Computer Supported Modeling and Reasoning

David Basin, Achim D. Brucker, Jan-Georg Smaus, and Burkhart Wolff

April 2005

http://www.infsec.ethz.ch/education/permanent/csmr/

Isabelle: Automation by Proof Search

Burkhart Wolff

Outline of this Part

- Proof search (à la tableaux proving) and backtracking
- Making Calculi more deterministic
- Proof procedures

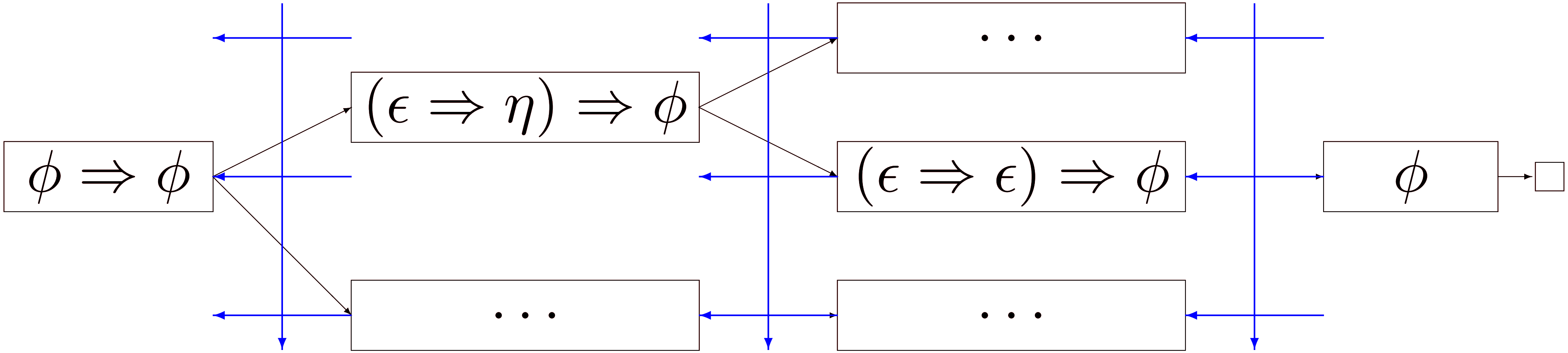
Proof Search and Backtracking

- Need for more automation
- Some aspects in proof construction are highly non-deterministic:
  - unification: which unifier to choose?
  - resolution: where to apply a rule (which 'subgoal')?
  - which rule to apply?
- How to organize proof-search technically?

Organizing Proof Search: Idea 1

Organize proof search as a tree of theorems (`thm`'s).

A sketch of an exemplary proof search:



Summary:

undo to go to previous proof state.

Problems with Idea 1

- Branching of the tree infinite in general (HO-unification)
- Explicit tree representation expensive in time and space
- Not very abstract

Organizing Proof Search: Idea 2

Organize proof search as a relation on theorems (thm's)

$$prooftrees = \mathcal{P}(\texttt{thm} \times \texttt{thm})$$

Advantage: an abstract algebra

- $PT_1 \circ PT_2$: sequential composition ("then")
- $PT_1 \cup PT_2$: alternative of proof attempts ("or")
- PT^* : reflexive transitive closure ("repeat ")
- $(\phi \Rightarrow \phi, \phi) \in PT^* \quad \equiv \quad$ "there is a proof for ϕ"

Problems with Idea 2

- Union $\cup$ is difficult to implement (needs comparison with all previous results).
- More operational, strategic interpretations of union $\cup$ are desirable (try this — then that, interleave attempts in PT_1 with attempts in PT_2, and so forth).

Organizing Proof Search: Idea 3

Organize proof search as a function on theorems (`thm`'s)

$$\texttt{type tactic} = \texttt{thm} \rightarrow \texttt{thm seq}$$

where `seq` is the type constructor for infinite lists.

This allows us to have in ISAR resp. in Isabelle/ML:

- "," or `THEN`
- "|" or `ORELSE`
- "$_*$" or `REPEAT`
- only at Isabelle/ML: `INTLEAVE`, `BREADTHFIRST`, `DEPTHFIRST`, . . .

