En déduction modulo (et d'autres formalismes) pas d'axiome

$$0 + x = x$$

mais une règle de calcul

$$0 + x \to x$$

Désir de transformer le maximum d'axiomes en règles de calcul

Jusqu'où peut-on aller ?

Quel est le rôle de la confluence dans cette discussion ?

Intégrer le calcul et le raisonnement : jusqu'où peut-on aller ?

Gilles Dowek

INRIA-Rocquencourt

JFLA 2001

Les langages applicatifs dans les langages mathématiques

Langages de définition de fonctions

Langages de programmation : tous les prog. sont des fonctions

Langages mathématiques : il y a des fonctions en mathématiques

Un langage mathématique doit comporter un langage de définition de fonctions (un langage de prog. applicatif) comme sous-langage

N'y a-t-il que des langages de programmation applicatifs ?

Un langage de programmation se réduit-il à un mécanisme de définition de fonctions ?

Non : les programmeurs ont aussi besoin de traits impératifs, objets, ...

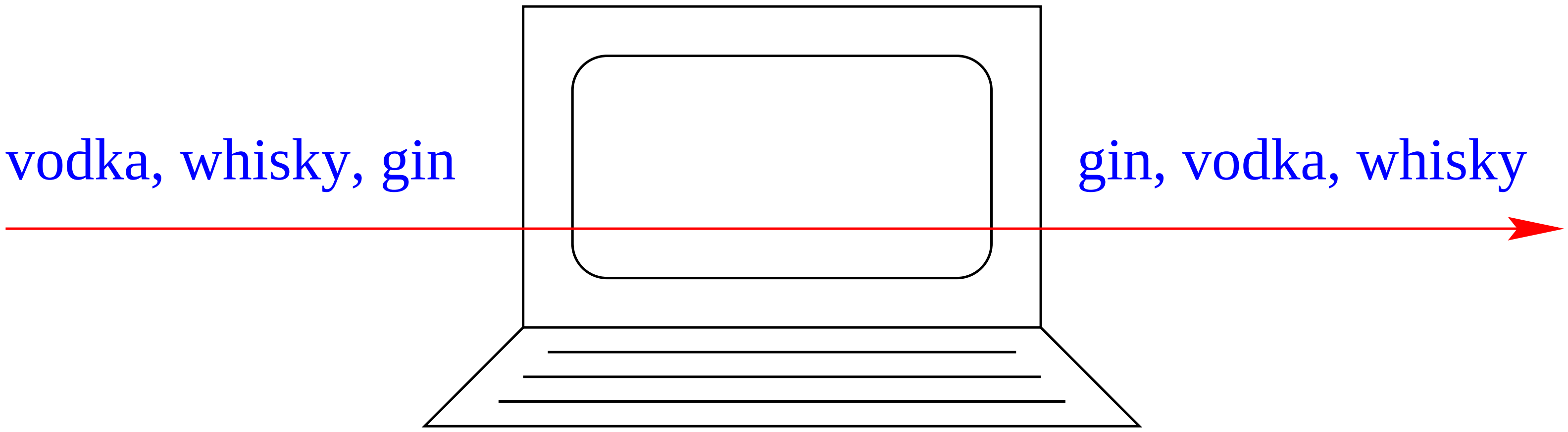
Exemple : algorithme de tri, 1000 objets, 998 objets invariants à chaque transposition

Plus efficace de ne pas recopier le tableau à chaque transposition

Conceptuellement plus simple de penser un tableau qui évolue dans le temps, qu'une suite de tableaux

Cependant ...

En programmant impérativement un algorithme de tri qu'avons nous fait d'autre que de définir une fonction ?



Un programme définit toujours une fonction du flot de symboles d'entrée vers le flot de symboles de sortie

Oui : tous les langages de programmation sont applicatifs

Un langage mathématique doit comporter un langage de programmation applicatif (c-à-d quelconque) comme sous-langage

Traits non fonctionnels : définitions plus simples dans les langages de programmation et dans les langages mathématiques

Pour définir une fonction de tri, il est plus simple pour l'informaticien de penser impérativement

Pourquoi cela serait-il différent pour le mathématicien ?

