

Construction de programmes à partir de preuves non constructives

Alexandre Miquel
U. Paris 7 (PPS) & ENS Lyon (LIP/Plume)

GDR-IM 22-23/01/2009

Le dictionnaire

- Sémantique de Brouwer-Heyting-Kolmogorov
- Correspondance de Curry-Howard

Théorie de la démonstration	Programmation fonctionnelle
Proposition (formule) Preuve (dérivation)	Type de données Programme (ou donnée)
$A \wedge B, A \vee B, A \Rightarrow B$	$A \times B, A + B, A \rightarrow B$
Règle de déduction Vérificateur de preuve	Règle de typage Vérificateur de type
Élimination des coupures Preuve sans coupure	Évaluation (calcul) Valeur
Preuve d'un lemme Théorie (énoncés et preuves)	Sous-programme Module (interface et implém.)

Conséquences (1/2)

Sur le plan théorique :

- Renouveau de la **théorie de la démonstration**
 \rightsquigarrow Dynamique calculatoire des démonstrations
- Sémantique : dénotationnelle / catégorique / des jeux (etc.)
- Logique linéaire [Girard 86]
- Systèmes de types très expressifs
 - Les systèmes $F, F\omega$ [Girard 70, 72]
 - La théorie des types [Martin-Löf 71, 79, 84]
 - Le calcul des constructions [Coquand & Huet 85]

Conséquences (2/2)

Sur le plan pratique :

- Vérification de type \rightsquigarrow **vérification de preuve**
- Assistants à la preuve basés sur ce principe :
 - Coq (Calcul des constructions)
 - Lego/Plastic (idem)
 - Agda/Alfa (Théorie des types)

Comment ça marche? (exemple 1)

- **Question :** qu'est-ce qui prouve

$$A \wedge B \Rightarrow B \wedge A ?$$

$$A \times B \rightarrow B \times A ?$$

Réponse : $\text{fun } (x, y) \mapsto (y, x)$

Comment ça marche? (exemple 2)

- **Question :** qu'est-ce qui prouve

$$(A \Rightarrow B) \wedge (B \Rightarrow C) \Rightarrow (A \Rightarrow C) ?$$

$$(A \rightarrow B) \times (B \rightarrow C) \rightarrow (A \rightarrow C) ?$$

Réponse : $\text{fun } (f, g) \mapsto \text{fun } x \mapsto g(f(x))$

Comment ça marche? (exemple 3)

- **Rappel :** $A \vee B \equiv A + B$
 $\equiv \text{Left of } A \mid \text{Right of } B$

- **Question :** qu'est-ce qui prouve

$$(A \vee B) \Rightarrow (B \vee A) ?$$

$$(A + B) \rightarrow (B + A) ?$$

Réponse : fonction
| $\text{Left}(x) \mapsto \text{Right}(x)$
| $\text{Right}(y) \mapsto \text{Left}(y)$

Constructivité de la logique intuitionniste (1/2)

Propriété de la disjonction

D'une preuve de $\vdash A \vee B$ (sans hypothèse), on peut extraire une preuve de $\vdash A$ ou une preuve de $\vdash B$

$$\begin{array}{c} \vdots p \\ \vdash A \vee B \end{array} \rightsquigarrow \begin{array}{c} \vdots p_1 \\ \vdash A \end{array} \text{ or } \begin{array}{c} \vdots p_2 \\ \vdash B \end{array}$$

- **Interprétation calculatoire :**

Un programme (clos) $p : A + B$ s'évalue :

- soit sur $\text{Left}(p_1)$ (avec $p_1 : A$),
- soit sur $\text{Right}(p_2)$ (avec $p_2 : B$)

Constructivité de la logique intuitionniste (2/2)

Propriété du témoin

D'une preuve de $\vdash \exists^{\mathbb{N}} x A(x)$ (sans hypothèse), on peut extraire un **témoin** t et une preuve de $\vdash A(t)$:

$$\begin{array}{c} \vdots p \\ \vdash \exists x A(x) \end{array} \rightsquigarrow t + \begin{array}{c} \vdots p' \\ \vdash A(t) \end{array}$$

