Forcing et réalisabilité classique

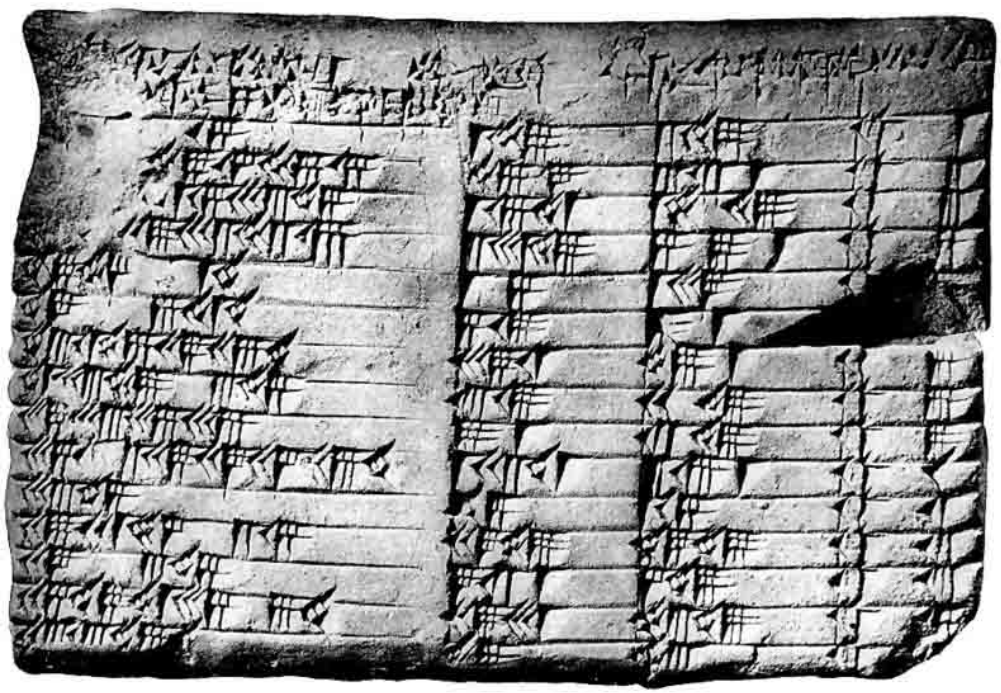
Lionel Rieg

Plume — LIP — ENS de Lyon
CPR — CÉDRIC — CNAM/ENSIIE

Thèse encadrée par Alexandre Miquel
17 juin 2014

Histoire des mathématiques : calcul et démonstration

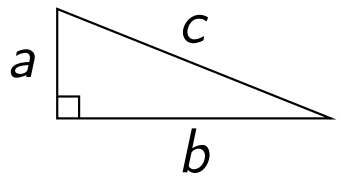
-3000 Multiplication, résolution d'équations quadratiques

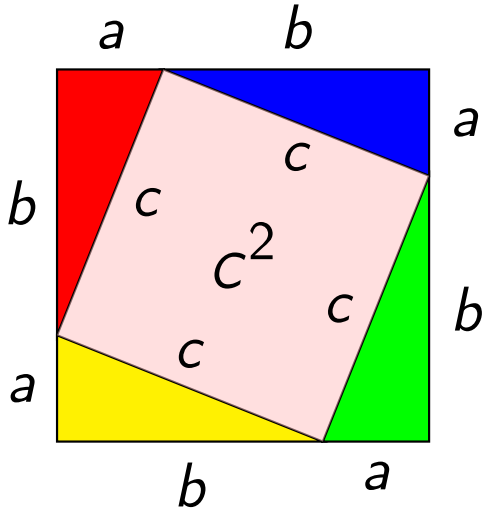


-3000 Multiplication, résolution d'équations quadratiques

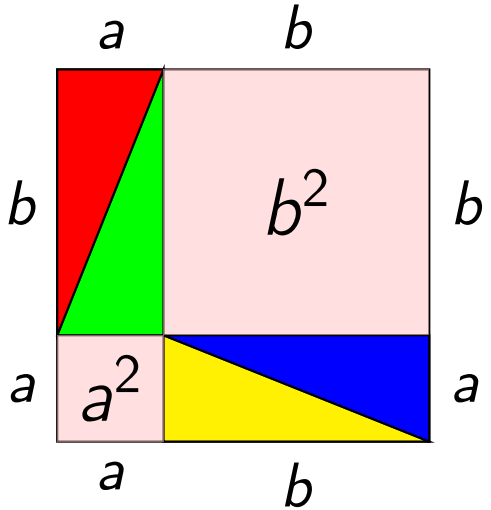
-500 Théorème de Pythagore

$\rightsquigarrow$ Arrivée des démonstrations





$$c^2 = a^2 + b^2$$

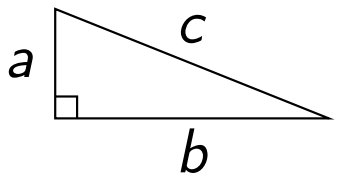


Histoire des mathématiques : calcul et démonstration

-3000 Multiplication, résolution d'équations quadratiques

-500 Théorème de Pythagore

$\rightsquigarrow$ Arrivée des démonstrations



XIX^{e} Structures algébriques

$\rightsquigarrow$ Axiomatisation des entiers naturels par Peano

$0, s$ (successeur) $\quad \forall n \forall m.\ n + 0 = n$

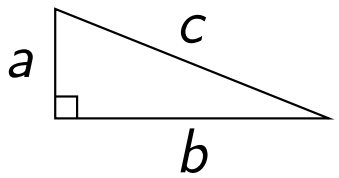
$2 = s(s(0)) \quad \forall n \forall m.\ n + s(m) = s(n + m)$

Histoire des mathématiques : calcul et démonstration

-3000 Multiplication, résolution d'équations quadratiques

-500 Théorème de Pythagore

$\rightsquigarrow$ Arrivée des démonstrations



XIX^{e} Structures algébriques

$\rightsquigarrow$ Axiomatisation des entiers naturels par Peano

XX^{e} Retour du calcul [Hilbert00]

- Entscheidungsproblem : mécanisation de la vérité [Turing36]
- Correspondance de Curry-Howard [Curry58,Howard69]

  $\rightsquigarrow$ Preuves = objets calculatoires

Qu'est-ce que le contenu calculatoire ?

1. Preuves : *quel contenu calculatoire ?*
   quelle notion de calcul ?
   $\rightsquigarrow$ Curry-Howard et réalisabilité

2. *Et pour le forcing ?*
   Contenu calculatoire d'une traduction logique
   $\rightsquigarrow$ Curry-Howard pour forcing

3. Illustration de Curry-Howard pour le forcing
   $\rightsquigarrow$ arbres de Herbrand
   $\rightsquigarrow$ nouvelle preuve du théorème de Herbrand

Le λ-calcul

[Church36]

Représenter le calcul :

- λ-calcul
- Fonctions récursives
- Machines de Turing

Représenter le calcul :

- λ-calcul
- Fonctions récursives
- Machines de Turing

λ-calcul = tout est fonction

mathématiques	λ-calcul
$x \mapsto \mathfrak{e}$	$\lambda x.\, \mathfrak{e}$
$f(x)$	$f\ x$

Exemples :

- $x \mapsto x + 3$ $\qquad$ $\lambda x.\, x + 3$
- décalage d'une fonction :
  $f \mapsto x \mapsto f(x + 2)$ $\qquad$ $\lambda f.\, \lambda x.\, f\ (x + 2)$

Calcul et expressivité

Calcul par β-réduction : $(\lambda x.\, M)\, N \to_\beta M[N/x]$

$$(\lambda x.\, x + 3)\, 2 \to_\beta 2 + 3$$

Calcul et expressivité

Calcul par β-réduction : $(\lambda x.\, M)\, N \to_\beta M[N/x]$

$$(\lambda x.\, x + 3)\, 2 \to_\beta 2 + 3$$

Entiers de Church $\overline{n} := \lambda x.\, \lambda f.\, \underbrace{f\, (f \ldots (f}_{n}\, x) \ldots)$

$\overline{0} := \lambda x.\, \lambda f.\, x$
$\overline{1} := \lambda x.\, \lambda f.\, f\, x$
$\overline{2} := \lambda x.\, \lambda f.\, f\, (f\, x)$
$\vdots$

Calcul et expressivité

Calcul par β-réduction : $(\lambda x.\, M)\, N \to_\beta M[N/x]$

$$(\lambda x.\, x + 3)\, 2 \to_\beta 2 + 3$$

Entiers de Church $\overline{n} := \lambda x.\, \lambda f.\, \underbrace{f\, (f \ldots (f}_{n}\, x) \ldots)$

$\overline{0} := \lambda x.\, \lambda f.\, x$
$\overline{1} := \lambda x.\, \lambda f.\, f\, x$
$\overline{2} := \lambda x.\, \lambda f.\, f\, (f\, x)$
$\vdots$

successeur $\overline{s} := \lambda n.\, \lambda x.\, \lambda f.\, f\, (n\, x\, f)$
addition, multiplication, ...

λ-calcul = fondement de la programmation fonctionnelle

Correspondance de Curry-Howard

Lien preuves formelles / programmes fonctionnels

Logique	**Programmes**
formule	type $(nat \to nat)$
preuve	programme $(\lambda x.\, x + 3)$

Correspondance de Curry-Howard

Lien preuves formelles / programmes fonctionnels

Logique	**Programmes**
formule	type $(nat \to nat)$
preuve	programme $(\lambda x.\, x + 3)$
calcul	évaluation (β-réduction)
vérification de preuves	vérification de types
règle d'inférence	construction de programmation
($\Rightarrow_i$, modus ponens)	(λ, App)
connecteurs ($\wedge$,$\vee$, $\Rightarrow$)	constructeurs de types ($\times$, $+$, $\to$)
récurrence	programme récursif

Exemple : transitivité de l'implication

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash B \Rightarrow C}\,\mathrm{Ax} \qquad \dfrac{\dfrac{}{\Gamma \vdash A \Rightarrow B}\,\mathrm{Ax} \qquad \dfrac{}{\Gamma \vdash A}\,\mathrm{Ax}}{\Gamma \vdash B}\Rightarrow_e}{A \Rightarrow B, B \Rightarrow C, A \vdash C}\Rightarrow_e}{A \Rightarrow B, B \Rightarrow C \vdash A \Rightarrow C}\Rightarrow_i}{A \Rightarrow B \vdash (B \Rightarrow C) \Rightarrow A \Rightarrow C}\Rightarrow_i}{\vdash (A \Rightarrow B) \Rightarrow (B \Rightarrow C) \Rightarrow A \Rightarrow C}\Rightarrow_i$$

avec $\Gamma := A \Rightarrow B, B \Rightarrow C, A$

$$\lambda f.\, \lambda g.\, \lambda x.\, g\,(f\,x) \;:\; (A \Rightarrow B) \Rightarrow (B \Rightarrow C) \Rightarrow A \Rightarrow C$$

Exemple : transitivité de l'implication

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash B \Rightarrow C}\ g\ \mathrm{Ax} \qquad \dfrac{\dfrac{}{\Gamma \vdash A \Rightarrow B}\ f\ \mathrm{Ax} \qquad \dfrac{}{\Gamma \vdash A}\ \mathrm{Ax}\ x}{\Gamma \vdash B}\Rightarrow_e \text{(App)}}{A \Rightarrow B, B \Rightarrow C, A \vdash C}\Rightarrow_e \text{(App)}}{A \Rightarrow B, B \Rightarrow C \vdash A \Rightarrow C}\Rightarrow_i \lambda x.}{A \Rightarrow B \vdash (B \Rightarrow C) \Rightarrow A \Rightarrow C}\Rightarrow_i \lambda g.}{\vdash (A \Rightarrow B) \Rightarrow (B \Rightarrow C) \Rightarrow A \Rightarrow C}\Rightarrow_i \lambda f.$$

avec $\Gamma := A \Rightarrow B, B \Rightarrow C, A$

$$\lambda f.\, \lambda g.\, \lambda x.\, g\,(f\,x) \;:\; (A \Rightarrow B) \Rightarrow (B \Rightarrow C) \Rightarrow A \Rightarrow C$$

Que fait cette fonction ?

Théorie des modèles pour Curry-Howard

$\rightsquigarrow$ relation entre programmes et formules : $t \Vdash A$

Fondée sur calcul plutôt que code

- $+$ significatif
- $+$ expressif (if $0 < 1$ then 0 else true)
- Contient typage (adéquation)

Extraction de témoin

passer de $t \Vdash \exists x.\, A(x)$ à n tel que $A(n)$ vrai (réalisable)

Limité à la logique constructive ?