Lui aussi veut écrire

$x \mapsto \{y := 1; for\ i := 1\ to\ x\ do\ y := 2 * y; return\ y\}$

Pour en finir avec la sémantique

Traits non fonctionnels $\rightarrow$ nécessité de définir la signification d'un programme précisément (c-à-d mathématiquement)

Une relation $\hookrightarrow$ sur $\mathcal{P} \times \mathbb{N} \times \mathbb{N}$

Sémantique naturelle : $\hookrightarrow$ définie inductivement

Sémantique opérationnelle : $\hookrightarrow \;=\; \rightarrow^*$, $\rightarrow$ définie inductivement

Sémantique dénotationnelle : $p, a \hookrightarrow b \;=\; (\llbracket p \rrbracket(a) = b)$,
$\llbracket p \rrbracket$ définie par récurrence sur la structure de p

Dans tous les cas : ensemble $\mathcal{P}$ des programmes (suites de symboles, arbres, ...)

`fun x -> {y:=1; for i := 1 to x do y := 2 * y; return y}` n'est pas une fonction mais une suite de symboles (contrairement à $p, . \hookrightarrow .$)

On utilise le langage mathématique pour parler des programmes

Mais les prog. ne sont pas des expressions du langage math.

Alternative : étendre le langage math. avec `p`

Plus besoin de sémantique $f, a \hookrightarrow b = (f(a) = b)$

Un renoncement ?

Renonce-t-on à définir précisément la signification d'un programme ?

Non : en mathématiques aussi on doit expliquer la signification des symboles utilisés

Pas en leur associant des objets mathématiques avec $[\![\,]\!]$

$$[\![\texttt{fun x -> x}]\!] = x \mapsto x$$

$$[\![x \mapsto x]\!] = x \mapsto x$$

Mais en posant des axiomes

$$\forall x\ ((x \mapsto x)(x) = x)$$

proches de la sémantique opérationnelle

A-t-on les moyens de la dissymétrie ?

$p, a \hookrightarrow b$ s'écrit désormais $p(a) = b$

Axiome : $(fun\ x \to x)(y) = y$

Prédicat = symétrique

Pourtant dissymétrie essentielle entre $(fun\ x \to x)2$ et 2

Peut-on remplacer = par un prédicat (dissymétrique) $\to^*$?

C'est difficile

L'expression "$(fun\ x \to x)(2)$" désigne le même nombre que l'expression "2"

On veut donc pouvoir déduire $(fun\ x \to x)(2) \in 2\mathbb{N}$ de $2 \in 2\mathbb{N}$

Plus généralement axiome (de Leibniz)

$a \to^* b \Rightarrow (a \in E \Leftrightarrow b \in E)$

Mais avec $E = \{x \mid x \to^* a\}$

$a \to^* b \Rightarrow (a \to^* a \Leftrightarrow b \to^* a)$

$a \to^* b \Rightarrow b \to^* a$

Même difficulté avec “normal”

$a \to^* b \Rightarrow (a \in E \Leftrightarrow b \in E)$

E ensemble termes normaux

$a \to^* b \Rightarrow (a\ normal \Leftrightarrow b\ normal)$

Tous les termes normalisables sont normaux ?

Il faut qu'une porte ...

Tension entre la volonté

- de dire que l'exécution préserve les propriété des objets (l'exécution répond à la question posée)
- d'avoir des prédicats ($\rhd^*$, $normal$, ...) qui ne sont pas préservés par l'exécution

Être normal n'est pas une propriété du nombre 2, mais de l'expression "2"

Pour l'exprimer il faut ou bien de la réflexion ou bien limiter l'axiome de Leibniz aux ensembles extentionnels

$\rightarrow^*$ n'est pas un prédicat du langage mathématique

Sémantique : un plongement profond du langage de prog. dans le langage math. ($\mathcal{P}$, $\hookrightarrow$, ...)

- Plongement superficiel (extension du langage mathématique par les opérateurs du langage de programmation)
- plongement profond du langage mathématique dans lui-même (réflexion) si on veut exprimer des propriétés des expressions mathématiques (normal, ...).

La déduction modulo (enfin)

Raisonner sur des programmes

$$\forall x\ ((if\ \ x\ then\ 2\ else\ 4) \in 2\mathbb{N})$$

Par récurrence (cas) sur x

$$(if\ \ true\ then\ 2\ else\ 4) \in 2\mathbb{N} \wedge (if\ \ false\ then\ 2\ else\ 4) \in 2\mathbb{N}$$

Axiome : $(if\ \ true\ a\ b) = a$

Simulation de l'exécution pas à pas

On n'est pas des debuggers ...

