Formal Methods and Specification (SS 2021)
Lecture 6:
Functions, Procedures

Stefan Ratschan

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

Evropský sociální fond Praha & EU: Investujeme do vaší budoucnosti

Function Specification

For using a function, it suffices to know its interface:

function sort(a, n):
Input: $a \in \mathcal{A}, n \in \mathcal{N}$
Output: $b \in \mathcal{A}$ s.t. $\mathrm{sorted}(b, n) \wedge \mathrm{perm}(a, b, n)$

function count(a, n, x)
Input: $a \in \mathcal{A}, n \in \mathcal{N}, x$ s.t. $\mathrm{sorted}(a, n)$
Output: $|\{i \mid i \in \{1, \dots, n\}, a[i] < x\}|$

We hide the implementation (information hiding),
it may change arbitrarily (as long as it fulfills the specification)

viz C/C++: header files

Usage

function sort(a, n):
Input: $a \in \mathcal{A}, n \in \mathcal{N}$
Output: $b \in \mathcal{A}$ s.t. sorted$(b, n) \wedge$ perm(a, b, n)

function count(a, n, x)
Input: $a \in \mathcal{A}, n \in \mathcal{N}, x$ s.t. sorted(a, n)
Output: $|\{i \mid i \in \{1, \ldots, n\}, a[i] < x\}|$

$s \leftarrow sort(a, n)$
$p \leftarrow count(s, n, v)$
@ $p = |\{i \mid i \in \{1, \ldots, n\}, a[i] < v\}|$

Verification condition:

$$[s = sort(a, n) \wedge p = count(s, n, v)] \Rightarrow p = |\{i \mid i \in \{1, \ldots, n\}, a[i] < v\}|$$

How to prove this? What do we know about those functions?

Input/output specifications correspond to logical formulas!

Representation of Specification using Output Variable

function sort(a, n):
Input: $a \in \mathcal{A}, n \in \mathcal{N}$
Output: $b \in \mathcal{A}$ s.t. $\text{sorted}(b, n) \wedge \text{perm}(a, b, n)$

$$\forall a \in \mathcal{A}, n \in \mathcal{N}, b \in \mathcal{A} \; . \; b = sort(a, n) \Rightarrow [\text{sorted}(b, n) \wedge \text{perm}(a, b, n)]$$

function count(a, n, x)
Input: $a \in \mathcal{A}, n \in \mathcal{N}, x$ s.t. $\text{sorted}(a, n)$
Output: $|\{i \mid i \in \{1, \ldots, n\}, a[i] < x\}|$

The output does not have a name!?

function count(a, n, x)
Input: $a \in \mathcal{A}, n \in \mathcal{N}, x$ s.t. $\text{sorted}(a, n)$
Output: r s.t. $r = |\{i \mid i \in \{1, \ldots, n\}, a[i] < x\}|$

$$\forall a \in \mathcal{A}, n \in \mathcal{N}, x, r \in \mathcal{N} \; . \; [\text{sorted}(a, n) \wedge r = count(a, n, x)] \Rightarrow r = |\{i \mid i \in \{1, \ldots, n\}, a[i] < x\}|$$

Representation of Specification using Function Name

function sort(a, n):
Input: $a \in \mathcal{A}, n \in \mathcal{N}$
Output: $b \in \mathcal{A}$ s.t. $\mathrm{sorted}(b, n) \land \mathrm{perm}(a, b, n)$

$$\forall a \in \mathcal{A}, n \in \mathcal{N} \,.\, \mathrm{sorted}(sort(a, n), n) \land \mathrm{perm}(a, sort(a, n), n)$$

function count(a, n, x)
Input: $a \in \mathcal{A}, n \in \mathcal{N}, x$ s.t. $\mathrm{sorted}(a, n)$
Output: $|\{i \mid i \in \{1, \dots, n\}, a[i] < x\}|$

$$\forall a \in \mathcal{A}, n \in \mathcal{N}, x \in \mathcal{N} \,.\, \mathrm{sorted}(a, n) \Rightarrow$$
$$count(a, n, x) = |\{i \mid i \in \{1, \dots, n\}, a[i] < x\}|$$