Théorie des modèles pour Curry-Howard

$\rightsquigarrow$ relation entre programmes et formules : $t \Vdash A$

Fondée sur calcul plutôt que code

- $+$ significatif
- $+$ expressif (if $0 < 1$ then 0 else true)
- Contient typage (adéquation)

Extraction de témoin

passer de $t \Vdash \exists x.\, A(x)$ à n tel que $A(n)$ vrai (réalisable)

Limité à la logique constructive ?

Non, ajouter un principe classique : [Griffin90]

$$\texttt{callcc} \ : \ ((A \Rightarrow B) \Rightarrow A) \Rightarrow A \quad \text{(Peirce)}$$

Exemple : le tiers exclu $(P \vee \neg P)$

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \neg P \Rightarrow (P \vee \neg P)}\vee_i^2 \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, P \vdash (P \vee \neg P) \Rightarrow \bot}\text{Ax} \qquad \dfrac{\dfrac{}{\Gamma, P \vdash P \Rightarrow (P \vee \neg P)}\vee_i^1 \qquad \dfrac{}{\Gamma, P \vdash P}\text{Ax}}{\Gamma, P \vdash P \vee \neg P}\Rightarrow_e}{\Gamma, P \vdash \bot}\Rightarrow_e}{\Gamma \vdash \neg P}\Rightarrow_i}{(P \vee \neg P) \Rightarrow \bot \vdash P \vee \neg P}\Rightarrow_e \qquad \dfrac{\dfrac{}{\ldots}\text{Peirce}}{\vdash ((P \vee \neg P) \Rightarrow \bot) \Rightarrow (P \vee \neg P)}\Rightarrow_i}{\vdash P \vee \neg P}\Rightarrow_e$$

avec $\Gamma := P \vee \neg P \Rightarrow \bot$

$$\texttt{callcc}\,(\lambda k.\, \mathit{right}\,(\lambda x.\, k\,(\mathit{left}\, x))) \;:\; P \vee \neg P$$

Exemple : le tiers exclu $(P \vee \neg P)$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, P \vdash (P \vee \neg P) \Rightarrow \bot}\,k\ \mathrm{Ax}}{\Gamma, P \vdash \bot}\Rightarrow_e\ (\mathrm{App}) \quad \dfrac{\dfrac{}{\Gamma, P \vdash P \Rightarrow (P \vee \neg P)}\,\mathit{left}\ \vee_i^1}{\Gamma, P \vdash P \vee \neg P}\Rightarrow_e\ (\mathrm{App}) \quad \dfrac{}{\Gamma, P \vdash P}\,\mathrm{Ax}\ x}{\Gamma \vdash \neg P}\Rightarrow_i\ \lambda x.}{(P \vee \neg P) \Rightarrow \bot \vdash P \vee \neg P}\Rightarrow_e\ (\mathrm{App}) \quad \dfrac{\dfrac{}{\Gamma \vdash \neg P \Rightarrow (P \vee \neg P)}\,\mathit{right}\ \vee_i^2}{\vdash ((P \vee \neg P) \Rightarrow \bot) \Rightarrow (P \vee \neg P)}\Rightarrow_i\ \lambda k.}{\vdash P \vee \neg P}\Rightarrow_e\ (\mathrm{App})$$

$\mathtt{callcc}$ Peirce $\dfrac{}{\ldots}$

avec $\Gamma := P \vee \neg P \Rightarrow \bot$

$$\mathtt{callcc}\,(\lambda k.\, \mathit{right}\,(\lambda x.\, k\,(\mathit{left}\, x))) \;:\; P \vee \neg P$$

Backtrack !

Quid pour la sémantique ?

Réalisabilité à la Krivine

[Krivine94]

Reformulation réalisabilité dans un cadre classique
$\rightsquigarrow$ `callcc` = backtrack : nouvelle évaluation (KAM)

Paramétrée par un pôle $\perp\!\!\!\perp$
$\rightsquigarrow$ propriété à observer

Réalisateurs pour $\perp$ (False)

- $\perp$ pas interprété par $\emptyset$
- $\neg A$ conserve un sens calculatoire

Extraction de témoin

- cas général : témoins pas fiables (backtrack)
- fiables pour $\forall \vec{x}.\, \exists \vec{y}.\, A(\vec{x}, \vec{y})$ (formules Π_2^0)

Exemple d'évaluation : le principe du minimum

$f \ : \ nat \to nat$ [Berardi,Coquand,Miquel et al]

Principe du minimum : $\exists x. \forall y. f(x) \leq f(y)$ (non extractible)

$\Downarrow$

Corollaire : $\exists x. f(x) \leq f(3x+1)$ (extractible)

Exemple : $f := x \mapsto |x - 1000|$

x	$3x+1$	$f(x)$	$f(3x+1)$	backtrack ?
0	1	1000	999	✓
1	4	999	996	✓
4	13	996	987	✓
13	40	987	960	✓
40	121	960	879	✓
121	364	879	636	✓
364	1093	636	93	✓
1093	3280	93	2280	X

Mes contributions en réalisabilité classique

Réalisabilité classique comme langage de programmation

Types de données primitifs en réalisabilité classique :

- généralisation des nombres (entiers, rationnels)
- listes, arbres...

Nombres réels en réalisabilité classique

- cadre uniforme efficace pour tous les réels
- extraction sur les réels calculables (Σ^0_1)

Formalisation Coq de la réalisabilité classique (assistant de preuves)
$\rightsquigarrow$ avec ses extensions
$\rightsquigarrow$ utile pour vérifier des réalisateurs

Réalisabilité classique et forcing

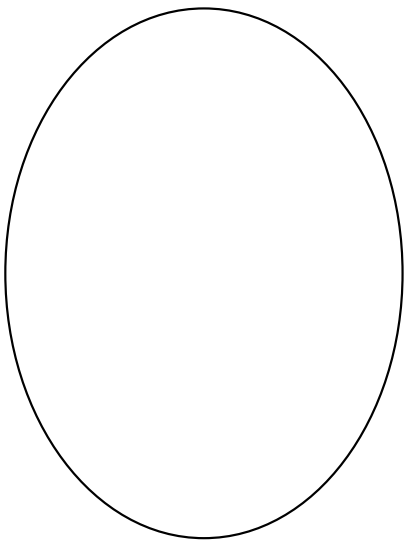
Qu'est-ce que le forcing ?

[Cohen63]

- Inventé par Paul Cohen en 63
  $\rightsquigarrow$ indépendance hypothèse continu $(\mathrm{CH} : 2^{\aleph_0} = \aleph_1)$
- But : ajouter des objets idéaux
  $\rightsquigarrow$ généralisation des extensions de corps
  $\rightsquigarrow$ filtre générique $\approx$ nouvelle racine
- Nouveaux sous-ensembles aux ensembles infinis
  $\rightsquigarrow$ pour casser CH : $\aleph_2$ nouveaux sous-ensembles de ω
- Modèles plus larges de ZF

Le forcing en dessin

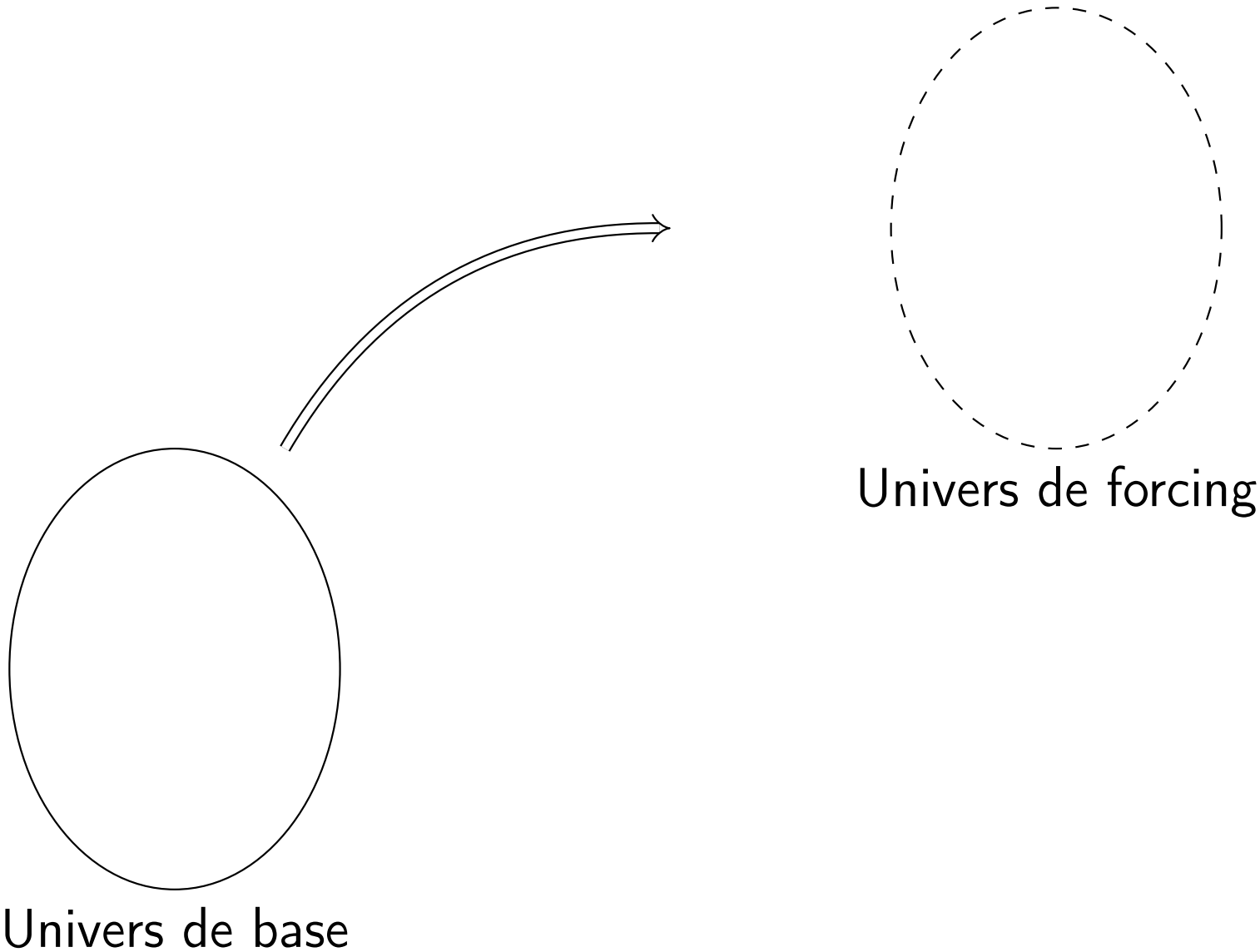
[Cohen63]



