Calcul des constructions

David Delahaye David.Delahaye@lirmm.fr

Université de Montpellier Faculté des Sciences

Licence Informatique L3 2024-2025





L3 Info. 2024-2025

Quel cadre logique pour vérifier les programmes?

Où en sommes-nous?

- Logique du premier ordre (classique et intuitionniste).
- Systèmes de preuves formelles (LK, LJ).
- Raisonnement avec l'égalité (LK_{EQ}, LJ_{EQ}).

Il nous faut plus de théories

- Une théorie générale : théorie des ensembles.
- Permet de tout encoder (fonctions, entiers, etc.).

Quel cadre logique pour vérifier les programmes?

Où en sommes-nous?

- Logique du premier ordre (classique et intuitionniste).
- Systèmes de preuves formelles (LK, LJ).
- Raisonnement avec l'égalité (LK_{EQ}, LJ_{EQ}).

Il nous faut plus de théories

- Une théorie générale : théorie des ensembles.
- Permet de tout encoder (fonctions, entiers, etc.).

Théorie « naïve » des ensembles

Un peu d'histoire

- Fin du 19ème siecle, volonté de formaliser certaines théories.
- Par exemple : les ensembles, la géométrie, etc.
- Théorie « naïve » de Georg Cantor en 1878.
- Théorie incohérente : paradoxe de Bertrand Russell en 1902.
 - $R = \{x \mid x \notin x\}, \text{ deux cas } :$
 - * Soit $R \in R$, donc par définition de R, $R \notin R$.
 - * Soit $R \notin R$, donc par définition de R, $R \in R$.
 - Le problème vient de l'axiome de compréhension :
 - $\star \exists x. \forall y. y \in x \Leftrightarrow P(y).$
 - * R se définit en prenant $P(x) = x \notin x$.

- Deux possibilités qui ont donné deux théories.
- Théorie des ensembles de Zermelo-Fraenkel :
 - Introduite par Ernst Zermelo en 1908, étendue plus tard par Abraham Fraenkel en 1922.
 - Restriction de l'axiome de compréhension : $\forall z. \exists x. \forall y. y \in x \Leftrightarrow y \in z \land P(y)$.
- Théorie des types de Whitehead-Russell :
 - Introduite par Alfred Whitehead et Bertrand Russell en 1911.
 - On ne restreint pas l'axiome de compréhension.
 - On type les expressions de manière à éviter $x \notin x$ ou $x \in x$.
- Alternatives : théorie des ensembles de von Neumann-Bernays-Gödel, appelée aussi théorie des classes, introduite en 1925.

- Deux possibilités qui ont donné deux théories.
- Théorie des ensembles de Zermelo-Fraenkel :
 - Introduite par Ernst Zermelo en 1908, étendue plus tard par Abraham Fraenkel en 1922.
 - Restriction de l'axiome de compréhension : $\forall z. \exists x. \forall y. y \in x \Leftrightarrow y \in z \land P(y).$
- Théorie des types de Whitehead-Russell :
 - Introduite par Alfred Whitehead et Bertrand Russell en 1911.
 - On ne restreint pas l'axiome de compréhension.
 - On type les expressions de manière à éviter $x \notin x$ ou $x \in x$.
- Alternatives : théorie des ensembles de von Neumann-Bernays-Gödel, appelée aussi théorie des classes, introduite en 1925.

- Deux possibilités qui ont donné deux théories.
- Théorie des ensembles de Zermelo-Fraenkel :
 - Introduite par Ernst Zermelo en 1908, étendue plus tard par Abraham Fraenkel en 1922.
 - Restriction de l'axiome de compréhension : $\forall z. \exists x. \forall y. y \in x \Leftrightarrow y \in z \land P(y).$
- Théorie des types de Whitehead-Russell :
 - Introduite par Alfred Whitehead et Bertrand Russell en 1911.
 - On ne restreint pas l'axiome de compréhension.
 - On type les expressions de manière à éviter $x \notin x$ ou $x \in x$.
- Alternatives : théorie des ensembles de von Neumann-Bernays-Gödel, appelée aussi théorie des classes, introduite en 1925.