L'exécution

Pas de prédicat $\longrightarrow^*$

Mais on peut identifier $(if\ true\ then\ 2\ else\ 4) \in 2\mathbb{N}$ et $2 \in 2\mathbb{N}$

Pas d'axiome

$$(if\ true\ a\ b) = a$$

ni

$$(if\ true\ a\ b) \longrightarrow^* a$$

Mais une règle de calcul

$$(if\ true\ a\ b) \longrightarrow a$$

$\longrightarrow$ extérieur au langage

Normalisation de $(if\ true\ then\ 2\ else\ 4) \in 2\mathbb{N}$

Une théorie n'est pas un ensemble d'axiomes,
mais un ensemble d'axiomes et de règles de calcul

Déjà dit (sous différentes formes) :

- démonstration automatique (unification AC, ...),
- calcul formel,
- théorie de la définition (de Bruijn, Plaisted, ...)
- lambda-calcul (Martin-Löf, Constructions, ...),
- réflexion,
- règles fold/unfold (λ) de Prawitz,
- logique d'ordre supérieur, ...
- ...

Réécrire les propositions

$$2 \in 2\mathbb{N} \wedge 4 \in 2\mathbb{N}$$

Si $. \in 2\mathbb{N}$ est défini algorithmiquement

$$\top \wedge \top$$

Si $. \in 2\mathbb{N}$ est défini par $. \in \mathbb{N} \wedge \exists y \in \mathbb{N}\ (. = 2 * y)$

$$2 \in \mathbb{N} \wedge \exists y \in \mathbb{N}\ (2 = 2 * y) \wedge 4 \in \mathbb{N} \wedge \exists y \in \mathbb{N}\ (4 = 2 * y)$$

$$x \in \{z \in \mathbb{N} \mid \exists y \in \mathbb{N}\ z = 2 * y\} \to x \in \mathbb{N} \wedge \exists y \in \mathbb{N}\ (x = 2 * y)$$

Jusqu'où peut-on aller ?

Premier cas : dans les théories équationnelles

Trois règles de réécriture

$$a \leftarrow b \rightarrow c \leftarrow d$$

Peut-on aller si loin ?

Non : $a = d$ conséquence équationnelle

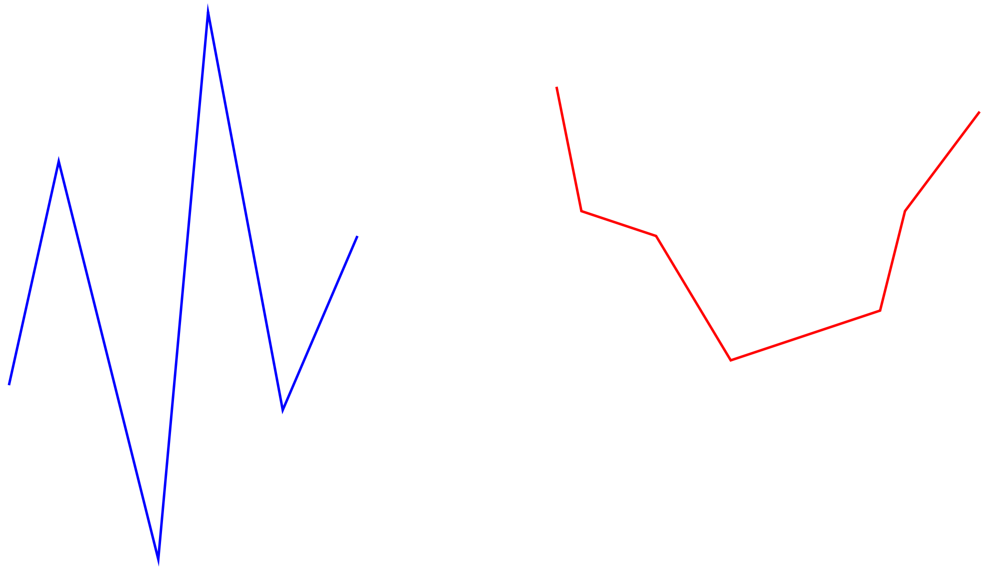
Mais a et d ne se calculent pas sur un terme commun

Utilisation à rebrousse poil de la règle $b \rightarrow a$

Contradictoire avec l'idée de règle de calcul

Confluence

Pour établir $t = u$ il suffit de trouver un réduit commun



Généralisation de cette idée au calcul des prédicats : confluence ?

Non

Trois règles de réécriture

$$a \leftarrow b \rightarrow c \leftarrow d$$

Est-il nécessaire d'utiliser la règle $b \rightarrow a$ à rebrousse poil pour établir $a = d$?

