Journal of Quantum Information Science, 2015, 5, 58-70
Published Online June 2015 in SciRes. http://www.scirp.org/journal/jqis
http://dx.doi.org/10.4236/jqis.2015.52008

Formal Verification of Robertson-Type Uncertainty Relation

Takaaki Masuhara, Toru Kuriyama, Masakazu Yoshida, Jun Cheng

Department of Intelligent Information Engineering and Sciences, Doshisha University, Kyoto, Japan
Email: masyoshi@mail.doshisha.ac.jp, jcheng@ieee.org

Received 30 April 2015; accepted 8 June 2015; published 11 June 2015

Abstract

Formal verification using interactive theorem provers have been noticed as a method of verification of proofs that are too big for humans to check the validity of them. The purpose of this work is to verify the validity of Robertson-type uncertainty relation toward verifying unconditional security of quantum key distributions. We verify the validity of the relation by using proof assistant Coq and it is turned out that the theorem regarding the relation formally holds. The source code for Coq which represents the validity of the theorem is printed in Appendix.

Keywords

Formal Verification, Proof Assistant Coq, Uncertainty Relation

1. Introduction

Formal verification is a technique for verifying the validity of proofs in mathematics, algorithms, computer systems, and so on. In the formal verification by using logical reasoning, interactive theorem provers (HOL [1], ACL2 [2], Isabelle [3], Coq [4], just to a name a few) are used for verifying mathematical proofs. In accordance with the Curry-Howard correspondence [5], the validity is verified through coding of the proofs in functional languages.

Formal verification by using the interactive theorem provers has been noticed as a technique for verifying proofs of theorems which are too large for humans to check the validity. Such a theorem as the Feit-Thompson theorem (also known as the odd order theorem) [6] can be enumerated as an example. This theorem was proven in 1963. However, the verification of the validity of the proof was highly difficult at 1980s [7] since its total number of pages are about 300. Gonthier *et al*. [8] verified the proof by using SSReflect which is an extension of the proof assistant Coq. The formalized theorem and lemmas which are formalized in the process of the verification are utilized for the formalization of mathematical science. Therefore, the formal verification of rela-

How to cite this paper: Masuhara, T., Kuriyama, T., Yoshida, M. and Cheng, J. (2015) Formal Verification of Robertson-Type Uncertainty Relation. *Journal of Quantum Information Science*, **5**, 58-70. http://dx.doi.org/10.4236/jqis.2015.52008

tively small lemmas which are used for large proofs is useful as a library for the verification of the other large proofs.

The formal verification is considered valid at information theory which is a branch of mathematical science. Affeldt *et al*. [9] formally verify basic definitions and theorems in information theory by using SSReflect, and also Shannon's channel coding theorem and source coding theorem [10] which are famous theorems of all results of information theory. These formalized theorems are valuable not only toward the formally verification of integrity between coding and decoding algorithms and the software implemented ones, but also further facilitating of the formal verification by utilizing the formalized types.

In quantum information theory, the axioms of quantum physics are described mathematically [11]. Therefore, the formal verification can be applied to verifying the validity of quantum information theory. In this work, we verify the validity of uncertainty relation toward verifying unconditional security of quantum key distributions and encouraging the formal verification of large proofs in quantum information theory. Specifically, we verify the validity of Robertson-type uncertainty relation [12] by using the proof assistant Coq.

This paper is organized as follows. In Section 2, we review the theorem regarding Robertson-type uncertainty relation and its mathematically proof. In Section 3, we formally verify the validity of the theorem with Coq. In Section 4, this paper is summarized. Coq source code is printed in **Appendix**.

2. Robertson-Type Uncertainty Relation

Robertson-type uncertainty relation imposes a restriction on probability distributions of measurement outcomes with observables. In this type, uncertainty of the measurement is characterized by standard deviation of the distribution.

In quantum information theory, a quantum system and a quantum pure state in the system are regarded in the same light as a Hilbert space and a unit vector in the space, respectively. In addition, an observable is regarded as an Hermitian operator on the Hilbert space. Let $V_\psi[A]$ be variance of outcomes which are obtained by measuring a quantum system in a quantum state ψ with an observable A. Then,

$$V_\psi[A] = \langle \psi | A^2\psi \rangle - \langle \psi | A\psi \rangle^2 \tag{1}$$

holds. Standard deviation of the outcomes $\Delta_\psi[A]$ is defined as $\sqrt{V_\psi[A]}$. The following theorem was given by Robertson [12].

Theorem 1 ([9] [12]). *For two observables A, B, and a quantum state* ψ,

$$\Delta_\psi[A]\Delta_\psi[B] \geq \frac{1}{2}\left|\langle \psi | [A,B]\psi \rangle\right| \tag{2}$$

holds, where $[A,B] := AB - BA$.

