Isabelle/HOL Exercises
Advanced

Interval Lists

Sets of natural numbers can be implemented as lists of intervals, where an interval is simply a pair of numbers. For example the set *{2, 3, 5, 7, 8, 9}* can be represented by the list *[(2, 3), (5, 5), (7, 9)]*. A typical application is the list of free blocks of dynamically allocated memory.

Definitions

We introduce the type

```
type_synonym intervals = "(nat*nat) list"
```

This type contains all possible lists of pairs of natural numbers, even those that we may not recognize as meaningful representations of sets. Thus you should introduce an *invariant*

```
consts inv :: "intervals => bool"
```

that characterizes exactly those interval lists representing sets. (The reason why we call this an invariant will become clear below.) For efficiency reasons intervals should be sorted in ascending order, the lower bound of each interval should be less than or equal to the upper bound, and the intervals should be chosen as large as possible, i.e. no two adjacent intervals should overlap or even touch each other. It turns out to be convenient to define *ex.inv* in terms of a more general function

```
consts inv2 :: "nat => intervals => bool"
```

such that the additional argument is a lower bound for the intervals in the list.

To relate intervals back to sets define an *abstraction function*

```
consts set_of :: "intervals => nat set"
```

that yields the set corresponding to an interval list (where the list satisfies the invariant).

Finally, define two primitive recursive functions

```
consts add :: "(nat*nat) => intervals => intervals"
       rem :: "(nat*nat) => intervals => intervals"
```

for inserting and deleting an interval from an interval list. The result should again satisfy the invariant. Hence the name: `inv` is invariant under application of the operations `add` and `rem` – if `inv` holds for the input, it must also hold for the output.

Proving the invariant

```
declare Let_def [simp]
declare split_split [split]
```

Start off by proving the monotonicity of `inv2`:

```
lemma inv2_monotone: "inv2 m ins ⟹ n≤m ⟹ inv2 n ins"
```

Now show that `add` and `rem` preserve the invariant:

```
theorem inv_add: "⟦ i≤j; inv ins ⟧ ⟹ inv (add (i,j) ins)"
theorem inv_rem: "⟦ i≤j; inv ins ⟧ ⟹ inv (rem (i,j) ins)"
```

Hint: you should first prove a more general statement (involving `inv2`). This will probably involve some advanced forms of induction discussed in Section 9.3.1 of the “Tutorial on Isabelle/HOL”.

Proving correctness of the implementation

Show the correctness of `add` and `rem` wrt. their counterparts ∪ and `-` on sets:

```
theorem set_of_add:
  "⟦ i≤j; inv ins ⟧ ⟹ set_of (add (i,j) ins) = set_of [(i,j)] ∪ set_of ins"
theorem set_of_rem:
  "⟦ i ≤ j; inv ins ⟧ ⟹ set_of (rem (i,j) ins) = set_of ins - set_of [(i,j)]"
```

Hints: in addition to the hints above, you may also find it useful to prove some relationship between `inv2` and `set_of` as a lemma.

General hints

- You should be familiar both with simplification and predicate calculus reasoning. Automatic tactics like `blast` and `force` can simplify the proof.
- Equality of two sets can often be proved via the rule `set_eqI`:

  ```
  (⋀x. (x ∈ A) = (x ∈ B)) ⟹ A = B
  ```

  .

- Potentially useful theorems for the simplification of sets include
  `Un_Diff: A ∪ B - C = A - C ∪ (B - C)` and
  `Diff_triv: A ∩ B = {} ⟹ A - B = A`.

- Theorems can be instantiated and simplified via `of` and `[simplified]` (see the Isabelle/HOL tutorial).