Le λ-calcul : réductions, causalité et déterminisme

Gérard Berry

Collège de France
Chaire Informatique et sciences numériques

Cours 2, 2 décembre 2009

Le λ-calcul pur

- x, y, z,... : variables
- $\lambda x.\, M$: abstraction fonction de x de corps M
- (MN) : application d'une fonction à un argument
- $(\lambda x.\, M)\, N \rightarrow M[N/x]$: β-réduction

$\underline{0} = \lambda f.\, \lambda x.\, x$

$\underline{1} = \lambda f.\, \lambda x.\, fx$

...

$\underline{n} = \lambda f.\, \lambda x.\, f^{n}(x) = f(f(\ldots f(x)\ldots))$

let rec Fact m = si m=1 alors 1 sinon Mult (m, Fact (m–1))

Fact = Y (FACT)

avec $Y = (\lambda x.\, \lambda y.y(xxy))\, (\lambda x.\, \lambda y.y(xxy))$

Boucle, récursion, ou point fixe

- Récursion / point fixe

  Fact(m) = if m=1 then 1 else Mult(m, Fact(m–1))

  Fact = Y (λf. λm if m=1 then 1 else Mult(m, f(m–1)))

- Boucle

  while e do p   for (i=0; i<n; i++) p

- Boucle → récursion / point fixe : facile !

  while e do p → if e then { p; while e do p }

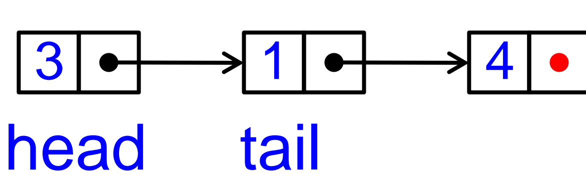
  While e p = if e then { p; While e p }

  While = Y (λW. λe. λp. if e then { p; W e p })

- Récursion → boucle : dur !

La récursion est plus primitive que la boucle

CAML : programmation fonctionnelle typée



En C / C++ : pointeurs, boucles, etc.
Bien mieux et plus sûr en CAML !

A faire : [3; 1; 4] → [4; 2; 5]

```
# let rec map f lst = match lst with
                           []→ []
                         | head :: tail → (f head) :: (map f tail);;
val map : ('a -> 'b) -> 'a list -> 'b list = <fun>

# let f n = n+1 in map f [3; 1; 4];;
: int list = [4; 2; 5]
```

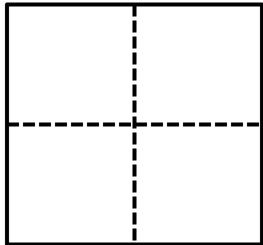
Théorème : un programme bien typé ne peut pas faire d'erreur non détectée à l'exécution

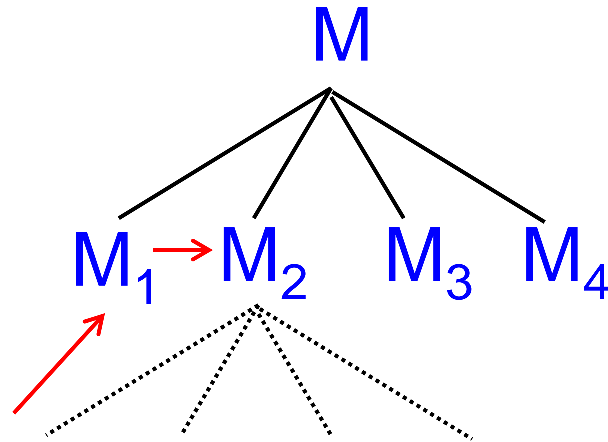
Erreur de type

```
# let rec badmap f lst = match lst with
                            []-> []
                          | head :: tail -> (f head) (badmap f tail);;
val badmap : ('a -> 'b list -> 'b list) -> 'a list -> 'b list = <fun>

# let f n = n+1 in badmap f [1; 2; 3];;
Characters 24-25:
  let f n = n+1 in badmap f [1; 2; 3];;
                          ^
Error: This expression has type int -> int but is here used with type
        int -> 'a list -> 'a list
```

Récursion et ordre supérieur en calcul numérique





$$F\,(f, x) = T\,(F, f, x) \times U\,(F, f, x)$$

Agenda

1. Cohérence et confluence
2. Causalité
3. Normalisation relative
4. Approximations, stabilité, séquentialité
5. Le λ-calcul simplement typé
6. Les types d’ordre supérieur

Agenda

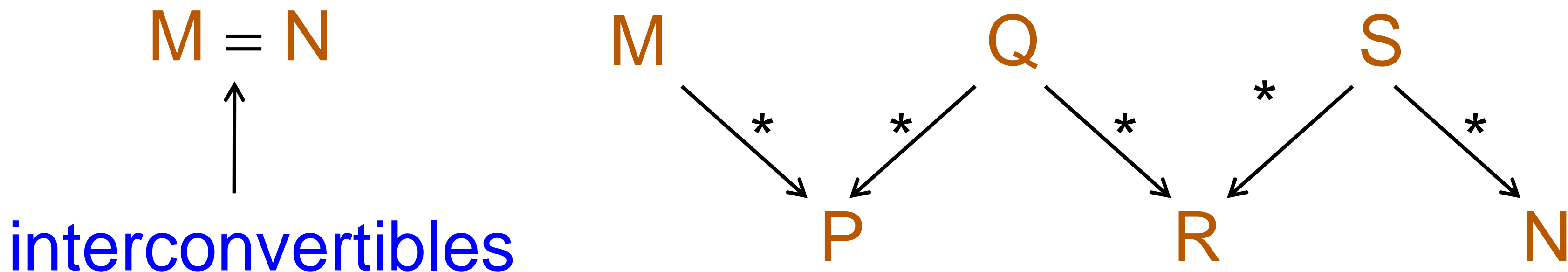
1. Cohérence et confluence
2. Causalité
3. Normalisation relative
4. Approximations, stabilité, séquentialité
5. Le λ-calcul simplement typé
6. Les types d'ordre supérieur

Interconvertibilité et cohérence

- Mais nous avons caché quelque chose : le λ-calcul est-il cohérent? Par exemple, les entiers sont ils bien tous différents? $\underline{m} \neq \underline{n}$ si $m \neq n$?