- Deux possibilités qui ont donné deux théories.
- Théorie des ensembles de Zermelo-Fraenkel :
 - Introduite par Ernst Zermelo en 1908, étendue plus tard par Abraham Fraenkel en 1922.
 - Restriction de l'axiome de compréhension : $\forall z. \exists x. \forall y. y \in x \Leftrightarrow y \in z \land P(y).$
- Théorie des types de Whitehead-Russell :
 - Introduite par Alfred Whitehead et Bertrand Russell en 1911.
 - On ne restreint pas l'axiome de compréhension.
 - On type les expressions de manière à éviter $x \notin x$ ou $x \in x$.
- Alternatives : théorie des ensembles de von Neumann-Bernays-Gödel, appelée aussi théorie des classes, introduite en 1925.

Que choisir?

- Inconvénients de la théorie des ensembles :
 - Statut des fonctions :
 - Les fonctions ne sont pas primitives (ensembles de couples).
 On doit introduire des notations (lieurs, application, domaines de définitions, etc.).
 - On ne peut pas calculer, tout repose sur le raisonnement. Pour démontrer la validité de 2 + 2 = 4, on doit utiliser les axiomes

$$\begin{cases} \forall y.o + y = y \\ \forall x, y.s(x) + y = s(x + y) \end{cases}$$

Statut des démonstrations :
Les démonstrations sont à part dans la théorie.
Ce sont des objets périphériques et non des objets du langage.
Ainsi, on peut démontrer que 5 est premier mais on ne peut pas exprimer qu'un certain objet est une démonstration de cela.

Que choisir?

- Inconvénients de la théorie des ensembles :
 - Statut des fonctions :
 - Les fonctions ne sont pas primitives (ensembles de couples).

 On doit introduire des notations (lieurs, application, domaines de la complexión de la complexión
 - On doit introduire des notations (lieurs, application, domaines de définitions, etc.).
 - On ne peut pas calculer, tout repose sur le raisonnement.
 - Pour démontrer la validité de 2 + 2 = 4, on doit utiliser les axiomes

$$\begin{cases} \forall y.o + y = y \\ \forall x, y.s(x) + y = s(x + y) \end{cases}$$

Statut des démonstrations :

Les démonstrations sont à part dans la théorie.

Ce sont des objets périphériques et non des objets du langage.

Ainsi, on peut démontrer que 5 est premier mais on ne peut pas exprimer qu'un certain objet est une démonstration de cela.

4□ > 4酉 > 900

Que choisir?

- Inconvénients de la théorie des ensembles :
 - Statut des fonctions :
 - Les fonctions ne sont pas primitives (ensembles de couples).
 On doit introduire des notations (lieurs, application, domaines de définitions, etc.).
 - On ne peut pas calculer, tout repose sur le raisonnement.
 - Pour démontrer la validité de 2 + 2 = 4, on doit utiliser les axiomes

$$\begin{cases} \forall y.o + y = y \\ \forall x, y.s(x) + y = s(x + y) \end{cases}$$

Statut des démonstrations :
Les démonstrations sont à part dans la théorie.
Ce sont des objets périphériques et non des objets du langage.
Ainsi, on peut démontrer que 5 est premier mais on ne peut pas exprimer qu'un certain objet est une démonstration de cela.

Que choisir?

- Inconvénients de la théorie des ensembles :
 - Statut des fonctions :
 - * Les fonctions ne sont pas primitives (ensembles de couples). On doit introduire des notations (lieurs, application, domaines de définitions, etc.).
 - * On ne peut pas calculer, tout repose sur le raisonnement. Pour démontrer la validité de 2+2=4, on doit utiliser les axiomes :

$$\begin{cases} \forall y.o + y = y \\ \forall x, y.s(x) + y = s(x + y) \end{cases}$$

Statut des démonstrations :
Les démonstrations sont à part dans la théorie.
Ce sont des objets périphériques et non des objets du langage.
Ainsi, on peut démontrer que 5 est premier mais on ne peut pas exprimer qu'un certain objet est une démonstration de cela.

Que choisir?