Proof. We observe $\Delta_\psi[A]^2 = \|A\psi\|^2$ and $\Delta_\psi[B]^2 = \|B\psi\|^2$ from the definition of standard deviation and Hermiticity of the observable. $\Delta_\psi[A]^2\,\Delta_\psi[B]^2 \geq |\langle \psi | AB\psi \rangle|^2$ holds from the Schwarz inequality $|\langle A\psi | B\psi \rangle| \leq \|A\psi\|\|B\psi\|$. We obtain $AB = \{A,B\}/2 + i[A,B]/2i$, where $\{A,B\} := AB + BA$. We observe

$$\begin{aligned}
|\langle \psi | AB\psi \rangle|^2 &= \left|\left\langle \psi \middle| \left(\frac{\{A,B\}}{2} + i\frac{[A,B]}{2i}\right)\psi \right\rangle\right|^2 = \left|\left\langle \psi \middle| \frac{\{A,B\}}{2}\psi \right\rangle + i\left\langle \psi \middle| \frac{[A,B]}{2i}\psi \right\rangle\right|^2 \\
&= \left|\left\langle \psi \middle| \frac{\{A,B\}}{2}\psi \right\rangle\right|^2 + \left|\left\langle \psi \middle| \frac{[A,B]}{2i}\psi \right\rangle\right|^2 = \left|\left\langle \psi \middle| \frac{\{A,B\}}{2}\psi \right\rangle\right|^2 + \left|\left\langle \psi \middle| \frac{[A,B]}{2}\psi \right\rangle\right|^2,
\end{aligned}$$

where we thank to Hermiticity of $\{A,B\}/2$ and $[A,B]/2i$.

$\left|\left\langle \psi \middle| \frac{\{A,B\}}{2}\psi \right\rangle\right|^2 + \left|\left\langle \psi \middle| \frac{[A,B]}{2}\psi \right\rangle\right|^2 \geq \frac{1}{4}|\langle \psi | [A,B]\psi \rangle|^2$ holds from $\left|\left\langle \psi \middle| \frac{[A,B]}{2}\psi \right\rangle\right|^2 = \frac{1}{4}|\langle \psi | [A,B]\psi \rangle|^2$. Then,

$\Delta_\psi[A]^2\,\Delta_\psi[B]^2 \geq |\langle \psi | AB\psi \rangle|^2 \geq \frac{1}{4}|\langle \psi | [A,B]\psi \rangle|^2$ holds. Therefore, Equation (2) can be observed immediately.

A relation between two observables represented by Equation (2) is called Robertson-type uncertainty relation.

The right-hand side of the inequality always takes 0 if the observables are commutative. Therefore, both of standard deviations of the observables may take 0. On the other hand, for non-commutative observables, the right-hand side of the inequality dose not take 0. Then, both of standard deviations of the observables dose not take 0. This implies that Equation (2) is a tradeoff between uncertainties of the observables. In this case, the uncertainty is characterized by standard deviation.

The relation between the non-commutative observables often plays crucial role in discussion of unconditional security of quantum key distributions. In BB84 [13] which is the most famous quantum key distribution protocol, eigenstates of non-commutative observables σ_x and σ_z are used for sharing secret key between a sender (called Alice) and a receiver (called Bob). Alice prepares random bits and sends quantum bits (qubits) to Bob, where each qubit is prepared in one of the eigenstates of σ_x and σ_z with a procedure of the protocol. Bob measures each qubit with σ_x or σ_z randomly and obtains outcomes as a candidate key. For obtaining a sifted key, Alice and Bob check the choices of the observables in the state preparation and the quantum measurement. They calculate error rate of a part of the sifted key. The protocol is aboded if the rate is grater than preset value (this implies that they presume the existence of some kind of eavesdropping). Otherwise, they perform the leftover sifted key to make the secret key with error correction and privacy amplification. A purpose of a eavesdropper (called Eve) is to gain information of the secret key without being detected by Alice and Bob on the channel. Eve can gain information if she can distinguish the eigenstates of σ_x and σ_z. However, there is a tradeoff between information gain for Eve and the error rate. That is, she cannot gain information of the secret key generated by σ_x without increasing the error rate of the part of the sifted key generated by σ_z, and vice versa. The fact is known as the information disturbance theorem [14]-[16] and this theorem is applied to a proof of unconditional security of BB84 [14]. The information disturbance theorem can be regarded as an information theoretic version of uncertainty relation [15]. The theorem is obtained directly [16] from entropic uncertainty relation [17] [18] which is a kind of types of uncertainty relation. In this work, we verify the validity of Robertson-type uncertainty relation using the proof assistant Coq toward verifying entropic uncertainty relation, the information disturbance theorem, and unconditional security of quantum key distributions.

3. Formal Verification of Robertson-Type Uncertainty Relation

In this section, we verify the validity of Robertson-type uncertainty relation by using the proof assistant Coq. We define *types* as follows:

- C: a type of a complex number
- Vec C n: a type of an *n*-dimensional complex vector
- UnitVec C n: a type of an *n*-dimensional complex unit vector
- Mat C v: a type of a v = $n \times n$ complex matrix
- HMat v: a type of a v = $n \times n$ Hermitian matrix

We define *functions* as follows:

- var: takes a pair of a variable of UnitVec C n and a variable of HMat v and returns variant with respect to the variables (see Equation (1))
- cabs: takes a variable of C and returns absolute value of it
- mMinus: takes a pair of variables of Mat C v and returns addition of them
- mMinus: takes a pair of variables of Mat C v and returns subtraction of them
- mMult: takes a pair of variables of Mat C v and returns product of them
- innerProd: takes a pair of variables of Vec C n and returns inner product of them
- mvMult: takes a pair of a variable of Mat C v and a variable of Vec C n and returns product of them