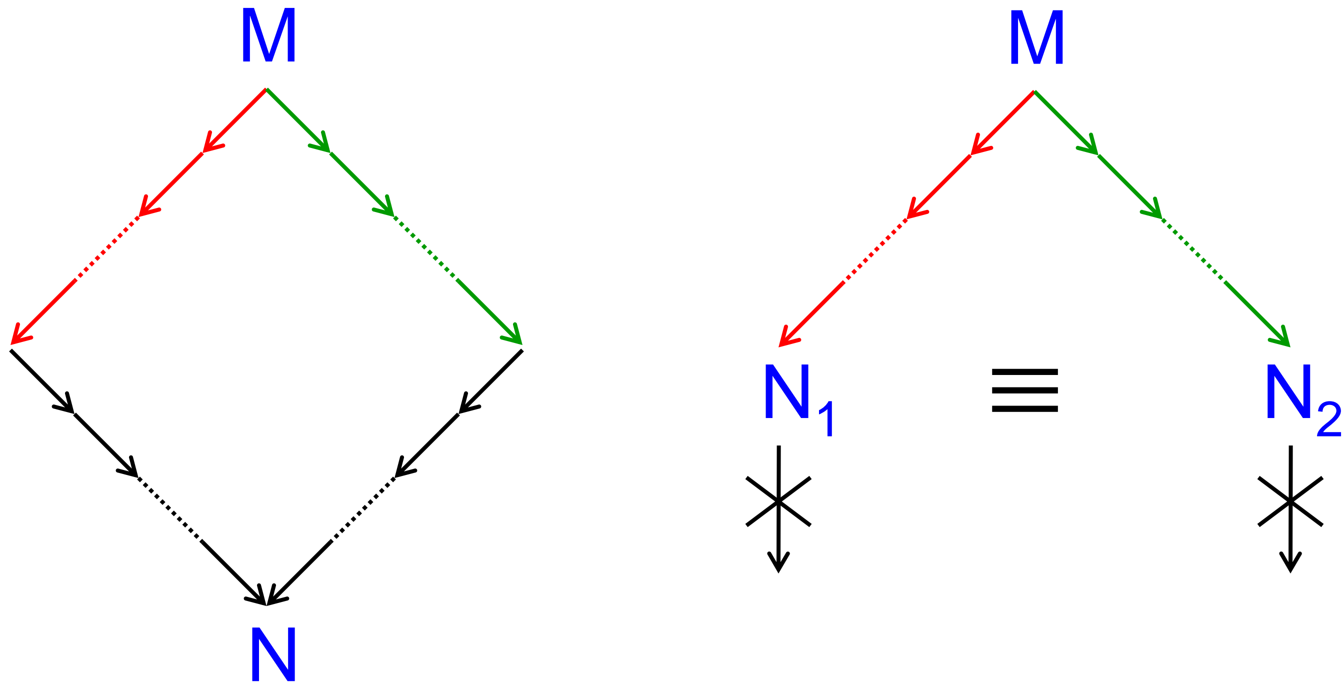
$$3+4 = 7 = 5+2$$

3+4 se calcule en 7, forme normale
et 7 est aussi la valeur de 5+2



Le calcul ne change pas la valeur, il la rend plus lisible

Confuence (Church-Rosser)