Univers de base

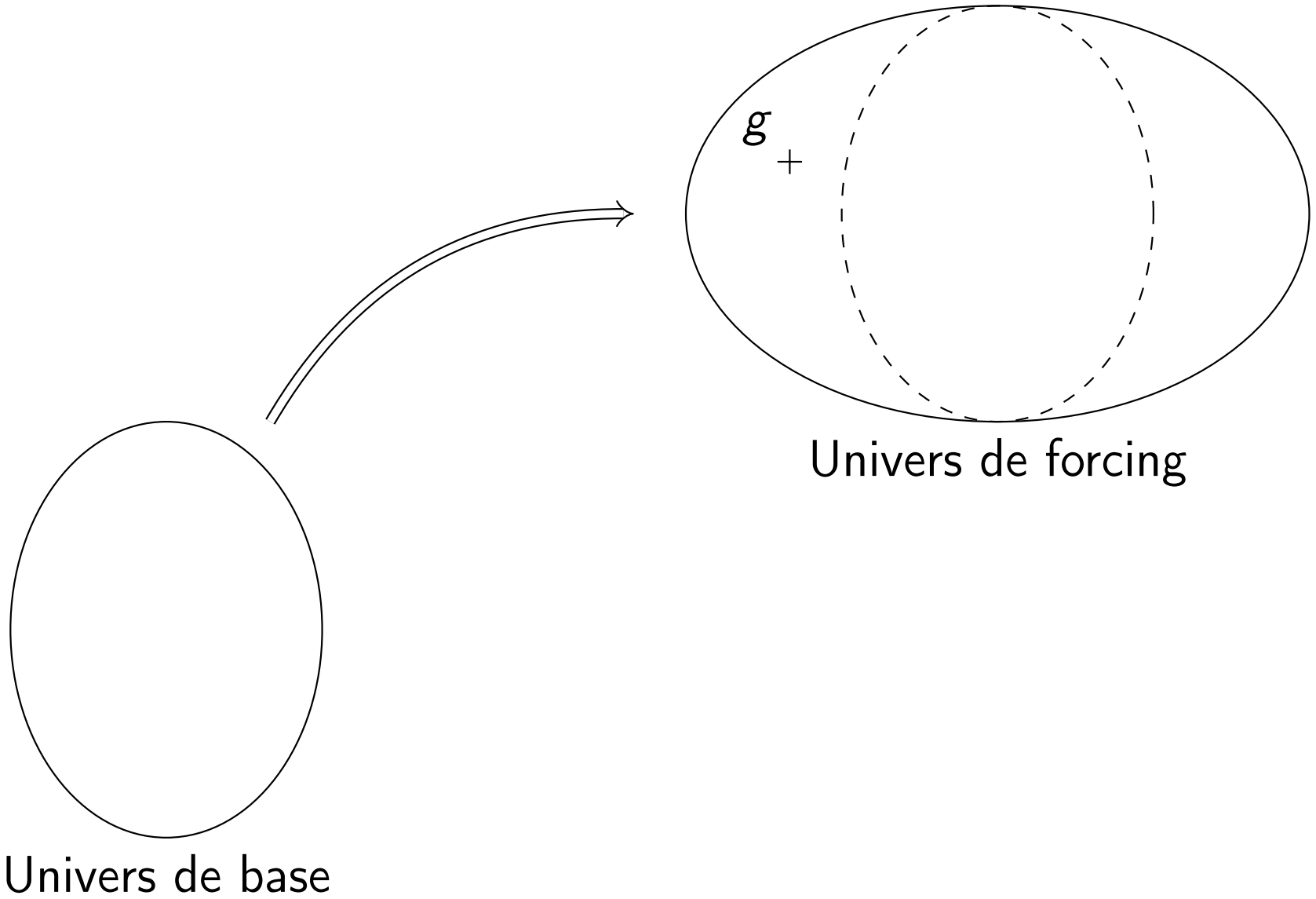
Le forcing en dessin

[Cohen63]



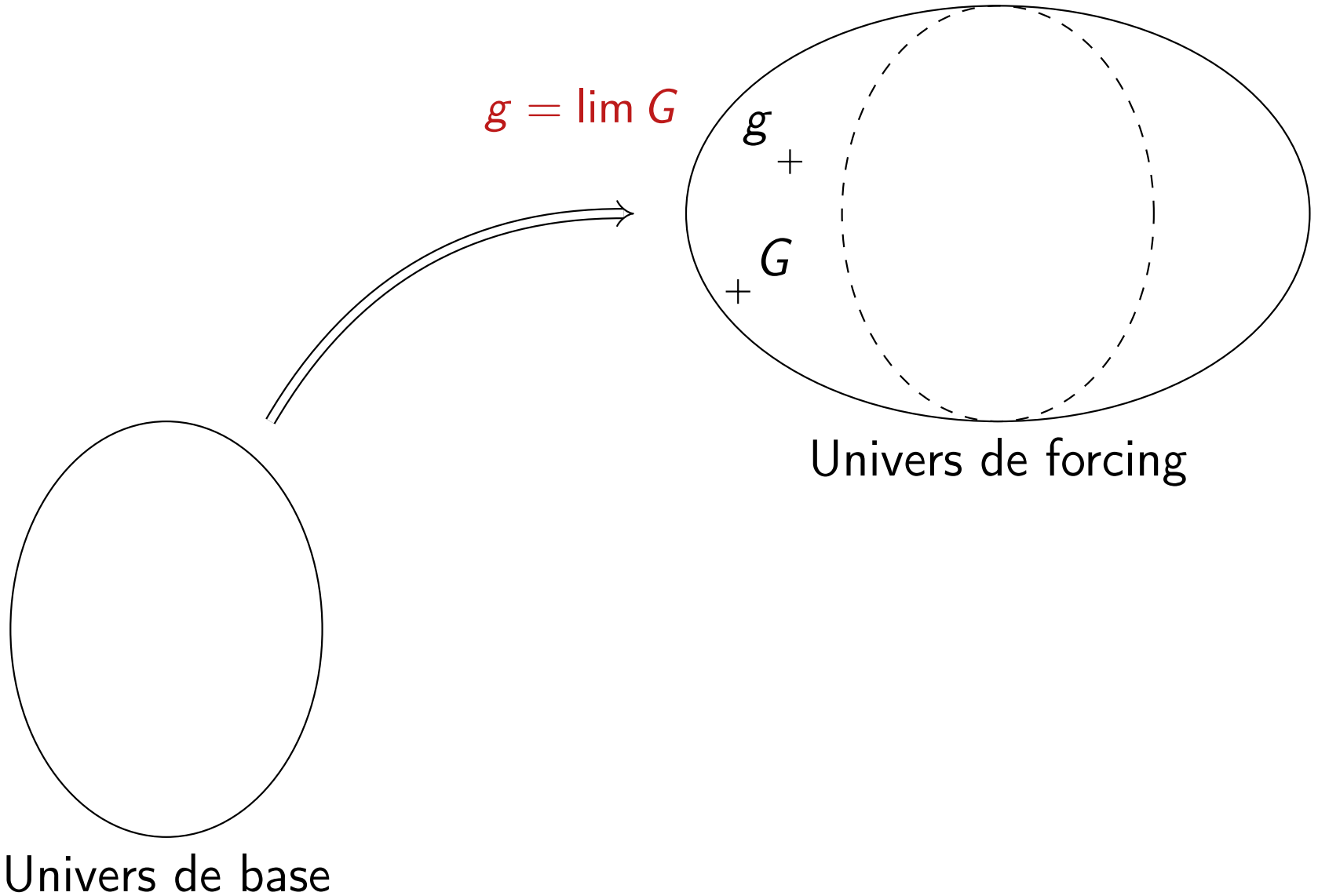
Le forcing en dessin

[Cohen63]



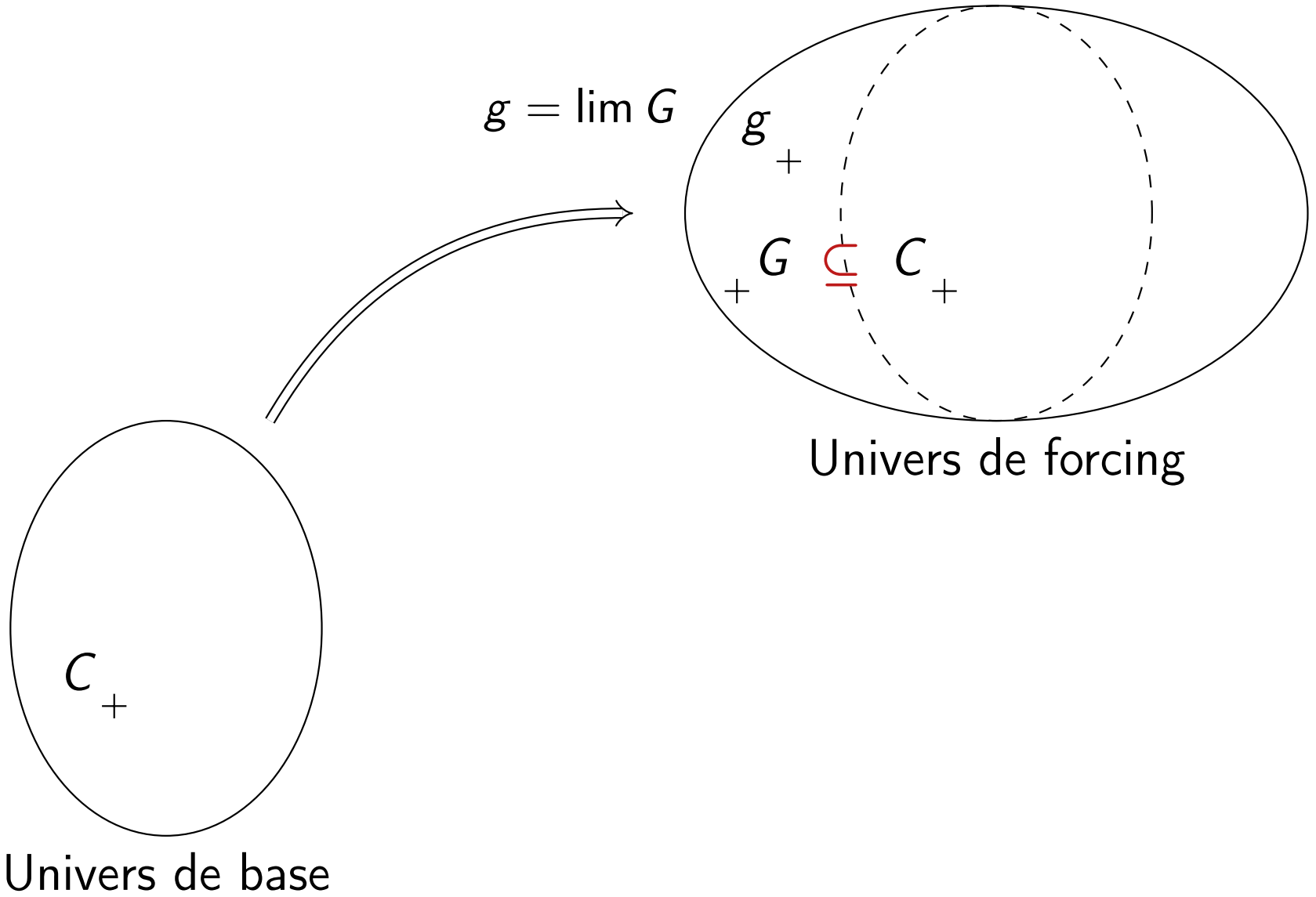
Le forcing en dessin

[Cohen63]



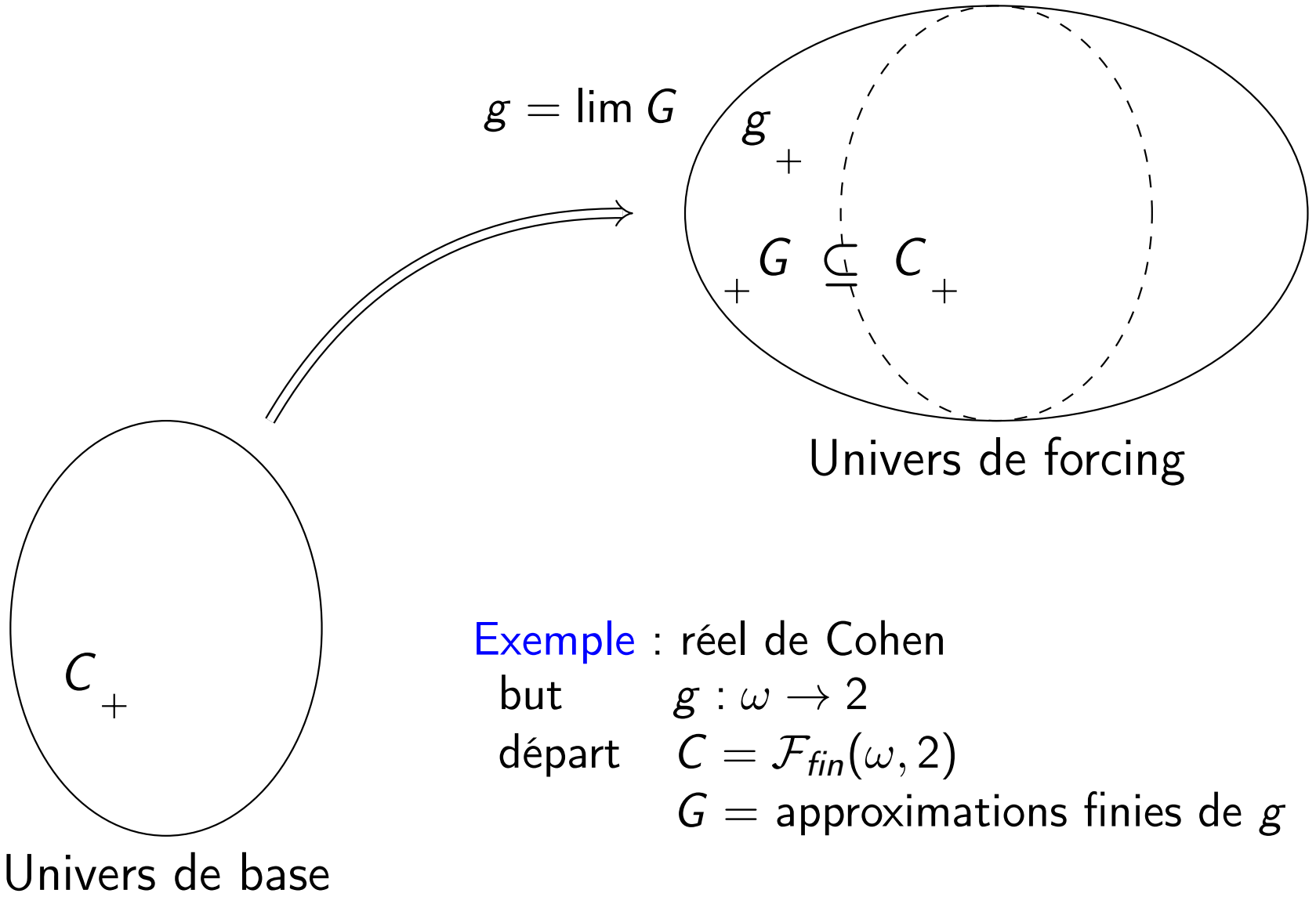
Le forcing en dessin

[Cohen63]