Theorem 2. *We declare a formalized statement of Robertson-type uncertainty relation in Coq:*
Theorem RobertsonUR:

```
forall (n : nat) (v : mlenghts n) (A B: HMat v) (psi : UnitVec C n),
(sqrt (var psi A)) * (sqrt (var psi B)) >=
(1/2) * (cabs ( innerProd psi (mvMult (mMinus (mMult A B) (mMult B A)) psi))).
```

Before giving the formally proof described by Coq, we show the broad outlines of it. Firstly, we define the *types* and the *functions* in linear algebra. The types and functions are already shown above. The next step is to

obtain *subgoals* through applying small lemmas to some inequalities. The subgoals are formally verified through interactive proof sessions. Finally, we complete the verification of the the theorem regarding Robertson-type uncertainty relation when we have no new subgoals.

Proof. We execute the above statement and obtain a response from Coq:

```
1 subgoals
_____________________________________(1/1)
forall (n : nat) (v : mlengths n) (A B : HMat v) (psi : UnitVec),
sqrt (var psi A) * sqrt (var psi B) >=
1/2* cabs (innerProd psi (mvMult (mMinus (mMult A B) (mMult B A)) psi))
```

For the above *subgoal* (statement), a command called *tactic* is inputted and executed, then, new subgoal which is rewritten according to the tactic is obtained as a response from Coq. We prove the statement through interactive proof sessions until there no exist new subgoals (for complete Coq source code, see **Appendix**).
We input the following tactic and execute it:

```
intros.
```

where intros is a tactic which adds a condition described by universal quantifier to preconditions. Then, we get the following response from Coq:

```
1 subgoals
n : nat
v : nlengths n
A : HMat v
B : HMat v
psi : UnitVec C n
_____________________________________(1/1)
sqrt (var psi A) * sqrt (var psi B) >=
1/2* cabs (innerProd psi (mvMult (mMinus (mMult A B) (mMult B A)) psi))
```

For readability, $\sqrt{V_\psi[A]}$, $\sqrt{V_\psi[B]}$, and $\left|\langle\psi|(AB-BA)\psi\rangle\right|$ are replaced with *a*, *b*, and *c*, respectively, by using a tactic set:

```
set ( a := sqrt (var psi A) ).
set ( b := sqrt (var psi B) ).
set ( c := (cabs (innerProd psi (mvMult (mMinus (mMult A B) (mMult B A)) psi)))).
```

Then, we get the following response from Coq:

```
1 subgoals
n: nat
v: mlengths n
A: HMat v
B: HMat v
psi: UnitVec C n
a:= sqrt (var psi A) : R
b:= sqrt (var psi B) : R
c:= cabs (innerProd psi (mvMult (mMinus (mMult A B) (mMult B A)) psi)):R
_____________________________________(1/1)
a * b >= 1/2 * c
```

Hereafter, we show only subgoals which are responses from Coq. $a^2b^2 \geq (1/4)c^2$ is observed from

$ab \geq (1/2)c$ by using a tactic replace:

replace (a * b >= 1 / 2 * c) with ((a * a) * (b * b) >= (1/4) * (c * c)).

Then, we get the following response from Coq:

```
_____________________________________(1/2)
a * a * (b * b) >= 1 / 4 * (c * c)
_____________________________________(2/2)
(a * a * (b * b) >= 1 / 4 * (c * c)) = (a * b >= 1 / 2 * c)
```

For proving the inequality $a^2b^2 \geq (1/4)c^2$, we prove $a^2b^2 \geq |\langle\psi|AB\psi\rangle|^2$ and $|\langle\psi|AB\psi\rangle|^2 \geq (1/4)c^2$ from a transitive relation $a^2b^2 \geq |\langle\psi|AB\psi\rangle|^2 \wedge |\langle\psi|AB\psi\rangle|^2 \geq (1/4)c^2 \Rightarrow a^2b^2 \geq (1/4)c^2$. The transitive relation is added to the preconditions by using a tactic assert:

```
assert ( a * a * (b * b) >=
(cabs ( innerProd psi (mvMult (mMult A B ) psi))) *
(cabs ( innerProd psi (mvMult (mMult A B ) psi))) /\
(cabs ( innerProd psi (mvMult (mMult A B ) psi))) *
(cabs ( innerProd psi (mvMult (mMult A B ) psi)))
>= (1 / 4) * (c * c) -> a * a * (b * b) >= 1 / 4 * (c * c)).
```

For proving the transitive relation $a^2b^2 \geq |\langle\psi|AB\psi\rangle|^2 \wedge |\langle\psi|AB\psi\rangle|^2 \geq (1/4)c^2 \Rightarrow a^2b^2 \geq (1/4)c^2$, we use lemma R_leq_eq2 $\left(\forall a,b,c,d \in \mathbb{R}, a^2b^2 \geq c^2 \wedge c^2 \geq (1/4)d^2 \Rightarrow a^2b^2 \geq (1/4)d^2\right)$. The lemma is applied to the subgoal by using a tactic apply:

apply R_leq_eq2.

