Proyecto de Verificación Formal – Curso 2021-I
Biblioteca – Problema de las cartas rusas
Eduardo Pascual Aseff

Introducción

En los protocolos de transmisión de información frecuentemente es necesario garantizar seguridad en la comunicación con respecto a algún espía o adversario. En la práctica se utiliza la criptografía para esto, la cual presupone que los adversarios cuentan con un poder de cómputo limitado. Para estudiar la seguridad en presencia de un adversario con capacidad de cómputo ilimitada, en varias ocasiones se han estudiado juegos de cartas. Un ejemplo de esto es el ***Problema de las cartas rusas*** [2], cuya versión clásica u original puede enunciarse de la siguiente manera:

Un mazo de 7 cartas etiquetadas del 0 al 6, es repartido entre tres jugadores de la siguiente manera: tres cartas a Alice, tres cartas a Bob y una carta a Cath. Mostrar si es posible para Alice y Bob aprender las cartas que tiene el otro mediante anuncios públicos, mientras puedan asegurar que Cath no pueda saber cuándo Alice o Bob posee una carta en particular.

La propiedad o requerimiento de que Alice y Bob aprendan las cartas del otro se conoce como "*Informatividad*", mientras que la propiedad de que Cath no aprenda quien posee una carta en particular se conoce como "*Seguridad*". Este problema tiene diferentes generalizaciones, variando con respecto al total de cartas y las cantidades repartidas a cada jugador, así como la cantidad de jugadores o nociones de informatividad y seguridad.

Un aspecto importante a tener en cuenta en este problema, es que si Alice logra hacer un anuncio informativo a Bob, entonces Bob aparte de saber las cartas de Alice, también por descarte puede saber las cartas de Cath. Entonces, en esta situación, Bob puede anunciarle a Alice las cartas de Cath aprendidas, lo cual no le aporta ninguna información nueva a Cath, y le permite a Alice conocer la mano de Bob de manera segura. Por lo tanto, los estudios normalmente solo se enfocan en los posibles anuncios de Alice a Bob.

En los estudios de seguridad, teniendo en cuenta el principio de Kerckhoffs, que planeta que "*la efectividad del sistema no debe depender de que su diseño permanezca en secreto*", se considera que el protocolo utilizado es de conocimiento común. Por otro lado, puede demostrarse que todo anuncio de Alice es equivalente a que ella diga a que su mano es una de la presentes en un conjunto de manos. Por ejemplo, considerando la repartición de cartas $\big((234),(156),(0)\big)$, Alice pudiera anunciar lo siguiente: "mi mano se encuentra en el conjunto $\{(012),(234),(056)\}$". Con este anuncio, Bob puede llegar a la conclusión de que

Alice tiene la mano (234), ya que las otras dos manos no son compatibles con su mano, pero de manera similar, Cath también aprende la mano de Alice (y por lo tanto la de Bob).

Este trabajo tiene como ***objetivo*** demostrar propiedades del Problema de las cartas rusas y sus soluciones, auxiliándose del sistema de gestión de pruebas formales Coq[1]. Además, se desarrolló una biblioteca para trabajar con conjuntos en Coq. En la Biblioteca de Conjuntos se describe la biblioteca de conjuntos antes mencionada, y en la Problema Ruso de las cartas se describe el modelado del Problema de las cartas rusas en Coq, así como las propiedades de este demostradas.

[1] https://coq.inria.fr/

1. Biblioteca de Conjuntos

En el modelado del Problema de las cartas rusas, es necesario abstraer las manos de cartas, así como colecciones de manos de cartas. Una mano de cartas es simplemente un conjunto de cartas, y una colección de manos sin elementos repetidos es a su vez un conjunto de manos, por eso la utilidad de poder representar conjuntos en este proyecto. Como parte de este proyecto, se implementó una biblioteca de conjuntos que se encuentra en los ficheros Defs_ListSets.v y Props_ListSets.v. Esta biblioteca y sus propiedades se pueden utilizar con solo importar el fichero Props_ListSets.v

Aquí se representan los conjuntos a través de listas (de la biblioteca estándar de Coq) que satisfacen la propiedad *isSet*, la cual garantiza que no haya elementos repetidos dentro de la lista. En el caso de las manos, o conjuntos de cartas, nos conviene poder determinar cuándo dos son iguales, y esto se cumple cuando una es permutación de la otra; para esto se utilizamos la propiedad *Permutation* de la biblioteca estándar de Coq.