Proof of Verification Condition using Output Variables

$s \leftarrow sort(a, n)$
$p \leftarrow count(s, n, v)$
@ $p = |\{i \mid i \in \{1, \dots, n\}, a[i] < v\}|$

$$\forall a \in \mathcal{A}, n \in \mathcal{N}, b \in \mathcal{A} \,.\; b = sort(a, n) \Rightarrow [\text{sorted}(b, n) \wedge \text{perm}(a, b, n)]$$

We assume
$s = sort(a, n)$
$p = count(s, n, v)$
and prove
$p = |\{i \mid i \in \{1, \dots, n\}, a[i] < v\}|$.

$$\forall a \in \mathcal{A}, n \in \mathcal{N}, x, r \in \mathcal{N} \,.\; [\text{sorted}(a, n) \wedge r = count(a, n, x)] \Rightarrow r = |\{i \mid i \in \{1, \dots, n\}, a[i] < x\}$$

We start by proving properties of the result of the first program line s. From the assumption describing the function *sort*, we know sorted(s, n) and perm(a, s, n). From the assumption describing the function *count*, after the choices $a \leftarrow s$, $n \leftarrow n$, $x \leftarrow v$ we know:

$$[\text{sorted}(s, n) \wedge p = count(s, n, v)] \Rightarrow p = |\{i \mid i \in \{1, \dots, n\}, s[i] < v\}.$$

Since we know both assumptions on the left-hand side, we also know the right-hand side. Since perm(a, s, n), both a and s contain exactly the same elements, and hence

$$p = |\{i \mid i \in \{1, \dots, n\}, a[i] < v\}|,$$

which is what we wanted to prove.

Proof of Verification Condition using Function Names

We assume what we know about the functions

$$\forall a \in \mathcal{A}, n \in \mathcal{N} \ . \ \mathrm{sorted}(sort(a,n),n) \wedge \mathrm{perm}(a, sort(a,n),n)$$

$$\forall a \in \mathcal{A}, n \in \mathcal{N}, x \ . \ \mathrm{sorted}(a,n) \Rightarrow count(a,n,x) = |\{i \mid i \in \{1,\dots,n\}, a[i] < x\}|$$

To prove the verification condition

$$[s = sort(a,n) \wedge p = count(s,n,v)] \Rightarrow p = |\{i \mid i \in \{1,\dots,n\}, a[i] < v\}|$$

we assume the left-hand side (representing the program), and prove the right-hand side (representing the assertions). Due to substitutions resulting from the assumptions, it suffices to prove

$$count(sort(a,n),n,v) = |\{i \mid i \in \{1,\dots,n\}, a[i] < v\}|.$$

We start by proving properties of the result of the first program line $\mathrm{sort}(a,n)$. From the first assumption above, we know $\mathrm{sorted}(\mathrm{sort}(a,n),n)$ and $\mathrm{perm}(a,\mathrm{sort}(a,n),n)$. From the second assumption we know, after the choices $a \leftarrow \mathrm{sort}(a,n)$, $n \leftarrow n$, $x \leftarrow v$:

$$[\mathrm{sorted}(\mathrm{sort}(a,n),n) \wedge p = count(\mathrm{sort}(a,n),n,v)] \Rightarrow p = |\{i \mid i \in \{1,\dots,n\}, \mathrm{sort}(a,n)[i] < v$$

Since we know both assumptions on the left-hand side, we also know the right-hand side. Since $\mathrm{perm}(a, sort(a,n),n)$, both a and $sort(a,n)$ contain exactly the same elements, and hence

$$p = |\{i \mid i \in \{1,\dots,n\}, a[i] < v\}|$$

which is what we wanted to prove.