Then, we get the following response from Coq:

```
H : a * a * (b * b) >=
cabs (innerProd psi (mvMult (mMult A B) psi)) *
cabs (innerProd psi (mvMult (mMult A B) psi)) /\
cabs (innerProd psi (mvMult (mMult A B) psi)) *
cabs (innerProd psi (mvMult (mMult A B) psi)) >= 1/4 * (c*c)
-> a * a * (b * b) >= 1/4 * (c*c)
_____________________________________(1/2)
a * a * (b * b) >= 1 / 4 * (c * c)
_____________________________________(2/2)
(a * a * (b * b) >= 1 / 4 * (c * c)) = (a * b >= 1 / 2 * c)
```

Accordingly, the inequality which was performed with assert is added to the precondition as an assumption H. The assumption H is applied to the subgoal for proving $a^2b^2 \geq (1/4)c^2$:

apply H.

We input a tactic split to split $a^2b^2 \geq |\langle\psi|AB\psi\rangle|^2 \wedge |\langle\psi|AB\psi\rangle|^2 \geq (1/4)c^2$ into $a^2b^2 \geq |\langle\psi|AB\psi\rangle|^2$ and $|\langle\psi|AB\psi\rangle|^2 \geq (1/4)c^2$:

split.

For proving $a^2b^2 \geq |\langle\psi|AB\psi\rangle|^2$, RUR_Var_geq_InnerProd is applied to the subgoal:

apply RUR_Var_geq_InnerProd.

$|\langle\psi|AB\psi\rangle|$ is replaced with *d* for readability:

set (d := cabs (innerProd psi (mvMult (mMult A B) psi))).

We get the following response from Coq:

```
_____________________________________(1/2)
d * d >= 1 / 4 * (c * c)
_____________________________________(2/2)
(a * a * (b * b) >= 1 / 4 * (c * c)) = (a * b >= 1 / 2 * c)
```

d^2 and $(1/4)c^2$ is replaced with $|\langle\psi|\{(AB+BA)/2\}\psi\rangle|^2 + |\langle\psi|\{(AB-BA)/2\}\psi\rangle|^2$ and $|\langle\psi|\{(AB-BA)/2\}\psi\rangle|^2$, respectively:

```
replace ( d * d ) with ((((1/2) * (cabs ((innerProd psi (mvMult (mPlus
(mMult A B) (mMult B A) ) psi))))) * ((1/2) * (cabs ( (innerProd psi (mvMult
(mPlus (mMult A B) (mMult B A) ) psi)))))) + (( (1/2) * (cabs ( (innerProd psi (mvMult
(mMinus (mMult A B) (mMult B A)) psi))))) * ((1/2) * (cabs ( (innerProd psi (mvMult
(mMinus (mMult A B) (mMult B A)) psi))))))).
replace ((1/4) * (c * c)) with (((1/2) * (cabs ((innerProd psi
(mvMult (mMinus (mMult A B) (mMult B A)) psi))))) * ((1/2) *
(cabs ((innerProd psi (mvMult (mMinus (mMult A B) (mMult B A)) psi)))))).
```

For proving $|\langle\psi|\{(AB+BA)/2\}\psi\rangle|^2 + |\langle\psi|\{(AB-BA)/2\}\psi\rangle|^2 \geq |\langle\psi|\{(AB-BA)/2\}\psi\rangle|^2$, we apply Square_Plus_geq $(\forall a,b \in \mathbb{R}, b^2 + a^2 \geq a^2)$ as follows:

apply Square_Plus_geq.

For proving $|\langle\psi|\{(AB+BA)/2\}\psi\rangle|^2 = (1/4)c^2$ and $|\langle\psi|\{(AB+BA)/2\}\psi\rangle|^2 + |\langle\psi|\{(AB-BA)/2\}\psi\rangle|^2 = d^2$, we apply RUR_eq_AB_0 and RUR_eq_AB_1 as follows:

apply RUR_eq_AB_0. apply RUR_eq_AB_1.

Then, we get the following response from Coq:

```
_____________________________________(1/1)
(a * a * (b * b) >= 1 / 4 * (c * c)) = (a * b >= 1 / 2 * c)
```

For proving $a^2b^2 \geq (1/4)c^2 \equiv ab \geq (1/2)c$, we apply R_leq_eq $\left(\forall x, y \in \mathbb{R}, z \in \mathbb{C}, \left(\sqrt{x}\right)^2 \left(\sqrt{y}\right)^2 \geq (1/4)|z|^2 \equiv \sqrt{x}\sqrt{y} \geq (1/2)|z|\right)$:

apply R_leq_eq.

Then, we get the following response from Coq:

No more subgoals.

All of the subgoals are proven. Input Qed and end the proof:

Qed.

The validity of Robertson-type uncertainty relation is verified formally.

4. Conclusion

In this work, we verified formally the validity of Robertson-type uncertainty relation by using the proof assistant Coq. We expect that the formalized theorem utilizes for facilitating the formal verification of the other theorems in quantum information theory. In future work, we will verify entropic uncertainty relation and the information disturbance theorem toward verifying unconditional security of quantum key distributions formally.

Acknowledgements

This work was partly supported by JSPS KAKENHI (24300030) Grant-in-Aid for Scientific Research (B), and MEXT-Supported Program for the Strategic Research Foundation at Private Universities (2014-2018, S1411030).

References

[1] Gordon, M.J.C. (1986) Why Higher-Order Logic Is a Good Formalism for Specifying and Verifying Hardware. In: Milne, G.J. and Subrahmanyam, P.A., Eds., *Proceedings of the* 1985 *Edinburgh Workshop on VLSI*, North-Holland, 153-177.

