量子計算

Jacques Garrigue, 2021 年 12 月 22 日

1　量子計算の基礎

古典的な計算では状態がビットの列として考えられる。各ステップはその状態の一部を更新することになる。mathcomp で書くとこのようになる。

```
From mathcomp Require Import all_ssreflect all_algebra complex.

Section classical.
Definition state n := {ffun 'I_n -> bool}.
Definition op n := state n -> state n.

Definition set n i b : op n :=
  fun s => [ffun j => if i == j then b else s j].
Definition flip n i : op n :=
  fun s => [ffun j => if i == j then negb (s j) else s j].
Definition cnot n i j : op n :=
  fun s => if s j then flip n i s else s.
End classical.
```

n ビットがあると、2^n 個の状態がある。

量子計算は量子力学の原理を利用することで、古典的な計算より大きい状態空間を一気に扱える。ただし制約が多い。

量子計算の状態は、各ビットの状態の組合せでできておらず、そういう組合せの上の C 上のベクトル空間 H として表現できる。$\dim H = 2^n$ ということになる。H の状態は直接観測できないが、測定の確率分布として観測される。各ステップでは空間全体を変えるが、ユニタリな変換しかできない。

```
Section quantum.
Local Open Scope ring_scope.
Local Open Scope complex_scope.
Variable R : rcfType.  (* 実数 *)
Let C := R[i].         (* 複素数 *)

(* 通常の分布 *)
Definition dist (U : finType) :=
  {f : {ffun U -> R} | [forall i, f i >= 0] && (\sum_i f i == 1)}.

Variable n : nat.
Definition qstate := {ffun state n -> C^o}.  (* C^o は C をベクトル空間として *)
Definition qbasis (i : state n) : qstate := [ffun j => (i == j)%:R].
Definition norm1 (s : qstate) := \sum_i (s i)^* * (s i) == 1.

(* 許されている変換は線形でユニタリなものだけ *)
Definition qop := {linear qstate -> qstate}.
Definition unitary (f : qop) := forall s, norm1 s -> norm1 (f s).
```

上記の qbasis が $H =$ qstate の正規直交規定になる。ある s : qstate が i : state n として観測される確率が $|s_i| = \bar{s_i} s_i =$ (s i)^* * (s i) になる。また qop が H の線形変換なので、unitary はノルムを保存するとい意味になる。

古典のときと同じように qflip と qcnot が定義できる。線形性は簡単だが、ユニタリ性はいきなり難しそうだ。H の次元が大きいからである。

```
Definition qflip i (s : qstate) : qstate :=
  [ffun v => s (flip n i v)].

Definition qcnot i j (s : qstate) : qstate :=
  [ffun v => s (cnot n i j v)].

Lemma linear_qflip i : linear (qflip i).
Proof. move=> x y z. apply/ffunP => v. by rewrite !ffunE. Qed.

Lemma linear_qcnot i j : linear (qcnot i j).
Proof. move=> x y z. apply/ffunP => v. by rewrite !ffunE. Qed.

Lemma unitary_qflip i : unitary (Linear (linear_qflip i)).
Proof.
move=> s. rewrite /norm1 => Hs /=.
  Hs : \sum_v (s v)^* * s v == 1
  ============================
  \sum_v (qflip i s v)^* * qflip i s v == 1
Abort.
End quantum.
```

2　レンズの概念

レンズは元々データベースなどに使われる概念である。更新したいデータを全体から抽出し、更新結果を挿入する。

$$\begin{array}{lcl} \texttt{extract} & : & A \to B \\ \texttt{inject} & : & A \times B \to A \\ \texttt{inject}(x, \texttt{extract}(x)) & = & x \end{array}$$

