Extraction de programmes

David Delahaye

Faculté des Sciences
David.Delahaye@lirmm.fr

Master M1 2017-2018

Extraction de programmes

Idée

- Extraire des programmes à partir de preuves ;
- Preuves avec un comportement calculatoire ;
- Leitmotiv : « prouver = programmer ».

Extraction : deux cas possibles

- On a une spécification, un programme, et une preuve :
  - On élimine la spécification et la preuve, et on garde le programme.
- On a une spécification et une preuve :
  - On élimine la spécification, et on extraie le programme de la preuve.

Interprétation de Brouwer-Heyting-Kolmogorov

Interprétation BHK

- Interprétation de la logique intuitionniste (sans le tiers exclu) ;
- Proposée par Brouwer et Heyting, et aussi par Kolmogorov ;
- Appelée aussi « interprétation par réalisabilité » (Kleene) ;
- Idée : donner une interprétation fonctionnelle aux preuves.

Interprétation de Brouwer-Heyting-Kolmogorov

Définition

Par induction sur les formules :

- Une preuve de $A \Rightarrow B$ est une fonction qui associe à une preuve de A une preuve de B ;
- Une preuve de $A \wedge B$ est un couple (π_1, π_2), où π_1 est une preuve de A et π_2 une preuve de B ;
- Une preuve de $A \vee B$ est soit une preuve de A, soit une preuve de B ;
- Une preuve de $\forall x.A(x)$ est une fonction qui associe à tout objet t une preuve de $A(t)$;
- Une preuve de $\exists x.A(x)$ est un couple (t, π), où t est un objet et π est une preuve de $A(t)$;
- Une preuve de $\neg A$ (vue comme $A \Rightarrow \bot$) est une fonction qui associe à toute preuve de A une preuve de $\bot$;
- On désigne par I la preuve de $\top$, et il n'existe pas de preuve $\bot$.

Isomorphisme ou correspondance de Curry-Howard

Principe et historique

- Basé sur une double correspondance :
  - Correspondance preuves/programmes ;
  - Correspondance formules/types.
- Curry : analogie entre les preuves dans les systèmes à la Hilbert et la logique combinatoire ;
- Howard : analogie entre les preuves en déduction naturelle intuitionniste et les termes du λ-calcul typé.

Cas de la logique implicative minimale

Règles en déduction naturelle (avec séquent)

$$\frac{}{\Gamma, A \vdash A}\,\mathrm{ax}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\Rightarrow_I \qquad \frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}\Rightarrow_E$$

Preuve de $(A \Rightarrow B) \Rightarrow A \Rightarrow B$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash A \Rightarrow B}\,\mathrm{ax} \qquad \dfrac{}{\Gamma \vdash A}\,\mathrm{ax}}{\Gamma = A \Rightarrow B, A \vdash B}\Rightarrow_E}{A \Rightarrow B \vdash A \Rightarrow B}\Rightarrow_I}{\vdash (A \Rightarrow B) \Rightarrow A \Rightarrow B}\Rightarrow_I$$

λ-calcul simplement typé

Termes et types

- Termes :
  - Les variables x, y, ... sont des variables ;
  - Si x est une variable, τ un type, et t un terme, alors $\lambda x : \tau.t$ est un terme (notation à la Church) ;
  - Si t_1 et t_2 sont des termes, alors $t_1\ t_2$ est un terme.
- Types :
  - Les types de base ι_1, ι_2, ... sont des types ;
  - Si τ_1 et τ_2 sont des types, alors $\tau_1 \rightarrow \tau_2$ est un type.

Règles de typage

$$\frac{(x, \tau) \in \Gamma}{\Gamma \vdash x : \tau}\ \mathrm{Var}$$

$$\frac{\Gamma, (x, \tau_1) \vdash t : \tau_2}{\Gamma \vdash \lambda x : \tau_1.t : \tau_1 \rightarrow \tau_2}\ \mathrm{Fun} \qquad \frac{\Gamma \vdash t_1 : \tau_1 \rightarrow \tau_2 \qquad \Gamma \vdash t_2 : \tau_1}{\Gamma \vdash t_1\ t_2 : \tau_2}\ \mathrm{App}$$

Isomorphisme de Howard

Correspondance formules/types : Φ

- $\Phi(A) = \iota_A$;
- $\Phi(A \Rightarrow B) = \Phi(A) \rightarrow \Phi(B)$.