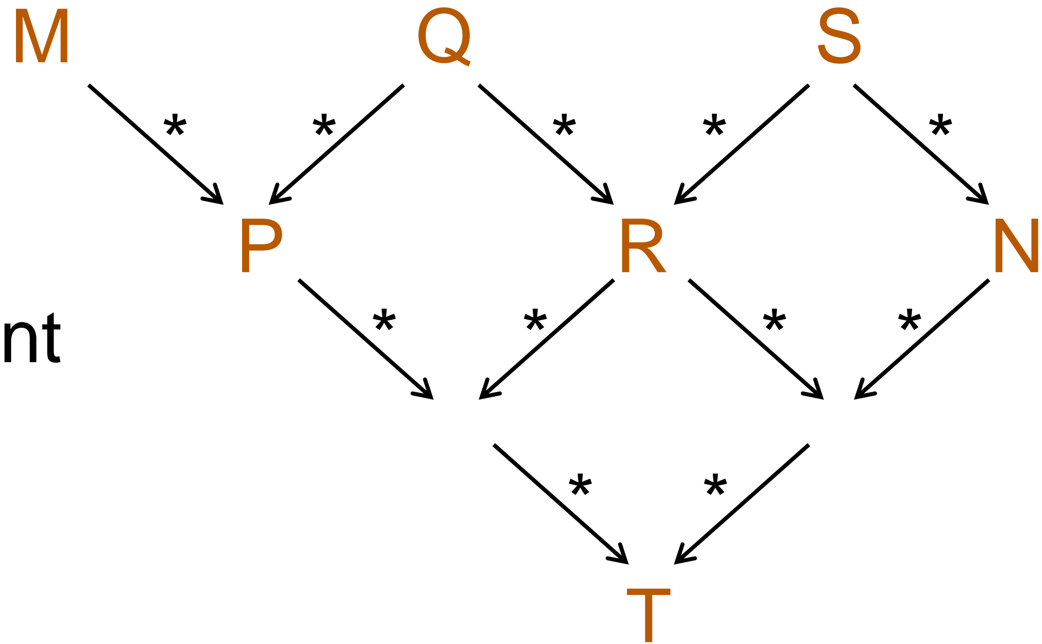


Théorème de Church-Rosser
=> forme normale unique
=> résultat déterministe

Interconvertibilité et cohérence

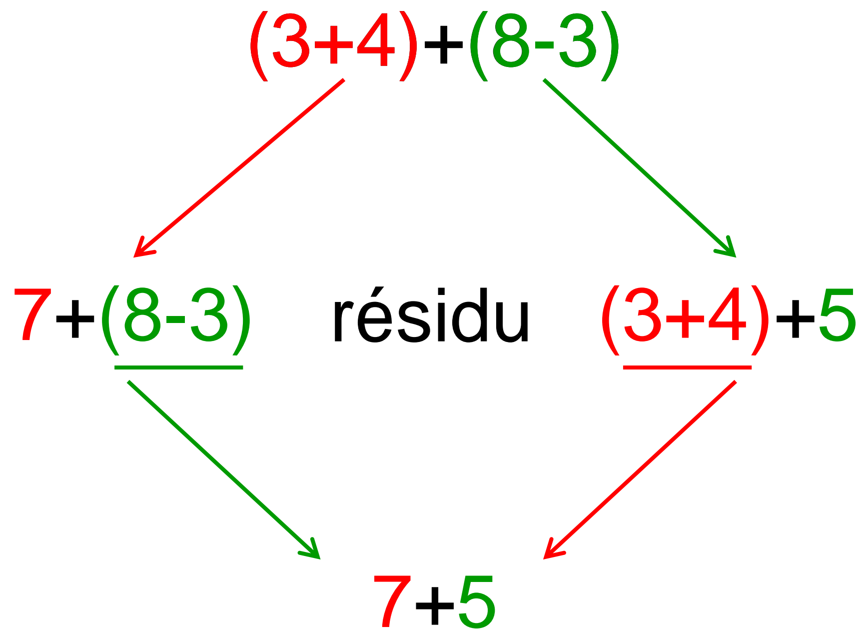
M = N

=> M et N confluent

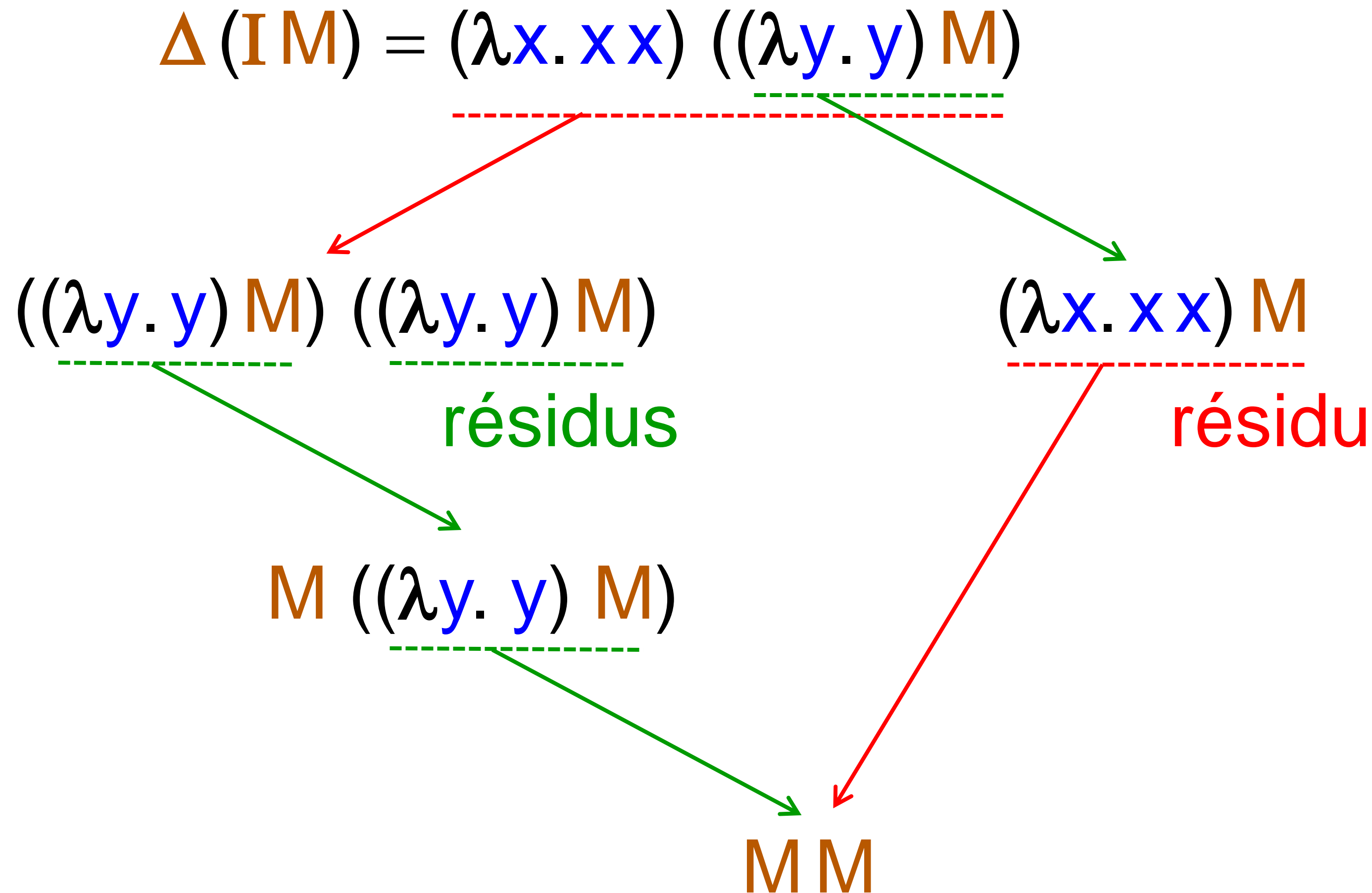


=> cohérence : <u>m</u> ≠ <u>n</u> si m ≠ n

Confluence locale – le cas simple

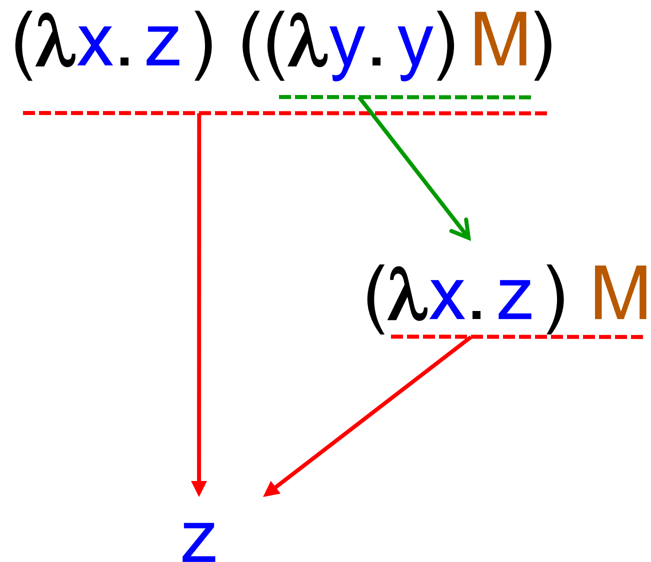


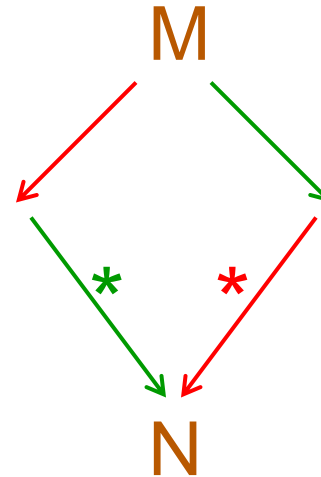
Confluence locale du λ-calcul



Les résidus de redex non réduits sont les mêmes

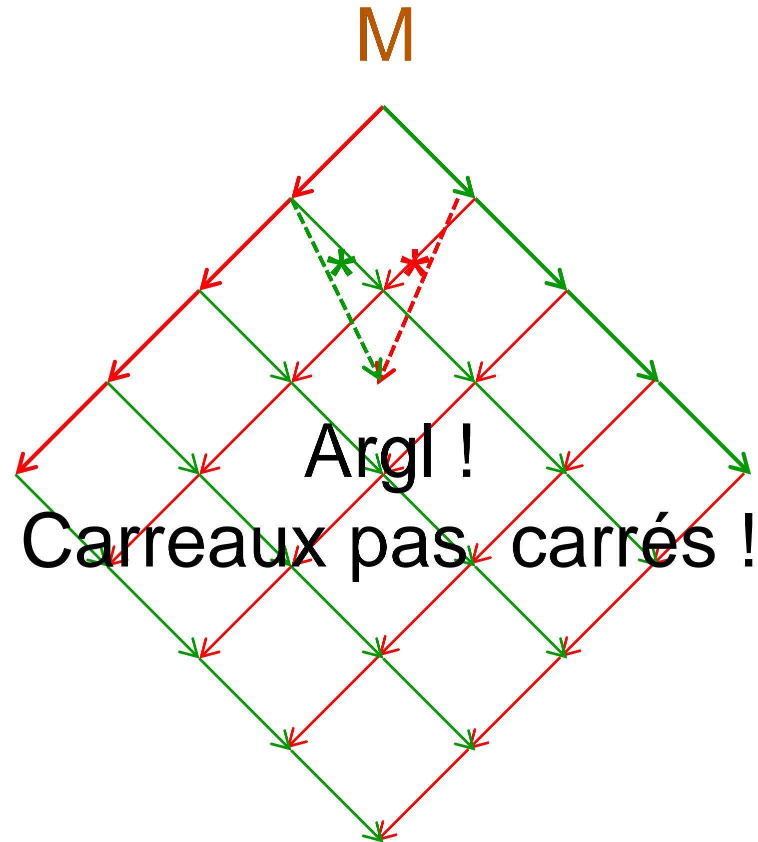
Disparition de redex





Dans une réduction, un redex non réduit a 0, 1 ou plusieurs résidus
Certains calculs peuvent être inutiles

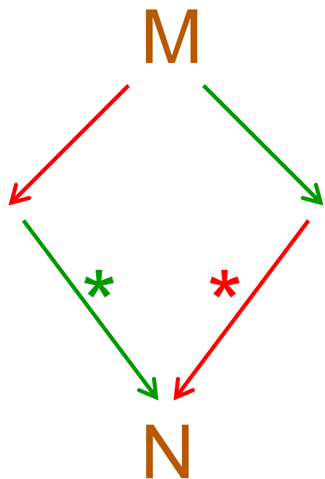
Problème de récurrence

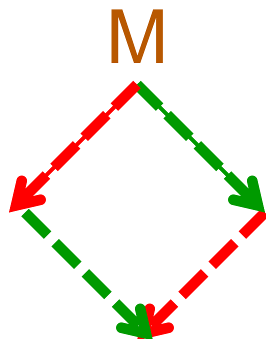


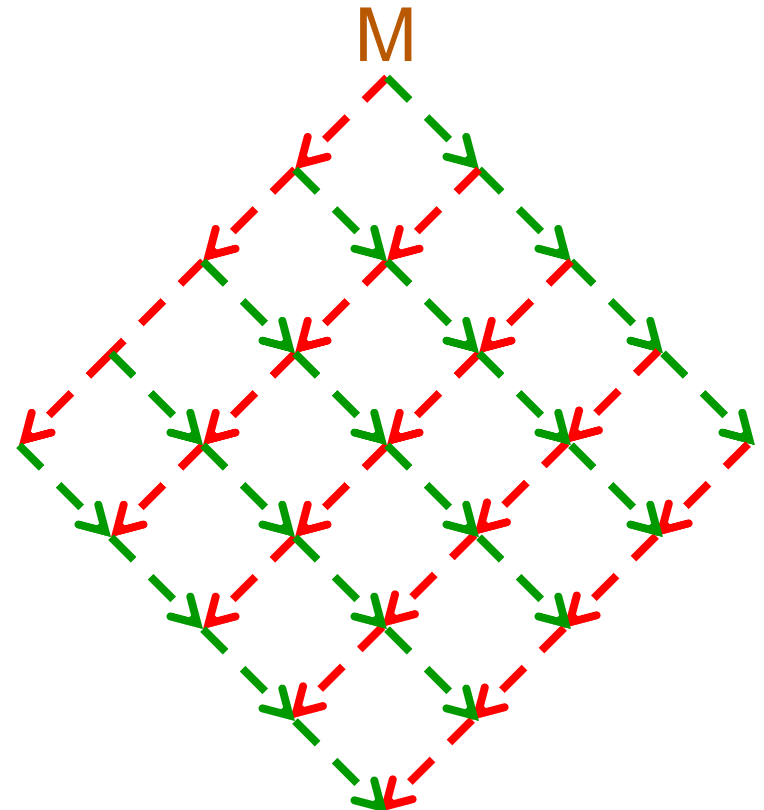
Développements finis (Curry & Feys)