Relaciones entre conjuntos

Algunas propiedades del Problema de las cartas Rusas, surgen de relaciones entre posibles manos, o entre relaciones entre manos y anuncios, por lo tanto es necesario abstraer las relaciones más importantes que existen entre conjuntos. Entre estas se encuentran, por ejemplo, cuando dos conjuntos son disjuntos, o cuando un conjunto es la unión entre otros dos. Las relaciones definidas se listan a continuación:

- Conjuntos Disjuntos. Se expresa mediante la función *aredisjoint h1 h2*.
- Subconjuto. Se expresa mediante la propiedad *subset h1 h2*.
- Intersección. Se expresa a través de la propiedad ternaria *intersection h1 h2 inters*, la cual se satisface si y solo si, los elementos de *inters* se encuentran en *h1* o *h2* simultáneamente.
- Unión. Se expresa mediante la propiedad ternaria *union h1 h2 uni*, la cual se cumple si y solo los elementos de *uni*, se encuentran en el conjunto *h1* o en el *h2*.
- Diferencia. Se expresa mediante la propiedad ternaria *difference h1 h2 dif*, la cual se satisface si y solo sí, los elementos en *dif* están en *h1* y además no es están en *h2*.

Propiedades de los conjuntos.

En el Conclusiones

En este proyecto, utilizando el Sistema de gestión de pruebas formales Coq, se modelaron y demostraron algunas propiedades que satisfacen las soluciones del Problema de las cartas rusas. El modelado y las demostraciones desarrollados permiten seguir investigando y realizar otras demostraciones sobre propiedades de este problema. Por otro lado, se diseñó e implementó una biblioteca para representar conjuntos mediante listas en

Coq, la cual es independiente del Problema bajo estudio, y puede ser de utilidad para otros trabajos.

Anexo 1 – Propiedades de la biblioteca de conjuntos. se listan las propiedades relacionadas con los conjuntos, que se demostraron en este proyecto para ser utilizados en la demostración de propiedades del Problema de las cartas rusas.

2. Problema Ruso de las cartas

En esta sección describiremos como fue modelado el problema, así como los resultados obtenidos en el proyecto. El código del proyecto en relación con el Problema de las cartas rusas está divido en dos ficheros: Defs_RCP.v y Props_RCP.v. En Defs.v se encuentran las definiciones necesarias para modelar el problema. En el fichero Properties.v se demuestran algunas propiedades inherentes al problema bajo estudio.

Modelado del problema

Se trató de modelar el problema de la manera más general posible, por lo que las cartas son representadas mediante un tipo no específico $Card$. Además no se restringió la cantidad de cartas repartidas a Alice, Bob y Cath; aquí consideramos que ellos recibieron A, B y C cartas respectivamente, los cuales son naturales con la única condición dada por $A > 1$. Si A fuese 0, no tuviese mucho sentido este problema, ya que no hay nada que Alice quiera o pueda comunicar a Bob. Por otro lado, todas las cartas pertenecen a un mazo, denominado $Deck$, el cual tiene tamaño $N = A + B + C$. Todo lo anterior se representa en Coq de la siguiente manera:

```
Parameter (Card: Type)
          (A B C: nat)
          (Deck: list Card).
Definition N : nat := A + B + C.
Axiom Alice_has_cards: A > 0.
Axiom Deck_size: length Deck = N.
Axiom Deck_def: forall (c : Card), In c Deck.
```