Correspondance preuves/termes : φ

- Pour chaque contexte de preuve $\Gamma = A_1, \ldots, A_n$,
  $\varphi(\Gamma) = (x_{A_1}, \Phi(A_1)), \ldots, (x_{A_n}, \Phi(A_n))$
  (une variable unique par formule) ;
- Si la preuve π est de la forme :

$$\frac{}{\Gamma, A \vdash A}\ \mathrm{ax}$$

alors $\varphi(\pi) = x_A$;

Isomorphisme de Howard

Correspondance formules/types : Φ

- $\Phi(A) = \iota_A$;
- $\Phi(A \Rightarrow B) = \Phi(A) \rightarrow \Phi(B)$.

Correspondance preuves/termes : φ

- Si la preuve π est de la forme :

$$\dfrac{\dfrac{\pi'}{\Gamma, A \vdash B}}{\Gamma \vdash A \Rightarrow B} \Rightarrow_I$$

alors $\varphi(\pi) = \lambda x_A : \Phi(A).\varphi(\pi')$;

Isomorphisme de Howard

Correspondance formules/types : Φ

- $\Phi(A) = \iota_A$;
- $\Phi(A \Rightarrow B) = \Phi(A) \rightarrow \Phi(B)$.

Correspondance preuves/termes : φ

- Si la preuve π est de la forme :

$$\dfrac{\dfrac{\pi_1}{\Gamma \vdash A \Rightarrow B} \quad \dfrac{\pi_2}{\Gamma \vdash A}}{\Gamma \vdash B} \Rightarrow_E$$

alors $\varphi(\pi) = \varphi(\pi_1)\ \varphi(\pi_2)$;
- Théorème : pour une preuve π de $\Gamma \vdash A$, on a donc $\varphi(\Gamma) \vdash \varphi(\pi) : \Phi(A)$.

Retour sur l'exemple

Preuve de $(A \Rightarrow B) \Rightarrow A \Rightarrow B$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash A \Rightarrow B}\,\text{ax} \quad \dfrac{}{\Gamma \vdash A}\,\text{ax}}{\Gamma = A \Rightarrow B, A \vdash B}\Rightarrow_E}{A \Rightarrow B \vdash A \Rightarrow B}\Rightarrow_I}{\vdash (A \Rightarrow B) \Rightarrow A \Rightarrow B}\Rightarrow_I$$

Retour sur l'exemple

Preuve de $(A \Rightarrow B) \Rightarrow A \Rightarrow B$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash A \Rightarrow B}\,\text{ax} \quad \dfrac{}{\Gamma \vdash A}\,\text{ax}}{\Gamma = A \Rightarrow B, A \vdash B}\Rightarrow_E}{A \Rightarrow B \vdash A \Rightarrow B}\Rightarrow_I}{\vdash (A \Rightarrow B) \Rightarrow A \Rightarrow B}\Rightarrow_I$$

Arbre de typage correspondant

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : \iota_A \rightarrow \iota_B}\,\text{Var} \quad \dfrac{}{\Gamma \vdash y : \iota_A}\,\text{Var}}{\Gamma = (x, \iota_A \rightarrow \iota_B), (y, \iota_A) \vdash x\ y : \iota_B}\,\text{App}}{(x, \iota_A \rightarrow \iota_B) \vdash \lambda y : \iota_A . x\ y : \iota_A \rightarrow \iota_B}\,\text{Fun}}{\vdash \lambda x : \iota_A \rightarrow \iota_B . \lambda y : \iota_A . x\ y : (\iota_A \rightarrow \iota_B) \rightarrow \iota_A \rightarrow \iota_B}\,\text{Fun}$$

Extensions

Autres connecteurs et quantificateurs

- $\wedge$: produit cartésien, couple/projections ;
- $\vee$: union disjointe, injections/filtrage ;
- $\neg$: revient à l'implication ($\neg A \equiv A \Rightarrow \bot$) ;
- $\forall$: produit (implication dépendante), fonction/application ;
- $\exists$: sigma (produit cartésien dépendant), couple/projections.

Exercice

Logique implicative minimale

Démontrer les propositions suivantes en déduction naturelle et en extraire les termes correspondants en λ-calcul simplement typé :

- $A \Rightarrow B \Rightarrow A$;
- $(A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C$.