• Interprétation calculatoire :

Un programme (clos) $p : \Sigma x : T. A(x)$ s'évalue sur un couple (t, p') tel que

- $t : T$ (type des individus), et
- $p' : A(t)$

Le tiers-exclu

• Question : comment prouver constructivement

$$\begin{array}{c} A \vee \neg A \\ A + (A \rightarrow \emptyset) \end{array}$$

- ... si A est le grand théorème de Fermat ?
- ... si A est la conjecture de Riemann ?
- ... si A est **indécidable** ? (ex. : hypothèse du continu)
- ... si **je n'ai pas la connaissance de A** ?

Extraction de programme

• Principe : Une preuve **constructive** de

$$(\forall x : T) (\exists y : U) A(x, y)$$

est une fonction (calculable) qui, appliquée à un $x : T$ arbitraire, retourne un couple **témoin/justification** (y, z) (avec $y : U, z : A(x, y)$)

• En pratique : On garde le témoin ; on jette la justification

\rightsquigarrow Nécessite d'**éliminer le code mort**

• Extraction de programmes dans Coq [Paulin 89, Letouzey 02]

- Distinction : Set, Type / Prop (contenu **informatif** / **non informatif**)
- L'extraction jette ce qui est dans Prop (non informatif)
- L'utilisateur décide de ce qui est informatif ou non (lors de la preuve)

Non constructivité de la logique classique

• Disjonction sans alternative :

- $A \vee \neg A$ ($A \equiv$ conj. de Riemann, hyp. du continu, etc.)
- $e + \pi$ transcendant $\vee e \times \pi$ transcendant

• Existentiel sans témoin :

- $(\exists x : \mathbb{N}) ((A \wedge x = 1) \vee (\neg A \wedge x = 0))$ (mêmes exemples pour A)

• Fonctions non calculables (oracles) :

- Fonction d'arrêt des machines de Turing :

$$(\forall x : \mathbb{N}) (\exists y : \mathbb{N}) ((\text{Halt}(x) \wedge y = 1) \vee (\neg \text{Halt}(x) \wedge y = 0))$$

- Principe du minimum ($T \neq \emptyset$) :

$$(\forall f : T \rightarrow \mathbb{N}) (\exists x : T) (\forall y : T) f(x) \leq f(y)$$

Logique classique et continuations

• La découverte fondamentale

[Felleisen & Griffin 90]

- L'opérateur de contrôle **call/cc** [Felleisen] a pour type :

$$((A \rightarrow B) \rightarrow A) \rightarrow A$$

- Côté logique [Griffin 90], il s'agit de la **loi de Peirce...**
... qui entraîne (constructivement) le tiers-exclu

- Logique classique = programmation par **continuations**
= méthode **essai/erreur**

• λ -calculs classiques

- $\lambda\mu$ [Parigot 92]
- λ -sym [Barbanera & Berardi 96]
- λ_c (call/cc) + **réalisabilité classique** [Krivine 94]
- $\tilde{\lambda}\mu\tilde{\mu}$ [Herbelin & Curien 00]

Tiers-exclu, le retour

- **Question :** comment prouver

$$A \vee \neg A$$
$$A + (A \rightarrow \emptyset)$$

Réponse : call-cc $\lambda k. \text{Right} (\lambda x. k (\text{Left}(x)))$

La réalisabilité classique

[Krivine 94]

- Un langage de **réalisateurs** : $\lambda_c = \lambda + \text{call/cc}$
- Processus = terme (**preuve**) + pile (**contre-preuve**)
- À chaque formule A on associe :
 - Un ensemble de piles $\|A\|$ (**valeur de fausseté**)
 - Un ensemble de termes $|A|$ (**valeur de vérité**)

Exemple : $\|A \Rightarrow B\| = |A| \cdot \|B\| = \{t \cdot \pi : t \in |A|, \pi \in \|B\|\}$

- Définition de la valeur de vérité $|A|$ par **orthogonalité** :

$$|A| = A^\perp = \{t \in \Lambda : \forall \pi \in \|A\| \ t \star \pi \in \perp\}$$

où \perp est un ensemble de processus, paramètre de la construction

Théorème d'adéquation

À toute preuve classique de A on sait associer un réalisateur $t \in |A|$

Le quantificateur existentiel classique

Le programme extrait à partir d'une preuve de $(\exists x : \mathbb{N}) A(x)$ s'évalue sur un couple **témoin/justification** (n, p) , avec $n \in \mathbb{N}$ et $p : A(n)$.

