

CS576 Topics in Automated Deduction

Elsa L Gunter
2112 SC, UIUC
egunter@illinois.edu

<http://courses.grainger.illinois.edu/cs576>

Slides based in part on slides by Tobias Nipkow

February 26, 2026

Proving Properties of Even Numbers

Induction leads to two cases:

- **rule:** $0 \in \text{Ev}$

1. $0 + 0 \in \text{Ev}$ case $m = 0$

- **rule:** $n \in \text{Ev} \implies n + 2 \in \text{Ev}$

- z 2. $\forall n. [n \in \text{Ev}; n + n \in \text{Ev}] \implies \text{Suc}(\text{Suc}n) + \text{Suc}(\text{Suc}n) \in \text{Ev}$

case $m = n + 2$

Rule Induction for Ev

To prove

$$n \in Ev \implies P \ n$$

by *rule induction* on $n \in Ev$ we must prove

- $P \ 0$
- $P \ n \implies P(n + 2)$

Uses rule `Ev.induct`:

$$\llbracket n \in Ev; P \ 0; \ \wedge n. P \ n \implies P(n + 2) \rrbracket \implies P \ n$$

An elimination rule

Rule Induction in General

Set S is defined inductively. To prove

$$x \in S \implies P x$$

by *rule induction* on $x \in S$ we must prove for every rule

$$\llbracket a_1 \in S; \dots; a_n \in S \rrbracket \implies a \in S$$

that P is preserved:

$$\llbracket P a_1; \dots; P a_n \rrbracket \implies P a$$

In Isabelle/HOL:

```
proof(rule S.induct)
```

or

```
apply(erule S.induct)
```

Demo: Inductive Set Definition

Demo: Evens are infinite

Format for Inductive Relations Definitions

inductive $R :: \tau \rightarrow \text{bool}$ where

$\llbracket R(a_{1,1}); \dots; R(a_{1,n}); A_{1,1}; \dots; A_{1,k} \rrbracket \Longrightarrow R(a_1) \mid$

$\dots \mid$

$\llbracket R(a_{m,1}); \dots; R(a_{m,l}); A_{m,1}; \dots; A_{m,j} \rrbracket \Longrightarrow R(a_m)$

where $A_{i,j}$ are side conditions not involving R .

Format for Inductive Relations Definitions

inductive R :: " $\tau \rightarrow \text{bool}$ " where

$\llbracket R(a_{1,1}); \dots; R(a_{1,n}); A_{1,1}; \dots; A_{1,k} \rrbracket \implies R(a_1) \mid$
... \mid

$\llbracket R(a_{m,1}); \dots; R(a_{m,1}); A_{m,1}; \dots; A_{m,j} \rrbracket \implies R(a_m)$

where $A_{i,j}$ are side conditions not involving R.

Format for Mutual Inductive Relations Definitions

inductive

$R_1 :: \tau_1 \longrightarrow \text{bool}$ and

...

$R_n :: \tau_n \longrightarrow \text{bool}$ where

$\llbracket R_i(a_{1,1}); \dots; R_j(a_{1,n}); A_{1,1}; \dots; A_{1,k} \rrbracket \implies R_k(a_1) \mid$
... |

$\llbracket R_m(a_{m,1}); \dots; R_n(a_{m,1}); A_{m,1}; \dots; A_{m,j} \rrbracket \implies R_p(a_m)$

where $A_{i,j}$ are side conditions not involving any R_k .

Example with Mutual Recursion

```
inductive
Even :: "nat  $\Rightarrow$  bool" and
Odd  :: "nat  $\Rightarrow$  bool" where
ZeroEven [intro!]: "Even 0" |
OddOne  [intro!]: "Odd (Suc 0)" |
OddSucEven [intro]: "Odd n  $\Rightarrow$  Even (Suc n)" |
EvenSucOdd [intro]: "Even n  $\Rightarrow$  Odd (Suc n)"
```

General Recursive Functions: `fun`

Example:

```
fun fib :: "nat  $\Rightarrow$  nat" where
  "fib 0 = 0" |
  "fib 1 = 1" |
  "fib (Suc(Suc x)) = (fib x + fib (Suc x))"
```

Not primitive recursive because of `fib(Suc(Suc x))` on left, and because of `fib(Suc x)` on right.

fun: Rules of Use

Compared to `primrec`, very few restrictions:

- Can be used to define functions over any type
- Clauses in `fun` must be equations
- Left-hand side is function being defined applied to terms built from data constructors, distinct variables and wildcards
- Right-hand side is a expression made from the function being defined, the variables in the argument on the left, and previously defined terms
- If clauses overlap, first takes precedence.
- Calculates a measure from lexicographic ordering of some collection of arguments

Example: `sep`

Define a function for putting a separator between all adjacent elements in a list:

```
fun sep :: "'a * 'a list => 'a list" where
  "sep(a, []) = []" |
  "sep(a, [x]) = [x]" |
  "sep(a, x#y#zs) = x # a # sep(a,y#zs)"
```