- Inconvénients de la théorie des ensembles :
 - Statut des fonctions :
 - Les fonctions ne sont pas primitives (ensembles de couples).
 On doit introduire des notations (lieurs, application, domaines de définitions, etc.).
 - On ne peut pas calculer, tout repose sur le raisonnement. Pour démontrer la validité de 2+2=4, on doit utiliser les axiomes :

$$\begin{cases} \forall y.o + y = y \\ \forall x, y.s(x) + y = s(x + y) \end{cases}$$

Statut des démonstrations :
 Les démonstrations sont à part dans la théorie.
 Ce sont des objets périphériques et non des objets du langage.

Ainsi, on peut démontrer que 5 est premier mais on ne peut pas exprimer qu'un certain objet est une démonstration de cela.

Que choisir?

- Inconvénients de la théorie des ensembles :
 - Statut des fonctions :
 - Les fonctions ne sont pas primitives (ensembles de couples).
 On doit introduire des notations (lieurs, application, domaines de définitions, etc.).
 - * On ne peut pas calculer, tout repose sur le raisonnement. Pour démontrer la validité de 2+2=4, on doit utiliser les axiomes :

$$\begin{cases} \forall y.o + y = y \\ \forall x, y.s(x) + y = s(x + y) \end{cases}$$

Statut des démonstrations :
 Les démonstrations sont à part dans la théorie.
 Ce sont des objets périphériques et non des objets du langage.
 Ainsi, on peut démontrer que 5 est premier mais on ne peut pas exprimer qu'un certain objet est une démonstration de cela.

- Introduire du calcul dans le langage.
 - On souhaite avoir une représentation intentionnelle des fonctions :
 - Les fonctions sont des objets du langage (contrairement à la théorie des ensembles où elles sont un cas particulier de prédicats).
 - Une fonction peut être définie par un algorithme la calculant.
 - Théorie des types simples de Church en 1940, logique d'ordre supérieur basée sur le λ -calcul typé.

- Introduire du calcul dans le langage.
 - On souhaite avoir une représentation intentionnelle des fonctions :
 - Les fonctions sont des objets du langage (contrairement à la théorie des ensembles où elles sont un cas particulier de prédicats).
 - Une fonction peut être définie par un algorithme la calculant
 - Théorie des types simples de Church en 1940, logique d'ordre supérieur basée sur le λ -calcul typé.

- Introduire du calcul dans le langage.
 - On souhaite avoir une représentation intentionnelle des fonctions :
 - Les fonctions sont des objets du langage (contrairement à la théorie des ensembles où elles sont un cas particulier de prédicats).
 - * Une fonction peut être définie par un algorithme la calculant.
 - Théorie des types simples de Church en 1940, logique d'ordre supérieur basée sur le λ -calcul typé.

- Introduire du calcul dans le langage.
 - On souhaite avoir une représentation intentionnelle des fonctions :
 - Les fonctions sont des objets du langage (contrairement à la théorie des ensembles où elles sont un cas particulier de prédicats).
 - * Une fonction peut être définie par un algorithme la calculant.
 - Théorie des types simples de Church en 1940, logique d'ordre supérieur basée sur le λ -calcul typé.

- Les démonstrations comme objets du langage.
 - Interprétation de Brouwer-Heyting-Kolmogorov en logique intuitionniste : les démonstrations sont vues comme des fonctions.
 - Isomorphisme (correspondance) de Curry-Howard :
 - Proposition = type et preuve = fonction.
 - Haskell Curry en 1958, Nicolaas de Bruijn en 1968,
 - William Alvin Howard en 1969.

- Les démonstrations comme objets du langage.
 - Interprétation de Brouwer-Heyting-Kolmogorov en logique intuitionniste : les démonstrations sont vues comme des fonctions.
 - ▶ Isomorphisme (correspondance) de Curry-Howard :
 - * Proposition = type et preuve = fonction.

 Haskell Curry en 1958, Nicolaas de Bruijn en 1968,

- Les démonstrations comme objets du langage.
 - Interprétation de Brouwer-Heyting-Kolmogorov en logique intuitionniste : les démonstrations sont vues comme des fonctions.
 - Isomorphisme (correspondance) de Curry-Howard :
 - ★ Proposition = type et preuve = fonction.
 - Haskell Curry en 1958, Nicolaas de Bruijn en 1968, William Alvin Howard en 1969.

