Coq でプログラムの正しさを保証する

クイックソートとその性質の証明が RyanGIScott/QuickSort.v にあるので利用させていただく。

```
Require Extraction.
Require Import ExtrOcamlNatInt.
Extraction "quicksort.ml" quickSort.
```

を末尾に追加し、ターミナル上で `coqc QuickSort.v` を実行すると `quicksort.ml` が作られる。

※元のコードにOmegaライブラリが使われており、最新バージョンではコンパイルできない（少なくとも `v8.8.2` では可能）
coq/coq

Coq のリストについて

```
Inductive list (A : Type) : Type :=
    | nil : list A
    | cons : A -> list A -> list A.
```

略記が用意されている。

- 要素を1つ追加するときは `x :: xs`
- リストを連結するときは `l1 ++ l2`

たとえば `cons 1 (cons 2 (cons 3 nil))` は `1 :: 2 :: 3 :: nil` となる。

各種注釈

- `flip_ltb x y` : y < x なら true
- `flip_geb x y` : x <= y なら true
- `filter f v` : v の要素で f に渡して true になるものだけ残す
- `QSAcc l` : リスト l がピボットで分割でき、さらに分割された2つの列も各々分割でき…という状態を表しているっぽい。分割統治できるという性質?（QuickSort Accumulateの略と思われる）
- `induction_ltofl_Prop` : 任意の n について、「n以下の自然数について正しい」ならすべての自然数について正しい（そんな気はする）

- `quickSort'` : リスト l と、l が分割統治できることの証明を受け取って、クイックソートを行う。クイックソートのアルゴリズムを思い浮かべると、この関数が行うのは連結だけ。
- `filter_length` : filter すると長さが元のリスト以下になる。
- `qsAcc` : 任意のリストは分割統治できる。
- `quickSort` : 任意のリストが分割統治できることが分かったので quickSort' をラップした関数を作る。
- `unit_test` : 動作確認。auto をやるだけ。
- `quickSort'_PI'` : 整列の結果は一意。

- `quickSort'_cons`：元の列のクイックソートは、ピボットで分割した各列をクイックソートしてつなげるのと同じ。
- `quickSort_cons_QSAccCons`：ピボットを分割したうちのどっちにくっつけても同じ。
- `Sorted_partition`：l1,l2 が整列済みで x が l1 の上界、l2 の下界なら `[l1++x::l2]` も整列済み。
- `Sorted_filter`：整列済みの列の部分列も整列済み。
- `In_quickSort'`：整列前に含まれていた要素は整列後も含まれている。
- `Permutation_filter_ltb_geb`：ピボットでの分割は置換で表せる。

- `quickSort'_Sorted`：整列済みなら小さい順に並んでいる。
- `quickSort_correct`：クイックソート後のリストは整列済みかつ元のリストの並べ替えになっている。

OCaml に抽出

- 付け加えた行の2行目は、Coq の nat を Ocaml の int に変換してくれる。実用的になる。
- 抽出されたコードを眺めると、証明に関する部分は省かれている。（quickSort' の引数からQSAccが消えているなど。）
- `ocaml` で対話環境を起動し、 `#use "quicksort.ml";;` してから `quickSort [1;4;3;2];;` などと打つと、性質の保証されたクイックソートが使える（今回は実装がおなじみなので証明の情報量は少ない）。

参考文献

- RyanGlScott/QuickSort.v
- 関数型言語OCaml入門 (PDF), Ocaml爆速入門
- 依存型の話 (PDF)