[2] Boyer, R.S., Moore, J.S. and Kaufmann, M. (1996) ACL2 Version 7.0. http://www.cs.utexas.edu/users/moore/acl2/

[3] University of Cambridge Computer Laboratory (1989) Isabelle. http://isabelle.in.tum.de/

[4] INRIA (1985) The Coq Proof Assistant. http://coq.inria.fr/

[5] Heine, M., Sørensen, B. and Urzyczyn, P. (2006) Lectures on the Curry-Howard Isomorphism. Warsaw University, Poland.

[6] Feit, W. and Thompson, J.G. (1963) Solvability of Groups of Odd Order. *Pacific Journal of Mathematics*, **13**, 775-1029. http://dx.doi.org/10.2140/pjm.1963.13.775

[7] Bender, H. and Glaubernan, G. (1995) Local Analysis for the Odd Order Theorem. Cambridge University Press, Cambridge. http://dx.doi.org/10.1017/CBO9780511665592

[8] Gonthier, G., Asperti, A., Avigad, J., Bertot, Y., Cohen, C., Garillot, F., Roux, S.L., Mahboubi, A., O'Connor, R., Biha, S.O., *et al*. (2013) A Machine-Checked Proof of the Odd Order Theorem. *Lecture Notes in Computer Science*, **7998**, 163-179. http://dx.doi.org/10.1007/978-3-642-39634-2_14

[9] Affeldt, R., Hagiwara, M. and Sénizergues, J. (2014) Formalization of Shannon's Theorems. *Journal of Automated Reasoning*, **53**, 63-103. http://dx.doi.org/10.1007/s10817-013-9298-1

[10] Shannon, C.E. (1948) A mathematical Theory of Communication. *Bell System Technical Journal*, **27**, 379-423. http://dx.doi.org/10.1002/j.1538-7305.1948.tb01338.x

[11] Hayashi, M., Ishizaka, S., Kawachi, A., Kimura, G. and Ogawa, T. (2014) Introduction to Quantum Information Science. Springer.

[12] Robertson, H.P. (1929) The Uncertanty Principle. *Physical Review*, **34**, 163-164. http://dx.doi.org/10.1103/PhysRev.34.163

[13] Bennett, C.H. and Brassard, G. (1984) Quantum Cryptography: Public Key Distribution and Coin Tossing. *Proceedings of IEEE International Conference on Computers Systems and Signal Processing*, Bangalore, 175-179.

[14] Biham, E., Boyer, M., Boykin, P.O., Mor, T. and Roychowdhury, V. (2000) A Proof of the Security of Quantum Key Distribution. *Proceedings of* 32*nd Annual ACM Symposium on the Theory of Computing*, New York, 715-724.

[15] Miyadera, T. and Imai, H. (2006) Information-Disturbance Theorem for Mutually Unbiased Observables. *Physical Review A*, **73**, Article ID: 042317. http://dx.doi.org/10.1103/PhysRevA.73.042317

[16] Miyadera, T. and Imai, H. (2007) Information-Disturbance Theorem and Uncertainty Relation. http://arxiv.org/abs/0707.4559

[17] Maassen, H. and Uffink, J.B.M. (1988) Generalized Entropic Uncertainty Relations. *Physical Review Letters*, **60**, 1103. http://dx.doi.org/10.1103/PhysRevLett.60.1103

[18] Krishna, M. and Parthasarathy, K.R. (2002) An Entropic Uncertainty Principle for Quantum Measurements. *Sankhya*: *The Indian Journal of Statistics, Series A*, **64**, 842-852.