これを古典的な状態に応用する。{ffun 'I_n -> T} の代わりに n 組 n.-tuple T を使う。

```
Section lens.
Context [n m : nat]. (* Section を出た後、n と m が省略される *)
Record lens := mkLens {lens_t :> m.-tuple 'I_n ; lens_uniq : uniq lens_t}.
Canonical lens_subType := Eval hnf in [subType for lens_t].
Canonical lens_predType := PredType (pred_of_seq : lens -> pred 'I_n).

Lemma lens_inj (l : lens) : injective (tnth l). Admitted.

Context (l : lens) [T : Type].

Definition extract (t : n.-tuple T) := [tuple of map (tnth t) l].

(* i が l に含まれていないと index i l = size l *)
(* j >= size l のとき、nth a l j = a *)
Definition inject (t : n.-tuple T) (t' : m.-tuple T) :=
  [tuple nth (tnth t i) t' (index i l) | i < n].

Definition endo1 := m.-tuple T -> m.-tuple T.
```

```
Definition focus1 (f : endo1) t := inject t (f (extract t)).
focus1
     : n.-tuple T -> n.-tuple T
End lens.
Arguments lens n m : clear implicits.

Definition lens_comp n m p (l1 : lens n m) (l2 : lens m p) : lens n p.
exists (extract l2 l1).
abstract (rewrite map_inj_uniq ?lens_uniq //; apply lens_inj).
Defined.
```

これによって、各操作で対象ビットを取る必要がなくなる。

```
Section lens_ops.
Definition bits n := n.-tuple bool.
Definition flip1 (v : bits 1) := [tuple negb (tnth v ord0)].
Definition cnot1 (v : bits 2) :=
  let c := tnth v (lift ord0 ord0) in
  if c then [tuple negb (tnth v ord0); c] else v.

Definition state_bits {n} (v : state n) := mktuple v.
Definition bits_state {n} (v : bits n) := finfun (tnth v).

Lemma flip1E n i :
  focus1 (mkLens [tuple i] isT) flip1 =1
  fun v => state_bits (flip n i (bits_state v)).
Proof.
move=> v. apply eq_from_tnth => j.
rewrite !tnth_mktuple !ffunE /=.
case: ifP => ij //=.
by rewrite tnth_map /= (tnth_nth i _ ord0) /= (eqP ij).
Qed.

Lemma cnot1E n (i j : 'I_n) (H : uniq [:: i; j]) :
  focus1 (mkLens _ H) cnot1 =1
  fun v => state_bits (cnot n i j (bits_state v)).
Admitted.
End lens_ops.
```

練習問題 2.1 cnot1E を証明しなさい。

3 量子状態のレンズ

同じレンズを量子状態に使える。ただし、inject ではなく、merge_indices を使わなければならない。

```
Section merge_lens.
Context [n m : nat] (l : lens n m).

Definition others := [seq i <- enum 'I_n | i \notin l].
Lemma size_others : size others == n - m.
Proof. (* 黒魔法 *) Admitted.

Definition lothers : lens n (n-m).
exists (Tuple size_others).
```

```
abstract (by rewrite filter_uniq // enum_uniq).
Defined.

Definition merge_indices (v : bits m) (w : bits (n-m)) :=
  [tuple nth (nth dI w (index i lothers)) v (index i l) | i < n].

Lemma extract_merge v1 v2 : extract l (merge_indices v1 v2) = v1.
Admitted.
Lemma extract_lothers_merge v1 v2 : extract lothers (merge_indices v1 v2) = v2.
Admitted.
Lemma merge_indices_extract v :
  merge_indices (extract l v) (extract lothers v) = v.
Admitted.
End merge_lens.
```

