Examen du cours Calcul des Constructions Inductives

B. Barras, H. Herbelin et C. Paulin

2 janvier 2004

1 Rappels sur la librairie standard de Coq

Cette section ne comporte pas de question. Elle rappelle les définitions utiles de la librairie standard de Coq.

```
Inductive nat : Set := O : nat | S : nat -> nat.
Inductive option [A : Set]  : Set :=
      Some : A->(option A) | None : (option A).
Inductive sumor [A : Set; B : Prop]  : Set :=
      inleft : A->A+{B} | inright : B->A+{B}
Inductive sig [A : Set; P : A->Prop]  : Set :=
      exist : (x:A)(P x)->(sig A P)
Inductive sumbool [A,B:Prop] : Set :=
      left  : A -> (sumbool A B)
    | right : B -> (sumbool A B).
```

On écrira `{A}+{B}` pour `(sumbool A B)`, et `{ y:N | P } + { Q }` est une notation pour `(sumor (sig N [y:N]P) Q)`.

2 Modélisation à l'aide de définitions inductives

Le but de cet exercice est d'utiliser les définitions inductives de Coq pour modéliser un protocole de télévision payante.

Les acteurs de ce protocole sont : la carte à puce C, le décodeur D et un intrus I, ces différents acteurs forment un type `agent`. On suppose donné un ensemble `key` de clés de cryptage symétriques et privées (connues uniquement du décodeur et de la carte à puces) ainsi qu'un ensemble de canaux `channel` qui contient deux canaux distincts c_1 et c_2. Des messages sont échangés sur le réseau. Ces messages peuvent être de la forme suivante :

- le nom d'un acteur
- une clé de cryptage
- un canal
- un message crypté par une clé de cryptage
- une paire formée de deux messages

Les messages envoyés peuvent être reçus par n'importe quel agent. L'intrus a de plus la possibilité d'envoyer des messages parasites. Les seuls messages que peut envoyer l'intrus sont les messages qu'il "connaît" et qui sont formés de :

- les messages qui ont transité sur le réseau,
- la paire de deux messages qu'il connaît,
- l'une ou l'autre des composantes d'un message formé d'une paire,
- un message qu'il connaît crypté par une clé qu'il connait
- un message crypté qu'il connaît décrypté par la bonne clé qu'il doit connaître.

On introduit les relations `send` et `receive` de type `agent`→`message`→`Prop` qui représentent respectivement les messages envoyés ou reçus par un agent ainsi qu'un prédicat `known` de type `message`→`Prop` qui représente les messages connus de l'intrus.

1. Donner les définitions inductives correspondant aux types `agent` et `message`.
2. On suppose que la relation `send` est donnée. La relation `receive` est alors définie par :

```
Inductive receive [A:agent;m:message] : Prop :=
  receive_intro : (B:agent)(send B m)->(receive A m)
```

   Donner un équivalent de cette formule à l'aide des connecteurs logiques usuels.

3. Soit le protocole suivant :
   - Le décodeur D envoie à la carte C le message m_1 formé d'une paire $(D,(c_1)_K)$ où $(c_1)_K$ est le canal c_1 crypté par la clé K.
   - Lorsque la carte reçoit le message du décodeur, elle vérifie que l'abonnement a été payé, et si c'est le cas renvoie le message $m_2 = (C,(c_1)_K)$ au décodeur.

   (a) Donner une définition mutuellement inductive des prédicats `send`, `receive` et `known` paramétrés par une proposition `paid` qui est vraie si l'abonnement est payé.

   (b) Montrer qu'il existe une preuve de $\neg$`paid` $\rightarrow$ (`receive` D m_2) (on ne détaillera pas la preuve)

   (c) Le protocole précédent est modifié en remplaçant m_2 par le message $m_3 = (C,(c_2)_K)$. Comment pourriez-vous faire une preuve de $\neg$`paid` $\rightarrow \neg$(`receive` D m_3) ?

3 Logique classique et CCI

On suppose la logique classique dans le monde calculatoire.

```
Variable em : (A:Prop){A}+{~A}.
```

1. Construire dans le CCI la fonction `phi` de `Prop` dans `Prop` qui à `A` associe la proposition vraie `True` si `A` est vrai et la proposition false `False` sinon.

2. On admet l'élimination dépendante de `sumbool` (ce qui est le cas du CCI) :

```
sumbool_ind : (A,B:Prop) (P:{A}+{B}->Prop)
  ((a:A)(P (left A B a))) -> ((b:B)(P (right A B b))) -> (s:({A}+{B})) (P s)
```

Montrer que l'énoncé `(A:Prop)(phi A)<->A` est prouvable.