Appendix: Coq Source Code

```
Require Import Reals Fourier.
Require Import Arith Omega Eqdep_dec.
Set Implicit Arguments.

Definition C : Type := prod R R.
Definition cplus (a b : C) : C := ((fst a + fst b), (snd a + snd b) ).
Definition cminus (a b : C) :C := ((fst a - fst b), (snd a - snd b) ).
Definition cmult (a b : C) : C := ((fst a * fst b - snd a * snd b),
(fst a * snd b + fst a * snd b)).
Definition conjC (a : C) : C := ( (fst a), (-snd a) ).
Definition cabs (a : C) : R := (sqrt ( fst a * fst a + snd a * snd a)).
Definition Re (a : C ) : R := fst a.
Definition Im (a : C ) : R := snd a.

Section Vector.
Variable A: Type.
Inductive Vec : nat -> Type :=
  | vnil : Vec 0
  | vcons : forall (a:A) (n:nat), Vec n -> Vec (S n).
Definition head {n} : Vec (S n) -> A.
 intros.
 inversion X as [| m x _].
 exact m.
Qed.
Definition tail : forall n, Vec(S n) -> Vec n.
 intros.
 inversion X as [| _ m xs].
 exact xs.
Qed.
End Vector.
Implicit Arguments vnil [A].
Implicit Arguments head [A].
Implicit Arguments tail [A].

Section VMap.
Variable A B : Type.
Variable length : nat.
Definition map : (A -> B) -> Vec A length -> Vec B length.
 intros.
 induction length.
 exact vnil.
 exact (vcons (X (head _ X0) ) (IHn (tail _ X0) ) ).
Qed.
End VMap.

Delimit Scope Vector_Scope with Vec.
Open Scope Vector_Scope.
Infix "::" := vcons (at level 60, right associativity) : Vector_Scope.
Notation "[]" := vnil: Vector_Scope.
Notation "[ x , .. , z ]" := (x :: .. (z :: []) ..) (at level 0) : Vector_Scope.

Section VZip.
Variables X Y Z: Type.
Variable f: X -> Y -> Z.
Fixpoint vzip {n} (v1: Vec X n) (v2: Vec Y n): Vec Z n.
 destruct v1.
 constructor.
 inversion v2.
 constructor.
 apply f.
 exact a.
 exact a0.
 apply vzip.
 exact v1.
 exact X0.
Qed.
End VZip.

Delimit Scope Matrix_Scope with Mat.
Open Scope Matrix_Scope.
Notation "' mlengths' " := (Vec nat) (at level 0) : Matrix_Scope.

Section Matrix.
Variable A: Type.
```

```
Inductive Mat: forall n, mlengths n -> Type :=
  | mscalar: A -> Mat []
  | mvector: forall n m v, Vec (@Mat n v) m -> @Mat (S n) (m::v).
End Matrix.

Notation "' #{' x , .. , z }" := (mvector (x :: .. (z :: []) ..)%Vec)
(at level 0) : Matrix_Scope.
Notation "' ' {' x , .. , z }" := (mvector (mscalar x :: .. (mscalar z :: []) ..)%Vec)
(at level 0) : Matrix_Scope.

Section MZip.
Variables X Y Z: Type.
Variable f: X -> Y -> Z.
Fixpoint mzip {n} (v: mlengths n) (m1: Mat X v) (m2: Mat Y v): Mat Z v.
 destruct n as [|n'].
 inversion m1.
 inversion m2.
 apply mscalar; apply f; assumption.
 inversion m1 as [|? m1' l1 v1 H1 Heq1]; subst.
 inversion m2 as [|? m2' l2 v2 H2 Heq2]; subst.
 apply (inj_pair2_eq_dec _ eq_nat_dec _ _ _) in Heq1.
 apply (inj_pair2_eq_dec _ eq_nat_dec _ _ _) in Heq2.
 assert (Heq: m1' :: l1 = m2' :: l2).
 eapply eq_trans.
 apply Heq1.
 symmetry.
 exact Heq2.
 clear Heq1.
 clear Heq2.
 assert (Heqm: m1' = m2') by (inversion Heq; reflexivity).
 assert (Heql: l1 = l2).
 inversion Heq.
 apply (Eqdep_dec.inj_pair2_eq_dec _ eq_nat_dec _ _ _) in H1.
 exact H1.
 clear Heq.
 rename l2 into l.
 rename m2' into m'.
 subst.
 apply mvector.
 specialize (mzip n' l).
 apply (vzip mzip); assumption.
Qed.
Fixpoint mPlus {n} (l : mlengths n) (m1 m2: Mat X l) : Mat X l.
 auto.
Qed.
Fixpoint mMinus {n} (l : mlengths n) (m1 m2: Mat X l) : Mat X l.
 auto.
Qed.
Fixpoint mMult {n} (l : mlengths n) (m1 m2: Mat X l) : Mat X l.
 auto.
Qed.
Fixpoint mvMult {n} (l: mlengths n) (m: Mat X l) (v: Vec Y n) : Vec Y n.
 auto.
Qed.
End MZip.

Parameter innerProd : forall (n : nat), Vec C n -> Vec C n -> C.
Definition UnitVec (t : Type) (n : nat) : Type := Vec t n.
Definition HMat (n : nat) (v : mlengths n): Type := Mat C v.
Definition vecNorm {n} (psi : Vec C n) : R := sqrt (Re (innerProd psi psi)).

Parameter O_leq_Sqrt : forall (a:R), 0 <= sqrt a.
Parameter O_leq_vecNorm : forall (n : nat) (psi : Vec C n), 0 <= vecNorm psi.
