Theorem Proving in Isabelle

Lutz Schröder

December 21, 2001

Recursion

(Primitive) recursion

(Primitive) recursion

- Recursion (in general) defines a function f on a datatype uniquely by means of certain equations

(Primitive) recursion

- Recursion (in general) defines a function f on a datatype uniquely by means of certain equations
- Primitive recursion:

(Primitive) recursion

- Recursion (in general) defines a function f on a datatype uniquely by means of certain equations
- Primitive recursion:
  - recursion over one argument only

(Primitive) recursion

- Recursion (in general) defines a function f on a datatype uniquely by means of certain equations
- Primitive recursion:
  - recursion over one argument only
  - one equation

$$f(\ldots, C_i(y_1, \ldots, y_{m_i}), \ldots) = \ldots$$

for each constructor C_i

(Primitive) recursion

- Recursion (in general) defines a function f on a datatype uniquely by means of certain equations
- Primitive recursion:
  - recursion over one argument only
  - one equation

    $$f(\ldots, C_i(y_1, \ldots, y_{m_i}), \ldots) = \ldots$$

    for each constructor C_i
  - recursive calls of f on the right side use only the y_i.

(Primitive) recursion

- Recursion (in general) defines a function f on a datatype uniquely by means of certain equations
- Primitive recursion:
  - recursion over one argument only
  - one equation

    $$f(\ldots, C_i(y_1, \ldots, y_{m_i}), \ldots) = \ldots$$

    for each constructor C_i
  - recursive calls of f on the right side use only the y_i.

Primitive recursion: syntax

Primitive recursion: syntax

- Declare functions by `consts`

Primitive recursion: syntax

- Declare functions by `consts`
- Define functions by `primrec`

Primitive recursion: syntax

- Declare functions by `consts`
- Define functions by `primrec`
- Put all equations in quotes!

Primitive recursion: syntax

- Declare functions by `consts`
- Define functions by `primrec`
- Put all equations in quotes!

Examples

General recursion

General recursion

- Primitive recursion doesn't always suffice

General recursion

- Primitive recursion doesn't always suffice
- Classical example: the Fibonacci numbers

$$\begin{aligned} f(0) = f(1) &= 1 \\ f(Suc(Suc(n))) &= f(Suc(n)) + f(n) \end{aligned}$$

General recursion

- Primitive recursion doesn't always suffice
- Classical example: the Fibonacci numbers

$$\begin{aligned} f(0) = f(1) &= 1 \\ f(Suc(Suc(n))) &= f(Suc(n)) + f(n) \end{aligned}$$

- Generalized recursion (on datatype t):

General recursion

- Primitive recursion doesn't always suffice
- Classical example: the Fibonacci numbers

$$\begin{aligned} f(0) = f(1) &= 1 \\ f(Suc(Suc(n))) &= f(Suc(n)) + f(n) \end{aligned}$$

- Generalized recursion (on datatype t):
  - relies on measure function $t \longrightarrow nat$

General recursion

- Primitive recursion doesn't always suffice
- Classical example: the Fibonacci numbers

$$\begin{aligned} f(0) = f(1) &= 1 \\ f(Suc(Suc(n))) &= f(Suc(n)) + f(n) \end{aligned}$$

- Generalized recursion (on datatype t):
  - relies on measure function $t \longrightarrow nat$
  - arbitrary patterns on the left side of equations

General recursion

- Primitive recursion doesn't always suffice
- Classical example: the Fibonacci numbers

$$\begin{aligned} f(0) = f(1) &= 1 \\ f(Suc(Suc(n))) &= f(Suc(n)) + f(n) \end{aligned}$$

- Generalized recursion (on datatype t):
  - relies on measure function $t \rightarrow nat$
  - arbitrary patterns on the left side of equations
  - arguments of f on the right side must have provably smaller measure (this proves termination: a natural number can only become smaller so many times.)

General recursion: syntactic details

General recursion: syntactic details

- Use `recdef` in place of `primrec`

General recursion: syntactic details

- Use `recdef` in place of `primrec`
- Explicitly name function f to be defined

General recursion: syntactic details

- Use `recdef` in place of `primrec`
- Explicitly name function f to be defined
- Provide measure function by `"measure(%x. ...)"`

General recursion: syntactic details

- Use `recdef` in place of `primrec`
- Explicitly name function f to be defined
- Provide measure function by `"measure(%x. ...)"`
- Recursion only for argument before the first '$\Rightarrow$':

General recursion: syntactic details

- Use `recdef` in place of `primrec`
- Explicitly name function f to be defined
- Provide measure function by `"measure(%x. ...)"`
- Recursion only for argument before the first '$\Rightarrow$':
  - If necessary, rearrange arguments, or

General recursion: syntactic details

- Use `recdef` in place of `primrec`
- Explicitly name function f to be defined
- Provide measure function by `"measure(%x. ...)"`
- Recursion only for argument before the first '$\Rightarrow$':
  - If necessary, rearrange arguments, or
  - collect several arguments into a tuple (type $\ldots * \ldots$)

General recursion: syntactic details

- Use `recdef` in place of `primrec`
- Explicitly name function f to be defined
- Provide measure function by `"measure(%x. ...)"`
- Recursion only for argument before the first '$\Rightarrow$':
  - If necessary, rearrange arguments, or
  - collect several arguments into a tuple (type $\ldots * \ldots$)
- Special induction rule f`.induct`, applied via `induct_thm_tac`

General recursion: syntactic details

- Use `recdef` in place of `primrec`
- Explicitly name function f to be defined
- Provide measure function by `"measure(%x. ...)"`
- Recursion only for argument before the first '$\Rightarrow$':
  - If necessary, rearrange arguments, or
  - collect several arguments into a tuple (type $\ldots * \ldots$)
- Special induction rule f`.induct`, applied via `induct_thm_tac`

Another example