- En réalisabilité intuitionniste
 - On a la garantie que n vérifie $A(n)$
 - La justification $p : A(n)$ est donc superflue (code mort)
- En réalisabilité classique
 - Aucune garantie que n vérifie $A(n)$! (possibilité de **faux-témoin**)
 - La justification $p : A(n)$ n'offre aucune garantie... (**parapreuve**)
... mais contient le matériel nécessaire pour backtracker
- Solution pour obtenir des témoins fiables :
 - Interroger le témoin (procédure de test)
 - Répudier un faux-témoignage $p : A(n)$...
... en lui opposant une **réfutation** $p' : \neg A(n)$

Recherche de témoin par essai/erreur

- Les ingrédients :
 - Une preuve (classique) $p : (\exists x : \mathbb{N}) A(x)$
 - Une fonction de décision $d : \mathbb{N} \rightarrow \mathbb{B}$ du prédicat A
 - Une fonction de réfutation potentielle r , telle que $r n : \neg A(n)$ pour tout n tel que $\neg A(n)$
- L'algorithme :
 - 1 Extraire $n : \mathbb{N}$ et $p_0 : A(n)$ de la preuve p
 - 2 Avec la fonction d , tester si $A(n)$. Si oui, sortir.
 - 3 Sinon, construire une réfutation $r_0 := r n : \neg A(n)$
 - 4 Appliquer $r_0 : \neg A(n)$ à $p_0 : A(n)$ pour déduire \perp
(En pratique, cette dernière étape déclenche le backtrack)
- L'implémentation : $p (M (\lambda xy . d x (\text{stop } x) (r x y)))$

Lemme : la procédure converge en temps fini sur un témoin fiable

Autres problèmes

- **Problème de la spécification**
 - Étant donnée une formule A , quel est le comportement calculatoire commun à tous les réalisateurs de A ?
 - Exemple : la loi de Peirce et call/cc
- **Existence de réalisateurs**
 - Quelles formules vraies admettent des réalisateurs?
 - Exemple : formules arithmétiques (1er ordre)
- **Structure des modèles obtenus**
 - Chaque choix du paramètre \perp définit un modèle (en fait : une classe)
 - La plupart de ces modèles sont non standard

Exemple : le principe du minimum et ses conséquences

Principe du minimum

Si T est un type de données habité :

$$(\forall f : T \rightarrow \mathbb{N}) (\exists x : T) (\forall y : T) f(x) \leq f(y)$$

Preuve. Par l'absurde, avec une récurrence forte.

- **Remarque :** Pas de preuve intuitionniste (oracle)

Corollaire

$$(\forall f : \mathbb{N} \rightarrow \mathbb{N}) (\exists x : \mathbb{N}) f(x) \leq f(x + 7)$$

Plus généralement : $(\forall f, g : \mathbb{N} \rightarrow \mathbb{N}) (\exists x : \mathbb{N}) f(x) \leq f(g(x))$

Preuve. On prend le point x donné par le principe du minimum.

- **Remarque :** Il y a aussi une preuve intuitionniste, qui ne nous intéresse pas ici.

Extensions de la réalisabilité classique

Définie à l'origine pour l'**arithmétique classique du 2nd ordre**, la réalisabilité classique s'étend :

- À l'axiome du choix dépendant [Krivine 01]
 - "quote", horloge, compteur, etc.
- À la théorie des ensembles de Zermelo-Fraenkel [Krivine 99]
- Au calcul des constructions avec univers (Coq) [Miquel 07]
- Au forcing : *Existence d'un ultrafiltre sélectif sur \mathbb{N}* [Krivine 08]
- Aux logiques modales [Miquel 09]
- Décomposition du forcing?