Implementation

Write input/output specifications as assertions:

function sort(a, n):
Input: $a \in \mathcal{A}, n \in \mathcal{N}$
Output: $b \in \mathcal{A}$ s.t. $\text{sorted}(b, n) \wedge \text{perm}(a, b, n)$
@ $a \in \mathcal{A}, n \in \mathcal{N}$
...
@ $\text{sorted}(b, n) \wedge \text{perm}(a, b, n)$
return b

function count(a, n, x):
Input: $a \in \mathcal{A}, n \in \mathcal{N}, x$ s.t. $\text{sorted}(a, n)$
Output: $|\{a[i] < x \mid i \in \{1, \dots, n\}\}|$
@ $a \in \mathcal{A}, n \in \mathcal{N}, x$ s.t. $\text{sorted}(a, n)$
...
@ $r = |\{i \mid i \in \{1, \dots, n\}, a[i] < x \,|\}|$
return r

Functions: Summary

Each function should have an input/output specification.

function $f(v_1, \dots, v_n)$
Input: $I(v_1, \dots, v_n)$
Output: v' s.t. $O(v_1, \dots, v_n, v')$

For proving verification conditions of code that uses the function:
add the logical formula corresponding to the I/O specification
as an assumption

$$\forall v_1, \dots, v_n, v' \,.\, [I(v_1, \dots, v_n) \wedge v' = f(v_1, \dots, v_n)] \Rightarrow O(v_1, \dots, v_n, v')$$

$$\forall v_1, \dots, v_n \,.\, I(v_1, \dots, v_n) \Rightarrow O(v_1, \dots, v_n, f(v_1, \dots, v_n))$$

Here, we can fully ignore internal code of function

Function implementation: I/O specification as first and last assertion

Attention: Here, function must be side-effect free:
no input, no global variables

Recursive Function Calls: Example

$$r = \gcd(x, y) :\Leftrightarrow r|x \wedge r|y \wedge \neg\exists r' \,.\, r'|x \wedge r'|y \wedge r' > r$$

function GCD(x,y)
Input: $x \in \mathcal{N}, y \in \mathcal{N}, x \geq y$
Output: $\gcd(x, y)$

$$\forall x \in \mathcal{N}, y \in \mathcal{N} \,.\, [x \geq y \wedge r = \mathtt{GCD}(x, y)] \Rightarrow r = \gcd(x, y)$$

Implementation:

function GCD(x,y)
@ $x, y \in \mathcal{N}, x \geq y$
if $y = 0$ **then**
    $r \leftarrow x$
else
    $r \leftarrow \mathtt{GCD}(y, x \bmod y)$
@ $r = \gcd(x, y)$
return r

Correctness proof?

Recursive Function Calls: Example

function $\texttt{GCD}(x,y)$
$@\ x, y \in \mathcal{N}, x \geq y$
if $y = 0$ **then**
    $r \leftarrow x$
else
    $r \leftarrow \texttt{GCD}(y, x \bmod y)$
$@\ r = \gcd(x, y)$
return r

Basic paths:

assume $x, y \in \mathcal{N}, x \geq y$
assume $y = 0$
$r \leftarrow x$
$@\ r = \gcd(x, y)$

assume $x, y \in \mathcal{N}, x \geq y$
assume $y \neq 0$
$r \leftarrow \texttt{GCD}(y, x \bmod y)$
$@\ r = \gcd(x, y)$

Verification conditions:

- $[x \geq y \wedge y = 0 \wedge r = x] \Rightarrow r = \gcd(x, y)$
- $[x \geq y \wedge y \neq 0 \wedge r = \texttt{GCD}(y, x \bmod y)] \Rightarrow r = \gcd(x, y)$

For proving the second condition, we use our assumption ...

Proof of Verification Condition

We assume that the recursive call returns a correct result:

$$\forall x, y, r \,.\, [x \geq y \wedge r = \mathtt{GCD}(x, y)] \Rightarrow r = \gcd(x, y)$$

For proving

$$[x \geq y \wedge y \neq 0 \wedge r = \mathtt{GCD}(y, x \bmod y)] \Rightarrow r = \gcd(x, y)$$

we assume $x \geq y$, $y \neq 0$, and $r = \mathtt{GCD}(y, x \bmod y)$ and and try to prove

$$r = \gcd(x, y).$$

From the formula above, after choosing $x \leftarrow y, y \leftarrow x \bmod y$ we conclude

$$[y \geq x \bmod y \wedge r = \mathtt{GCD}(y, x \bmod y)] \Rightarrow r = \gcd(y, x \bmod y)$$

The part $y \geq x \bmod y$ is a mathematical fact, we already know $r = \mathtt{GCD}(y, x \bmod y)$, and so we can conclude $r = \gcd(y, x \bmod y)$. Now it suffices to prove $\gcd(y, x \bmod y) = \gcd(x, y)$, which is a well-known mathematical fact.