Making Calculi more Deterministic

Observation: Some rules can always be applied blindly in backward reasoning, e.g. $\longrightarrow$-*I* or $\wedge$-*I*.

$$\dfrac{\dfrac{\dfrac{\rho, \phi, \psi \vdash \phi}{\rho \wedge \phi, \psi \vdash \phi}\ \wedge\text{-}E'}{\rho \wedge \phi \vdash \psi \longrightarrow \phi}\ \longrightarrow\text{-}I}{\vdash (\rho \wedge \phi) \longrightarrow \psi \longrightarrow \phi}\ \longrightarrow\text{-}I$$

The topmost connective is $\longrightarrow$, which asks for $\longrightarrow$-*I*.Again $\longrightarrow$-*I*.To decompose the assumption $\rho \wedge \phi$, use $\wedge$-*E*′.The proof can be completed by assumption.

Problematic Rules

Others are problematic, e.g.:

$$\frac{\Gamma \vdash B}{A, \Gamma \vdash B}\ \textit{weaken} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \text{disjI2} \qquad \frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}\ \text{notnotD}$$

But: proof rules can be tailored such that they may be applied blindly.

Example: $\wedge$-E'

First approach: getting rid of critical rules by fusing them into others.

Consider:

$$\frac{A, B, \Gamma \vdash C}{A \wedge B, \Gamma \vdash C} \; \wedge\text{-}E'$$

It is instructive to reconsider the derivation of $\wedge$-E' which uses weakining inside.

The method `erule` (corresponding to `etac`) has the effect of "internalizing" weakening.

Example: $contraposXX$

Following the fusion approach, we also get alternative versions of contraposition rules:

$$\frac{B, \Gamma \vdash A}{\neg A, \Gamma \vdash \neg B}\ \text{contraposNN} \qquad \frac{\neg B, \Gamma \vdash A}{\neg A, \Gamma \vdash B}\ \text{contraposNP}$$

$$\frac{B, \Gamma \vdash \neg A}{A, \Gamma \vdash \neg B}\ \text{contraposPN} \qquad \frac{B, \Gamma \vdash A}{\neg A, \Gamma \vdash \neg B}\ \text{contraposPP}$$

Thus, with contraposNN, we incorporate the elimination of superfluous negations. contraposPN is useful but can not be applied “blindly” (non-termination).

Example: $\wedge$-E'

Second approach: Use only rules that transform the proof state equivalently (only use “safe rules” or “analytic tableaux rules”).

Instead of

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \text{disjI2}$$

we use:

$$\frac{\neg B, \Gamma \vdash A}{\Gamma \vdash A \vee B}\ \text{disjCI}$$

which does not lose information and avoids backtracking.

Adapting Rules for Automated Proof Search

Based on disjCI and the contraposXX-rules, the following example is deterministic:

$$\dfrac{\dfrac{\dfrac{\dfrac{\neg\alpha, \alpha, \beta \vdash \beta}{\neg\alpha, \beta \vdash \alpha \longrightarrow \beta}\ \longrightarrow\text{-}I}{\neg(\alpha \longrightarrow \beta), \beta \vdash \alpha}\ \text{contraposNP}}{\neg(\alpha \longrightarrow \beta) \vdash \beta \longrightarrow \alpha}\ \longrightarrow\text{-}I}{\vdash (\alpha \longrightarrow \beta) \vee (\beta \longrightarrow \alpha)}\ \text{disjCI1}$$

Neither $\vee$-I_L nor $\vee$-I_R would work here. Use derived rules.
Principle: Emulate sequent calculus with classical logic.
The safe, but non-terminating contraposNP can be avoided by fusing it with all logical junctors.(In this case: $\longrightarrow$).

Handling Quantifiers

Can derive $\forall$-E' ($\equiv$ `allE`) using $\forall$-E ($\equiv$ `spec`):

$$\dfrac{\forall x. A(x) \qquad \begin{matrix}[A(x), \forall x. A(x)]\\ \vdots \\ B\end{matrix}}{B}\ \forall\text{-}dupE$$

What is the difference to $\exists$-E?

Problem: $\forall x. A(x)$ may still be needed.

Principle: Introduce duplicating rules. Turns search infinite!

Check out `allE` and `all_dupE` in `IFOL`!

