Interactive Formal Verification
7: Inductive Definitions

Tjark Weber
(Slides: Lawrence C Paulson)
Computer Laboratory
University of Cambridge
Defining a Set Inductively
Defining a Set Inductively

- The set of even numbers is the least set such that
  - 0 is even.
  - If \( n \) is even, then \( n+2 \) is even.
Defining a Set Inductively

• The set of even numbers is the least set such that
  • 0 is even.
  • If $n$ is even, then $n+2$ is even.
• These can be viewed as *introduction rules*. 
Defining a Set Inductively

- The set of even numbers is the least set such that
  - 0 is even.
  - If $n$ is even, then $n+2$ is even.
- These can be viewed as *introduction rules*.
- We get an *induction principle* to express that no other numbers are even.
Defining a Set Inductively

- The set of even numbers is the least set such that
  - 0 is even.
  - If $n$ is even, then $n+2$ is even.
- These can be viewed as *introduction rules*.
- We get an *induction principle* to express that no other numbers are even.
- Induction is used throughout mathematics, and to express the semantics of programming languages.
Inductive Definitions in Isabelle

theory Ind
imports Main
begin

subsubsection{Inductive definition of the even numbers}

inductive_set Ev :: "nat set" where
  ZeroI: "0 ∈ Ev"
  Add2I: "n ∈ Ev ==> Suc(Suc n) ∈ Ev"

Proofs for inductive predicate(s) "Evp"
  Proving monotonicity ...

*response*
Even Numbers Belong to Ev

```text
*All even numbers belong to this set.*

lemma "2*k : Ev"
apply (induct k)
apply auto
apply (auto simp add: ZeroI Add2I)
done
```

proof (prove): step 1

goal (2 subgoals):
1. 2 * 0 ∈ Ev
2. ∀k. 2 * k ∈ Ev → 2 * Suc k ∈ Ev
Even Numbers Belong to $\text{Ev}$

ordinary induction yields two subgoals
Proving Set Membership

```
lemma "2\cdot k ∈ Ev"  
apply (induct k)  
apply auto  
apply (auto simp add: ZeroI Add2I)  
done
```

```
proof (prove): step 2

goal (2 subgoals):
1. 0 ∈ Ev
2. ∀k. 2 * k ∈ Ev ⇒ Suc (Suc (2 * k)) ∈ Ev
```

```
tool-bar next
```
Proving Set Membership

After simplification, the subgoals resemble the introduction rules.
Finishing the Proof

```plaintext
*All even numbers belong to this set.*

lemma "2*k : Ev"
apply (induct k)
apply auto
apply (auto simp add: ZeroI Add2I)
done
```

```
proof (prove): step 3

goal:
No subgoals!
```
Finishing the Proof

We have used these as *conditional rewrite rules*. 
Finishing the Proof

We have used these as conditional rewrite rules.
Finishing the Proof

We have used these as conditional rewrite rules. Isabelle also supports introduction rules (backward chaining).
Rule Induction
Rule Induction

• Proving something about every element of the set.
Rule Induction

- Proving something about every element of the set.
- It expresses that the inductive set is minimal.
Rule Induction

• Proving something about every element of the set.
• It expresses that the inductive set is minimal.
• It is sometimes called “induction on derivations”
Rule Induction

- Proving something about every element of the set.
- It expresses that the inductive set is \textit{minimal}.
- It is sometimes called “induction on derivations”
- There is a \textit{base case} for every non-recursive introduction rule
Rule Induction

- Proving something about every element of the set.
- It expresses that the inductive set is *minimal*.
- It is sometimes called “induction on derivations”
- There is a *base case* for every non-recursive introduction rule
- ...and an *inductive step* for the other rules.
\textbf{Ev Has only Even Numbers}

\begin{verbatim}
lemma "n \in Ev \implies \exists k. n = 2*k"
  apply (induct n rule: Ev.induct)
  apply auto
  apply arith
  done

proof (prove): step 0

goal (1 subgoal):
  1. n \in Ev \implies \exists k. n = 2 * k
\end{verbatim}
\textbf{Ev} Has only Even Numbers

\begin{verbatim}
lemma "n \in Ev \implies \exists k. n = 2 \times k"
apply (induct n rule: Ev.induct)
apply auto
apply arith
done

proof (prove): step 0

goal (1 subgoal):
1. \( n \in Ev \implies \exists k. n = 2 \times k \)
\end{verbatim}
Ev Has only Even Numbers

```isar
lemma "n ∈ Ev ⟷ ∃k. n = 2*k"
apply (induct n rule: Ev.induct)
apply auto
apply arith
done
```

```
proof (prove): step 0

goal (1 subgoal):
1. n ∈ Ev ⟷ ∃k. n = 2 * k
```
An Example of Rule Induction

```isar
lemma "\(\forall n \in \text{Ev} \Rightarrow \exists k. n = 2*k\)"
apply (induct n rule: Ev.induct)
apply auto
apply arith
done
```

proof (prove): step 1

goal (2 subgoals):
1. \(\exists k. 0 = 2 * k\)
2. \(\forall n. [n \in \text{Ev}; \exists k. n = 2 * k] \Rightarrow \exists k. \text{Suc} (\text{Suc} n) = 2 * k\)
An Example of Rule Induction

base case: $n$ replaced by 0
An Example of Rule Induction

**Base Case:** $n$ replaced by 0

**Induction Step:** $n$ replaced by $\text{Suc} \ (\text{Suc} \ n)$
Nearly There!

text{*All elements of this set are even.*}
lemma "n ∈ Ev ⇒ ∃k. n = 2*k" 
apply (induct n rule: Ev.induct)
apply auto
apply arith
done

proof (prove): step 2

goal (1 subgoal):
1. ∀k. 2 * k ∈ Ev ⇒ ∃ka. Suc (Suc (2 * k)) = 2 * ka

tool-bar next
Nearly There!