Recursive Function Calls: Summary

For proving correctness of a recursive function
add the logical formula corresponding to the
I/O specification of the function itself
as an assumption

This is not a cyclic argument! See chapter “Functions” in lecture notes.

Again: the function must be side-effect free:
no input, no global variables

Specification of Procedures (Call by Reference): Example

procedure `reverse`(a, n)
Input: $a \in \mathcal{A}$, $n \in \mathcal{N}$
Output: $a^* \in \mathcal{A}$ s.t. $\forall i \in \{0, \ldots, n-1\} \,.\, a^*[i] = a[n-1-i]$

Modifies a, hence we need a different name for the input and output value.

Further possibilities, e.g. a^{in}/a^{out}, $a\!\downarrow / a\!\uparrow$

Corresponding logical formula?

$$\forall a, a^*, n \,.\, reverse(a, n) \Rightarrow \forall i \in \{0, \ldots, n-1\} \,.\, a^*[i] = a[n-1-i]$$

Separately list input and output argument:

$$\forall a, a^*, n \,.\, reverse(a, a^*, n) \Rightarrow \forall i \in \{0, \ldots, n-1\} \,.\, a^*[i] = a[n-1-i]$$

Here, *reverse* is a predicate!

Usage

Assumption:

$$\forall a, a^*, n \;.\; \mathit{reverse}(a, a^*, n) \Rightarrow \forall i \in \{0, \dots, n-1\} \;.\; a^*[i] = a[n-1-i]$$

Usage in a program:

assume $\forall i \in \{0, \dots, n-1\} \;.\; a[i] \geq 0$
`reverse`(a, n)
`@` $\forall i \in \{0, \dots, n-1\} \;.\; a[i] \geq 0$

SSA:

assume $\forall i \in \{0, \dots, n-1\} \;.\; a[i] \geq 0$
assume $\mathit{reverse}(a, a_1, n)$ // reverse is a predicate now
`@` $\forall i \in \{0, \dots, n-1\} \;.\; a_1[i] \geq 0$

Verification condition

$$[\forall i \in \{0, \dots, n-1\} \;.\; a[i] \geq 0 \land \mathit{reverse}(a, a_1, n)] \Rightarrow \forall i \in \{0, \dots, n-1\} \;.\; a_1[i] \geq 0$$

Example: Proof of Verification Condition

Assumptions:

- $\forall i \in \{0, \ldots, n-1\} \;.\; a[i] \geq 0$
- $\mathit{reverse}(a, a_1, n)$

To prove:

- $\forall i \in \{0, \ldots, n-1\} \;.\; a_1[i] \geq 0$

Assumption from specification:

$$\forall a, a^*, n \;.\; \mathit{reverse}(a, a^*, n) \Rightarrow \forall i \in \{0, \ldots, n-1\} \;.\; a^*[i] = a[n-1-i]$$

For concrete call (substitutions $a \leftarrow a_1, n \leftarrow n$):

$$\mathit{reverse}(a, a_1, n) \Rightarrow \forall i \in \{0, \ldots, n-1\} \;.\; a_1[i] = a[n-1-i]$$

Resulting new assumption: $\forall i \in \{0, \ldots, n-1\} \;.\; a_1[i] = a[n-1-i]$

Proof

Assumptions:

- $\forall i \in \{0, \ldots, n-1\} \,.\, a[i] \geq 0$
- $\forall i \in \{0, \ldots, n-1\} \,.\, a_1[i] = a[n-1-i]$

To prove $\forall i \in \{0, \ldots, n-1\} \,.\, a_1[i] \geq 0$

Let i be arbitrary, but fixed.

We prove $a_1[i] \geq 0$.

Due to the second assumption this means to prove $a[n-1-i] \geq 0$, which we know after substituting $i \leftarrow n-1-i$ in the first assumption.