- Calcul des constructions (CC) :
 - Introduit par Thierry Coquand et Gérard Huet en 1985.
 - \triangleright Basé sur un λ -calcul typé et sur l'isomorphisme de Curry-Howard.
 - Polymorphisme, types dépendants, constructions.
- On aura aussi besoin de l'induction :
 - Calcul des constructions inductives (prochain cours).
 - Introduit par Thierry Coquand et Christine Paulin-Mohring en 1989.
 - Ajout des types inductifs primitifs
 - Coq est basé sur ce calcul

- Calcul des constructions (CC) :
 - Introduit par Thierry Coquand et Gérard Huet en 1985.
 - \triangleright Basé sur un λ -calcul typé et sur l'isomorphisme de Curry-Howard.
 - Polymorphisme, types dépendants, constructions.
- On aura aussi besoin de l'induction :
 - Calcul des constructions inductives (prochain cours).
 - Introduit par Thierry Coquand et Christine Paulin-Mohring en 1989.
 - Ajout des types inductifs primitifs.
 - Coq est basé sur ce calcul.

Syntaxe (abstraite)

- On fixe un ensemble de variables $x, y, ... \in \mathbb{V}$;
- ullet L'ensemble des termes ${\mathcal T}$ est le plus petit ensemble tel que :
 - ▶ Si $x \in \mathbb{V}$ alors $x \in \mathcal{T}$ (variable);
 - ▶ Si $x \in \mathbb{V}$ et $M \in \mathcal{T}$ alors $\lambda x.M \in \mathcal{T}$ (fonction ou λ -terme);
 - ▶ Si $M, N \in \mathcal{T}$ alors $M, N \in \mathcal{T}$ (application).

Notation pointée

- La portée d'un lieur (λ) va jusqu'à la parenthèse fermante du terme du lieur;
- Si le terme du lieur n'est pas parenthésé, la portée du lieur va jusqu'à la fin du terme;
- Donc, si on veut arrêter la portée d'un lieur, il suffit d'utiliser des parenthèses pour limiter explicitement la portée du lieur;
- Exemples :
 - $\lambda x.f \times y \equiv \lambda x.(f \times y);$
 - Si on veut que le λ ne porte que sur f(x), on doit écrire : $(\lambda x.f(x))y$.
- Notation : $\lambda x, y.M \equiv \lambda x.\lambda y.M$.

Associativité

- L'application associe à gauche :
 - $M N L \equiv ((M N) L).$

Variables libres, variables liées

- Une variable x est libre dans un terme M si et seulement s'il existe une occurrence de x dans M qui n'est sous la portée d'un lieur;
- Une variable x est liée dans un terme M si et seulement s'il existe une occurrence de x dans M qui est sous la portée d'un lieur;
- Occurrence \equiv position d'un terme dans un terme.

Variables libres (« Free Variables »)

- L'ensemble des variables libres d'un terme M, noté FV(M), est défini par récurrence structurelle par :
 - ▶ Si $x \in \mathbb{V}$ alors $FV(x) = \{x\}$;
 - ▶ Si $x \in \mathbb{V}$ et $M \in \mathcal{T}$ alors $FV(\lambda x.M) = FV(M) \{x\}$;
 - Si $M, N \in \mathcal{T}$ alors $FV(M, N) = FV(M) \cup FV(N)$.

Variables liées (« Bound Variables »)

- L'ensemble des variables liées d'un terme M, noté BV(M), est défini par récurrence structurelle par :
 - ▶ Si $x \in \mathbb{V}$ alors $BV(x) = \emptyset$;
 - ► Si $x \in \mathbb{V}$ et $M \in \mathcal{T}$ alors $BV(\lambda x.M) = BV(M) \cup \{x\}$;
 - ▶ Si $M, N \in \mathcal{T}$ alors $BV(M, N) = BV(M) \cup BV(N)$.

Variables libres, variables liées : exemples

- y est libre dans $\lambda x.f \times y$;
- x est liée dans $\lambda x.f \times y$;
- Dans la formule $\lambda x.f \times y$:
 - L'ensemble des variables libres est $\{f, y\}$;
 - L'ensemble des variables liées est $\{x\}$.
- Une variable peut être libre et liée à la fois (c'est-à-dire qu'elle possède une occurrence où elle est libre et une autre où elle est liée), par exemple : (λx.f x y) x, où x est libre (deuxième occurrence) et liée (première occurrence) à la fois.