<u>Théorème</u> : Soient R_1, R_2, ..., R_n des redex de M. Les réductions qui réduisent tous les R_i et leurs résidus sont finies et conduisent au même terme

réduire un
paquet de redex en un coup

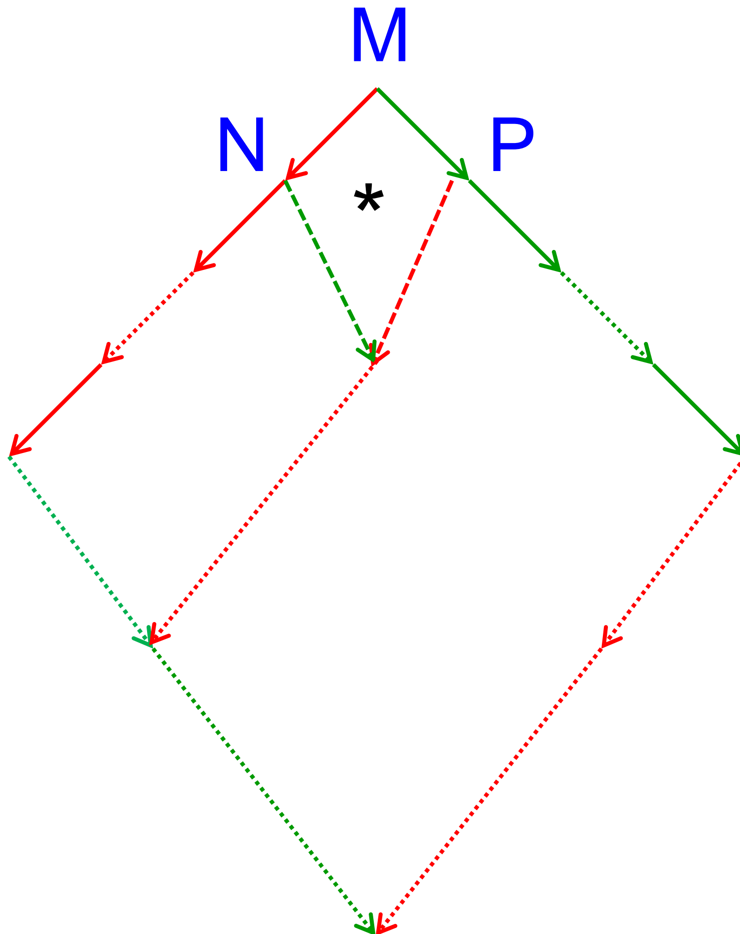






Solution 2 : lemme de Newman

Soit M fortement normalisable (toutes les réductions terminent).
Soit p(M) la longueur de la plus longue réduction de M.
montrons la confluence par récurrence sur p(M)