Proof Procedures (Simplified)

Proof Procedures (Simplified)

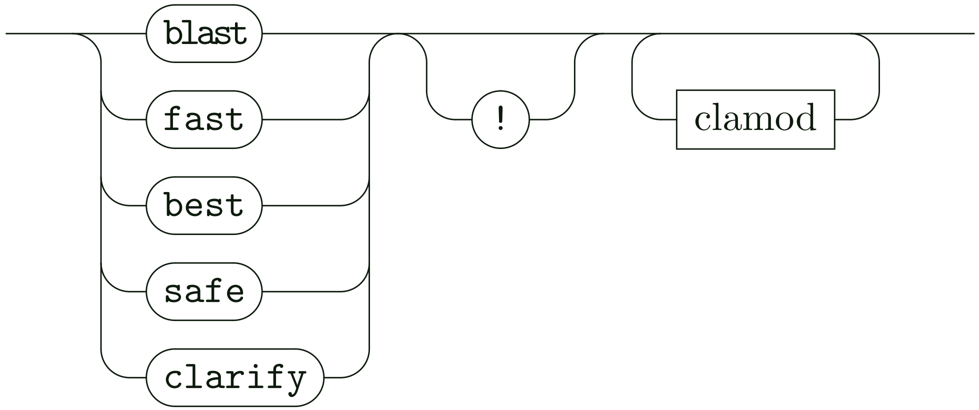
Tactics in Isabelle are performed in order:

1. `DEPTHSOLVE(`
   `REPEAT(rtac` $safe\_I\_rules$ `ORELSE etac` $safe\_E\_rules$`))`
2. canonize: propagate "$x = t$" throughout subgoal
3. `rtac` $unsafe\_I\_rules$ `ORELSE etac` $unsafe\_E\_rules$
4. `atac`

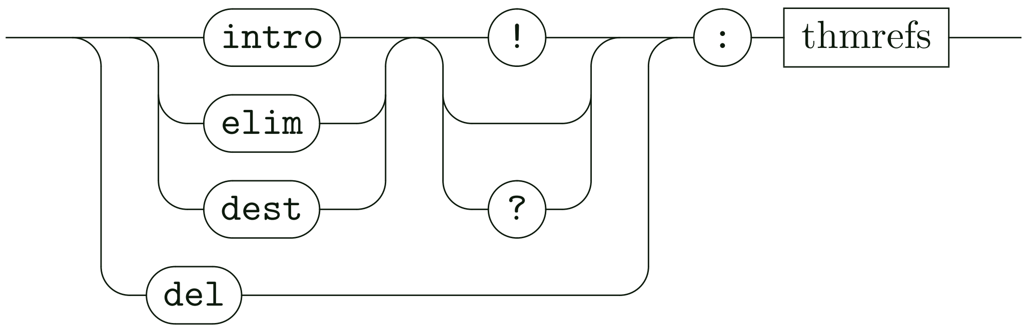
In ISAR, `rtac` is `rule`,`etac` is `erule`, . . .

Combined Proof Search Tactics in ISAR

On the ISAR-level, the syntax for commands accessing the "provers" looks as follows:



clamod



`clamod` allows for introducing new rules (`thm`'s) as introduction, elimination or destruction rules. Rules classified with bang "!" were applied earlier and more agressively as "safe rules".

These commands were mapped to the SML-tactics (described in more detail in the Isabelle Reference Manual [Pau03]).

Safe and Unsafe Rules

On the Isabelle SML level, the rules and their classification were maintained in the data structure `claset`, and accessed by functions of type `claset * thm list` $\longrightarrow$ `claset`.

Class:	To add use function:
Safe introduction rules	`addSIs`
Safe elimination rules	`addSEs`
Unsafe introduction rules	`addIs`
Unsafe elimination rules	`addEs`

Combined Proof Search Tactics

- `fast_tac` : `claset` → `int` → `tactic`
  (safe and unsafe steps in depth-first stategy)
- `best_tac` : `claset` → `int` → `tactic`
  (safe and unsafe steps in breadth-first stategy)
- `blast_tac` : `claset` → `int` → `tactic`
  (like `fast_tac`, but often more powerful)

More details can be found in the Isabelle Reference Manual[Pau03].

Summary on Automated Proof Search

- Proof search can be organized as a tree of theorems.
- Calculi can be set up to facilitate proof search (although this must be done by specialists).
- Combined with search strategies, powerful automatic procedures arise. Can prove well-known hard problems such as $((\exists y.\forall x.J(y,x) \vee \neg J(x,x)) \rightarrow \neg(\forall x.\exists y.\forall z.J(z,y) \vee \neg J(z,x))$
- Unfortunately, failure is difficult to interpret.

More Detailed Explanations

Notion

In this lecture we use both, the ISAR synatx and the “classical” ML based syntax of Isabelle. We first denote the ISAR syntax, followed by the ML syntax, e.g. `assume`/`atac`.