Parameter O_leq_Square_R : forall (a : R ), 0 <= a * a.
Parameter O_leq_Square_vecNorm : forall (n : nat) (psi : Vec C n),
```

```
0 <= (vecNorm psi) * (vecNorm psi).
Lemma R_eq : forall (a : R), a = a.
  auto.
Qed.
Parameter Square_vecNorm_eq_innerProd : forall (n : nat) (psi : Vec C n),
(vecNorm psi) * (vecNorm psi) = cabs (innerProd psi psi).
Parameter SchwarzIneq : forall (n : nat) (psi phi : Vec C n),
cabs (innerProd psi phi ) <= (vecNorm psi) * (vecNorm phi).
Parameter Squared_SI : forall (n : nat) (psi phi : UnitVec C n),
(cabs ( innerProd psi phi ) ) * ( cabs ( innerProd psi phi ) )
  <= ( (vecNorm psi) * (vecNorm psi) ) * ((vecNorm phi) * (vecNorm phi)).
Parameter Squared_SI2 : forall (n : nat) (psi phi: UnitVec C n),
((vecNorm psi) * (vecNorm psi)) * ((vecNorm phi) * (vecNorm phi))
  >= (cabs ( innerProd psi phi ) ) * (cabs ( innerProd psi phi ) ).
Definition eval (n : nat ) (v : mlengths n) ( psi : UnitVec C n ) (A : HMat v) :
R := Re ( innerProd psi (mvMult A psi) ).
Definition var (n : nat) (v : mlengths n) ( psi : UnitVec C n) (A : HMat v ) : R :=
 Re ( cminus ( innerProd psi (mvMult (mMult A A) psi))
              ( cmult (innerProd psi (mvMult A psi)) (innerProd psi (mvMult A psi)) )
).
Parameter var_A : forall (n : nat) (v : mlengths n) (A : HMat v) (psi : UnitVec C n),
(vecNorm (mvMult A psi) ) * (vecNorm (mvMult A psi) )
= (sqrt (var psi A)) * (sqrt (var psi A)).
Parameter var_B : forall (n : nat) (v : mlengths n) (B : HMat v) (psi : UnitVec C n),
(vecNorm (mvMult B psi) ) * (vecNorm (mvMult B psi) )
= (sqrt (var psi B)) * (sqrt (var psi B)).
Lemma Mat_Vec_SI  : forall (n : nat) (v : mlengths n) (psi phi : Vec C n) (A B : Mat C v),
cabs (innerProd (mvMult A psi) (mvMult B phi) )
<= (vecNorm (mvMult A psi) ) * (vecNorm (mvMult B phi) ).
  intros.
  set (xi := mvMult A psi).
  set (eta := mvMult B psi).
  apply SchwarzIneq.
Qed.

Parameter O_leq_Square_R2 : forall (a : R ), a * a >= 0.
Parameter a_Plus_b_leq_a : forall (a b :R), b >= 0 -> b + a >= a.
Lemma Square_Plus_geq : forall (a b : R), Rplus (Rmult b b) (Rmult a a) >= Rmult a a.
  intros.
  set ( c := (Rmult a a) ).
  assert ( 0 <= b * b  ).
  apply O_leq_Square_R.
  apply a_Plus_b_leq_a.
  apply O_leq_Square_R2.
Qed.
Parameter innerProd_HMat : forall (n : nat) (v : mlengths n)
(A B : HMat v) (psi phi : Vec C n),
innerProd (mvMult A psi) (mvMult B phi) = innerProd psi (mvMult (mMult A B) phi).
Parameter innerProd_HMat_eq : forall (n : nat) (v : mlengths n)
(A B : HMat v) (psi : Vec C n),
innerProd (mvMult A psi) (mvMult B psi) = innerProd psi (mvMult (mMult A B) psi).
Parameter RUR_eq_AB_0 : forall (n : nat) (v : mlengths n) (A B : HMat v) (psi : Vec C n),
( (1/2) * (cabs ( (innerProd psi (mvMult (mMinus (mMult A B) (mMult B A) ) psi) ) ) ) ) *
( (1/2) * (cabs ( (innerProd psi (mvMult (mMinus (mMult A B) (mMult B A) ) psi) ) ) ) )
 = (1/4) * ( (cabs ( innerProd psi (mvMult (mMinus (mMult A B) (mMult B A) ) psi))) *
              (cabs ( innerProd psi (mvMult (mMinus (mMult A B) (mMult B A) ) psi)))).
Parameter RUR_eq_AB_1 : forall (n : nat) (v : mlengths n) (A B : HMat v) (psi : Vec C n),
  ( ( (1/2) * (cabs ( (innerProd psi (mvMult (mPlus (mMult A B) (mMult B A) ) psi) ) ) ) ) *
    ( (1/2) * (cabs ( (innerProd psi (mvMult (mPlus (mMult A B) (mMult B A) ) psi) ) ) ) ) )
+ ( ( (1/2) * (cabs ( (innerProd psi (mvMult (mMinus (mMult A B) (mMult B A) ) psi) ) ) ) ) *
    ( (1/2) * (cabs ( (innerProd psi (mvMult (mMinus (mMult A B) (mMult B A) ) psi) ) ) ) ) )
= (cabs ( innerProd psi (mvMult (mMult A B ) psi))) *
(cabs ( innerProd psi (mvMult (mMult A B ) psi))) .
Parameter RUR_eq_AB_2 : forall (n : nat) (v : mlengths n) (A B : HMat v) (psi : Vec C n),
sqrt ( (cabs ( innerProd psi (mvMult (mMult A B ) psi)))
```

```
* (cabs ( innerProd psi (mvMult (mMult A B ) psi))) )
 =  cabs ( innerProd psi (mvMult (mMult A B ) psi)).