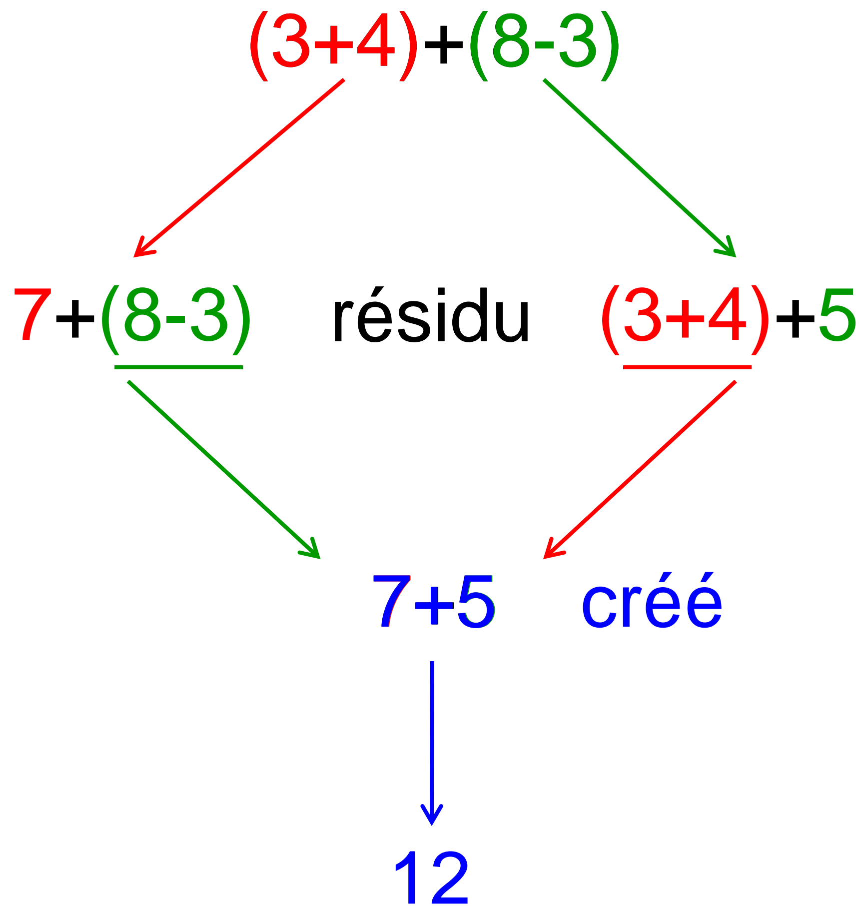


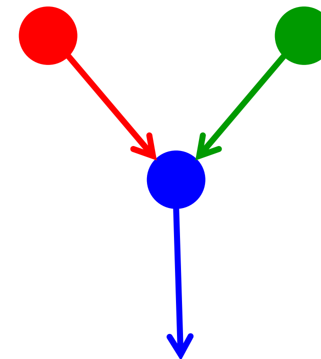
1. p(N) < p(M)
2. p(P) < p(M)
3. CQFD

Agenda

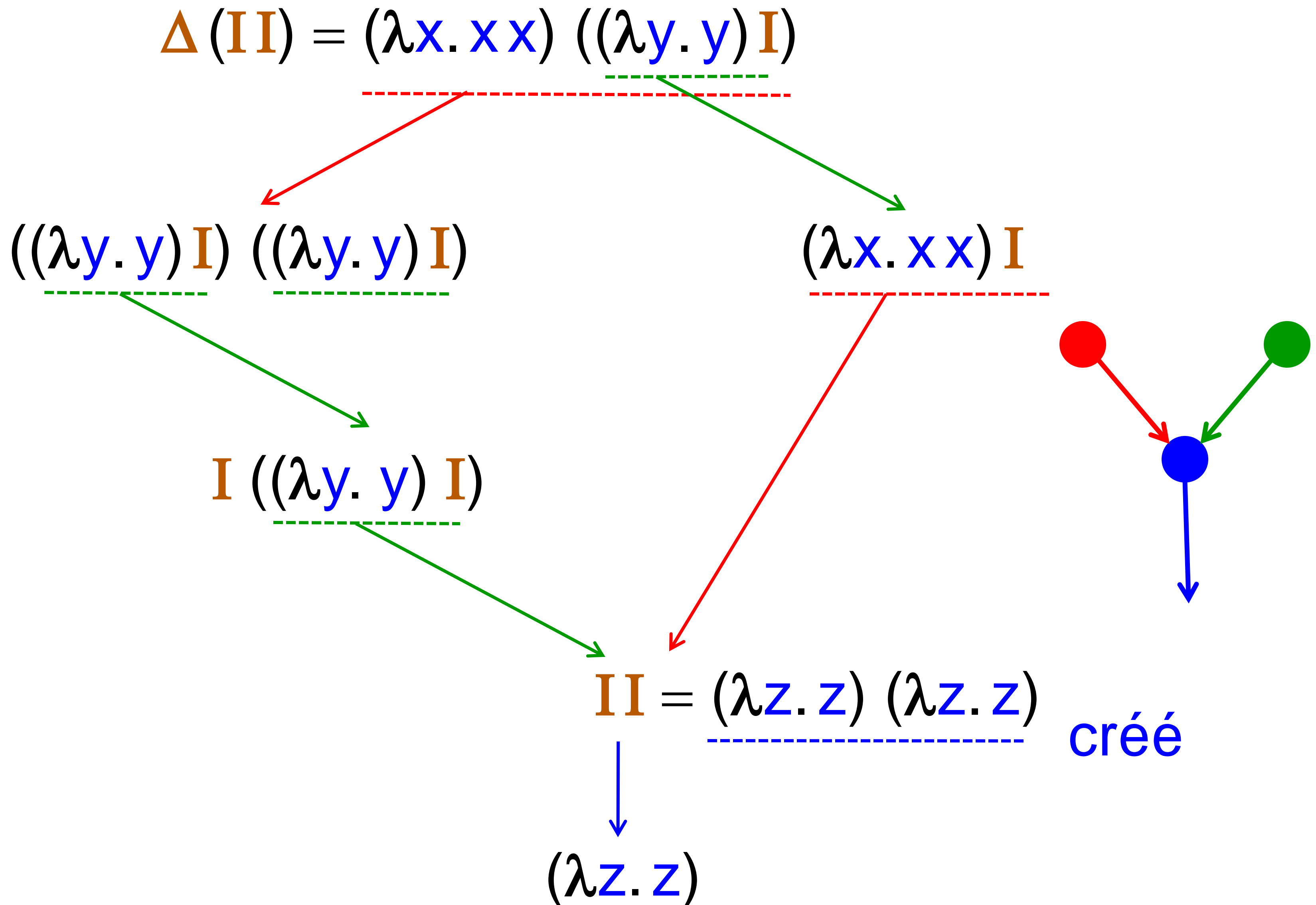
1. Cohérence et confluence
2. Causalité
3. Normalisation relative
4. Approximations, stabilité, séquentialité
5. Le λ-calcul simplement typé
6. Les types d'ordre supérieur

Création de redex – le cas simple





Causalité : la création de redex

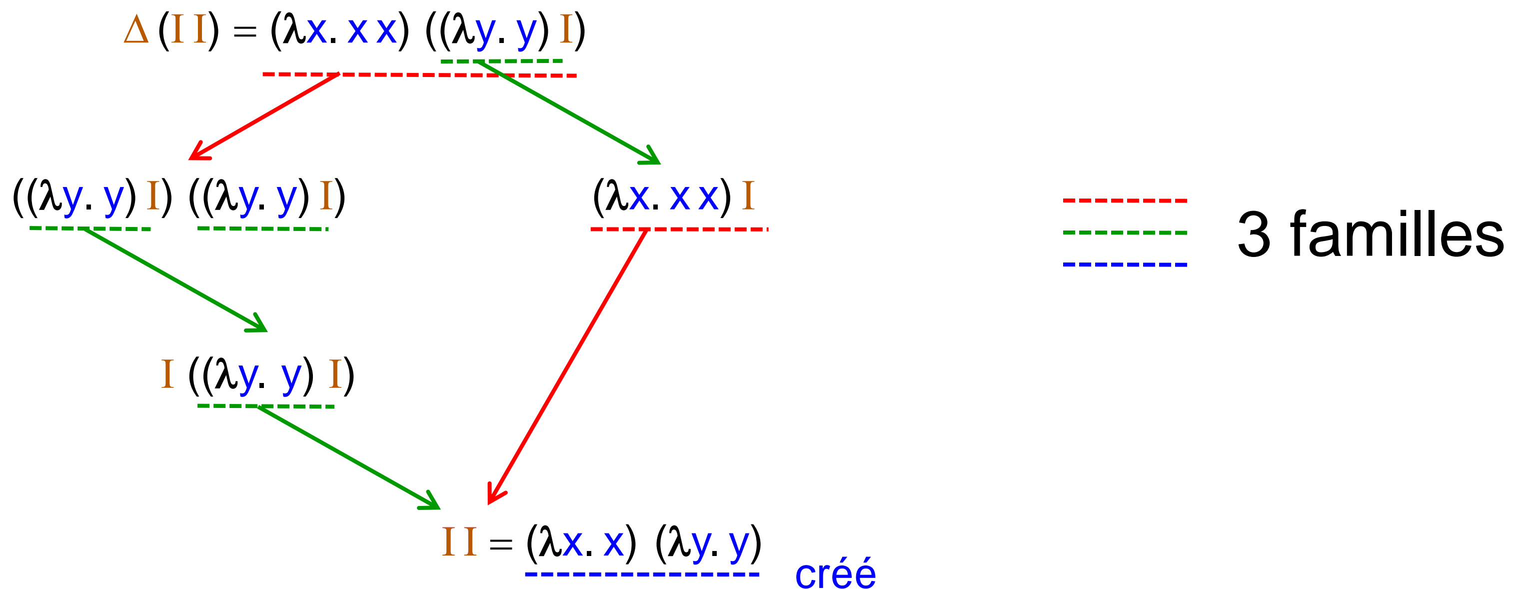


Agenda

1. Cohérence et confluence
2. Causalité
3. Normalisation relative
4. Approximations, stabilité, séquentialité
5. Le λ-calcul simplement typé
6. Les types d'ordre supérieur

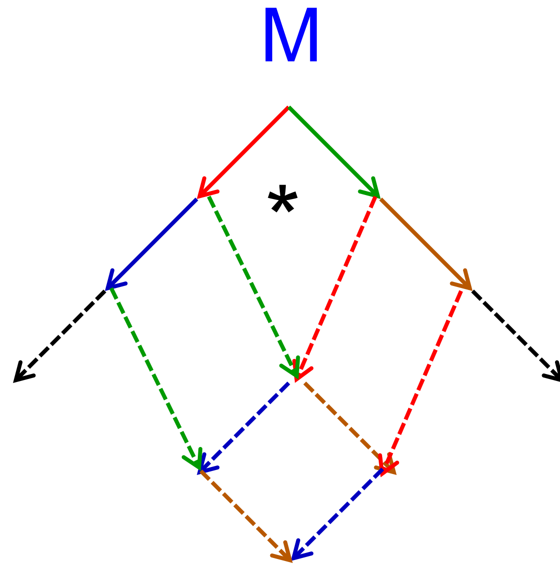
Développements finis généralisés

- Familles de redex d'un terme : les classes obtenues par fermeture réflexive, transitive et symétrique de résidu