Le forcing en dessin

[Cohen63]

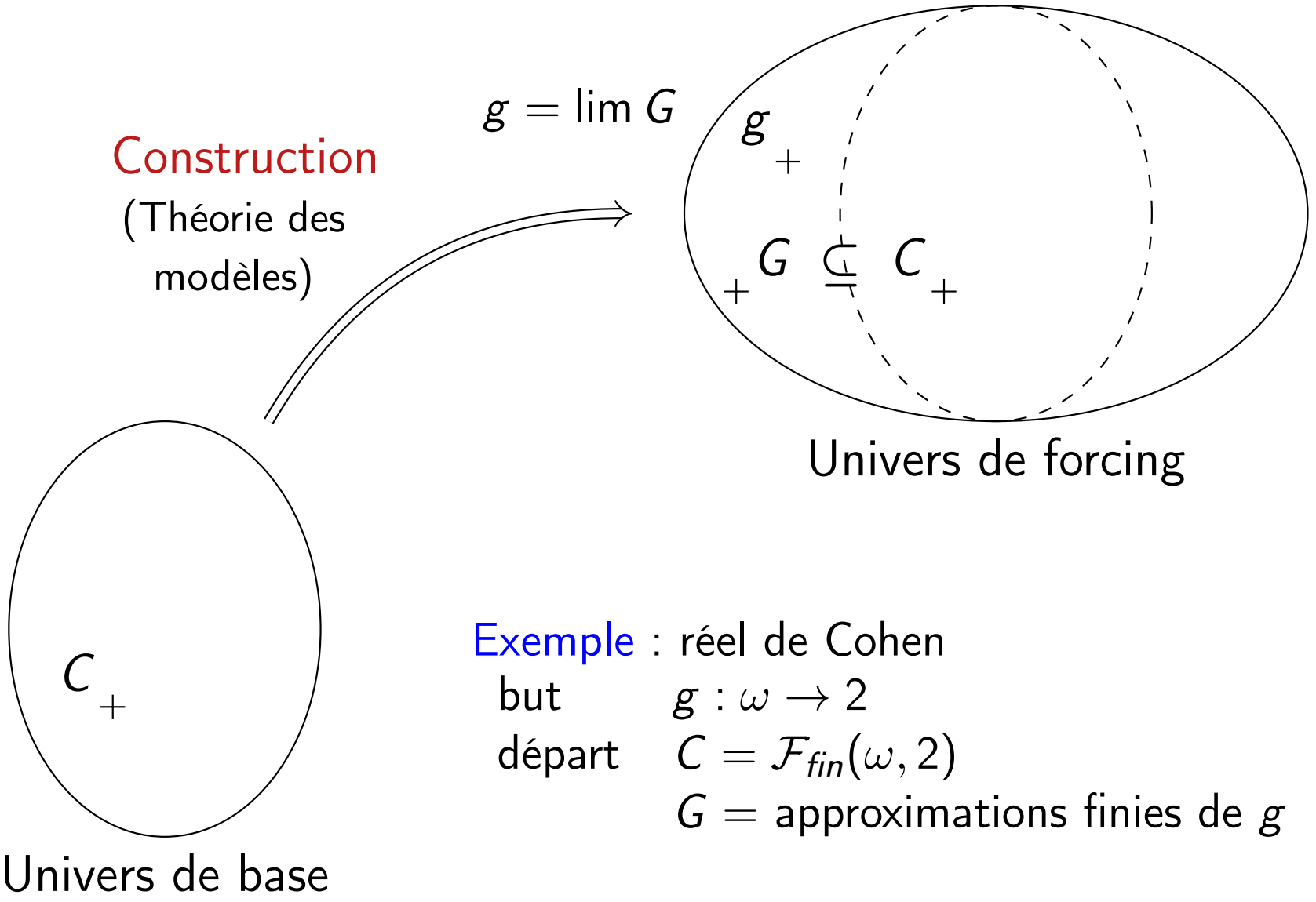


Exemple : réel de Cohen

but	$g : \omega \to 2$
départ	$C = \mathcal{F}_{fin}(\omega, 2)$
	$G =$ approximations finies de g

Le forcing en dessin

[Cohen63]

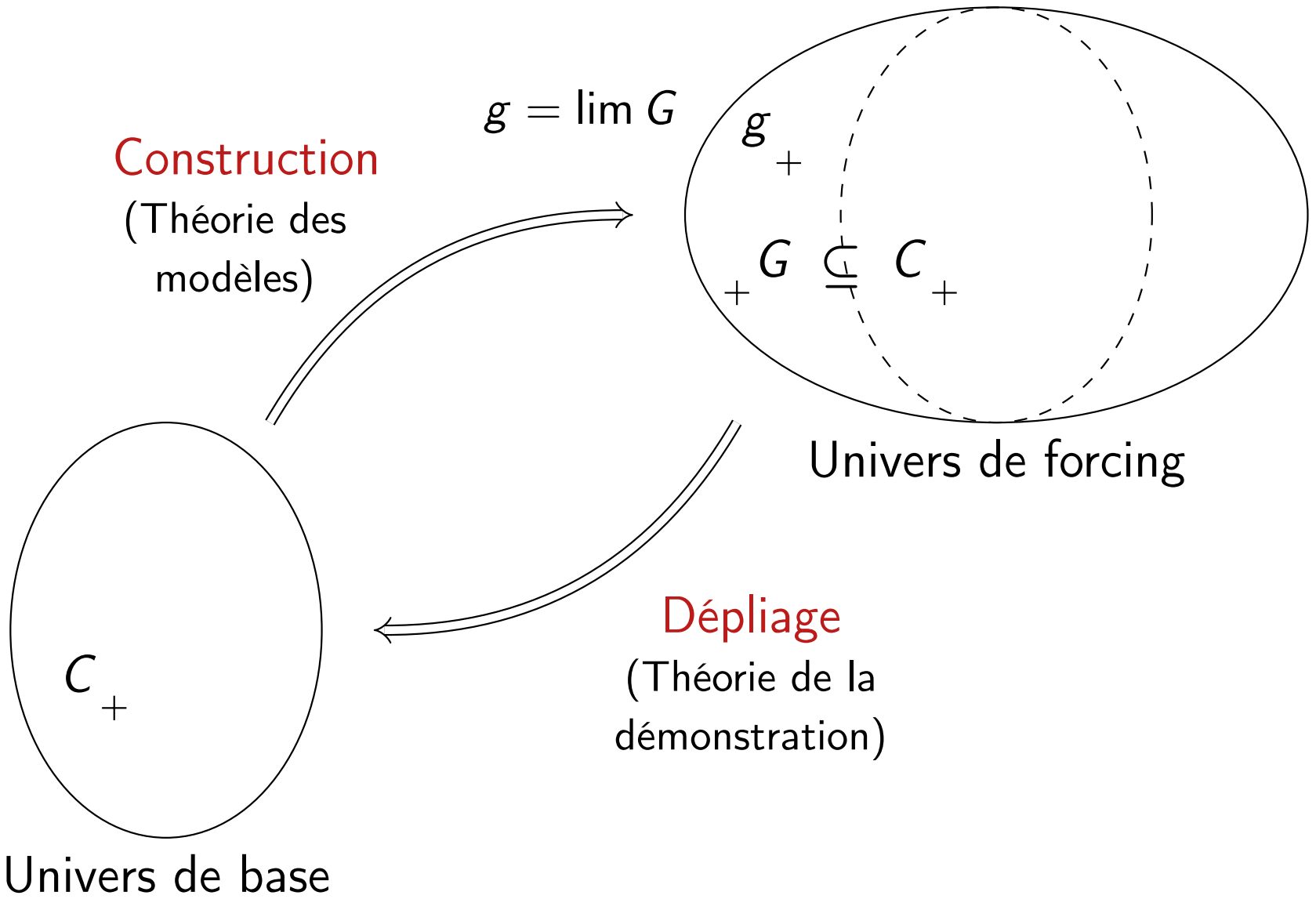


Exemple : réel de Cohen

but	$g : \omega \to 2$
départ	$C = \mathcal{F}_{fin}(\omega, 2)$
	$G =$ approximations finies de g

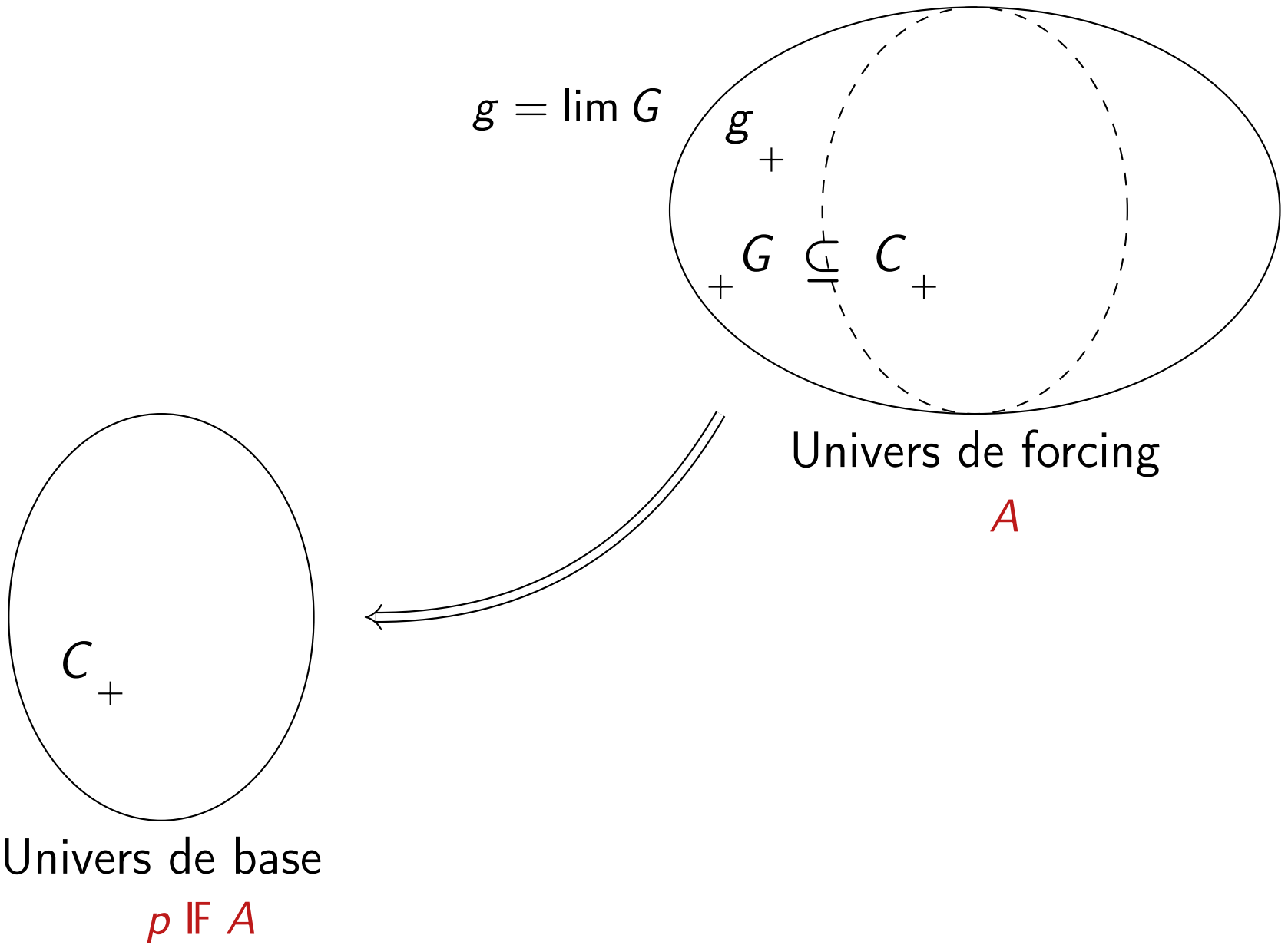
Le forcing en dessin

[Cohen63]