Figura 1. Parametros para modelar el problema

Las manos se denominan como $Tuple$, y específicamente son representadas mediante una lista de cartas. Para validar que una $Tuple$ representa realmente una mano, se utiliza el predicado *isHand*, el cual es equivalente a decir que es un conjunto. Además se cuenta con el predicado $isHandLen$ para garantizar que una mano tenga una longitud dada. Para estudiar los posibles anuncios de Alice, se definió el predicado $isAnnouncement$, el cual nos indica que una lista de manos sea un conjunto donde todas las manos tienen longitud A.

```
Definition isHand (tup : Tuple)  := isSet tup.
Definition isHandLen (len : nat) (tup : Tuple) : Prop :=
  length tup = len /\ isHand tup.
Definition isAnnouncement (tup: list Tuple): Prop :=
  isSet tup /\
  forall (elem : Tuple), In elem tup -> isHandLen A elem.
```

Figura 2. Modelado de manos de cartas y de anuncios de Alice

Seguridad

Considerando y uniendo los axiomas CA2 y CA3 de [1], un anuncio L se considera *seguro*, si para cualquier mano posible que tenga Cath, y para cualquier carta x que Cath no posee, es posible que Alice tenga o no la carta x. Esto se puede expresar como que existe una mano en L, compatible con la mano de Cath, tal que x está en esa mano, y además existe otra mano en L, también compatible con la mano de Cath, tal que x no está en esa mano. Esto lo podemos expresar en Coq de la siguente manera:

```
Definition isSafe (ann: list Tuple) : Prop :=
  forall (cH : Tuple), isHandLen C cH ->
    forall (x: Card), ~ In x cH ->
      (exists (t1 : Tuple),  isHandLen A t1  /\ areDisjoint cH t1 /\
                              In x t1 /\ In t1 ann )   /\
      (exists (t2 : Tuple),  isHandLen A t2  /\ areDisjoint cH t2 /\
                            ~ In x t2 /\ In t2 ann ).
```

Figura 3. Definición de seguridad

Informatividad

Basándonos en el axioma CA1 definido en [1], un anuncio L es informativo si para cualquier mano posible de Bob, hay a lo sumo una mano en L que es compatible con la mano de Bob. Esto es equivalente a decir que no existen dos manos en L que sean disjuntas a la mano de Bob, lo cual expresemos en Coq de la siguiente manera:

```
Definition isInformative (ann: list Tuple) : Prop :=
  forall (b : Tuple), (isHandLen B b) ->
    ~ exists (l1 l2 : Tuple),
      l1 <> l2 /\ In l1 ann /\ In l2 ann /\
      areDisjoint b l1 /\ areDisjoint b l2.
```

Figura 4. Definición de informatividad

Demostraciones

En este proyecto se demostraron varias propiedades del Problema de las cartas rusas, y dichas demostraciones aparecen en el fichero Properties.v. A continuación se listan las propiedades demostradas:

- El Lema 1 de [1], el cual plantea lo siguiente: Un anuncio L es informativo si y solo si cualquier par de manos $h1$ y $h2$ de L, se satisface que $|h1 \cap h2| < A - C$.
- El corolario 1 de [1], que plantea que existen anuncios informativos solo cuando $C < A$.
- El Lema 2 de [1], el cual dice que $C < B$ es una condición necesaria para que exista un protocolo informativo y seguro.

- El Lema 3 de [1], que dice que en todo anuncio informativo y seguro, cada carta aparece en al menos $C + 1$ manos.

Las propiedades anteriores se plantearon en Coq de la siguiente manera:

```
Lemma informativeAlternative: forall (ann : list Tuple),
isAnnouncement ann ->
( isInformative ann <->
  forall (l1 l2 inters: Tuple),
   l1 <> l2 -> In l1 ann -> In l2 ann -> isSet inters ->
     intersection l1 l2 inters -> length (inters) < A - C ).

Lemma corollary1: forall (ann: list Tuple), isAnnouncement ann ->
  length ann >= 2 -> isInformative ann -> C < A.

Lemma goodAnnouncemnt_onlyif_cIsLowerThanB:
  forall (ann: list Tuple), isAnnouncement ann -> isGood ann -> C < B.

Lemma goodAnnouncement_x_appears_more_than_C_times:
  forall (ann : list Tuple) (x : Card), A > 1 ->
    isAnnouncement ann -> isGood ann ->
    exists (xHands : list Tuple),
      isSet xHands /\
      (forall elem, In elem xHands -> In x elem) /\
       subset xHands ann /\
       length xHands >= C+1.
```