Parameter RUR_eq_AB_3 : forall (n : nat) (v : mlengths n) (A B : HMat v) (psi : Vec C n),
sqrt ( (1/4) * ( cabs ( innerProd psi (mvMult (mMinus (mMult A B) (mMult B A) ) psi)) ) )
 = (1/2) * ( cabs ( innerProd psi (mvMult (mMinus (mMult A B) (mMult B A) ) psi)) ).
Parameter R_Square : forall (a b c d: R),
(a * a * (b * b) >= c * c * (d * d) ) = (a * b >= c * d).
Parameter Square_Half : (1 / 2) * (1 / 2) = (1 / 4).
Parameter R_Transitivity : forall ( a b c d e: R),
(a * a * (b * b) >= (c * c) /\ (c * c) >= d * (e * e) ) -> (a * a * (b * b) >= d * (e * e) ).
Lemma R_Square_leq : forall (a b c: R),
( a * a * (b * b) >= 1 / 4 * (c * c) ) = ( a * b >= 1/ 2 * c ).
  intros.
  replace ( (1 / 4) ) with ( (1 / 2) * (1 / 2) ).
  apply R_Square.
  apply Square_Half.
Qed.
Lemma R_leq_eq : forall ( a b : R) (c : C),
( sqrt a * sqrt a * (sqrt b * sqrt b) >= (1 / 4) * (cabs c * cabs c) )
= (sqrt a * sqrt b >= (1 / 2) * cabs c ).
  intros.
  set( x := sqrt a ).
  set( y := sqrt b ).
  set( z := cabs c ).
  replace ( (1 / 4) ) with ( (1 / 2) * (1 / 2) ).
  apply R_Square.
  apply Square_Half.
Qed.
Lemma R_leq_eq2 : forall (a b c d : R),
( (a * a) * (b * b) >= c * c /\ c * c >= ( 1 / 4 ) * (d * d) )
-> ( (a * a) * (b * b) >= (1 / 4) * (d * d) ).
  intros.
  assert( (a * a * (b * b) >= c * c /\ c * c >= ( 1 / 4 ) * (d * d) )
  -> ( (a * a) * (b * b) >= (1 / 4) * (d * d) ) ).
  apply R_Transitivity.
  apply H0.
  apply H.
Qed.
Lemma RUR_Var_geq_InnerProd : forall (n : nat) (v : mlengths n)
(A B : HMat v ) (psi : UnitVec C n),
  ((sqrt (var psi A)) * (sqrt (var psi A)) ) * ( (sqrt (var psi B)) * (sqrt (var psi B))) >=
  (cabs ( innerProd psi (mvMult (mMult A B ) psi))) *
  (cabs ( innerProd psi (mvMult (mMult A B ) psi))).
  intros.
  replace ( (sqrt (var psi A) ) * (sqrt (var psi A) ) )
  with ( ( vecNorm (mvMult A psi) ) * (vecNorm (mvMult A psi) ) ).
  replace ( (sqrt (var psi B) ) * (sqrt (var psi B) ) )
  with ( ( vecNorm (mvMult B psi) ) * (vecNorm (mvMult B psi) ) ).
  replace (innerProd psi (mvMult (mMult A B) psi) )
  with (innerProd (mvMult A psi) (mvMult B psi) ).
  set ( xi := mvMult A psi ).
  set ( eta := mvMult B psi ).
  apply Squared_SI2.
  apply innerProd_HMat_eq.
  apply var_B.
  apply var_A.
Qed.

Section RobertsonUR.
Theorem RobertsonUR : forall (n : nat) (v : mlengths n) (A B: HMat v) (psi : UnitVec C n),
 (sqrt (var psi A)) * (sqrt (var psi B))
 >= (1/2) * (cabs ( innerProd psi (mvMult (mMinus (mMult A B) (mMult B A) ) psi)) ) .
 intros.
 set ( a := sqrt (var psi A) ).
 set ( b := sqrt (var psi B) ).
```

```
 set ( c := (cabs ( innerProd psi (mvMult (mMinus (mMult A B) (mMult B A) ) psi)) ) ).
 replace ( a * b >= 1 / 2 * c ) with ( (a * a) * (b * b) >= (1/4) * (c * c) ).
 assert ( a * a * (b * b) >= (cabs ( innerProd psi (mvMult (mMult A B ) psi)))
 * (cabs ( innerProd psi (mvMult (mMult A B ) psi)))
/\ (cabs ( innerProd psi (mvMult (mMult A B ) psi)))
* (cabs ( innerProd psi (mvMult (mMult A B ) psi)))
>= (1 / 4) * (c * c) -> a * a * (b * b) >= 1 / 4 * (c * c) ).
 apply R_leq_eq2.
 apply H.
 split.
 apply RUR_Var_geq_InnerProd.
 set ( d := cabs ( innerProd psi (mvMult (mMult A B ) psi)) ).
 replace ( d * d ) with
 ( ( ( (1/2) * (cabs ( (innerProd psi (mvMult (mPlus (mMult A B) (mMult B A) ) psi) ) ) ) ) *
    ( (1/2) * (cabs ( (innerProd psi (mvMult (mPlus (mMult A B) (mMult B A) ) psi) ) ) ) ) )
+ ( ( (1/2) * (cabs ( (innerProd psi (mvMult (mMinus (mMult A B) (mMult B A) ) psi) ) ) ) ) *
    ( (1/2) * (cabs ( (innerProd psi
    (mvMult (mMinus (mMult A B) (mMult B A) ) psi) ) ) ) ) ) ).
 replace ( (1 / 4) * (c * c) ) with
 ( ( (1/2) * (cabs ( (innerProd psi (mvMult (mMinus (mMult A B) (mMult B A) ) psi) ) ) ) ) *
( (1/2) * (cabs ( (innerProd psi (mvMult (mMinus (mMult A B) (mMult B A) ) psi) ) ) ) ) ).
 apply Square_Plus_geq.
 apply RUR_eq_AB_0.
 apply RUR_eq_AB_1.
 apply R_leq_eq.
Qed.
End RobertsonUR.
```