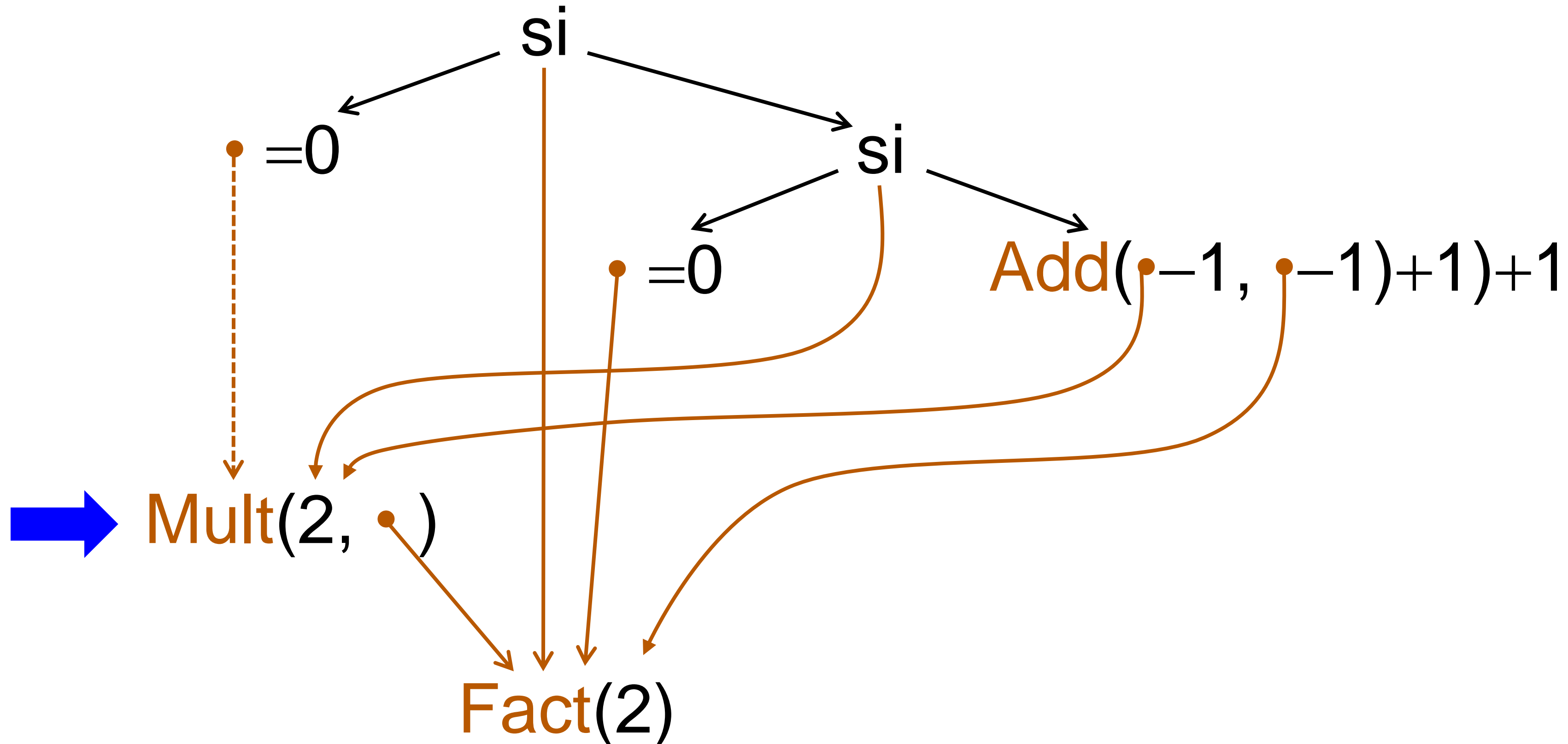
<u>Théorème (Lévy)</u> : toute réduction qui réduit seulement des redex d'un nombre fini de familles est finie

Corollaire immédiat : la confluence



La confluence locale n'introduit pas de nouvelles familles
le lemme de Newman s'applique aux réductions finies des familles de redex initiales

si Mult(2, Fact(2))=0 alors Fact(2)
sinon si Fact(2)=0 alors Mult(2, Fact(2))
sinon (Add(Mult(2, Fact(2))–1, Fact(2)–1)+1)+1



Calculer sur les arbres
en partageant les sous-arbres communs

Les résidus peuvent s'emboîter !

$(\lambda x.\, x\, x)\ (\lambda y.\, (\lambda z.z)\, y\, y)$

$\rightarrow (\lambda y.\, (\lambda z.z)\, y\, y)\ (\lambda y.\, (\lambda z.z)\, y\, y)$

$\rightarrow (\lambda z.z)\ (\lambda y.\, (\lambda z.z)\, y\, y)\ (\lambda y.\, (\lambda z.z)\, y\, y)$

<u>Théorème de Lévy</u> : partager les redex de même famille donnerait des réductions optimales.

<u>Abadi-Gonthier-Lévy</u> : mais les structures de données adaptées sont dures et chères!

Agenda

1. Cohérence et confluence
2. Causalité
3. Normalisation relative
4. Approximations, stabilité, séquentialité
5. Le λ-calcul simplement typé
6. Les types d’ordre supérieur

Le problème du ou parallèle

- En C, trois disjonctions booléennes sont possibles
  - OuS (e, e') = e | e' : boucle si e ou e' boucle
  - OuG (e, e') = e || e' : si e alors vrai sinon e'
    rend vrai si e rend vrai,
    même si e' boucle
  - OuD (e, e') = e' || e : symétrique
- Le parallélisme est-il possible ?
  - existe-t-il OuP (e, e') qui rend vrai si l'un de e ou e' rend vrai, même si l'autre boucle ?

Réponse : non !

Approximations de λ-termes

- On ajoute un nouveau terme Ω, sans réduction. Un Ω-terme est un terme avec des Ω.
- Pour un terme M, le terme M_Ω est obtenu par remplacement des redex externes de M par Ω
- On écrit $M \subseteq N$ si on peut obtenir N en remplaçant des Ω de M par des Ω-termes quelconques

$$\lambda x.\, x\, ((\lambda y.\, y)\, x)\, (z\, ((\lambda u.\, u)\, x)) \qquad \lambda x.\, x\, P\, (z\, \Omega)$$

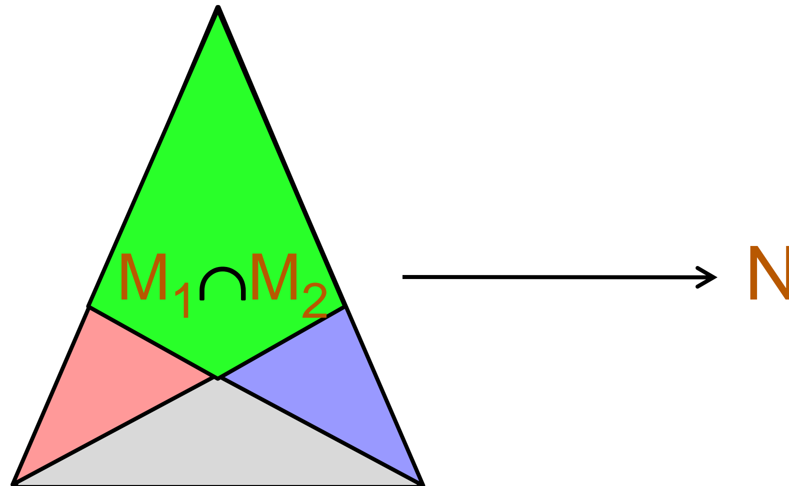
$$\supseteq \qquad \subseteq$$

$$\lambda x.\, x\, \Omega\, (z\, \Omega)$$

Théorèmes sur l'approximation

<u>Théorème de monotonie</u> : si $M \subseteq N$ et si M a une forme normale sans Ω, alors N a la même forme normale

<u>Théorème de stabilité</u> : si $M_1 \subseteq M$ et $M_2 \subseteq M$ et si M_1 et M_2 ont une forme normale N sans Ω, alors $M_1 \cap M_2$ a la même forme normale.



