函数式编程原理

Lecture 5

split函数的特点

- If length(L)>1 and split(L) = (A, B), then A and B have *smaller* length than L.
- This follows from the spec, using some fairly obvious facts:

A@B is a perm of L, so
length(A)+length(B)=length(L)

length(A) & length(B) differ by 0 or 1

if n>1 and n odd, (n div 2)+1 < n
if n>1 and n even, n div 2 < n

用归纳法证明Merge函数的正确性

For all <-sorted lists A and B,
merge(A, B) = a <-sorted permutation of A@B.

- **Method**: *strong* induction on length(A)*length(B).
- **Base cases**: (A, [ ]) and ([ ], B).
  (i) Show: if A is <-sorted, merge(A,[ ]) = a <-sorted perm of A@[ ].
  (ii) Show: if B is <-sorted, merge([ ],B) = a <-sorted perm of [ ]@B.
- **Inductive case**: (x::A, y::B).
  Induction Hypothesis: for all *smaller* (A′, B′), if A′ & B′ are <-sorted, merge(A′, B′) = a <-sorted perm of A′@B′.
  Show: if x::A and y::B are <-sorted,
  merge(x::A, y::B) = a <-sorted perm of (x::A)@(y::B).

Merge函数的特点

```
fun merge (A, [ ])  = A
  | merge ([ ], B) = B
  | merge (x::A, y::B) = case compare(x, y) of
                           LESS  => x :: merge(A, y::B)
                         | EQUAL  => x::y::merge(A, B)
                         | GREATER  => y :: merge(x::A, B)
```

Does clause order matter? **NO**

Patterns are { Exhaustive
Overlap of first two clauses is harmless

Each yields merge([ ], [ ]) = [ ]

Could use *nested* **if-then-else** instead of **case**.

But we need a 3-way branch, so **case** is *better style*.

开始使用帮助(helper)函数

- We *defined* split and merge
- We *proved* they meet their specs
- Now let's use them to implement the **mergesort** algorithm...

归并排序—— mergesort

```
msort : int list -> int list

(* REQUIRES  true                                  *)
(* ENSURES   msort(L) = a <-sorted perm of L      *)

fun msort [ ] = [ ]
  | msort [x] = [x]
  | msort L =  let
                  val (A, B) = split L
                  val A′ = msort A
                  val B′ = msort B
               in
                  merge (A′, B′)
               end
```

an alternative version

msort的正确性验证

For all L:int list,
msort(L) = a <-sorted permutation of L.

- **Method**: by strong induction on *length* of L
- **Base cases**: L = [ ], L = [x]
  (i) Show msort [ ] = a sorted perm of [ ]
  (ii) Show msort [x] = a sorted perm of [x]
- **Inductive case**: length(L)>1.
  Inductive hypothesis: for all *shorter* lists R,
  msort R = a sorted perm of R.
  Show msort L = a sorted perm of L.

插入排序程序性能分析

```
fun ins (x, [ ]) = [x]
  | ins (x, y::L) = case compare(x, y) of
                      GREATER => y::ins(x, L)
                    |    _    => x::y::L
```

- $W_{ins}(n)$: the work for ins(x, L) (length L =n)
- $W_{ins}(n)$ is O(n)

```
fun isort [ ] = [ ]
  |  isort (x::L) = ins (x, isort L)
```

- $W_{isort}(n)$: the work for isort(L) (length L = n)
- $W_{isort}(n)$ is $O(n^2)$

归并排序程序性能分析

```
fun split [ ]  = ([ ],  [ ])
    | split [x] = ([x], [ ])
    | split (x::y::L) =
          let val (A, B) =split L
in (x::A, y::B)
          end
```

```
merge (A, [ ]) = A
    | merge ([ ], B) = B
    | merge (x::A, y::B) = case compare(x, y) of
                              LESS => x :: merge(A, y::B)
                            | EQUAL => x::y::merge(A, B)
                            | GREATER => y :: merge(x::A, B)
```

- $W_{split}(n)$: work of split(L) (length(L)=n)
- $W_{split}(n)$ is O(n)

- $W_{merge}(n)$: work of merge(A,B) (length(A)+length(B)=n)
- $W_{merge}(n)$ is O(n)

归并排序程序性能分析

```
fun msort [ ] = [ ]
   | msort [x] = [x]
   | msort L =  let
                    val (A, B) = split L
               in
                    merge (msort A, msort B)
               end
```

- $W_{mosrt}(n)$: work of msort(L)
  (length(L)=n)
- $W_{msort}(n)$ is O(n log n)

- 有没有新的数据类型能并发执行?
  - Tree结构
  - 用树结构进行排序

msort(L)性能优于isort(L)，还能继续提升性能吗?

•从程序算法上考虑：……

•从数据结构上考虑：List结构为线性（顺序）结构（在[ ]基础上利用'::'进行线性扩展），因此很难提升性能（无法并发/并行）