, \ldots, \mathtt{S}_{i-1}, \mathtt{S}_{i+1}, \ldots, \mathtt{S}_n\mathtt{\},\ K,\ U]\ :>\ T}$$

with $free(S_i) = x$ would achieve exactly what we need.

APPLICATION 2: GOAL REDUCTION

"Goal-oriented" application of Modus Ponens ("backward chaining"): exactly what `quickCheck` does, but now on the top-level. One reduction step at the time and verbose documentation in the proof (not silent).

$free(T) = z$ and $y = x \setminus z$ and $\{C_1, \ldots, C_k\} = \{S_i \mid free(S_i) \cap y = \emptyset\}$. Then

$$\underset{x}{\forall}(S_1 \wedge S_2 \wedge \cdots \wedge S_n \Rightarrow T) \quad \equiv \quad \underset{z}{\forall}\Big((C_1 \wedge \cdots \wedge C_k) \Rightarrow (\underset{y}{\exists} S'_1 \wedge \cdots \wedge S'_m) \Rightarrow T\Big).$$

View $C_1, \ldots, C_k$ as side-conditions, under which we reduce the goal T!

$$\mathtt{T}^* \ \mathtt{/;}\ \mathtt{qCQ[\{C_1, \ldots, C_k\},\ K,\ U]\ :>}\ \underset{y}{\exists} \mathtt{S}'_1 \wedge \cdots \wedge \mathtt{S}'_m$$

would achieve exactly what we need.

Special case: If $y = \emptyset$, then the goal reduces to `True`, i.e., the proof is finished.

GOAL REDUCTION: EXAMPLES

Example. The quantified implication

$$\underset{f,X,Y,B}{\forall} \left((f\colon X \to Y \wedge B \subseteq Y \wedge \mathcal{I}(X,f) = B) \Rightarrow \mathit{surjective}(f,X,B) \right)$$

would lead to the rule

$$\mathtt{surjective[f\_,X\_,B\_]\ /;\ qCQ[I(X,f) = B,K,U]\ :>\ \underset{Y}{\exists}(f\colon X \to Y \wedge B \subseteq Y)}$$

Proving the surjectivity of $f(x) := x^2$ from $\mathbb{R}$ to $\mathbb{R}_0^+$ reduces to

- finding a Y such that $f\colon \mathbb{R} \to Y$ and $\mathbb{R}_0^+ \subseteq Y$
- provided that we can “easily show” that $\mathcal{I}(\mathbb{R}, f) = \mathbb{R}_0^+$.
- Goal reduction would wait until this is the case.

GOAL REDUCTION: EXAMPLES

Example. The quantified implication

$$\underset{f,X,Y}{\forall} \left((f\colon X \to Y \wedge \mathcal{I}(X,f) = Y) \Rightarrow \mathit{surjective}(f,X,Y) \right)$$

would lead to the rule

$$\mathtt{surjective[f\_,X\_,Y\_]\ /;\ qCQ[\{f\colon X \to Y, I(X,f) = Y\}, K, U]\ :>\ True}$$

FURTHER APPLICATIONS

- Handling of explicit definitions
- Handling of implicit definitions
- Replacement based on (conditional) equalities
- Replacement based on (conditional) equivalences

EXAMPLE

See demo.

JKU

JOHANNES KEPLER
UNIVERSITY LINZ