4 量子作用素をの応用

レンズを量子状態に応用するとき、状態の残りを分けて取っておくことができない。だから、空間の見方を変えて、変換したい部分を前にして、残りを後に隠す。endo はその残りの空間 T に対して線形的に働くようになっている。さらに naturality がその働きが一様であることを保証する。f : endo n で naturality f がなりたつとき、f がある 2^n 次元行列で表現できる。

```
Section tensor_space.
Variable R : comRingType.
Local Open Scope ring_scope.
Import GRing.Theory.

Definition tpower n T := {ffun bits n -> T}.
Definition endofun n := forall T : lmodType R, tpower n T -> tpower n T.
Definition endo n :=
  forall T : lmodType R, {linear tpower n T -> tpower n T}.
Definition tsquare n := tpower n (tpower n R^o).

Definition eq_endo n (f1 f2 : endo n) := forall T : lmodType R, f1 T =1 f2 T.
Notation "f1 =e f2" := (eq_endo f1 f2) (at level 70).

Definition comp_endo n (tr  tr' : endo n) : endo n :=
  fun A => GRing.comp_linear (tr A) (tr' A).
Notation "f \v g" := (comp_endo f g) (at level 50).

Definition tsendo_fun n (M : tsquare n) : endofun n :=
  fun T v => [ffun vi => \sum_vj (M vi vj : R) *: v vj].

Lemma tsendo_is_linear n M T : linear (@tsendo_fun n M T).
Proof.
move=> /= x y z; apply/ffunP => /= vi; rewrite !ffunE.
rewrite scaler_sumr -big_split; apply eq_bigr => /= vj _.
by rewrite !ffunE scalerDr !scalerA mulrC.
Qed.

Definition tsendo n M : endo n := fun T => Linear (@tsendo_is_linear n M T).

(* 自然変換 *)
```

```
Definition map_tpower m T1 T2 (f : T1 -> T2) (nv : tpower m T1)
  : tpower m T2 := [ffun v : bits m => f (nv v)].
Definition naturality n (f : endo n) :=
  forall (T1 T2 : lmodType R) (h : linear T1 -> T2) (v : tpower n T1),
    map_tpower h (f T1 v) = f T2 (map_tpower h v).

Lemma naturalityP n (f : endo n) :
  naturality f <-> exists M, f =e tsendo M.
Admitted.

Section curry.
Context [T : lmodType R] [n m : nat] (l : lens n m).

Definition curry (st : tpower n T) : tpower m (tpower (n-m) T) :=
  [ffun v : bits m =>
   [ffun w : bits (n-m) => st (merge_indices l v w)]].

Definition uncurry (st : tpower m (tpower (n-m) T)) : tpower n T :=
  [ffun v : bits n => st (extract l v) (extract (lothers l) v)].

Lemma curry_is_linear : linear curry. Admitted.
Definition curry_lin := Linear curry_is_linear.
End curry.

Section focus.
Context [n m : nat] (l : lens n m) (tr : endo m).
Definition focus_fun : endofun n :=
  fun T (v : tpower n T) => uncurry l (tr _ (curry_lin l v)).

Lemma focus_is_linear T : linear (@focus_fun T).
Admitted.

Definition focus : endo n := fun T => Linear (@focus_is_linear T).
End focus.

(* 縦の合成 *)
Lemma focus_comp n m (tr tr' : endo m) (l : lens n m) :
  focus l (tr \v tr') =e focus l tr \v focus l tr'.
Admitted.

(* 横の合成 *)
Lemma focusC n m p (l : lens n m) (l' : lens n p) tr tr' :
  [disjoint l & l'] -> naturality tr -> naturality tr' ->
  focus l tr \v focus l' tr' =e focus l' tr' \v focus l tr.
Admitted.

(* 結合律 *)
Lemma focusM n m p (l : lens n m) (l' : lens m p) tr : naturality tr ->
  focus (lens_comp l l') tr =e focus l (focus l' tr).
End tensor_space.
```

5　qecc ライブラリ

今回の資料は qecc というライブラリから重要なところを抜き出している。GitHub から qecc をダウンロードすると実際の証明ができるようになる。

練習問題 5.1 *https://github.com/t6s/qecc* の *Code/Download zip* から *qecc* をダウンロードして、`qexamples.v` を *CoqIde* で開き、*Compile / Make* でコンパイルした上で、`flip1` を元に `qflip` と同様に定義した `qnot1` が `tsmor qnot` と同値であることを証明しなさい。