Implementation

procedure `reverse`(a, n)
Input: $a \in \mathcal{A}$, $n \in \mathcal{N}$
Output: $a^* \in \mathcal{A}$ s.t. $\forall i \in \{0, \ldots, n-1\} \,.\, a^*[i] = a[n-1-i]$

procedure `reverse`(a, n)
`@` $a \in \mathcal{A}$, $n \in \mathcal{N}$
$a^{in} \leftarrow a$
$\ldots$
`@` $\forall i \in \{0, \ldots, n-1\} \,.\, a[i] = a^{in}[n-1-i]$
return

Procedures, More Parameters: Example

procedure `swap`(x, y)
Input:
Output: x^*, y^* s.t. $x^* = y$, $y^* = x$

$$\forall x, y, x^*, y^* \,.\, swap(x, x^*, y, y^*) \Rightarrow [x^* = y \wedge y^* = x]$$

Usage:

assume $x \geq 0 \wedge y \geq 1$
`swap`(x, y)
`@` $x \geq 1 \wedge y \geq 0$

SSA:

assume $x \geq 0 \wedge y \geq 1$
assume $swap(x, x_1, y, y_1)$
`@` $x_1 \geq 1 \wedge y_1 \geq 0$

Verification condition:

$$[x \geq 0 \wedge y \geq 1 \wedge swap(x, x_1, y, y_1)] \Rightarrow [x_1 \geq 1 \wedge y_1 \geq 0]$$

Proof

Assumption from function specification:

$$\forall x, y, x^*, y^* \;.\; swap(x, x^*, y, y^*) \Rightarrow [x^* = y \land y^* = x]$$

We want to prove:

$$[x \geq 0 \land y \geq 1 \land swap(x, x_1, y, y_1)] \Rightarrow [x_1 \geq 1 \land y_1 \geq 0]$$

Due to the assumption, it suffices to prove

$$[x \geq 0 \land y \geq 1 \land x_1 = y \land y_1 = x] \Rightarrow [x_1 \geq 1 \land y_1 \geq 0]$$

which holds obviously.

Procedures: Summary

procedure $\mathrm{p}(v_1, \ldots, v_n)$
Input: $I(v_1, \ldots, v_n)$
Output: $v_1^*, \ldots, v_n^*$ s.t. $O(v_1, v_1^*, \ldots, v_n, v_n^*)$

Assumption from specification:

$$\forall v_1, v_1^*, \ldots, v_n, v_n^* \,. \quad [I(v_1, \ldots, v_n) \wedge p(v_1, v_1^*, \ldots, v_n, v_n^*)] \Rightarrow O(v_1, v_1^*, \ldots, v_n, v_n^*)$$

A call $\mathrm{p}(v_1, \ldots, v_n)$ becomes **assume** $p(v_1, v_1^*, \ldots, v_n, v_n^*)$ in SSA, with p being a predicate symbol.

Further Examples

1. Recursive function calls (binary search)
2. Program development by assertion based stepwise refinement (sorting)

... in both cases array indices from 1 to n

Recursive Binary Search

function $\texttt{search}(a, l, u, x)$
Input: $a \in \mathcal{A}(\mathcal{N})$, $n \in \mathcal{N}$, $\texttt{sorted}(a, n)$, $l, u \in \{1, \ldots, n\}$, $x \in \mathcal{N}$
Output: r, s.t. $\begin{array}{l} r = 0 \Rightarrow [\neg \exists i \,.\, l \leq i \leq u \wedge a[i] = x] \wedge \\ r \neq 0 \Rightarrow [l \leq r \leq u \wedge a[r] = x] \end{array}$

As a formula:

$$\forall a, l, u, x, r \,.\, r = \texttt{search}(a, l, u, x) \Rightarrow \left[\begin{array}{c} r = 0 \Rightarrow [\neg \exists i \,.\, l \leq i \leq u \wedge a[i] = x] \\ \wedge \\ r \neq 0 \Rightarrow [l \leq r \leq u \wedge a[r] = x] \end{array} \right]$$

Implementation

function `search`(a, l, u, x)
assume $a \in \mathcal{A}(\mathcal{N})$, $n \in \mathcal{N}$, `sorted`(a, n), $l, u \in \{1, \ldots, n\}$, $x \in \mathcal{N}$
if $u < l$ **then**
    $r \leftarrow 0$
else if $a[(l + u)/2] = x$ **then**
    $r \leftarrow (l + u)/2$
else if $a[(l + u)/2] < x$ **then**
    $r \leftarrow \texttt{search}(a, (l + u)/2 + 1, u, x)$
else
    $r \leftarrow \texttt{search}(a, l, (l + u)/2 - 1, x)$
@ $r = 0 \Rightarrow [\neg \exists i \,.\, l \leq i \leq u \wedge a[i] = x] \wedge$
  $r \neq 0 \Rightarrow [l \leq r \leq u \wedge a[r] = x]$
return r

In the correctness proof we ignore the input condition, it is preserved obviously.

