



Interactive Theorem Proving using Isabelle/HOL

Session 8

René Thiemann

Department of Computer Science

Outline

- Sets and Lists in Isabelle
- Practical Example: Binary Search Trees

Sets and Lists in Isabelle

Sets in Isabelle

- type `'a set` for sets with elements of type `'a`

Set Basics

- $x \in A$ – membership
- $A \cap B$ – intersection
- $A \cup B$ – union
- $\neg A$ – complement
- $A - B$ – difference
- $A \subseteq B$ and $A \subset B$ – subset
- $\{\}$ – empty set
- UNIV – universal set (all elements of specific type)
- $\{x\}$ – singleton set
- $\text{insert } x \ A$ – insertion of single elements ($\text{insert } x \ A = \{x\} \cup A$)
- $f ` A$ – image of function with respect to set (“map f over elements of A ”)

Further Operations on Sets

- `set` – convert list to set
- `Collect p` – convert predicate `p :: 'a ⇒ bool` to set of type `'a set`
- `finite A` – is set finite?
- `card A :: nat` – cardinality of set (note: `card A = 0` whenever `A` is infinite)
- `sum f A` – $\sum_{x \in A} f(x)$ (note: `sum f A = 0` whenever `A` is infinite)
- `prod f A` – similar to `sum`, just product
- `Ball A p` / `Bex A p` – do all / any elements of `A` satisfy predicate `p`?
- `Max A` and `Min A` – maximum and minimum of finite, non-empty set `A`
- `{x .. y}` – all elements between `x` and `y`

Syntax for Set Comprehension

- `{x . p x}` – same as `Collect p`
- `{t | x y. p x y}` – same as `{z. ∃ x y. t = z ∧ p x y}`
- example: `{ (x + 5, y) | x y. x < 7 ∧ odd y }`

Remarks on Finiteness and Cardinality

- **properties like finiteness and cardinality do not work** well in combination with set-comprehension or Collect
- in these cases it is often required to manually rewrite or estimate such sets **by using images, products, intersections and unions**
- since card returns a natural number, card does not work well with infinite sets; consequence: **many lemmas on cardinalities have finiteness as assumption**
- therefore, cardinality proofs are often accompanied by finiteness proofs

Demo – Example Proof

```
lemma "card { (x * 3, y) :: nat × bool | x y. x < 10 ∧ P y } ≤ 20"
```

Remarks on Sums and Products

- $\text{sum } f \ S = 0$ and $\text{prod } f \ S = 1$ whenever S is infinite
- infinite sums are available as limits, and will not be covered in this course
- there are several congruence lemmas on sums and products available, e.g., where the function f can be changed by a pointwise comparison
- there is ample special syntax for sums and products

Demo – Example Proof

`lemma "sum (λ i. i) {.. $(n :: \text{nat})$ } $\leq n^2$ "`

question: is lemma true, if `nat` is replaced by `int`?

Lists in Isabelle

- type `'a list` for lists with elements of type `'a`

List Basics – Selection of Functions

- `[]` or `Nil` and `#` or `Cons` – Nil and Cons
- `set` – conversion of list to set
- `length`, `take`, `drop`, `map`, `filter`, `concat`, `foldl`, `foldr` – as in Haskell
- `@` or `append` – append
- `hd` and `tl` – head and tail of list
- `xs ! n` – `n`-th element of `xs`
- `xs [i := a]` – list update, similar to function update `f (x := a)`

List Basics – Predicates

- `x ∈ set xs` – membership test via set
- `set xs ⊆ set ys` – sublist test via set
- `distinct`, `sorted`, ...

Syntax for Lists

- `[1, 3, x, 11, a + b]` – explicit finite list
- `[n ..< m]` – range, restricted to `nat` list
- `[n .. m]` – range, restricted to `int` list
- **list comprehension** is available, internally converted to `concat` and `map`; example
 - `[(a, 2 * b) . a <- [0 ..< n], even a, b <- [2 .. 5]]`
 - `concat (map
 (λ a. if even a then map (λ b. (a, 2 * b)) [2..5] else [])
 [0..<n])`

Reasoning on Lists and Sets

- automation works quite well for lists and sets
- still there are some lemmas which often have to be applied manually
 - all kinds of congruence rules or rules that work pointwise
 - `sum.cong` – $\text{sum } f \ A = \text{sum } g \ B$ whenever $A = B$ and $f \ x = g \ x$ for all $x \in B$
 - `sum_mono` – $\text{sum } f \ A \leq \text{sum } g \ A$ whenever $f \ x \leq g \ x$ for all $x \in A$
 - `sum.neutral` – $\text{sum } f \ A = 0$ whenever $f \ x = 0$ for all $x \in A$
 - `nth_equalityI` – two lists are identical if they have the same length and are pointwise identical
 - `set_conv_nth` – definition of `set xs` via n -th elements
 - `split_list` – whenever $x \in \text{set } xs$ then $xs = p @ x \# s$ for suitable `p` and `s`
- use find-theorems to gather existing results, e.g.,
`find_theorems "sum _ (_ U _) = _ + _"`

Practical Example: Binary Search Trees

Binary Search Tree

- binary tree: straight-forward datatype definition; tree is a leaf or a node storing an element with left- and right-subtree
- search tree: the tree is **ordered**, i.e., for each node with element x , left-subtree ℓ and right-subtree r , all elements in ℓ are strictly smaller than x and x is strictly smaller than all elements in r
- selected operations: insert, delete, and membership test
- optimizations are not included, e.g. balancing in splay-trees, AVL-trees, ...

Demo and Exercise Session: Formalize Binary Search Trees