Extraction dans Coq

Caractéristiques de l'extraction

- Utilisation de l'isomorphisme de Curry-Howard ;
- Preuves encodées comme des fonctions Coq ;
- Extraction du comportement calculatoire dans une spécification :
  - Extraction des fonctions ;
  - Extraction des parties purement calculatoires des preuves.
- Plusieurs langages cibles : OCaml, Haskell, et Scheme.

Commandes Coq

- Extraction d'une constante : `Extraction nom` ;
- Extraction d'une constante et de toutes ses dépendances : `Recursive Extraction nom`.

Extraction simple d'une fonction

Fonction successeur

```
Coq < Definition succ (n : nat) : nat := S n.
succ is defined

Coq < Extraction succ.
(** val succ : nat -> nat **)
let succ n =
  S n
```

Extraction récursive d'une fonction

Fonction successeur

```
Coq < Definition succ (n : nat) : nat := S n.
succ is defined

Coq < Recursive Extraction succ.
type nat =
| O
| S of nat
(** val succ : nat -> nat **)
let succ n =
  S n
```

Extraction d'une preuve

Fonction double

```
Coq < Lemma double : forall n : nat, {v : nat | v = 2 * n}.
1 subgoal

  ============================
   forall n : nat, {v : nat | v = 2 * n}

Coq < intro.
1 subgoal

  n : nat
  ============================
   {v : nat | v = 2 * n}
```

Extraction d'une preuve

Fonction double

```
Coq < exists (2 * n).
1 subgoal

  n : nat
  ============================
   2 * n = 2 * n

Coq < reflexivity.
No more subgoals.

Coq < Defined.
intro.
exists (2 * n).
reflexivity.
double is defined
```

Extraction d'une preuve

Fonction double

```
Coq < Extraction double.
(** val double : nat -> nat **)
let double n =
  mult (S (S O)) n
```

Jouer avec l'isomorphisme de Curry-Howard

Fonction successeur

```
Coq < Definition succ (n : nat) : nat.
1 subgoal

  n : nat
  ============================
   nat

Coq < exact (S n).
No more subgoals.

Coq < Defined.
exact (S n).
succ is defined
```

Jouer avec l'isomorphisme de Curry-Howard

Fonction successeur

```
Coq < Print succ.
succ = fun n : nat => S n
     : nat -> nat
Argument scope is [nat_scope]

Coq < Eval compute in (succ 2).
     = 3
     : nat
```

Jouer avec l'isomorphisme de Curry-Howard

Fonction factorielle

```
Coq < Definition fact (n : nat) : nat.
1 subgoal

  n : nat
  ============================
   nat

Coq < elim n.
2 subgoals

  n : nat
  ============================
   nat
subgoal 2 is:
 nat -> nat -> nat
```

Jouer avec l'isomorphisme de Curry-Howard

Fonction factorielle

```
Coq < exact 1.
1 subgoal

  n : nat
  ============================
   nat -> nat -> nat

Coq < intros; exact ((S n0) * H).
No more subgoals.

Coq < Defined.
elim n.
 exact 1.
 intros; exact (S n0 * H).
fact is defined
```

Jouer avec l'isomorphisme de Curry-Howard

Fonction factorielle

```
Coq < Eval compute in (fact 3).
     = 6
     : nat
```

Exercices

Égalité sur les entiers naturels

- Démontrer le lemme de décidabilité de l'égalité sur les entiers naturels (utiliser le « ou constructif » : `{...} + {...}`) ;
- Extraire la fonction qui teste l'égalité sur les entiers naturels à partir du lemme précédent ;
- Tester la fonction OCaml extraite (si OCaml non installé sur les machines, utiliser : `https://try.ocamlpro.com/`).

Fonction factorielle

- Spécifier (avec un inductif) le comportement de la fonction factorielle ;
- Démontrer le lemme de définition et d'adéquation de factorielle ;
- Extraire la fonction factorielle du lemme précédent ;
- Tester la fonction OCaml extraite.

Exercices

Fonction « map » sur les listes d'entiers naturels

- Spécifier (avec un inductif) le comportement de la fonction « map » ;
- Démontrer le lemme de définition et d'adéquation de « map » ;
- Extraire la fonction « map » du lemme précédent ;
- Tester la fonction OCaml extraite.

Fonction d'addition sur les entiers naturels

- Définir la fonction d'addition avec des tactiques ;
- Tester la fonction ainsi définie.