Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Dipl.-Inf. Achim D. Brucker
Dr. Burkhart Wolff

Computer-supported Modeling and Reasoning
http://www.infsec.ethz.ch/education/permanent/csmr/
(rev. 16814)
Submission date: –

First-Order Logic

In this lecture you will deepen your knowledge about *first-order logic* (FOL). Theorem proving in FOL involves the issue of binding and substitution, which we treat at a fairly pragmatic level for the moment. (The issue will be revisited in the subsequent exercises on meta-theory and λ-calculus). We will learn to manage premises in backward proofs and tactic methods that manipulate assumptions in backward proofs.

1 More on Isabelle

1.1 Isabelle System Architecture

For using Isabelle it is sometimes helpful if one has a broad overview of Isabelle's system architecture (see Fig. 1). Isabelle is generic theorem prover providing a simple meta logic; it is implemented the functional language "Standard ML" (SML). On top of the Isabelle core, a variety of Isabelle instances are built, e.g. Isabelle/FOL (for first-order logic), Isabelle/HOL (for higher-order logic), or Isabelle/ZF (for Zermelo-Fränkel set theory). Isabelle instances can be programmed directly via programs written in SML or via the ISAR-proof language, which also provides powerful documentation facilities. On top of this, different SML

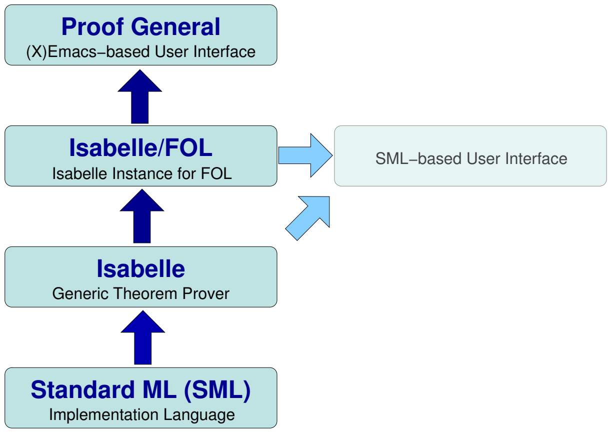

Figure 1: The System Architecture of Isabelle

Proof General

user interfaces are provided. In this lecture, we use the most modern interface, called "Proof General", which itself builds upon the (X)Emacs editor family.

1.2 Assumptions in Backward Proof

assumptions

In backward proof, Isabelle allows two notions for introducing assumptions into a proof context. For simple cases we can use

lemma *name*: "$\llbracket\, a_1; \ldots; a_n \rrbracket \Longrightarrow C$"

The latter format introduces assumptions as named objects that can be referenced identically to rules:

lemma *name*:	**lemma** *name*:
assumes $name_1$: "a_1"	**assumes** $name_1$: "a_1"
$\vdots$	$\vdots$
assumes $name_n$: "a_n"	and $name_n$: "a_n"
shows "C"	**shows** "C"

assumes
and
insert

One can use the **assumes** or and to enumerate several assumptions. Using this style, the assumptions are not automatically added to assumption list of the goals. If needed, you can insert them with the command *insert*.

Assumptions, derived rules, rules, axioms, theorems are all the same in Isabelle and can be combined in arbitrary ways in forward and backward proof! Remark: Internally, all these objects are represented as a particular abstract data type `thm`. The Isabelle kernel is a collection of SML-modules that imple-

`thm`

ment this data type (provers of this system architecture are often referred as LCS-style-provers). LCS-style

1.3 New FOL Rules

The distinctive feature of FOL compared to PL are the quantifiers $\forall$ and $\exists$. Note that quantifiers have low priority, e.g., we have to write $(\forall x.p(x)) \longrightarrow (\exists x.p(x))$.

Recall the introduction and elimination quantifier rules from the lecture:

$$\frac{P(x)}{\forall x.\,P(x)}\ \forall\text{-}I^{1.} \qquad \frac{\forall x.\,P(x)}{P(t)}\ \forall\text{-}E \qquad \frac{P(t)}{\exists x.\,P(x)}\ \exists\text{-}I \qquad \frac{\exists x.\,P(x) \qquad \begin{matrix}[P(x)]\\ \vdots \\ R\end{matrix}}{R}\ \exists\text{-}E^{2.}$$

quantifier rules

where the side conditions are:

1. x is not free in any assumption on which $P(x)$ depends.

2. x is not free in B or any assumption of the sub-derivation of B other than $A(x)$.

In Isabelle/FOL, these rules are represented (including the side conditions) as follows:

spec
allI
exI
exE

$$(\bigwedge x.P(x)) \Longrightarrow (\forall x.P(x)) \quad \texttt{allI} \qquad (\bigwedge x.P(x)) \Longrightarrow P(x) \quad \texttt{spec}$$

$$P(x) \Longrightarrow (\exists x.P(x)) \quad \texttt{exI} \qquad \llbracket \exists x.P(x); \bigwedge x.P(x) \Longrightarrow R \rrbracket \Longrightarrow R \quad \texttt{exE}$$

Where $\bigwedge$ is the meta-level universal quantification. If a goal is preceded by the meta-quantor $\bigwedge x.\ldots$, this means that Isabelle must be able to prove the subgoal in a way which is independent from x, i.e., without instantiating x. Another view on meta-level quantification is that they introduce "fresh free variables" on the fly (in fact, variables bound by outermost meta-level quantifiers were treated as free variables within substitutions).

meta-level universal quantifier
$\bigwedge$

Whenever an application of a rule leads to the introduction of meta variables in a goal preceded by $\bigwedge$, these introduced meta variables will be made dependent on x. You may also say that those meta-variables will be *Skolem* functions of x. When experimenting with rule applications introducing $\bigwedge$'s, you will notice that the order of these introductions is crucial.