Le forcing en dessin

[Cohen63]



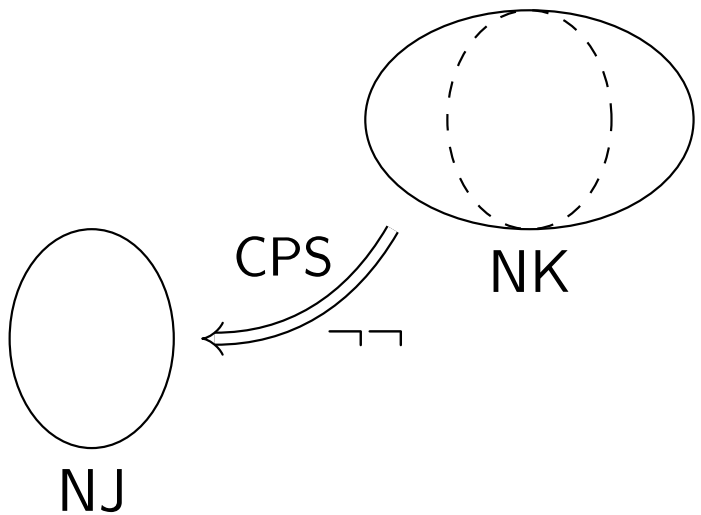
Interprétation calculatoire du forcing

Méthode :

1. Dépliage construction de forcing [Cohen63]
   $\rightsquigarrow$ transformation de formules/preuves
2. Vision à travers Curry-Howard [Krivine10]
   $\rightsquigarrow$ transformation de programmes non typés
3. Intégration à l'évaluation [Miquel13]
   $\rightsquigarrow$ de KAM à KFAM

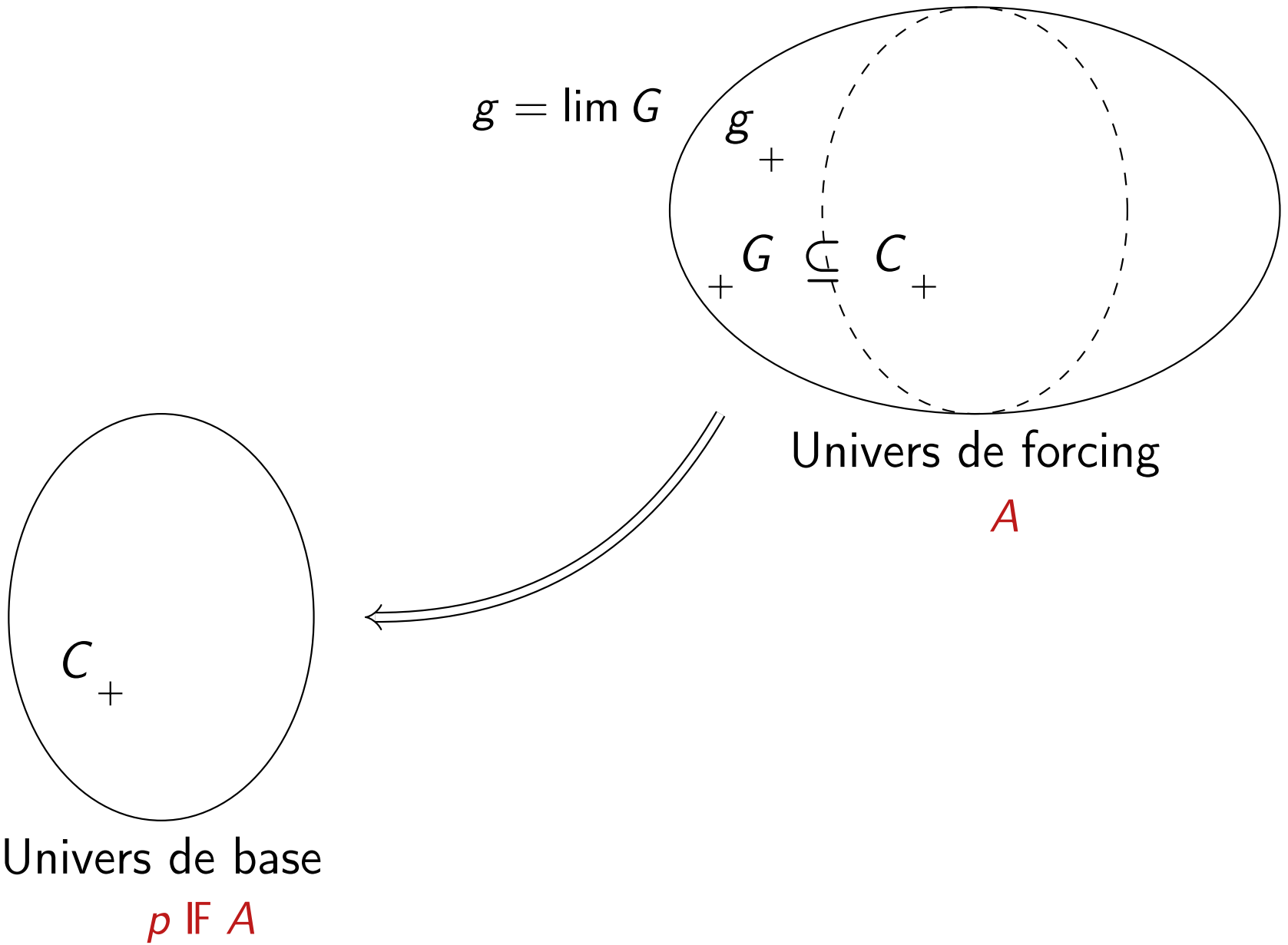
Similaire à :

1. $\neg\neg$-traductions
2. traductions CPS
3. `callcc`



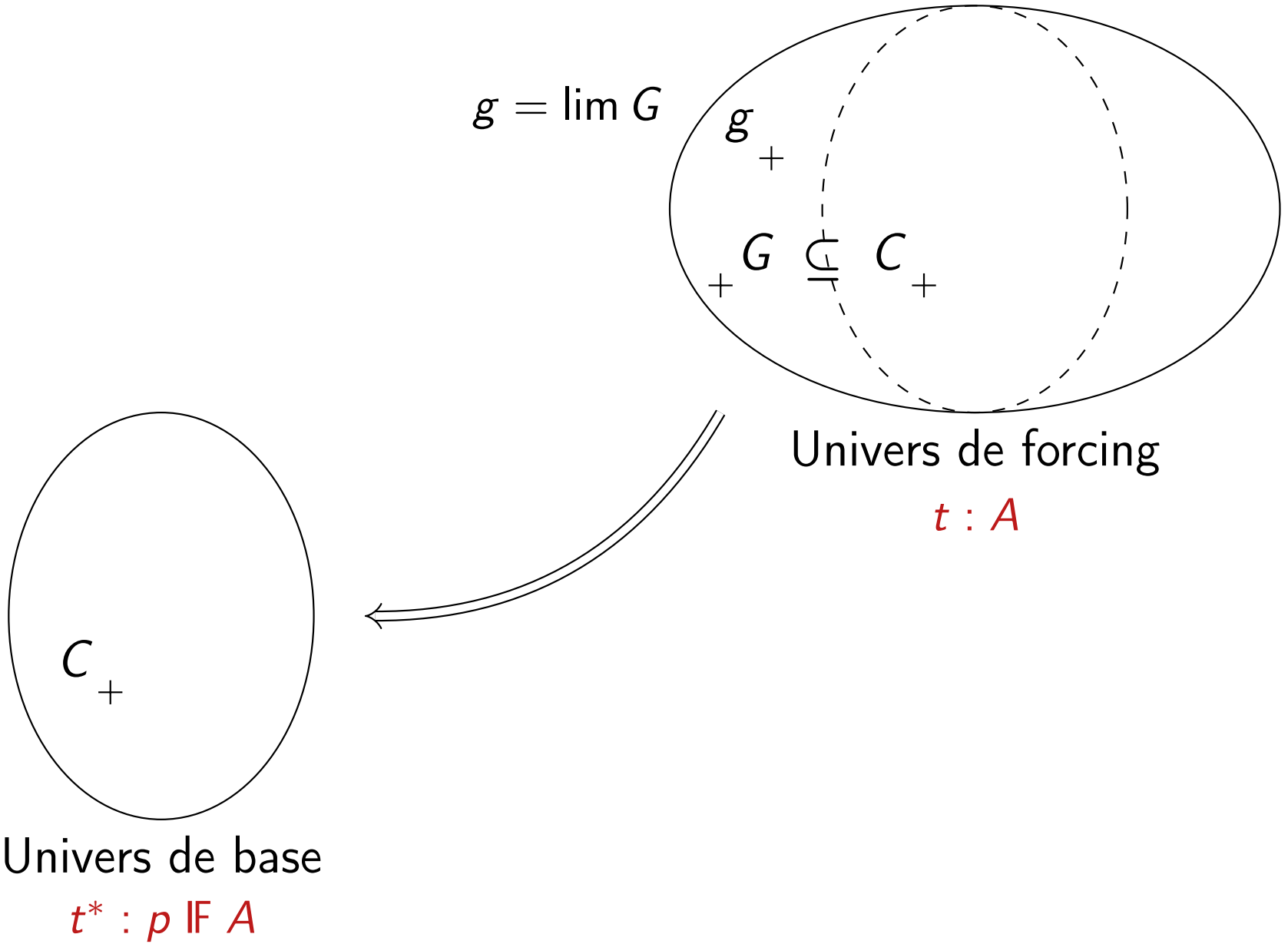
Forcing et calcul

[Cohen63]



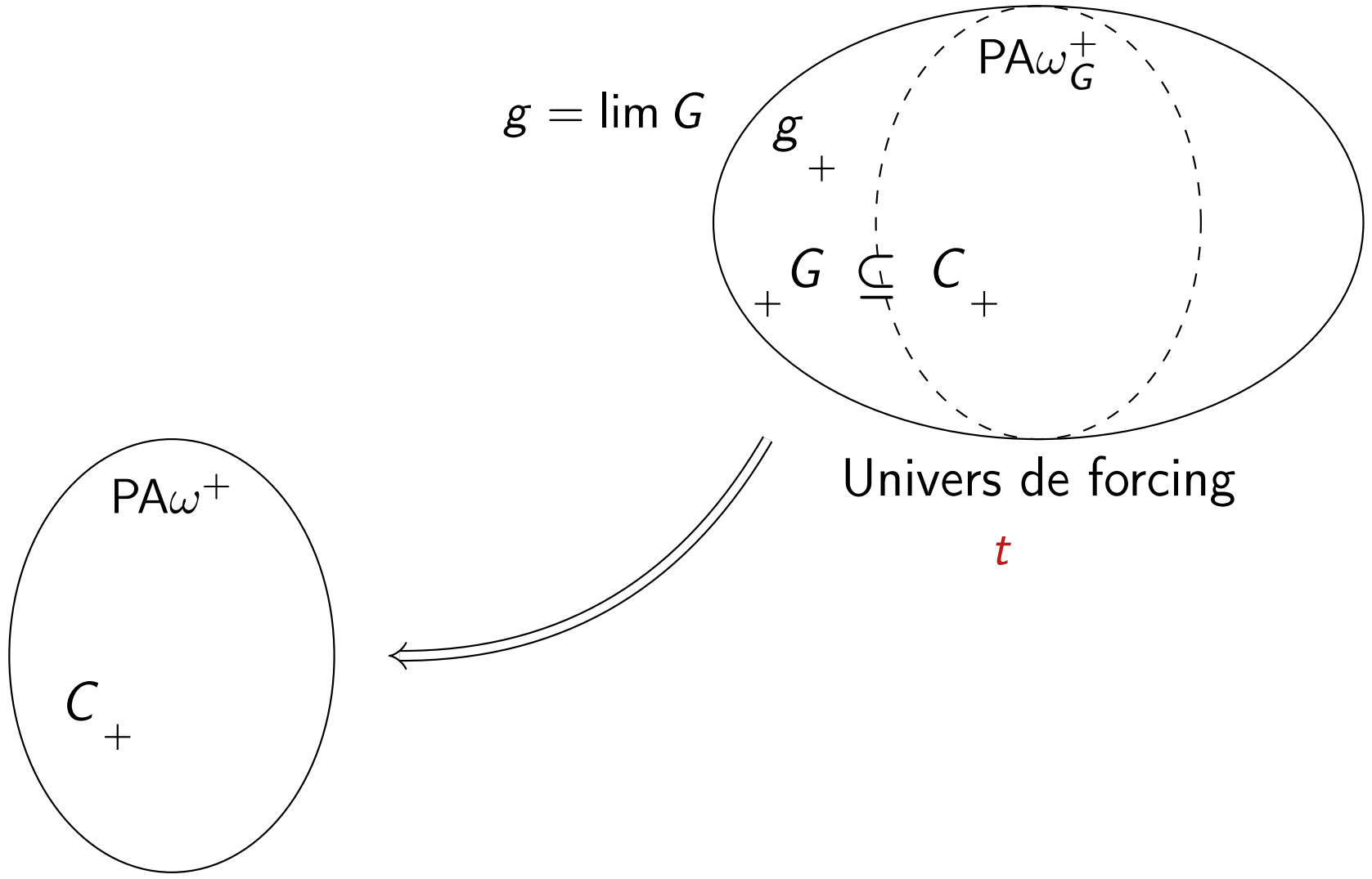
Forcing et calcul

[Cohen63,Krivine10]



Forcing et calcul

[Cohen63,Krivine10]



Quid du filtre générique G ?