Need for Automation

We have seen in the exercises that proving on a stepwise basis is very tedious and yearns for automation.

Efficiency considerations are also important for automation. The non-determinacy in proof search may lead to deep backtracking which should therefore be avoided.

Idea 1: A Tree of Theorems

We have seen in the previous lecture that resolution transforms a proof state into a new proof state. Since in general, a proof state has several successor states (states that can be obtained by one resolution step), conceptually one obtains a tree where the children of a state are the successors.

The essential point of idea 1 is that the tree is constructed explicitly, as a data-structure.

$\phi \Longrightarrow \phi$?

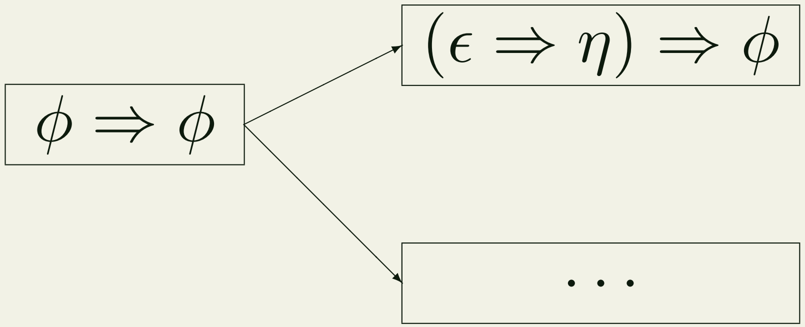
The initial proof state is $\phi \Longrightarrow \phi$. Isabelle will display this as

```
Level 1 : (1 subgoal)
φ
 1. φ
```

Technically, the proof state is an Isabelle theorem (`thm`), i.e. something which Isabelle considers as proven. The aim of a proof search in backward proof is to transform $\phi \Longrightarrow \phi$ into ϕ (ϕ can be shown if I assume nothing).

Idea 2: A Relation on Theorems

One can look at a fragment of a tree of theorems as in idea 1, e.g.:



One could say that each tactic application (with a particular rule) gives rise to a relations on theorems. That is to say, ϕ and ϕ' are in the relation if ϕ' is a successor proof state of ϕ.

This is abstract in that there is no order among the successors of a proof state.

Also, one does not represent a tree explicitly.

Sequential Composition

Given two relations between `thm`'s, PT_1 and PT_2, we define $PT_1 \circ PT_2$ as the relation

$$\{(\phi, \psi) \mid \text{there is } \eta \text{ such that } (\phi, \eta) \in PT_1 \text{ and } (\eta, \psi) \in PT_2\}$$

Union of Relations

The union of two relations is defined as usual for sets. If PT_1 and PT_2 each model the application of a particular tactic, then $PT_1 \cup PT_2$ models the application of "first tactic or second tactic".

Reflexive Transitive Closure

PT^* is inductively defined as the smallest set where

- $(\phi, \phi) \in PT^*$ for all ϕ;
- if $(\phi, \eta) \in PT$ and $(\eta, \psi) \in PT^*$ then $(\phi, \psi) \in PT^*$.

So if PT models the application of a particular tactic, then PT^* models the application of that tactic arbitrarily many times

Idea 2: A Function on Theorems

Idea 3 differs from idea 2 in that it is less abstract, more operational. Instead of saying that ϕ and ϕ' are in a relation, one says that ϕ' is in the sequence returned by the tactic applied to ϕ. There is an order among the successors of a proof state.

One still does not represent a tree explicitly, but by higher-order functions that can compute the rest of a sequence step by step.

Infinite Lists

For any type τ, the type τ `seq` (recall the notation) is the type of (possibly) infinite lists of elements of type τ. This is of course an abstract datatype. There should be functions to return the head and the tail of such an infinite list.

An abstract datatype is a type whose terms cannot be represented explicitly and accessed directly, but only via certain functions for that type.

Tacticals

- `THEN`
- `ORELSE`
- `REPEAT`
- `INTLEAVE`, `BREADTHFIRST`, `DEPTHFIRST`, . . .

are called tacticals.

Tacticals are operations on tactics. They play an important role in automating proofs in Isabelle. The most basic tacticals are `THEN` and `ORELSE`. Both of those tacticals are of type `tactic` $*$ `tactic` $\rightarrow$ `tactic` and are written infix: tac_1 `THEN` tac_2 applies tac_1 and then tac_2, while tac_1 `ORELSE` tac_2 applies tac_1 if possible and otherwise applies tac_2 [Pau03, Ch. 4].