L'impossibilité du vrai parallélisme

Corollaire : Il n'existe pas de terme « ou parallèle » OuP tel que

$M_1 = \text{OuP vrai } \Omega \rightarrow^* \text{vrai}$

$M_2 = \text{OuP } \Omega \text{ vrai} \rightarrow^* \text{vrai}$

$M_3 = \text{OuP faux faux} \rightarrow^* \text{faux}$

Preuve : Si OuP existe, on pose $M = \text{OuP vrai vrai}$

On a $M_1 \subseteq M$ et $M_2 \subseteq M$

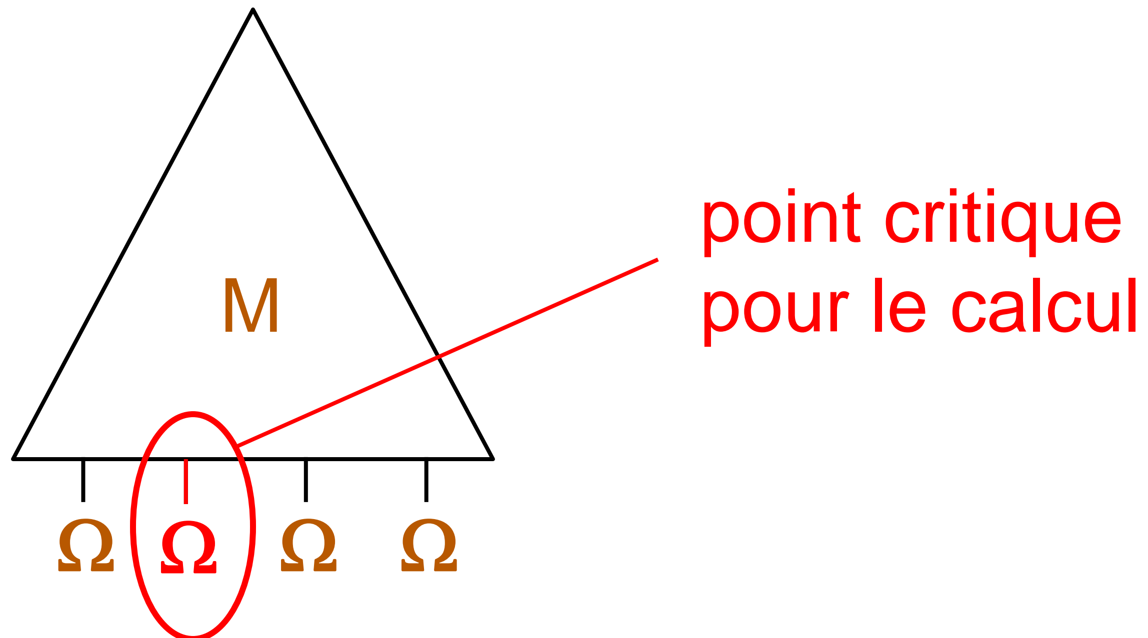
stabilité : $M_1 \rightarrow^* \text{vrai}$ et $M_2 \rightarrow^* \text{vrai} \Rightarrow M_1 \cap M_2 = \text{Oup } \Omega\ \Omega \rightarrow^* \text{vrai}$

monotonie : $\text{Oup } \Omega\ \Omega \rightarrow^* \text{vrai} \Rightarrow \text{Oup faux faux} \rightarrow^* \text{vrai}$

confluence : contredit $\text{Oup faux faux} \rightarrow^* \text{faux}$, contradiction !

La séquentialité

- Théorème de séquentialité : si M ⊂ N, si M n'a pas de forme normale mais N en a une, alors il existe un Ω dans M qui doit devenir non-Ω dans tout P ⊇ M qui a une forme normale.



Corollaire

- Corollaire : la fonction de Gustave suivante n'est pas définissable :

  Gus Ω vrai faux $\rightarrow^*$ vrai

  Gus faux Ω vrai $\rightarrow^*$ vrai

  Gus vrai faux Ω $\rightarrow^*$ vrai

- Preuve : le terme Gus Ω Ω Ω n'a pas de Ω critique

Agenda

1. Cohérence et confluence
2. Causalité
3. Normalisation relative
4. Approximations, stabilité, séquentialité
5. Le λ-calcul simplement typé
6. Les types d’ordre supérieur

Le λ-calcul simplement typé

$f : D \to E,\ \ x : D \Rightarrow f\,x : E$

$g : E \to F$

$g \;\; f : D \to F$

~~$: (E \to F) \times (D \to E) \to (D \to F)$~~

$= \lambda f.\, \lambda g.\, \lambda x.\, g(f(x))$

~~$: (E \to F) \to ((D \to E) \to (D \to F))$~~

$: (E \to F) \to (D \to E) \to D \to F$

- Variables de type : σ, τ, υ, ...
- Si σ et τ sont des types, alors $(\sigma \to \tau)$ est un type
- Un terme bien typé M^{σ} doit vérifier 4 conditions:
  1. Tout sous-terme doit être bien typé
  2. Toute abstraction doit avoir la forme $(\lambda x^{\sigma}. M^{\tau})^{(\sigma \to \tau)}$
  3. Toute application doit avoir la forme $(M^{(\sigma \to \tau)}\ N^{\sigma})^{\tau}$
  4. Toute les occurrences libres ou liées par le même lieur doivent avoir le même type

     $\ldots\ x^{\sigma} \ldots\ x^{\tau}\ \ldots \Rightarrow\ \sigma = \tau$

     $\lambda y^{\sigma}.\ \ldots\ y^{\tau} \ldots\ y^{\upsilon} \ldots \Rightarrow\ \sigma = \tau = \upsilon$

Un terme M est typable s'il existe des types pour lui

Exemples

$id_D : (D \to D) \qquad \mathrm{I} = (\lambda x^{\sigma}.\ x^{\sigma})^{(\sigma\to\sigma)}$

$: (E\to F)\to(D\to E)\to D\to F$

$= (\lambda g^{(\tau\to\upsilon)}.\lambda f^{(\sigma\to\tau)}.\lambda x^{\sigma}.\ (g^{(\tau\to\upsilon)}\ (f^{(\sigma\to\tau)}\ (x^{\sigma}))^{\tau})^{\upsilon})^{(\tau\to\upsilon)\to(\sigma\to\tau)\to(\sigma\to\upsilon)}$

ouch! ☹

$= \lambda g^{(\tau\to\upsilon)}.\lambda f^{(\sigma\to\tau)}.\lambda x^{\sigma}.\ g(f\,x)$

mieux ☺

$\underline{2} = \lambda f^{(\sigma\to\sigma)}.\lambda x^{\sigma}.\ (f(f\,x))^{\sigma}$ type $(\sigma\to\sigma)\to(\sigma\to\sigma)$

~~$\Delta = \lambda x^{\sigma}.\ x^{\sigma}\ x^{\sigma}$ demanderait $\exists\tau$ t.q. $\sigma = (\sigma\to\tau)$~~

La normalisation forte du λ-calcul typé

Proposition : Si M est typable de type σ et si $M \to^* N$ dans le calcul pur, alors N est typable de type σ

Théorème : Si M est typable, alors toutes les réductions de M sont finies

Preuve : par morphisme des familles de Lévy

Corollaire : il n'existe pas de combinateur de point fixe $Y^{(\sigma\to\sigma)\to\sigma}$ tel que $YM \to M(YM)$

Preuve : $Y^{(\sigma\to\sigma)\to\sigma} (\lambda x^\sigma.\ x^\sigma)$ bouclerait !

La limitation intrinsèque du λ -calcul typé

Théorème : sur les entiers de Church, seuls les polynômes étendus sont définissables

polynômes étendus : projections, constantes, test à 0, fermés par somme et produits

Augmenter la puissance en gardant la simplicité

- Ajouter explicitement ce qui a disparu
  - PCF : booléens, entiers, $Y^{(\sigma \to \sigma) \to \sigma}$ pour tout σ, cf cours 3
  - ML / CAML : types paramétriques, inférence de types, etc.
  - reste indécidable, mais raisonnablement domestiqué
- Rendre le système de types plus puissant
  - augmenter l'expressivité, préserver de la normalisation forte
  - Système T (Gödel)
  - Système F (Jean-Yves Girard, John Reynolds)
  - Théorie des constructions (G. Huet, T. Coquand)

Théorie des constructions

- On ne peut écrire que des programmes qui terminent, mais beaucoup de programmes qui terminent !
- La preuve de terminaison doit faire partie du programme
- On peut ensuite l'enlever et extraire automatiquement un programme CAML exécutable du terme source complet
- On peut aussi faire des maths vraiment formelles

CompCert, compilateur C prouvé (Leroy et. al.)