Substitution

- M[L/y] est la substitution dans le terme M de toutes les occurrences libres de la variable y par le terme L;
- Elle est définie par récurrence structurelle par :

Si
$$x \in \mathbb{V}$$
 alors $x[L/y] = \begin{cases} L, \text{ si } x = y \\ x, \text{ sinon} \end{cases}$

- Si $x \in \mathbb{V}$ et $M \in \mathcal{T}$ alors $(\lambda x.M)[L/y] = \begin{cases} \lambda x.M, \text{ si } x = y \\ \lambda x.M[L/y], \text{ sinor} \end{cases}$
- Si $M, N \in \mathcal{T}$ alors $(M \ N)[L/y] = M[L/y] \ N[L/y]$.

Exemples

- $(\lambda z.x)[y/x] = \lambda z.y$;
- $((\lambda x.f \ x) \ x \ z)[y/x] = (\lambda x.f \ x) \ y \ z.$



Substitution

- M[L/y] est la substitution dans le terme M de toutes les occurrences libres de la variable y par le terme L;
- Elle est définie par récurrence structurelle par :

Si
$$x \in \mathbb{V}$$
 alors $x[L/y] = \begin{cases} L, \text{ si } x = y \\ x, \text{ sinon} \end{cases}$

- Si $x \in \mathbb{V}$ et $M \in \mathcal{T}$ alors $(\lambda x.M)[L/y] = \begin{cases} \lambda x.M, \text{ si } x = y \\ \lambda x.M[L/y], \text{ sinon} \end{cases}$
- Si $M, N \in \mathcal{T}$ alors $(M \ N)[L/y] = M[L/y] \ N[L/y]$.

Exemples

- $(\lambda z.x)[y/x] = \lambda z.y$;
- $((\lambda x.f \ x) \ x \ z)[y/x] = (\lambda x.f \ x) \ y \ z.$



Substitution

- M[L/y] est la substitution dans le terme M de toutes les occurrences libres de la variable y par le terme L;
- Elle est définie par récurrence structurelle par :

Si
$$x \in \mathbb{V}$$
 alors $x[L/y] = \begin{cases} L, \text{ si } x = y \\ x, \text{ sinon} \end{cases}$

- Si $x \in \mathbb{V}$ et $M \in \mathcal{T}$ alors $(\lambda x.M)[L/y] = \begin{cases} \lambda x.M, \text{ si } x = y \\ \lambda x.M[L/y], \text{ sinon} \end{cases}$
- ▶ Si $M, N \in \mathcal{T}$ alors $(M \ N)[L/y] = M[L/y] \ N[L/y]$.

Exemples

- $(\lambda z.x)[y/x] = \lambda z.y$;
- $((\lambda x.f \ x) \ x \ z)[y/x] = (\lambda x.f \ x) \ y \ z.$



Substitution sûre

- Capture de variables :
 - $[\lambda y.x][y/x] = \lambda y.y$: l'occurrence libre de y devient liée.
- La substitution M[N/x] est dite sûre si $BV(M) \cap FV(N) = \emptyset$;
- On ne considérera que des substitutions sûres et on peut toujours se ramener à des substitutions sûres par α -renommage (par renommage des variables liées par des variables « fraîches » non libres).

Substitution sûre

- Capture de variables :
 - $(\lambda y.x)[y/x] = \lambda y.y$: l'occurrence libre de y devient liée.
- La substitution M[N/x] est dite sûre si $BV(M) \cap FV(N) = \emptyset$;
- On ne considérera que des substitutions sûres et on peut toujours se ramener à des substitutions sûres par α -renommage (par renommage des variables liées par des variables « fraîches » non libres).

lpha-renommage

- $\lambda x.M \leadsto_{\alpha} \lambda y.M[y/x]$, si $y \notin FV(M)$;
- La relation \to_{α} est la plus petite relation contenant la relation \leadsto_{α} et qui passe au contexte :
 - $\lambda x.M \rightarrow_{\alpha} \lambda x.M'$, si $M \rightarrow_{\alpha} M'$;
 - $M N \rightarrow_{\alpha} M' N$, si $M \rightarrow_{\alpha} M'$;
 - $M N \rightarrow_{\alpha} M N'$, si $N \rightarrow_{\alpha} N'$.
- La relation $=_{\alpha}$ est la clôture réflexive, symétrique et transitive de la relation \to_{α} ;
- On dira que deux termes M et N sont α -équivalents si et seulement si on a $M=_{\alpha}N$;
- Dans la suite, on quotientera l'ensemble $\mathcal T$ des termes par la relation $=_{\alpha}$, c'est-à-dire que deux termes α -équivalents seront considérés comme identiques.

