

Functional Data Structures

Exercise Sheet 14

```
theory ex14
imports
  "~/src/HOL/Library/Multiset"
  "~/src/HOL/Library/Code_Target_Nat"
  "~/src/HOL/Library/RBT"
  "~/src/HOL/Library/Char_ord"
  "~/src/HOL/Library/Code_Char"
begin
```

Exercise 14.1 Word Frequency — Down to ML code

Your task is to develop a program that reads a corpus and prints the words in the corpus together with their frequencies, sorted by descending frequencies.

Except input and output, your program shall be formalized in Isabelle/HOL. A corpus is described as $'a$ list, and the result is described by $('a \times nat)$ list

The frequency of a word in a corpus can be specified as:

```
definition freq :: "'a list  $\Rightarrow$  'a  $\Rightarrow$  nat"
  where "freq ws = count (mset ws)"
```

Specify a predicate that characterizes a correct result. Note: If words have the same frequency, any order is allowed.

```
definition is_freq_list :: "'a list  $\Rightarrow$  ('a  $\times$  nat) list  $\Rightarrow$  bool"
```

Tests:

```
lemma <is_freq_list ["a","b","b","c","c"] [("c",2),("b",2),("a",1)]>
lemma <is_freq_list ["a","b","b","c","c"] [("b",2),("c",2),("a",1)]>
lemma <¬is_freq_list ["a","b","b","c","c"] [("b",2),("c",3),("a",1)]>
lemma <¬is_freq_list ["a","b","b","c","c"] [("a",1),("c",2),("b",2)]>
lemma <¬is_freq_list ["a","b","b","c","c"] [("b",2),("c",2),("b",2),("a",1)]>
```

1 Refinement #1

Compute a function from words to their frequency by folding over the corpus, starting with $\lambda_. 0$.

definition *incr1* :: “ $'a \Rightarrow ('a \Rightarrow \text{nat}) \Rightarrow 'a \Rightarrow \text{nat}$ ”
definition “*freq1 ws* \equiv *fold incr1 ws* ($\lambda_ . 0$)”

lemma *freq1_correct[simp]*: “*freq1 ws* = *freq ws*”

2 Refinement #2

Use red black trees to implement the mapping from words to frequencies. Words that do not occur in the corpus must not be mapped!

Use the RBT implementation from *HOL/Library/RBT*! It provides, e.g., *RBT.empty*, *RBT.lookup*, *RBT.insert*.

definition *abs_fm* :: “ $('a::\text{linorder}, \text{nat}) \text{rbt} \Rightarrow 'a \Rightarrow \text{nat}$ ” **where**
“*abs_fm t k* \equiv *case RBT.lookup t k* of *None* $\Rightarrow 0$ | *Some v* $\Rightarrow v$ ”

definition *inv_fm* :: “ $('a::\text{linorder}, \text{nat}) \text{rbt} \Rightarrow \text{bool}$ ” **where**
“*inv_fm t* \equiv ($0 \notin \text{ran } (\text{RBT.lookup } t)$)”

lemma *empty2_correct[simp]*:
“*abs_fm RBT.empty* = ($\lambda_ . 0$)” “*inv_fm RBT.empty*”

definition *incr2* :: “ $'a::\text{linorder} \Rightarrow ('a, \text{nat}) \text{rbt} \Rightarrow ('a, \text{nat}) \text{rbt}$ ”

lemma *incr2_correct[simp]*:
“*inv_fm t* $\implies \text{abs_fm } (\text{incr2 } k \ t) = \text{incr1 } k \ (\text{abs_fm } t)$ ”
“*inv_fm t* $\implies \text{inv_fm } (\text{incr2 } k \ t)$ ”

Now we have refined the operations, we can refine the algorithm that uses the operations:

definition “*freq2 ws* \equiv *fold incr2 ws RBT.empty*”

lemma *freq2_correct[simp]*: “*abs_fm (freq2 ws)* = *freq1 ws*” “*inv_fm (freq2 ws)*”

2.1 Extracting Result from RBT

Extract the desired result — list of pairs of words and their frequencies, sorted by frequency — from the red black tree. Use *RBT.entries*.

definition *fsort* :: “ $'a::\text{linorder} \text{list} \Rightarrow ('a \times \text{nat}) \text{list}$ ”

Prove that your function is correct. Hint: You will need to prove some auxiliary lemmas on standard list functions. Use *find_theorems* to search for existing lemmas. Hint: A lemma of the form *RBT.lookup (freq2 ws) w* = *Some f* $\iff \dots$, derived from *freq2_correct freq1_correct* may be useful!

lemma *fsort_correct*: “*is_freq_list ws (fsort ws)*”

3 Code Generation

Now we can use Isabelle/HOL’s code generator to actually extract functional code from our Isabelle formalization.

First, we derive a specialized versions with strings:

```
definition fsort_string :: “String.literal list  $\Rightarrow$  (String.literal  $\times$  nat) list”  
where “fsort_string  $\equiv$  fsort”
```

Then we can use the code generator in different ways.

By the value command

```
value [code] “fsort_string [STR “foo”, STR “bar”, STR “foo”, STR “bara’]”
```

Export code to file

```
export_code fsort_string in SML module_name Fsort file “export.sml”
```

We can load the file into JEdit’s ML IDE:

```
SML_file “export.sml”
```

And use it from within some wrapper code for parsing a corpus and printing the result:

```
SML_file “fsort.sml”
```

Use code directly with Isabelle’s builtin ML interpreter

```
ML_val {* see template file *}
```

The code generator also supports other target languages

```
export_code fsort_string in Haskell  
export_code fsort_string in Scala  
export_code fsort_string in OCaml
```