Traduction du filtre générique

Restriction : C arithmétique

Extension de la traduction $A \mapsto p \Vdash A$ à G : (dans $\mathrm{PA}\omega^+$)

$$p \Vdash x \in G \;:=\; p \leq x \qquad (x \text{ condition})$$

Traduction du filtre générique

Restriction : C arithmétique

Extension de la traduction $A \mapsto p \Vdash A$ à G : (dans $\mathrm{PA}\omega^+$)

$$p \Vdash x \in G \;:=\; p \leq x \qquad (x \text{ condition})$$

Termes de preuves pour les propriétés de G : $t_i : p \Vdash A_i$

- non vide : $1 \in G$ (A_1)
- inclus dans C : $\forall p \in G.\, p \in C$ (A_2)
- filtre 1 : $\forall p \forall q.\, (p \cdot q) \in G \Rightarrow p \in G$ (A_3)
- filtre 2 : $\forall p \in G. \forall q \in G.\, (p \cdot q) \in G$ (A_4)
- généricité : $\forall S.\ \ldots$ (A_5)

$\rightsquigarrow$ extension de la traduction $t \mapsto t^*$ aux propriétés de G

Au final, ...

Traduction du filtre générique

Restriction : C arithmétique

Extension de la traduction $A \mapsto p \Vdash A$ à G : (dans $\mathrm{PA}\omega^+$)

$$p \Vdash x{\in}G \;:=\; p \leq x \qquad (x \text{ condition})$$

Termes de preuves pour les propriétés de G : $t_i : p \Vdash A_i$

- non vide : $1 \in G$ (A_1)
- inclus dans C : $\forall p \in G.\, p \in C$ (A_2)
- filtre 1 : $\forall p \forall q.\, (p \cdot q) \in G \Rightarrow p \in G$ (A_3)
- filtre 2 : $\forall p \in G. \forall q \in G.\, (p \cdot q) \in G$ (A_4)
- généricité : $\forall S.\ \ldots$ (A_5)

$\rightsquigarrow$ extension de la traduction $t \mapsto t^*$ aux propriétés de G

Au final, une transformation complète de $\mathrm{PA}\omega^+_G$ dans $\mathrm{PA}\omega^+$

Internaliser la transformation dans la KAM [Miquel13]

KAM = machine abstraite pour évaluer les programmes

Idée : passer de

$$t \xrightarrow{(\ldots)^*} t^* \xrightarrow{\text{KAM}} u^*$$

à

$$t \xrightarrow{\text{KFAM}} u$$

Internaliser la transformation dans la KAM [Miquel13]

KAM = machine abstraite pour évaluer les programmes

terme $\star$ pile $\succ$ terme' $\star$ pile'

Push	$t\,u$	$\star$	π	$\succ$	t	$\star$	$u \cdot \pi$
Grab	$\lambda x.\, t$	$\star$	$u \cdot \pi$	$\succ$	$t[u/x]$	$\star$	π
Save	`callcc`	$\star$	$t \cdot \pi$	$\succ$	t	$\star$	$k_\pi \cdot \pi$
Restore	$k_{\pi'}$	$\star$	$t \cdot \pi$	$\succ$	t	$\star$	π'

Exemple : $t := (\lambda x.\, \lambda y.\, x + y)\,2\,3$

$(\lambda x.\, \lambda y.\, x + y)\,2\,3 \star \pi$	$\succ$	$(\lambda x.\, x + y)\,2$	$\star$	$3 \cdot \pi$	(Push)
	$\succ$	$\lambda x.\, \lambda y.\, x + y$	$\star$	$2 \cdot 3 \cdot \pi$	(Push)
	$\succ$	$\lambda y.\, 2 + y$	$\star$	$3 \cdot \pi$	(Grab)
	$\succ$	$2 + 3$	$\star$	π	(Grab)

Internaliser la transformation dans la KFAM [Miquel13]

KAM = machine abstraite pour évaluer les programmes

terme $\star$ pile $\succ$ terme' $\star$ pile'

Push	$t\,u$	$\star$		π	$\succ$	t	$\star$	$u \cdot \pi$
Grab	$\lambda x.\,t$	$\star$	$u \cdot \pi$		$\succ$	$t[u/x]$	$\star$	π
Save	`callcc`	$\star$	$t \cdot \pi$		$\succ$	t	$\star$	$k_\pi \cdot \pi$
Restore	$k_{\pi'}$	$\star$	$t \cdot \pi$		$\succ$	t	$\star$	π'

$\Uparrow \quad \Downarrow$

Push*	$t\,u$	$\star$	$f \cdot$	π	$\succ$	t	$\star$	$\alpha_{11}\, f \cdot u$	$\cdot \pi$
Grab*	$\lambda x.\,t$	$\star$	$f \cdot u \cdot \pi$		$\succ$	$t[u/x]$	$\star$	$\alpha_6\ f \cdot$	π
Save*	`callcc`	$\star$	$f \cdot t \cdot \pi$		$\succ$	t	$\star$	$\alpha_{14}\, f \cdot k_\pi$	$\cdot \pi$
Restore*	$k_{\pi'}$	$\star$	$f \cdot t \cdot \pi$		$\succ$	t	$\star$	$\alpha_{15}\, f \cdot$	π'

Conclusion : forcing = case mémoire + monitoring

Utiliser le forcing : méthodologie générale

On cherche à prouver $\dfrac{A_1 \qquad \ldots \qquad A_n}{A}$.

Univers de base

Univers de forcing

Utiliser le forcing : méthodologie générale

On cherche à prouver $\dfrac{A_1 \quad \dots \quad A_n}{A}$.

Univers de base

1. Choisir C

Univers de forcing

Utiliser le forcing : méthodologie générale

On cherche à prouver $\dfrac{A_1 \qquad \ldots \qquad A_n}{A}$.

Univers de base

1. Choisir C
2. Poser les prémisses
   $x_1 : A_1, \ldots, x_n : A_n$

Univers de forcing

Utiliser le forcing : méthodologie générale

On cherche à prouver $\dfrac{A_1 \qquad \ldots \qquad A_n}{A}$.

Univers de base

1. Choisir C
2. Poser les prémisses
   $x_1 : A_1, \ldots, x_n : A_n$

Univers de forcing

3. Relever les prémisses

Utiliser le forcing : méthodologie générale

On cherche à prouver $\dfrac{A_1 \qquad \dots \qquad A_n}{A}$.

Univers de base

1. Choisir C
2. Poser les prémisses
   $x_1 : A_1, \dots, x_n : A_n$

Univers de forcing

3. Relever les prémisses
4. Faire la preuve (avec g, G)
   $t(x_1, \dots, x_n) : A$

Utiliser le forcing : méthodologie générale

On cherche à prouver $\dfrac{A_1 \qquad \dots \qquad A_n}{A}$.

Univers de base

1. Choisir C
2. Poser les prémisses
   $x_1 : A_1, \dots, x_n : A_n$

5. Traduire le forcing
   $t^*(x_1^*, \dots, x_n^*) : 1 \Vdash A$

Univers de forcing

3. Relever les prémisses
4. Faire la preuve (avec g, G)
   $t(x_1, \dots, x_n) : A$

Utiliser le forcing : méthodologie générale

On cherche à prouver $\dfrac{A_1 \qquad \dots \qquad A_n}{A}$.

Univers de base

1. Choisir C
2. Poser les prémisses
   $x_1 : A_1, \dots, x_n : A_n$

5. Traduire le forcing
   $t^*(x_1^*, \dots, x_n^*) : 1 \Vdash A$
6. Éliminer le forcing
   $w\, t^*(x_1^*, \dots, x_n^*) : A$

Univers de forcing

3. Relever les prémisses
4. Faire la preuve (avec g, G)
   $t(x_1, \dots, x_n) : A$

Utiliser le forcing : méthodologie générale

On cherche à prouver $\dfrac{A_1 \qquad \dots \qquad A_n}{A}$.

Univers de base

1. Choisir C
2. Poser les prémisses
   $x_1 : A_1, \dots, x_n : A_n$

5. Traduire le forcing
   $t^*(x_1^*, \dots, x_n^*) : 1 \Vdash A$
6. Éliminer le forcing
   $w\, t^*(x_1^*, \dots, x_n^*) : A$
7. Extraire un témoin
   (réalisabilité classique)

Univers de forcing

3. Relever les prémisses
4. Faire la preuve (avec g, G)
   $t(x_1, \dots, x_n) : A$

Motivations : théorème de Herbrand

Théorème : [Herbrand30]

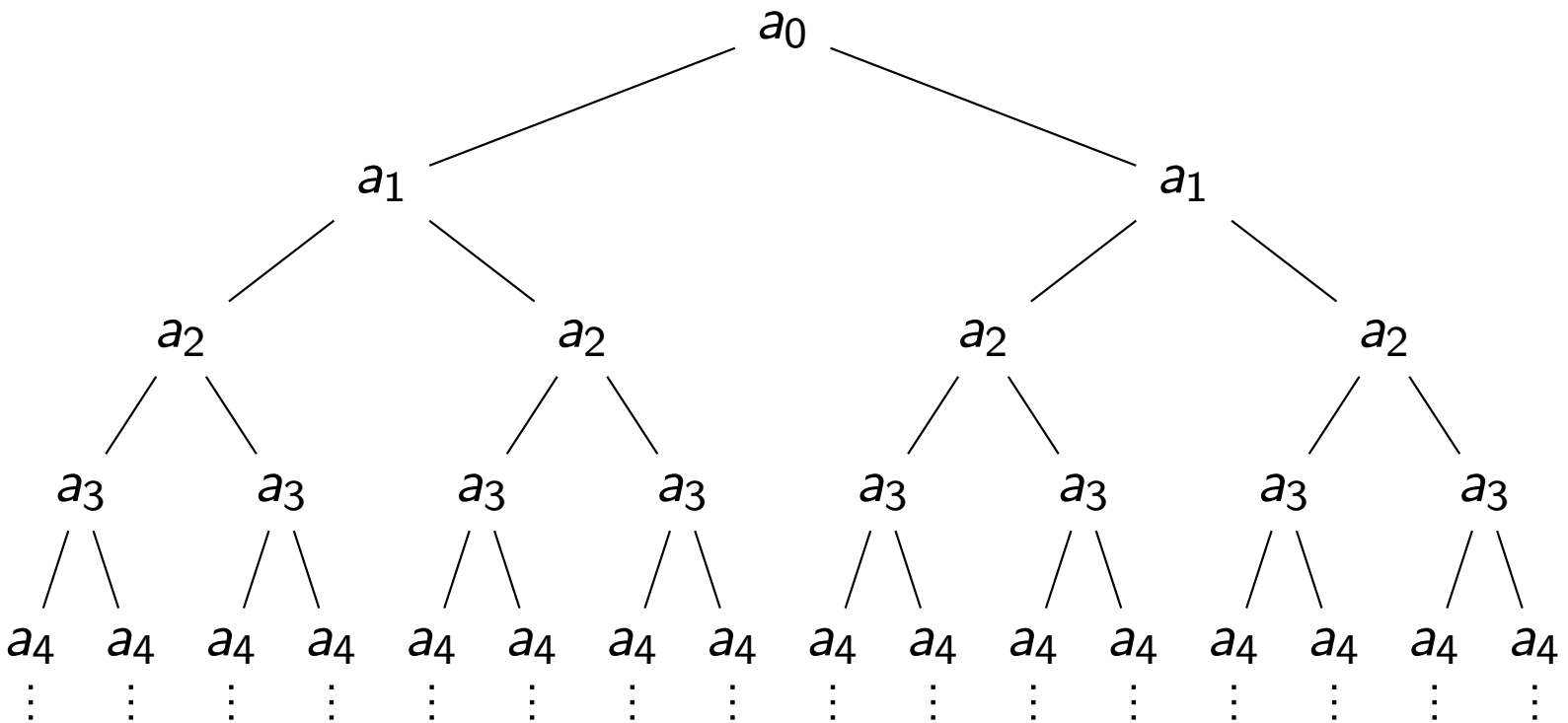
Si $\exists x.\, A(x)$ est vrai dans tout modèle syntaxique,
alors on a une preuve de $A(w_1) \vee \ldots \vee A(w_n)$