lpha-renommage

- $\lambda x.M \leadsto_{\alpha} \lambda y.M[y/x]$, si $y \notin FV(M)$;
- La relation \to_{α} est la plus petite relation contenant la relation \leadsto_{α} et qui passe au contexte :
 - $\lambda x.M \rightarrow_{\alpha} \lambda x.M'$, si $M \rightarrow_{\alpha} M'$;
 - $M N \rightarrow_{\alpha} M' N$, si $M \rightarrow_{\alpha} M'$;
 - $M N \rightarrow_{\alpha} M N'$, si $N \rightarrow_{\alpha} N'$.
- La relation $=_{\alpha}$ est la clôture réflexive, symétrique et transitive de la relation \to_{α} ;
- On dira que deux termes M et N sont α -équivalents si et seulement si on a $M=_{\alpha}N$;
- Dans la suite, on quotientera l'ensemble $\mathcal T$ des termes par la relation $=_{\alpha}$, c'est-à-dire que deux termes α -équivalents seront considérés comme identiques.

lpha-renommage

- $\lambda x.M \leadsto_{\alpha} \lambda y.M[y/x]$, si $y \notin FV(M)$;
- La relation \to_{α} est la plus petite relation contenant la relation \leadsto_{α} et qui passe au contexte :
 - $\lambda x.M \rightarrow_{\alpha} \lambda x.M'$, si $M \rightarrow_{\alpha} M'$;
 - $M N \rightarrow_{\alpha} M' N$, si $M \rightarrow_{\alpha} M'$;
 - $M N \rightarrow_{\alpha} M N'$, si $N \rightarrow_{\alpha} N'$.
- La relation =_α est la clôture réflexive, symétrique et transitive de la relation →_α;
- On dira que deux termes M et N sont α -équivalents si et seulement si on a $M=_{\alpha}N$;
- Dans la suite, on quotientera l'ensemble $\mathcal T$ des termes par la relation $=_{\alpha}$, c'est-à-dire que deux termes α -équivalents seront considérés comme identiques.

lpha-renommage

- $\lambda x.M \leadsto_{\alpha} \lambda y.M[y/x]$, si $y \notin FV(M)$;
- La relation \to_{α} est la plus petite relation contenant la relation \leadsto_{α} et qui passe au contexte :
 - $\lambda x.M \rightarrow_{\alpha} \lambda x.M'$, si $M \rightarrow_{\alpha} M'$;
 - $M N \rightarrow_{\alpha} M' N$, si $M \rightarrow_{\alpha} M'$;
 - $M N \rightarrow_{\alpha} M N'$, si $N \rightarrow_{\alpha} N'$.
- La relation $=_{\alpha}$ est la clôture réflexive, symétrique et transitive de la relation \to_{α} ;
- On dira que deux termes M et N sont α -équivalents si et seulement si on a $M =_{\alpha} N$;
- Dans la suite, on quotientera l'ensemble $\mathcal T$ des termes par la relation $=_{\alpha}$, c'est-à-dire que deux termes α -équivalents seront considérés comme identiques.

lpha-renommage

- $\lambda x.M \leadsto_{\alpha} \lambda y.M[y/x]$, si $y \notin FV(M)$;
- La relation \to_{α} est la plus petite relation contenant la relation \leadsto_{α} et qui passe au contexte :
 - $\lambda x.M \rightarrow_{\alpha} \lambda x.M'$, si $M \rightarrow_{\alpha} M'$;
 - $M N \rightarrow_{\alpha} M' N$, si $M \rightarrow_{\alpha} M'$;
 - $M N \rightarrow_{\alpha} M N'$, si $N \rightarrow_{\alpha} N'$.
- La relation =_α est la clôture réflexive, symétrique et transitive de la relation →_α;
- On dira que deux termes M et N sont α -équivalents si et seulement si on a $M=_{\alpha}N$;
- Dans la suite, on quotientera l'ensemble $\mathcal T$ des termes par la relation $=_{\alpha}$, c'est-à-dire que deux termes α -équivalents seront considérés comme identiques.