Figura 5. Propiedades demostradas del Problema de las cartas rusas

Conclusiones

En este proyecto, utilizando el Sistema de gestión de pruebas formales Coq, se modelaron y demostraron algunas propiedades que satisfacen las soluciones del Problema de las cartas rusas. El modelado y las demostraciones desarrollados permiten seguir investigando y realizar otras demostraciones sobre propiedades de este problema. Por otro lado, se diseñó e implementó una biblioteca para representar conjuntos mediante listas en Coq, la cual es independiente del Problema bajo estudio, y puede ser de utilidad para otros trabajos.

Anexo 1 – Propiedades de la biblioteca de conjuntos.

En este anexo se listan las propiedades de conjuntos demostradas en este proyecto.

1. Si una lista contiene dos elementos diferentes, entonces su longitud es mayor o igual que dos.

```
Lemma twoIn_LenGE2:
  forall {X : Type} (l1 l2 : X) (l : list X),
  l1 <> l2 -> In l1 l -> In l2 l -> length l >= 2.
```

2. Una lista es un conjunto si y solo si satisface el predicado *NoDup* de la biblioteca estándar de Coq. El predicado *NoDup* se satisface cuando una lista no contiene elementos repetidos.

```
Lemma isSetNoDup_equiv: forall {X: Type} (l : list X),
  isSet l <-> NoDup l.
```

3. Para dos conjuntos *h1, h2* podemos decir siempre que existe otro conjunto que es su unión, intersección o unión respectivamente.

```
Lemma unionExists: forall {X : Type} (h1 h2 : list X),
  isSet h1 -> isSet h2 ->
  exists uni : list X, union h1 h2 uni /\ isSet uni.

Lemma intersectionExists: forall {X : Type} (h1 h2 : list X),
  isSet h1 -> isSet h2 ->
  exists inters : list X, intersection h1 h2 inters /\ isSet inters.

Lemma intersectionExists': forall {X: Type} (h1 h2 : list X),
  exists inters : list X, intersection h1 h2 inters.

Lemma differenceExists: forall {X: Type} (h1 h2 : list X),
  isSet h1 -> isSet h2 ->
  exists diff : list X, difference h1 h2 diff /\ isSet diff.
```

4. Para todo lista *l* que contenga una carta *c*, podemos decir que existe otra lista *l´* tal que al agregarle *c*, obtenemos a *l*.

```
Lemma extractFromSet: forall {X: Type} (l : list X) (c : X),
  In c l -> exists l', l = c::l'.
```

5. Dos conjuntos son disjuntos si y solo si su intersección es el conjunto vacío.

```
Lemma disjointIntersection: forall {X: Type} (h1 h2 inter: list X),
  intersection h1 h2 inter -> areDisjoint h1 h2 <-> inter = [].
```

6. Las relaciones de intersección, unión, y de que dos conjuntos son disjuntos es conmutativa.

```
Lemma intersectionComm: forall {X: Type} (h1 h2 inters: list X),
  intersection h1 h2 inters -> intersection h2 h1 inters.

Lemma disjointComm: forall {X: Type} (h1 h2: list X),
  areDisjoint h1 h2 -> areDisjoint h2 h1.

Lemma unionComm: forall {X: Type} (h1 h2 u: list X),
  union h1 h2 u -> union h2 h1 u.
```

7. Algunos resultados de aplicar la diferencia, intersección y unión de conjuntos, al conjunto vacío.

```
Lemma difference_nil_left: forall {X: Type} (h1 dif: list X),
  difference [] h1 dif -> dif = [].

Lemma difference_nil_right: forall {X: Type} (h1 dif: list X),
  isSet h1 -> isSet dif ->
  difference h1 [] dif -> dif = h1.

Lemma intersection_nil_left: forall {X: Type} (h1 inters: list X),
  intersection [] h1 inters -> inters = [].

Lemma union_nil_left: forall {X: Type} (h1 uni: list X),
  union [] h1 uni -> isSet h1 -> isSet uni -> h1 = uni.
```