Fondement logique du problème :
$\exists x.\, A(x) \mapsto A(w_1) \vee \ldots \vee A(w_n)$

Fondement informatique du problème :
arbre infini $\mapsto$ arbre fini

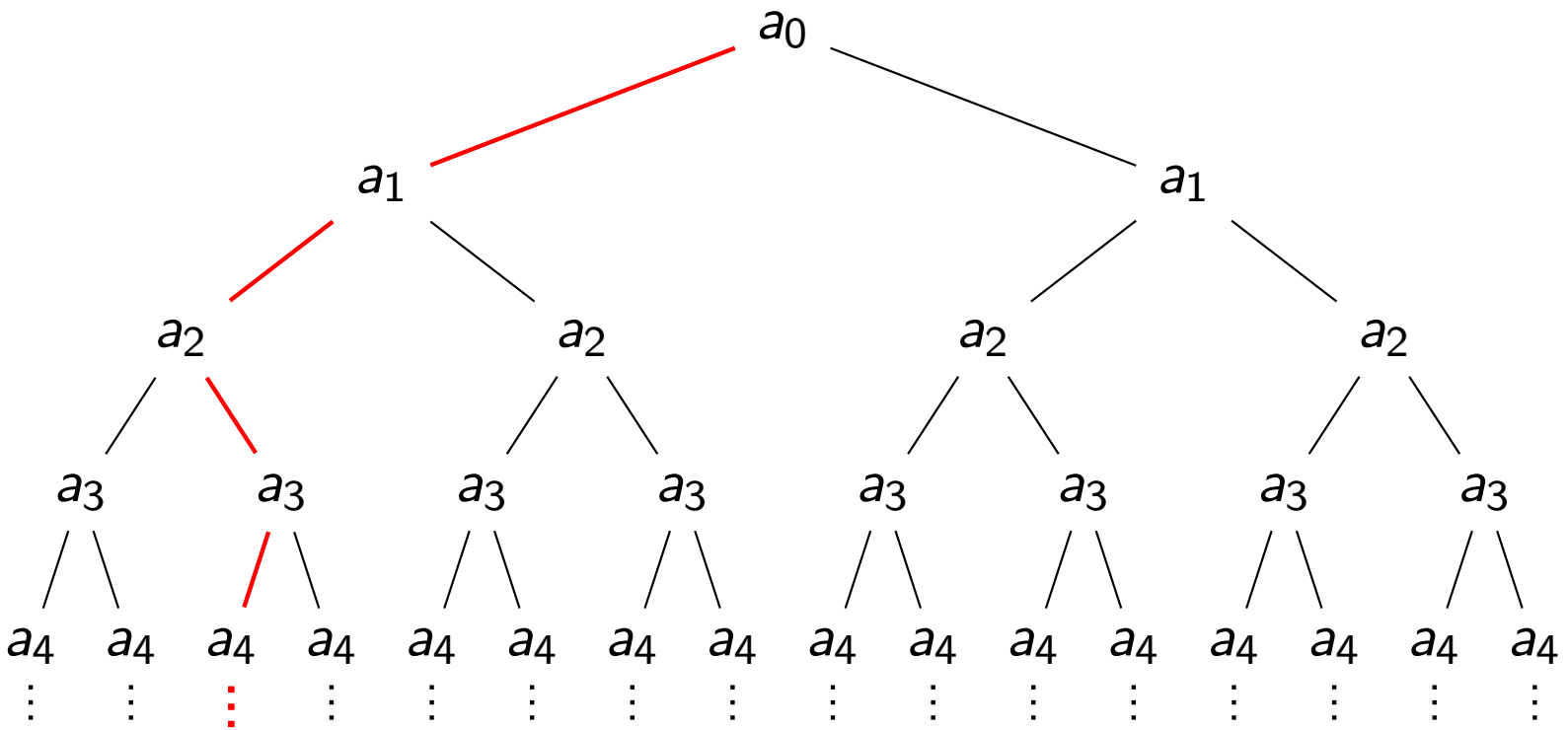
Hypothèse clé :
branche infinie $\mapsto$ branche finie

Idée de la méthode sans forcing



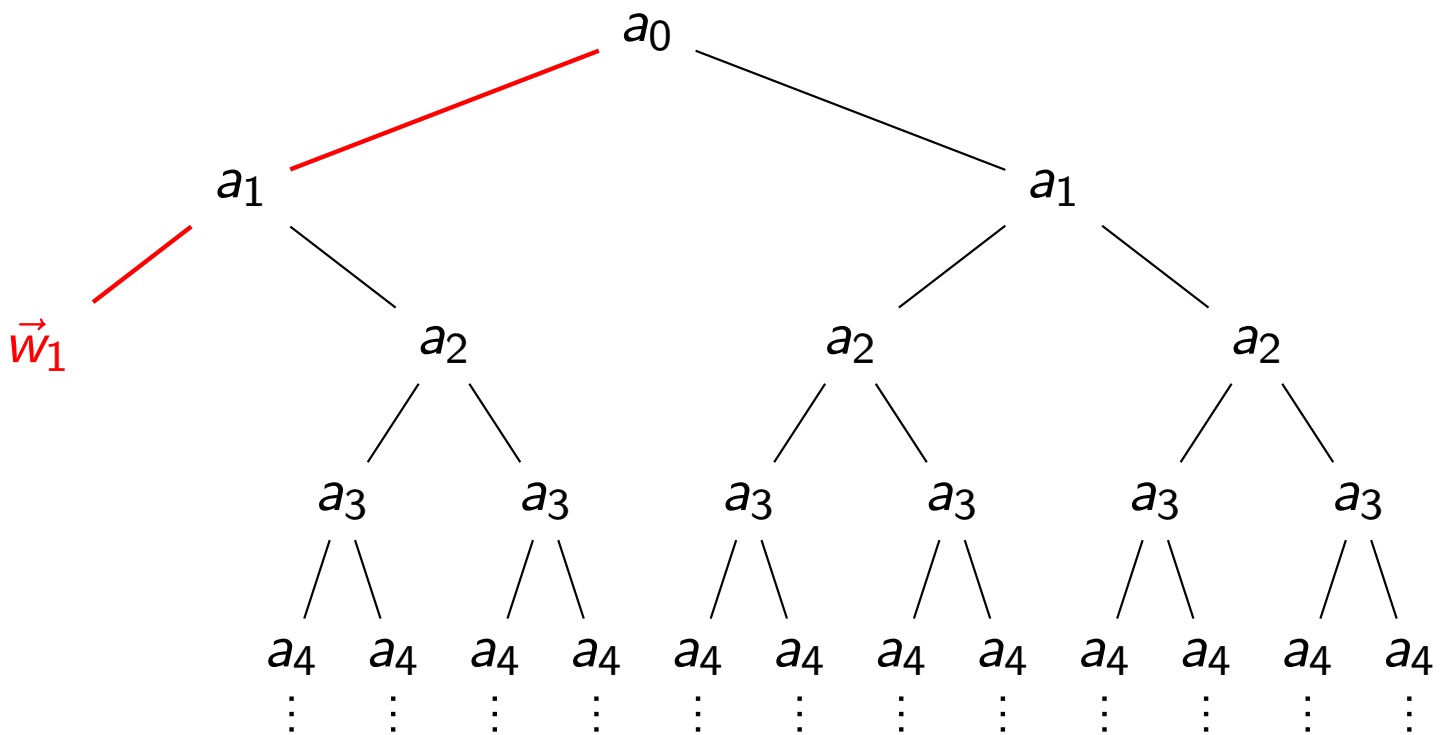
arbre binaire infini

Idée de la méthode sans forcing



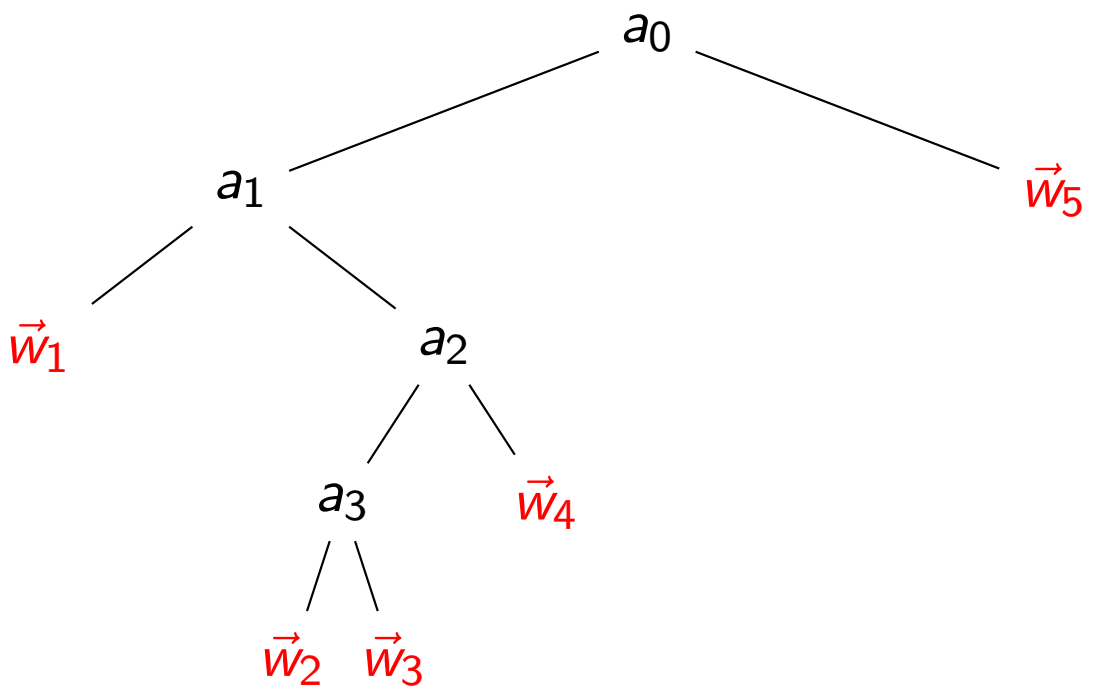
branche infinie quelconque

Idée de la méthode sans forcing



hypothèse clé : coupe à profondeur finie

Idée de la méthode sans forcing



conclusion par le théorème de l'éventail

Inconvénient de cette méthode

1. Gérer explicitement la structure arborescente
2. Besoin d'une énumération (point de vue logique)
3. Contenu calculatoire reporté au théorème de l'éventail

Le forcing résout ces problèmes :

1. Une seule branche à considérer
2. Branche non ordonnée
3. Interprétation calculatoire du forcing

Forcing pour les arbres de Herbrand

Intuition : $g =$ une branche infinie générique
qui représente toutes les autres

$\rightsquigarrow$ plus besoin de gérer la structure d'arbre

Forcing pour les arbres de Herbrand

Intuition : $g =$ une branche infinie générique
qui représente toutes les autres
$\rightsquigarrow$ plus besoin de gérer la structure d'arbre

$C :=$ branches finies dans l'arbre infini

$G :=$ certaines branches finies compatibles entre elles

$g := \bigcup G$

Forcing pour les arbres de Herbrand

Intuition : $g =$ une branche infinie générique
qui représente toutes les autres
$\rightsquigarrow$ plus besoin de gérer la structure d'arbre

$C :=$ branches finies dans l'arbre infini
+ information pour reconstruire l'arbre final

$G :=$ certaines branches finies compatibles entre elles

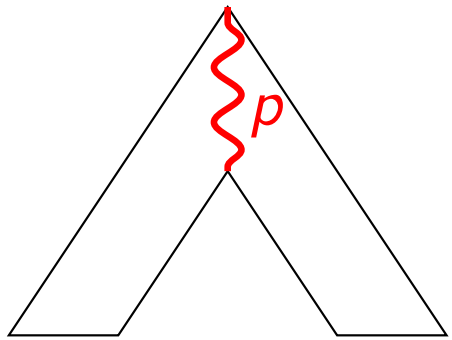
$g := \bigcup G$

Quel est leur contenu calculatoire précis ?