3. On enrichit maintenant le language de terme avec une famille de lieurs $\mathcal{C}$ typés par

$$\frac{\Gamma, k : A \rightarrow \bot \vdash M : \bot}{\Gamma, \vdash \mathcal{C}_k^A.M : A}$$

où $\bot$ abrège $\forall B : Type.B$.
On munit $\mathcal{C}$ des règles de réduction suivantes
- Substitution d'un contexte applicatif

$$(\mathcal{C}_k^{\Pi x:A.C}.Mu) \longrightarrow \mathcal{C}_{k'}^{C[x:=u]}.(M[k := \lambda y : (\Pi x : A.C).k'(yu)])$$

- Substitution d'un contexte de filtrage non dépendant

$$\texttt{<P>Cases}\ (\mathcal{C}_k^{I(u_1,\dots,u_p)}M)\ \texttt{of}\ f_1 \dots f_n\ \texttt{end} \longrightarrow \mathcal{C}_{k'}^{P(u_1,\dots,u_p)}(M[k := K])$$

où $K = \lambda y : I(u_1 \dots u_p).k'(\texttt{<P>Cases}\ y\ \texttt{of}\ f_1 \dots f_n\ \texttt{end})$

– Simplification

$$\mathcal{C}_k^A.(kM) \longrightarrow M \qquad \text{si } k \text{ n'apparaît pas dans } M$$

Avec $\mathcal{C}$, le tiers-exclu est prouvable

```
Lemma em : (A:Prop){A}+{~A}.
Proof [A](C k.(k (right A ~A [a].(k (left A ~A a) False))).
```

et ϕ acquiert un contenu calculatoire. Que vaut alors `phi(True)` ? Et `phi(False)` ? Surprenant ?

4. Donner une preuve alternative `phi'` de `(em A)` quand `A` a la forme `~B`. Que valent `phi(~~True)`, `phi'(~~True)`, `phi(~~False)` et `phi'(~~False)` ?

5. Dans le cas de l'élimination non dépendante de `sumbool` (le type d'élimination utilisée dans la définition de `phi`), $\mathcal{C}$ a permis de donner un contenu calculatoire à la logique classique. Peut-on généraliser la règle de substitution de filtrage dans le cas dépendant (c.-à.-d. lorsque le prédicat P dépend aussi de l'argument filtré) :

$$\texttt{<P>Cases } (\mathcal{C}_k^{I(u_1,\ldots,u_p)} M) \texttt{ of } f_1 \ldots f_n \texttt{ end} \longrightarrow \mathcal{C}_{k'}^{??}(M[k := K])$$

où $K = \lambda y : I(u_1 \ldots u_p).k'(\texttt{<P>Cases } y \texttt{ of } f_1 \ldots f_n \texttt{ end})$

En particulier, quel type faut-il donner à k' ? Conclusion ? Peut-on donner un contenu calculatoire à l'élimination dépendante d'un objet classique ? Quel est le sens de la preuve de `(A:Prop)(phi A)<->A` ?

4 Entiers naturels en notation binaire

1. Définir `N` le type des entiers naturels, engendré à partir de 0, $2n + 1$ et $2n + 2$.
2. Écrire une fonction `N_inj : N->nat` qui traduit un entier en représentation binaire (`N`) vers sa représentation unaire (`nat`).
3. Écrire la fonction qui calcule le successeur de type `N->N` et évaluer sa complexité en temps en fonction de la valeur de l'entier donné.
4. On se donne la spécification S suivante :

```
(x:N) { y:N | (N_inj x)=(plus (N_inj y) (N_inj y)) } +
      { (EX y:N | (N_inj x)=(S (plus (N_inj y) (N_inj y)))) }
```

– Donner le type des réalisations $\mathcal{E}$(`S`) et le prédicat que doit satisfaire cette réalisation $\mathcal{R}$(`S`), en admettant les réalisations suivantes pour `sumor` et `sig` :

```
      E(sig A P)  =  A
  E(exist A P x p)  =  x
     E(sumor A B)  =  (option A)
   E(inleft A B x)  =  (Some A x)
  E(inright A B p)  =  (None A)

      R(sig A P)  =  P
     R(sumor A B)  =  [x :(option E(A))]Cases x of
                                         (Some y) => (R(A) y)
                                       | None => B
                                       end
```

– Donner une preuve de cette spécification. Pour les parties de la preuve établissant des propositions d'égalité ou d'existentielle, on pourra se contenter d'utiliser un théorème que l'on énoncera et que l'on admettra.

– Donner la réalisation de la spécification obtenue à partir de cette preuve.

5. On considère maintenant la spécification

```
(x:N) { (EX y:N | (N_inj x)=(plus (N_inj y) (N_inj y))) } +
      { (EX y:N | (N_inj x)=(S (plus (N_inj y) (N_inj y)))) }
```

- Montrer que l'on peut prouver cette spécification très simplement en réutilisant la preuve de la spécification précédente.
- Donner une preuve plus directe de cette spécification (toujours en admettant les propriétés portant sur les égalités et les existentielles) donnant lieu à un programme plus efficace.

5 Cohérence de l'irrelevance des preuves

Montrer que dans le Calcul des Constructions pur, le principe d'irrelevence des preuves est cohérent. Ce principe traduit que pour toute proposition A, toutes les preuves de A sont égales.

En d'autres termes, il s'agit de montrer qu'il n'existe pas de preuve de la proposition absurde dans le contexte ne contenant que l'axiome d'irrelevance des preuves.