Proof of verification conditions corresponding to non-recursive branches: easy

Assumption from Function Specification

$$\forall a, l, u, x, r \,.\, r = \mathtt{search}(a, l, u, x) \Rightarrow \left[\begin{array}{l} r = 0 \Rightarrow [\neg \exists i \,.\, l \leq i \leq u \wedge a[i] = x] \wedge \\ r \neq 0 \Rightarrow [l \leq r \leq u \wedge a[r] = x] \end{array}\right]$$

For concrete call $\mathtt{search}(a, (l+u)/2+1, u, x)$, after the substitution $[a \leftarrow a, l \leftarrow (l+u)/2+1, u \leftarrow u, x \leftarrow x]$:

$$r = \mathtt{search}(a, (l+u)/2+1, u, x) \Rightarrow \left[\begin{array}{l} r = 0 \Rightarrow [\neg \exists i \,.\, (l+u)/2+1 \leq i \leq u \wedge a[i] = x] \wedge \\ r \neq 0 \Rightarrow [(l+u)/2+1 \leq r \leq u \wedge a[r] = x] \end{array}\right]$$

We can write this into the source code as documentation.

Recursive Binary Search

function $\texttt{search}(a, l, u, x)$
assume $a \in \mathcal{A}(\mathcal{N})$, $n \in \mathcal{N}$, $\texttt{sorted}(a, n)$, $l, u \in \{1, \dots, n\}$, $x \in \mathcal{N}$
if $u < l$ **then**
    $r \leftarrow 0$
else if $a[(l+u)/2] = x$ **then**
    $r \leftarrow (l+u)/2$
else if $a[(l+u)/2] < x$ **then**
    $r \leftarrow \texttt{search}(a, (l+u)/2+1, u, x)$
    $// \ [r = 0 \Rightarrow [\neg\exists i \,.\, (l+u)/2+1 \leq x \leq u \wedge a[i] = x]] \wedge$
        $[r \neq 0 \Rightarrow [(l+u)/2+1 \leq r \leq u \wedge a[r] = x]]$
else
    $r \leftarrow \texttt{search}(a, l, (l+u)/2-1, x)$
@ $r = 0 \Rightarrow [\neg\exists i \,.\, l \leq i \leq u \wedge a[i] = x] \wedge$
  $r \neq 0 \Rightarrow [l \leq r \leq u \wedge a[r] = x]$
return r

Verification condition of the $a[(l+u)/2] < x$ branch (formulation without substitution):

$$[\texttt{sorted}(a,n) \wedge a[(l+u)/2] < x \wedge r = \texttt{search}(a, (l+u)/2+1, u, x)] \Rightarrow \left[\begin{array}{l} r = 0 \Rightarrow [\neg\exists i \,.\, l \leq i \leq u \wedge a[i] = x] \wedge \\ r \neq 0 \Rightarrow [l \leq r \leq u \wedge a[r] = x] \end{array}\right]$$

Correctness Proof

From the function call we know

$$r = \mathtt{search}(a, (l{+}u)/2{+}1, u, x) \Rightarrow \left[\begin{array}{l} r = 0 \Rightarrow [\neg\exists i \,.\, (l+u)/2+1 \le i \le u \wedge a[i] = x] \wedge \\ r \ne 0 \Rightarrow [(l+u)/2+1 \le r \le u \wedge a[r] = x] \end{array}\right]$$