Pas vraiment

$$\dfrac{\dfrac{\vdash c = c}{\vdash b = d} \qquad \dfrac{a = d \vdash a = d}{b = d \vdash a = d}}{\vdash a = d}$$

Trois règles de réécriture

$$a \leftarrow b \rightarrow c \leftarrow d$$

Est-il nécessaire d'utiliser la règle $b \rightarrow a$ à rebrousse poil pour établir $a = d$?

Pas vraiment

$$\dfrac{\dfrac{\vdash c = c}{\vdash b = d}\,\text{réduction} \qquad \dfrac{a = d \vdash a = d}{b = d \vdash a = d}}{\vdash a = d}$$

Trois règles de réécriture

$$a \leftarrow b \rightarrow c \leftarrow d$$

Est-il nécessaire d'utiliser la règle $b \rightarrow a$ à rebrousse poil pour établir $a = d$?

Pas vraiment

$$\dfrac{\dfrac{\vdash c = c}{\vdash b = d}\,\text{réduction} \qquad \dfrac{a = d \vdash a = d}{b = d \vdash a = d}\,\text{réduction}}{\vdash a = d}$$

Trois règles de réécriture

$$a \leftarrow b \rightarrow c \leftarrow d$$

Est-il nécessaire d'utiliser la règle $b \rightarrow a$ à rebrousse poil pour établir $a = d$?

Pas vraiment

$$\dfrac{\dfrac{\vdash c = c}{\vdash b = d}\,\text{réduction} \qquad \dfrac{a = d \vdash a = d}{b = d \vdash a = d}\,\text{réduction}}{\vdash a = d}\,\text{coupure}$$

Trois règles de réécriture

$$a \leftarrow b \rightarrow c \leftarrow d$$

Est-il nécessaire d'utiliser la règle $b \rightarrow a$ à rebrousse poil pour établir $a = d$?

Pas vraiment

$$\dfrac{\dfrac{\vdash c = c}{\vdash b = d}\,\text{réduction} \qquad \dfrac{a = d \vdash a = d}{b = d \vdash a = d}\,\text{réduction}}{\vdash a = d}\,\text{coupure}$$

Mais injection d'une information : choix de $b = d$

Ma thèse (du jour)

Ce qui caractérise le calcul

- n'est pas l'orientation *en soi* (confluence)
- mais l'abscence de nécessité d'injecter de l'information (élimination des coupures)

Pour les théories équationnelles : les deux critères coïncident

Tranformation d'une preuve en preuve-vallée : processus d'élimination des coupures ?

Confluence et élimination des coupures

L'élimination des coupures traditionnellement rapprochée de la terminaison (normalisation)

Liens entre la confluence et l'élimination des coupures : deux formes de preuves de cohérence (indépendance)

$\mathcal{T}$ présentable par un système de réécriture confluent et deux termes normaux $\mathcal{T} \not\vdash t = u$

Complétion : preuve de cohérence

Dans les théories équationnelles : coïncidence entre confluence et élimination des coupures

Et dans le calcul des prédicats ?

Non

$$A \to B \wedge \neg A$$

Confluent

$$\dfrac{\dfrac{\dfrac{\dfrac{}{B \vdash B} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A, B \vdash A}}{A, B, \neg A \vdash}}{A, A \vdash}}{A \vdash}}{\vdash \neg A}}{B \vdash \neg A}}{B \vdash A} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{A, B \vdash A}}{A, B, \neg A \vdash}}{A, A \vdash}}{A \vdash}}{B \vdash} \ \text{coupure}}{\vdash \neg B}$$

$A \to B \wedge \neg A$ ne termine pas

Mais il y a également des systèmes confluents et terminants ne marchent pas

$$R \in R \to \forall y \; ((\forall x \; (\neg x \in R \Rightarrow \neg x \in y)) \Rightarrow \neg R \in y)$$

La confluence v.s. l'élimination des coupures

Si le calcul c'est l'analyticité la confluence ne suffit pas (sauf dans le cas particulier des théories équationnelles)

Confluence : forme spéciale de l'élimination des coupures dans le cas des théories équationnelles

Jusqu'où peut-on aller ?

Peu probable que l'élimination des coupures soit liée à des propriétés combinatoires simples des systèmes de réécriture : nouvelle propriété des systèmes de réécriture

Nombreuses preuves pour des cas particulier (`this` & Werner, Werner, Jouannaud et Okada, Fernandez, Walukiewicz, Blanqui, Stuber, ...)

Comment construire des systèmes de réécriture qui ont la propriété d'élimination des coupures (complétion) ?