Interprétations calculatoires

Pour $p \in C$:
un zipper dépendant pour arbre

- un contexte d'arbre de Herbrand
- avec un trou à la position p



Pour G : $\quad (p \Vdash x{\in}G \;:=\; p \leq x)$
un ensemble mouvant qui change en fonction de la position dans l'arbre

Pour les programmes $t_i : p \Vdash A_i$:
les primitive d'accès à l'information stockée dans $x \in G$

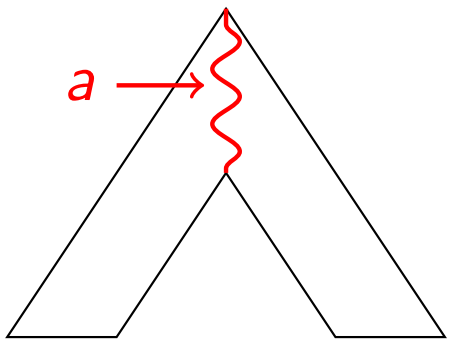
L'axiome de totalité

Cas particulier de la généricité :

p IF $\forall a \in \text{Atom}.\, \exists q \in G.\, \exists b \in \text{Bool}.\, \texttt{test}\, a\, b\, q = 1$

2 cas :

- $a \in p$: répondre b comme dans p
- $a \notin p$: on essaie à la fois `true` et `false`



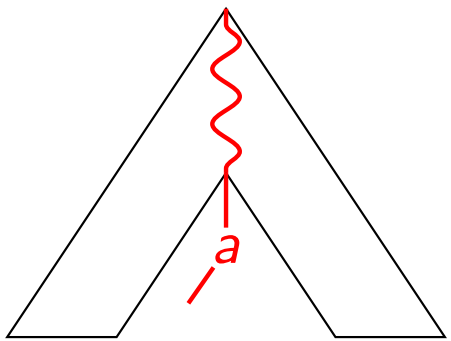
L'axiome de totalité

Cas particulier de la généricité :

p IF $\forall a \in \mathrm{Atom}.\, \exists q \in G.\, \exists b \in \mathrm{Bool}.\, \texttt{test}\, a\, b\, q = 1$

2 cas :

- $a \in p$: répondre b comme dans p
- $a \notin p$: on essaie à la fois `true` et `false`



Ordonnancement

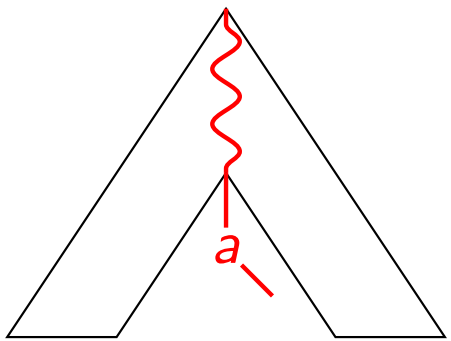
L'axiome de totalité

Cas particulier de la généricité :

p IF $\forall a \in \mathrm{Atom}.\, \exists q \in G.\, \exists b \in \mathrm{Bool}.\, \texttt{test}\, a\, b\, q = 1$

2 cas :

- $a \in p$: répondre b comme dans p
- $a \notin p$: on essaie à la fois `true` et `false`



Ordonnancement

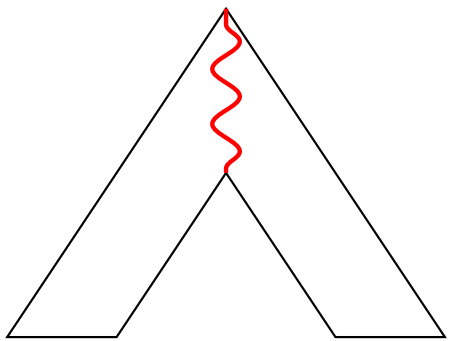
L'axiome de totalité

Cas particulier de la généricité :

p IF $\forall a \in \text{Atom}.\, \exists q \in G.\, \exists b \in \text{Bool}.\, \texttt{test}\, a\, b\, q = 1$

2 cas :

- $a \in p$: répondre b comme dans p
- $a \notin p$: on essaie à la fois `true` et `false`



Ordonnancement

$$
\begin{aligned}
&\lambda caf.\ \text{let } p, t := \alpha\, c \text{ in}\\
&\quad \text{if } \texttt{test}\, a\, \texttt{true}\, p \text{ then } f\,(\alpha\, c)\, \texttt{id}\, \texttt{true}^*\, \texttt{id}^* \text{ else}\\
&\quad \text{if } \texttt{test}\, a\, \texttt{false}\, p \text{ then } f\,(\alpha\, c)\, \texttt{id}\, \texttt{false}^*\, \texttt{id}^* \text{ else}\\
&\quad f\,\langle (a, \texttt{true}) \cup p, \lambda u.\\
&\qquad f\,\langle (a, \texttt{false}) \cup p, \lambda v.\\
&\qquad\quad t\,(\texttt{merge}\, a\, u\, v)\rangle\ \texttt{id}\, \texttt{false}^*\, \texttt{id}^*\rangle\ \texttt{id}\, \texttt{true}^*\, \texttt{id}^*
\end{aligned}
$$

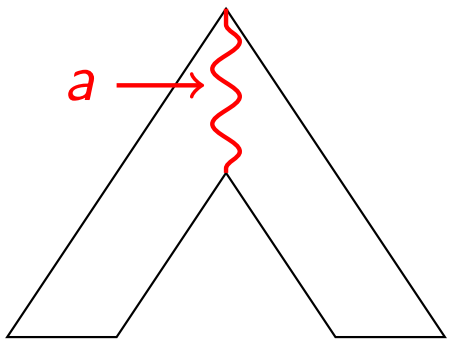
L'axiome de totalité

Cas particulier de la généricité :

p IF $\forall a \in \text{Atom}.\, \exists q \in G.\, \exists b \in \text{Bool}.\, \texttt{test}\, a\, b\, q = 1$

2 cas :

- $a \in p$: répondre b comme dans p
- $a \notin p$: on essaie à la fois `true` et `false`



Ordonnancement

$$\begin{aligned}
&\lambda caf.\ \text{let } p, t := \alpha\, c \text{ in}\\
&\quad \texttt{if test } a \texttt{ true } p \texttt{ then } f\ (\alpha\, c)\ \texttt{id true}^*\ \texttt{id}^*\ \texttt{else}\\
&\quad \texttt{if test } a \texttt{ false } p \texttt{ then } f\ (\alpha\, c)\ \texttt{id false}^*\ \texttt{id}^*\ \texttt{else}\\
&\quad f\ \langle (a, \texttt{true}) \cup p, \lambda u.\\
&\qquad f\ \langle (a, \texttt{false}) \cup p, \lambda v.\\
&\qquad\quad t\ (\texttt{merge}\, a\, u\, v)\rangle\ \texttt{id false}^*\ \texttt{id}^*\rangle\ \texttt{id true}^*\ \texttt{id}^*
\end{aligned}$$

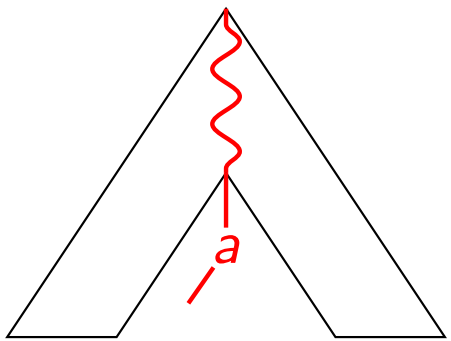
L'axiome de totalité

Cas particulier de la généricité :

p IF $\forall a \in \text{Atom}.\, \exists q \in G.\, \exists b \in \text{Bool}.\, \texttt{test}\, a\, b\, q = 1$

2 cas :

- $a \in p$: répondre b comme dans p
- $a \notin p$: on essaie à la fois `true` et `false`



Ordonnancement

$$\begin{aligned}
&\lambda caf.\ \text{let } p, t := \alpha\, c \text{ in}\\
&\quad \text{if } \texttt{test}\, a\, \texttt{true}\, p \text{ then } f\,(\alpha\, c)\, \texttt{id}\, \texttt{true}^*\, \texttt{id}^* \text{ else}\\
&\quad \text{if } \texttt{test}\, a\, \texttt{false}\, p \text{ then } f\,(\alpha\, c)\, \texttt{id}\, \texttt{false}^*\, \texttt{id}^* \text{ else}\\
&\quad f\,\langle (a, \texttt{true}) \cup p, \lambda u.\\
&\qquad f\,\langle (a, \texttt{false}) \cup p, \lambda v.\\
&\qquad\quad t\,(\texttt{merge}\, a\, u\, v)\rangle\ \texttt{id}\, \texttt{false}^*\, \texttt{id}^*\rangle\ \texttt{id}\, \texttt{true}^*\, \texttt{id}^*
\end{aligned}$$

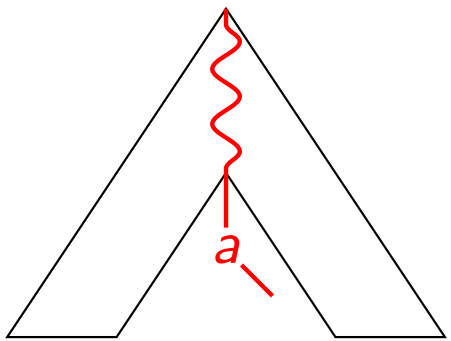
L'axiome de totalité

Cas particulier de la généricité :

p IF $\forall a \in \text{Atom}.\, \exists q \in G.\, \exists b \in \text{Bool}.\, \texttt{test}\, a\, b\, q = 1$

2 cas :

- $a \in p$: répondre b comme dans p
- $a \notin p$: on essaie à la fois `true` et `false`



Ordonnancement

$$
\begin{aligned}
&\lambda caf.\ \text{let } p, t := \alpha\, c \text{ in}\\
&\quad \text{if } \texttt{test}\, a\, \texttt{true}\, p \text{ then } f\,(\alpha\, c)\, \texttt{id}\, \texttt{true}^*\, \texttt{id}^* \text{ else}\\
&\quad \text{if } \texttt{test}\, a\, \texttt{false}\, p \text{ then } f\,(\alpha\, c)\, \texttt{id}\, \texttt{false}^*\, \texttt{id}^* \text{ else}\\
&\quad f\,\langle (a, \texttt{true}) \cup p, \lambda u.\\
&\qquad f\,\langle (a, \texttt{false}) \cup p, \lambda v.\\
&\qquad\quad t\,(\texttt{merge}\, a\, u\, v)\rangle\ \texttt{id}\, \texttt{false}^*\, \texttt{id}^*\rangle\ \texttt{id}\, \texttt{true}^*\, \texttt{id}^*
\end{aligned}
$$

Conclusion

Contributions :

- réalisabilité classique : vers un vrai langage de programmation
- Calcul par forcing :
  $\rightsquigarrow$ meilleurs algorithmes, pas plus d'expressivité
  $\rightsquigarrow$ ajout d'une case mémoire à un programme fonctionnel
- Compréhension complète du comportement du forcing
  $\rightsquigarrow$ Exemple des arbres de Herbrand

Perspectives :

- autres exemples de calcul par forcing (bar récursion...)
- autres traits impératifs (données)
- cas général du générique

Conclusion

Contributions :

- réalisabilité classique : vers un vrai langage de programmation
- Calcul par forcing :
  $\rightsquigarrow$ meilleurs algorithmes, pas plus d'expressivité
  $\rightsquigarrow$ ajout d'une case mémoire à un programme fonctionnel
- Compréhension complète du comportement du forcing
  $\rightsquigarrow$ Exemple des arbres de Herbrand

Perspectives :

- autres exemples de calcul par forcing (bar récursion...)
- autres traits impératifs (données)
- cas général du générique

Merci de votre attention

Conclusion

Contributions :

- réalisabilité classique : vers un vrai langage de programmation
- Calcul par forcing :
  $\rightsquigarrow$ meilleurs algorithmes, pas plus d'expressivité
  $\rightsquigarrow$ ajout d'une case mémoire à un programme fonctionnel
- Compréhension complète du comportement du forcing
  $\rightsquigarrow$ Exemple des arbres de Herbrand

Perspectives :

- autres exemples de calcul par forcing (bar récursion...)
- autres traits impératifs (données)
- cas général du générique

Merci de votre attention

Nombres réels en réalisabilité classique

Deux genres de constructions :

- coupures de Dedekind
- suites de Cauchy

Avantages de chaque :

- Dedekind : ordre
- Cauchy : opérations algébriques

Réels extractibles : définis par une formule Σ^0_2

$\rightsquigarrow$ extraction par totalité

$\rightsquigarrow$ exemple : racine de polynômes