8. Algunos resultados relacionados con la cardinalidad de la intersección, unión y diferencia de conjuntos, como por ejemplo el principio de inclusión y exclusión.

```
Lemma incExclPrinciple: forall {X: Type}
                               (h1 h2 inters uni : list X),
  isSet h1 -> isSet h2 -> isSet inters -> isSet uni ->
    intersection h1 h2 inters -> union h1 h2 uni ->
    length h1 + length h2 = length uni + length inters.

Lemma diffLength': forall {X: Type} (h1 h2 dif inters : list X),
  isSet h1 -> isSet h2 -> isSet dif -> isSet inters ->
```

```
  difference h1 h2 dif -> intersection h1 h2 inters ->
    length dif + length inters = length h1.

Lemma diffLength: forall {X: Type} (h1 h2 dif inters : list X),
  isSet h1 -> isSet h2 -> isSet dif -> isSet inters ->
  difference h1 h2 dif -> intersection h1 h2 inters ->
    length dif = length h1 - length inters.

Lemma unionLength: forall {X: Type} (h1 h2 uni inters : list X),
  isSet h1 -> isSet h2 -> isSet uni -> isSet inters ->
  union h1 h2 uni -> intersection h1 h2 inters ->
    length uni = length h1 + length h2 - length inters.
```

9. Si un conjunto es disjunto a la unión de otros dos, entonces es disjunto a esos dos conjuntos.

```
Lemma disjointUnionLeft: forall {X: Type} (h1 h2 uni d: list X),
  union h1 h2 uni -> areDisjoint d uni -> areDisjoint d h1.
```

10. Si un conjunto es disjunto a otros dos, entonces es disjunto a su unión.

```
Lemma disjointUnion: forall {X: Type} (b h1 h2 uni : list X),
  areDisjoint b h1 -> areDisjoint b h2 ->
  union h1 h2 uni -> areDisjoint b uni.
```

11. Si un conjunto *h2* tiene tamaño *L2*, y $L1 \leq L2$, entonces existe un subconjunto de *h2* con tamaño *L1*.

```
Lemma existsSubSet: forall {X: Type} (h2 : list X) (L1 L2 : nat),
  isSet h2 -> length h2 = L2 -> L1 <= L2 ->
  exists h1, (length h1 = L1 /\ isSet h1) /\ subset h1 h2.
```

12. Si *h1* es un subconjunto de *h2*, y *h2* es disjunto de un conjunto *d*, entonces *h1* también es disjunto con *d*.

```
Lemma disjointSubset: forall {X: Type} (h1 h2 d: list X),
  subset h1 h2 -> areDisjoint h2 d -> areDisjoint h1 d.
```

13. Un conjunto o lista de manos, se puede dividir en dos conjuntos, uno de los cuales satisface que todas las manos contienen una carta *x*, y el otro satisface que ninguna de sus manos contiene a la carta *x*.

```
Lemma setSplit: forall {X: Type} (ann: list (list X)) (x: X),
  isSet ann -> exists (ann_withx ann_nox : list (list X)),
  isSet ann_withx /\ isSet ann_nox /\ union ann_withx ann_nox ann /\
```

```
    (forall elem : list X, In elem ann_withx ->   In x elem) /\
    (forall elem : list X, In elem   ann_nox -> ~ In x elem).
```

14. Si un conjunto es más grande que otro, es porque existe una carta *x* tal que el más grande la contiene y el más pequeño no.

```
Lemma biggerSetHasOtherElements: forall {X: Type} (h1 h2 : list X),
  isSet h1 -> isSet h2 -> length h1 < length h2 ->
  exists x : X, ~ In x h1 /\ In x h2.
```

Referencias

[1] M. H. Albert, R. E. L. Aldred, M. D. Atkinson, H. P. van Ditmarsch, and C. C. Handley, “Safe communication for card players by combinatorial designs for two-step protocols,” *Australas. J. Comb.*, vol. 33, 2005.

[2] H. van Ditmarsch, “The Russian Cards Problem,” *Stud. Log.*, vol. 75, no. 1, 2003, doi: 10.1023/A:1026168632319.