Théorème des 4 couleurs (Gonthier – Werner)

Agenda

1. Cohérence et confluence
2. Causalité
3. Normalisation relative
4. Le λ-calcul simplement typé
5. Approximations, stabilité, séquentialité
6. Les types d'ordre supérieur

Types d'ordre supérieur

- Système F (Jean-Yves Girard / John Reynolds) :
  - quantificateurs explicites pour les types
  - deux réductions, une pour les termes et une pour les types

$$\mathbf{I} = \Lambda\sigma.\ \lambda x^{\sigma}.\ x^{\sigma} \quad \text{de type } \Pi\sigma.(\sigma\rightarrow\sigma)$$

$$\mathbf{I}\ b = \lambda x^{b}.\ x^{b}$$

$$\mathbf{I}\ b\ \underline{\text{vrai}} = (\lambda x^{b}.\ x^{b})\ \underline{\text{vrai}} = \underline{\text{vrai}}$$

<u>Théorème</u> : un terme typable dans Système F est fortement normalisable

<u>Preuve</u> : dur !

<u>Relations avec les familles de redex</u> : inconnue

Avantages de Système F

- Permet des définitions de types très riches (record, listes, arbres, types récursifs, types existentiels, etc.)
- Ne permet de coder que des fonctions qui terminent, mais permet de coder toutes les fonctions primitives récursives, plus d'autres
- Mais la typabilité est indécidable et il n'y a pas d'inférence de types !

Exemple : produit $\sigma \times \tau$

- Lambda-calcul pur

$$\langle x, y \rangle = \lambda z.\ z\ x\ y$$

$$\pi_1 = \lambda p.\ p\ (\lambda x. \lambda y.\ x)$$

$$\pi_2 = \lambda p.\ p\ (\lambda x. \lambda y.\ y)$$

- Système F

$$\sigma \times \tau = \Pi\, \upsilon.\ (\sigma \rightarrow \tau \rightarrow \upsilon)$$

$$\langle x^{\sigma}, y^{\tau} \rangle = \Lambda \upsilon.\ \lambda z^{\sigma \rightarrow \tau \rightarrow \upsilon}.\ z\ x\ y$$

$$\pi_1 = \lambda p.\ p\ \sigma\ (\lambda x^{\sigma}. \lambda x^{\tau}.\ x)$$

$$\pi_2 = \lambda p.\ \tau\ (\lambda x^{\sigma}. \lambda x^{\tau}.\ y)$$

- Entiers de Church, de type $int =_d \Pi\, \sigma\, .\, (\sigma\to\sigma)\to(\sigma\to\sigma)$

  $\underline{0}^{int} =_d \Lambda\sigma.\ \lambda f^{(\sigma\to\sigma)}.\lambda x^{\sigma}.\ x$

  $\underline{+1}^{int} =_d \Lambda\sigma.\ \lambda n^{int}.\ \lambda f^{(\sigma\to\sigma)}.\lambda x^{\sigma}.\ f\ (n\ u\ f\ x)$

- $\underline{It}$: itération n fois d'une fonction $f^{(\sigma\to\sigma)}$ sur y^{σ}

  $\underline{It} =_d \lambda n^{int}\ .\ \lambda f^{(\sigma\to\sigma)}.\ \lambda y^{\sigma}.\ n\ \sigma\ f\ y$

  - $\underline{It}\ \underline{0}\ f\ y = \underline{0}\ \sigma\ f\ y = (\Lambda\sigma.\ \lambda f^{(\sigma\to\sigma)}.\lambda x^{\sigma}.\ x)\ \sigma\ f\ y = y$
  - $\underline{It}\ \underline{n+1}\ f\ y = \underline{n+1}\ \sigma\ \ f\ y$

    $= (\Lambda\sigma.\ \lambda f^{(\sigma\to\sigma)}.\lambda x^{\sigma}.\ f(\underline{n}\ \sigma\ f\ x))\ \sigma\ f\ y$

    $= f(n\ \sigma\ y\ f)$

    $= f(\underline{It}\ f\ n\ y)$

Récursion primitive

- $R =_d \lambda f^{(int \to \sigma \to \sigma)}. \lambda n^{int}. \lambda y^{\sigma}.$

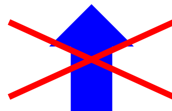
  $\pi_1(It\ n\ (\lambda x. \langle f(\pi_1\ x)\ (\pi_2\ x), \underline{+1}\ \pi_2\ x \rangle)\ \langle y, \underline{0} \rangle)$

$$R\ \underline{0}\ f\ y = y$$
$$R\ \underline{n+1}\ f\ y = f\ (R\ \underline{n}\ f\ y)\ \underline{n}$$

$$\underline{Pred} =_d R\ \underline{+1}$$

$$\underline{Pred}\ \underline{0} = \underline{0}$$
$$\underline{Pred}\ \underline{n+1} = \underline{n}$$

$$\underline{Pred}(\underline{+1}(\underline{+1}(\underline{+1}(\underline{+1}(\underline{0})))))$$



Agenda

1. Cohérence et confluence
2. Causalité
3. Normalisation relative
4. Le λ-calcul simplement typé
5. Approximations, stabilité, séquentialité
6. Les types d'ordre supérieur
7. Bonus: inférence de types en ML

Types polymorphes à la ML

1. Factoriser les types identiques

au lieu d'une identité $\mathrm{I}^{(\sigma\rightarrow\sigma)} = \lambda x^{\sigma}.\ x^{\sigma}$ pour chaque σ, avoir une seule identité $\mathrm{I}^{(\text{'a}\rightarrow\text{'a})}$ où 'a est un type générique, i.e. une variable de type quantifiée universellement.

2. Inférer les types pour ne pas les déclarer

```
# let rec append x L <=
      si null L alors cons x nil
                   sinon cons (hd L) (append x (tl L))
```

♥ append : ‘a →’a list→’a list

```
# let rec append x L <=
      si null L alors cons x nil
                   sinon cons (tl L) (append x (tl L))
```

♠ type-check error !

Conclusion

- Le λ-calcul est apparemment hyper-simple mais il est hyper-expressif.
- Les théorèmes syntaxiques sont fondamentaux mais durs.
- Les types d'ordre supérieurs permettent de définir toutes les fonctions totales d'intérêt => fondamental pour la preuve de programme (CoQ).
- Mais le λ-calcul reste strictement séquentiel.

Références (1)

- http://worrydream.com/AlligatorEggs/
  un très joli λ-calcul intuitif

- *Generalized Finite Developments*
  Jean-Jacques Lévy
  From Semantics to Computer Science, in honor of Gilles Kahn
  Cambridge University Press, 2009

- *The geometry of optimal lambda reduction*
  Georges Gonthier, Martin Abadi, Jean-Jacques Lévy
  Proc. ACM POPL conf., 1992

- *Proofs and Types*
  Jean-Yves Girard, traduit par Yves Lafont et Paul Taylor
  Cambridge University Press (1989).
  http://www.paultaylor.eu/stable/Proofs%2BTypes.html

Références (2)

- *Interactive Theorem Proving And Program Development. Coq'art: The Calculus Of Inductive Constructions*

  Yves Bertot, Pierre Castéran
  Springer, 2004

- *Formal Proof – The Four-Color Theorem*

  Georges Gonthier.
  Notices of the American Mathematical Society, 55(11), pp. 1382-1393 (Dec. 2008).