1.4 Substitutions in Backward Proof

As mentioned in the lecture, Isabelle uses meta-variables $?X$,$?Y$,.... These meta-variables are *logically* treated as *free variables*, but may be instantiated either interactively or automatically by Isabelle itself.

Sometimes the automatic instantiation is not appropriate for a proof; then the user must provide it interactively. In forward proof, this can be done by the **of** command you have already got to know.

In backward proof, variants of proof commands were provided. Instead of

apply(*rule* *name*)

rule_tac we might give several substitution during rule application :

apply(*rule_tac* **of** x_1= "*term*$_1$" and ...and x_n= "*term*$_n$" **in** *rule*)

Where *rule_tac* may contain syntactic elements and free variables of the proof context. Note that

apply(*rule_tac* **of** x = "*term*" **in** *rule*)

is *not* the same as

apply(*rule* [**of** ..."*term*" ...])

Can you figure out why?

1.5 Manipulating Assumptions in Backward Proof

So far, we never changed the assumptions a_i of a goal $[\![a_1; \ldots ; a_n]\!] \Longrightarrow C$. The command *rule* instantiates its argument rule such that its conclusion becomes equal to the conclusion C of the goal.

A collection of Isabelle tactic methods follows a different strategy:

erule

1. *erule* *rule* constructs an instantiation such that the first assumption b_1 of *rule* becomes equal to an a_i, and that the conclusion of *rule* becomes equal to an C. a_i is erased from the assumptions.

drule

2. *drule* *rule* constructs an instantiation such that the first assumption b_1 of *rule* becomes equal to an a_i, and that the conclusion of *rule* becomes a new assumption. a_i is erased from the assumptions.

frule

3. *drule* *rule* works like *drule* *rule* but does not erase a_i.

insert Moreover, with the command *insert*, an arbitrary theorem or assumption can be added to the assumption list.

Note that for some of these tactic methods are variants with explicit substitutions available: erule_tac, drule_tac, and frule_tac.

erule_tac
drule_tac
frule_tac

2 Exercises

2.1 Exercise 5

Derive the following rules in Isabelle:

$$\frac{P \wedge Q \qquad \begin{array}{c}[P, Q]\\ \vdots \\ R\end{array}}{R}\ \texttt{conjE} \qquad \frac{P \longrightarrow Q \quad P \quad \begin{array}{c}[Q]\\ \vdots \\ R\end{array}}{R}\ \texttt{impE} \qquad \frac{\begin{array}{c}[P]\\ \vdots \\ \bot\end{array}}{\neg P}\ \texttt{notI} \qquad \frac{\neg P \quad P}{R}\ \texttt{notE}$$

conjE, impE
notI, notE

2.2 Exercise 6

Prove the following theorems using `erule` and `disjE` and `conjE` wherever possible.

1. $(A \wedge B) \wedge C \longrightarrow A \wedge B \wedge C$
2. $(A \wedge B) \wedge (C \wedge D) \longrightarrow (B \wedge C) \wedge (D \wedge A)$
3. $(A \vee B) \vee (C \vee D) \longrightarrow (B \vee C) \vee (D \vee A)$

Compare the first two proofs with the proofs without `erule` in Ex. 1.

2.3 Exercise 7

Derive the rule

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array} \quad \begin{array}{c}[B]\\ \vdots \\ A\end{array}}{A \longleftrightarrow B}$$

in Isabelle. Recall that $\longleftrightarrow$ is defined by:

$$P \longleftrightarrow Q \equiv (P \longrightarrow Q) \wedge (Q \longrightarrow P) \qquad \texttt{iff\_def}$$

Use *erule* and *drule* wherever you can.

2.4 Exercise 8

Prove the following theorems of first-order logic in Isabelle:

1. $(\forall x.p(x)) \longrightarrow \exists x.p(x)$
2. $((\forall x.p(x)) \vee (\forall x.q(x))) \longrightarrow (\forall x.(p(x) \vee q(x)))$
3. $((\forall x.p(x)) \wedge (\forall x.q(x))) \longleftrightarrow (\forall x.(p(x) \wedge q(x)))$

4. $(\exists x.\forall y.p(x, y)) \longrightarrow (\forall y.\exists x.p(x, y))$
5. $(\exists x.p(f(x))) \longrightarrow (\exists x.p(x))$

What about: $(\forall x.(p(x) \vee q(x))) \longrightarrow ((\forall x.p(x)) \vee (\forall x.q(x)))$? Can you prove it?

2.5 Exercise 9

Prove

$$\frac{}{(\forall x.A \longrightarrow B(x)) \longleftrightarrow (A \longrightarrow \forall x.B(x))}\ \texttt{all\_distr}$$

in Isabelle. Reuse Exercise 7.

In lecture '1.5 FOL: Natural Deduction" it was said that in the above theorem it is crucial that "A does not contain x freely". How does Isabelle take this into account? Try to prove: $p(x) \longrightarrow \forall x.p(x)$

2.6 Exercise 10

Prove the following theorem of first-order logic in Isabelle:

$$s\,(s\,(s\,(s\,(zero)))) = four \wedge p(zero) \wedge (\forall x.p(x) \longrightarrow p(s(s(x)))) \longrightarrow p(four)$$