Verification condition of the $a[(l+u)/2] < x$ branch (formulation without substitution):

$$[\mathtt{sorted}(a, n) \wedge a[(l+u)/2] < x \wedge r = \mathtt{search}(a, (l+u)/2+1, u, x)] \Rightarrow$$
$$\left[\begin{array}{l} r = 0 \Rightarrow [\neg\exists i \,.\, l \le i \le u \wedge a[i] = x] \wedge \\ r \ne 0 \Rightarrow [l \le r \le u \wedge a[r] = x] \end{array}\right]$$

We assume: $\mathtt{sorted}(a, n)$, $a[(l+u)/2] < x$, $r = \mathtt{search}(a, (l+u)/2+1, u, x)$ which implies $\neg\exists i \,.\, 1 \le i \le (l+u)/2 \wedge a[i] = x$.

In the case $r = 0$ we know. that $\neg\exists i \,.\, (l+u)/2+1 \le i \le u \wedge a[i] = x$,
which proves $\neg\exists i \,.\, l \le i \le u \wedge a[i] = x$.

In the case $r \ne 0$ we known, that $(l+u)/2+1 \le r \le u \wedge a[r] = x$,
which proves $l \le r \le u \wedge a[r] = x$.

The proof for the second call is similar.

Sorting Algorithm: Basic Definitions

$\texttt{sorted}(a, n) :\Leftrightarrow \forall i \in \{1, \ldots, n-1\} \,.\, a[i] \leq a[i+1]$

$\texttt{perm}(a, b, n) :\Leftrightarrow \begin{array}{l} \exists c \,.\, \texttt{perm1n}(c, n) \wedge \\ \forall i \in \{1, \ldots, n\} \,.\, b[c[i]] = a[i] \end{array}$

$\texttt{perm1n}(a, n) :\Leftrightarrow \begin{array}{l} \forall i \in \{1, \ldots, n\}\, \exists j \in \{1, \ldots, n\} \,.\, a[j] = i \wedge \\ \forall i, j \in \{1, \ldots n\} \,.\, i \neq j \Rightarrow a[i] \neq a[j] \end{array}$

procedure $\texttt{swap}(a, n, i, j)$
Input: array a, $i, j, n \in \mathcal{N}$, $1 \leq i \leq n$, $1 \leq j \leq n$
Output: a^* s.t. $\begin{cases} a^*[i] = a[j], \\ a^*[j] = a[i], \\ \forall k \,.\, [1 \leq k \leq n, k \neq i, k \neq j] \Rightarrow a^*[k] = a[k] \end{cases}$

Sorting Algorithm: Stepwise Refinement

$minfirst(a, n, i) \;:\Leftrightarrow\; a[i] = \min\{a[k] \mid k \in \{i, \ldots n\}\}$

Example: $minfirst(\boxed{10 \mid 3 \mid 7 \mid 4 \mid 7}, 5, 2)$

procedure $\texttt{sort}(a, n)$
$a_{in} \leftarrow a$
for $i \leftarrow 1$ **to** n **do**
  @ $perm(a_{in}, a) \wedge \forall k \in \{1, \ldots, i-1\} \,.\, minfirst(a, n, k)$
  for $j \leftarrow n$ **down to** $i + 1$ **do**
    @ $perm(a_{in}, a) \wedge [\forall k \in \{1, \ldots, i-1\} \,.\, minfirst(a, n, k)] \wedge minfirst(a, n, j)$
    if $a[j-1] > a[j]$ **then**
      $\texttt{swap}(a, j-1, j)$
    @ $perm(a_{in}, a) \wedge [\forall k \in \{1, \ldots, i-1\} \,.\, minfirst(a, n, k)] \wedge minfirst(a, n, j-1)$
  @ $perm(a_{in}, a) \wedge \forall k \in \{1, \ldots, i\} \,.\, minfirst(a, n, k)$
@ $perm(a_{in}, a) \wedge sorted(a, n)$
return

assume $minfirst(a, n, j)$
assume $a[j-1] > a[j]$
$\texttt{swap}(a, j-1, j)$
@ $minfirst(a, n, j-1)$

Conclusion

For proving correctness of function and procedure calls

- we assume that the function/procedure returns
  for every input the correct result, and
- prove correctness of the call under this assumption.

The function/procedure specification allows us to
ignore its implementation!

Stepwise refinement:

1. What?
2. How?