text{*All elements of this set are even.*}
lemma "n ∈ Ev ⇒ ∃k. n = 2*k"
apply (induct n rule: Ev.induct)
apply auto
apply arith
done

proof (prove): step 2

goal (1 subgoal):
1. ∀k. 2 * k ∈ Ev ⇒ ∃ka. Suc (Suc (2 * k)) = 2 * ka

Too difficult for auto
Nearly There!

```
text{*All elements of this set are even.*}
lemma "n ∈ Ev ⇒ ∃k. n = 2*k"
apply (induct n rule: Ev.induct)
apply auto
apply arith
done
```

```
proof (prove): step 2

goal (1 subgoal):
  1. ∀k. 2 * k ∈ Ev ⇒ ∃ka. Suc (Suc (2 * k)) = 2 * ka
```

Too difficult for auto
The \textbf{arith} Proof Method

\begin{verbatim}
lemma "n ∈ Ev ⇒ ∃k. n = 2*k"
apply (induct n rule: Ev.induct)
apply auto
apply arith
done

proof (prove): step 3

goal:
No subgoals!
\end{verbatim}
The arith Proof Method

for hard arithmetic subgoals
Defining Finiteness

subsection{* Proofs about finite sets *}

text{*The finite powerset operator*}

inductive_set Fin :: "a set set" where
  emptyI: "{} ∈ Fin"
| insertI: "A ∈ Fin ==> insert a A ∈ Fin"

declare Fin.intros [intro]
Defining Finiteness

```
subsection{*

proofs about finite sets *}

text{*
The finite powerset operator*}

inductive_set Fin :: "a set set" where
   emptyI: "{} ∈ Fin"
| insertI: "A ∈ Fin ==> insert a A ∈ Fin"

declare Fin.intros [intro]
```

make the rules available to auto, blast
The Union of Two Finite Sets

proof (prove): step 1

goal (2 subgoals):
1. \( B \in \text{Fin} \rightarrow \{\} \cup B \in \text{Fin} \)
2. \( \forall A. [A \in \text{Fin}; B \in \text{Fin} \rightarrow A \cup B \in \text{Fin}; B \in \text{Fin}] \rightarrow \text{insert} \ a \ A \cup B \in \text{Fin} \)
The Union of Two Finite Sets

perform induction on A
A Subset of a Finite Set

proof (prove): step 1

goal (2 subgoals):
1. \( \forall B. B \subseteq \emptyset \Rightarrow B \in \text{Fin} \)
2. \( \forall A a B. (A \in \text{Fin}; \ \forall B. B \subseteq A \Rightarrow B \in \text{Fin}; B \subseteq \text{insert a A}) \Rightarrow B \in \text{Fin} \)
A Subset of a Finite Set

To prove that every subset of \( A \) is finite.

```isar
to prove that every subset of \( A \) is finite
```

```isar
proof (prove): step 1

goal (2 subgoals):
1. \( \forall B. \; B \subseteq \{\} \implies B \in \text{Fin} \)
2. \( \forall A \; a \in A. \; [A \in \text{Fin}; \; \forall B. \; B \subseteq A \implies B \in \text{Fin}; \; B \subseteq \text{insert \( a \) \( A \)}] \implies B \in \text{Fin} \)
```
A Subset of a Finite Set

To prove that every subset of A is finite, as seen in the induction hypothesis.
A Crucial Point in the Proof

now what??
Time to Try Sledgehammer!
Success!
Success!

this command should prove the goal
Success!

This command should prove the goal.

This one may return a more compact command.
The Completed Proof

```
lemma "[\forall A \in \text{Fin}; B \subseteq A \implies B \in \text{Fin}]
apply (induct A arbitrary: B rule: Fin.induct)
apply auto
apply (metis Fin.insert1 Int_absorb1 Int_commute Int_insert_right Int_lower1 mem_def subset_insert)
```

```
proof (prove): step 3

goal:
No subgoals!
```
How Sledgehammer Works

Isabelle
How Sledgehammer Works

Isabelle

Problem and 100s of lemmas

E

SPASS

Vampire
How Sledgehammer Works

Isabelle

Problem and 100s of lemmas

Proof

E

SPASS

Vampire
How Sledgehammer Works

Isabelle

Problem and 100s of lemmas

Proof

E

SPASS

Vampire

Theorem provers run in the *background*. Isabelle can still be used!
Notes on Sledgehammer
Notes on Sledgehammer

• It is always available, though it usually fails...
Notes on Sledgehammer

• It is always available, though it usually fails...

• It does not prove the goal, but returns a call to metis. This command usually works...
Notes on Sledgehammer

- It is always available, though it usually fails...
- It does not prove the goal, but returns a call to `metis`. This command *usually* works...
- The minimise option removes redundant theorems, increasing the likelihood of success.
Notes on Sledgehammer

• It is always available, though it usually fails...

• It does not prove the goal, but returns a call to `metis`. This command *usually* works...

• The minimise option removes redundant theorems, increasing the likelihood of success.

• Calling `metis` directly is difficult unless you know exactly which lemmas are needed.