$\wedge$-E

In Isabelle notation, it looks as follows:

$$[\![P \wedge Q;\ [\![P;\ Q]\!] \Longrightarrow R]\!] \Longrightarrow R$$

`claset`

`claset` is an abstract datatype. Overloading notation, `claset` is also an ML unit function which will return a term of that datatype when applied to $()$, namely, the current classifier set.

A classifier set determines which rules are safe and unsafe introduction, respectively elimination rules. The current classifier set is a classifier set used by default in certain tactics.

The current classifier set can be accessed via special functions for that purpose.

Accessing the `claset`

The functions `addSIs`, `addSEs`, `addIs`, `addEs` are all of type `claset` $*$ `thm list` $\rightarrow$ `claset`. They add rules to the current classifier set. For example, `addSIs` adds a rule as safe introduction rule.

Emulating the Sequent Calculus

The sequent calculus works with expressions of the form $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ which should be interpreted as: under the assumptions $A_1, \ldots, A_n$, at least one of $B_1, \ldots, B_m$ can be proven. So as a formula, this would be $A_1 \wedge \ldots \wedge A_n \to B_1 \vee \ldots \vee B_m$.

In Isabelle (and the proof trees we have seen, e.g,. in this lecture), we only have sequents with one formula to the right of the $\vdash$. We have said that we use sequent notation.

Deriving `allE`

You should do it in Isabelle. The rule is:

$$[\![\texttt{ALL}\, x.\ P(x);\ P(x) \Longrightarrow R]\!] \Longrightarrow R$$

The Rule $\vee$-*swap*

The rule `disjCI` is

$$\frac{\neg A, \Gamma \vdash B}{\Gamma \vdash A \vee B}\ \texttt{disjCI}$$

To derive it you need classical reasoning, as the rule exploits the equivalence of A and $\neg\neg A$ (then the rule follows immediately from $\rightarrow$-*I*).

The Rule impE

The rule impE is

$$\frac{A, \neg C, \Gamma \vdash B}{\neg(A \to B), \Gamma \vdash C}\ \text{impE}$$

It essentially "fuses" `contraposNP`, which can not be applied "blindly" due to non-termination, with $\to$-*I*.

This is a standard technique in Isabelle called swapping. In generally, if we have a formula $\neg(A \circ B)$ in the premises, where $\circ$ is some binary connective, swapping will put $(A \circ B)$ in the conclusion and put the old conclusion into the premises after negating it. Afterwards, an introduction rule for $\circ$ will be used [Pau03, Section 11.2].

Duplicating Rules

You should recall that elimination rules are used in combination with `erule`/`etac`. Using `allE` will eliminate the quantifier.

You should try a proof of the formula $(\forall x.P(x)) \to (P(a) \wedge P(b))$ in Isabelle to convince yourself that this is a problem since the quantified formula $\forall x.P(x)$ is needed twice as an assumption, with two different instantiations of x.

The duplicating rule $\forall$-*dupE* has the effect that the universally quantified formula will still remain as an assumption.

Proof Procedures

Tactics in Isabelle are performed in order:

1. `DEPTHSOLVE(REPEAT(rtac` $safe\_I\_rules$ `ORELSE etac` $safe\_E\_rules$`))`;
2. canonize: propagate "$x = t$" . . . throughout subgoal;
3. `rtac` $unsafe\_I\_rules$ `ORELSE etac` $unsafe\_E\_rules$;
4. `atac`.

One elementary proof step consists of trying a safe introduction rule with `rtac`, or, if that is not possible, a safe elimination rule with `etac`. This will be repeated as long as possible.

Then in the current subgoal, any assumption of the form $x = t$ (where x is a metavariable) will be propagated throughout the subgoal, i.e., all occurrences of x wil be replaced by t.

Then Isabelle will try one application of an unsafe introduction rule with `rtac`, or, if that is not possible, an unsafe elimination rule with `etac`. Finally, she will use `assumption`/`atac`. Note that `assumption`/`atac` is unsafe. In general, there are several premises in a subgoal and `atac` may unify the conclusion of the subgoal with the wrong premise. Different search strategies were applied.

References

[Pau03] Lawrence C. Paulson. *The Isabelle Reference Manual*. Computer Laboratory, University of Cambridge, March 2003.