β -réduction

- $(\lambda x.M)$ $N \leadsto_{\beta} M[N/x]$, si la substitution est sûre;
- La relation \to_{β} est la plus petite relation contenant la relation \leadsto_{β} et qui passe au contexte ;
- La relation \to_{eta}^* est la clôture réflexive et transitive de la relation \to_{eta} ;
- La relation $=_{\beta}$ est la clôture symétrique de la relation \to_{β}^* .

- $(\lambda x.x) y \rightarrow_{\beta} y$;
- $(\lambda x, y.f \times y) t u \rightarrow_{\beta}^* f t u;$
- $(\lambda x.(\lambda y.f\ y)\ x)\ t \rightarrow_{\beta}^* f\ t.$



β -réduction

- $(\lambda x.M) \ N \leadsto_{\beta} M[N/x]$, si la substitution est sûre;
- La relation \to_{β} est la plus petite relation contenant la relation \leadsto_{β} et qui passe au contexte;
- ullet La relation $ullet_{eta}^*$ est la clôture réflexive et transitive de la relation $ullet_{eta}$;
- La relation $=_{\beta}$ est la clôture symétrique de la relation \to_{β}^* .

- $(\lambda x.x) y \rightarrow_{\beta} y$;
- $(\lambda x, y.f \times y) t u \rightarrow_{\beta}^* f t u$;
- $(\lambda x.(\lambda y.f\ y)\ x)\ t \to_{\beta}^* f\ t.$



β -réduction

- $(\lambda x.M) \ N \leadsto_{\beta} M[N/x]$, si la substitution est sûre;
- La relation \to_β est la plus petite relation contenant la relation \leadsto_β et qui passe au contexte ;
- ullet La relation o_{eta}^* est la clôture réflexive et transitive de la relation o_{eta} ;
- La relation $=_{\beta}$ est la clôture symétrique de la relation \to_{β}^* .

- $(\lambda x.x) y \rightarrow_{\beta} y$;
- $(\lambda x, y.f \times y) t u \rightarrow_{\beta}^* f t u;$
- $(\lambda x.(\lambda y.f\ y)\ x)\ t \rightarrow_{\beta}^* f\ t.$



β -réduction

- $(\lambda x.M) \ N \leadsto_{\beta} M[N/x]$, si la substitution est sûre;
- La relation \to_β est la plus petite relation contenant la relation \leadsto_β et qui passe au contexte ;
- La relation \to_{eta}^* est la clôture réflexive et transitive de la relation \to_{eta} ;
- La relation $=_{\beta}$ est la clôture symétrique de la relation \to_{β}^* .

- $(\lambda x.x) y \rightarrow_{\beta} y$;
- $(\lambda x, y.f \times y) t u \rightarrow_{\beta}^* f t u;$
- $(\lambda x.(\lambda y.f\ y)\ x)\ t \rightarrow_{\beta}^* f\ t.$



β -réduction

- $(\lambda x.M) \ N \leadsto_{\beta} M[N/x]$, si la substitution est sûre;
- La relation \rightarrow_{β} est la plus petite relation contenant la relation \leadsto_{β} et qui passe au contexte;
- La relation \to_{eta}^* est la clôture réflexive et transitive de la relation \to_{eta} ;
- La relation $=_{\beta}$ est la clôture symétrique de la relation \to_{β}^* .

- $(\lambda x.x) y \rightarrow_{\beta} y$;
- $(\lambda x, y.f \times y) t u \rightarrow_{\beta}^* f t u$;
- $(\lambda x.(\lambda y.f\ y)\ x)\ t \to_{\beta}^* f\ t.$



Rédex et forme normale

- Un rédex est un terme de la forme $(\lambda x.M)$ N;
- Un terme est en forme normale s'il ne contient aucun rédex;
- Un terme en forme normale ne peut plus être β -réduit.

- $\lambda x.f x$ est en forme normale;
- $(\lambda x.f \ x) \ y$ n'est pas en forme normale.

Normalisation

- Un terme E est (faiblement) normalisable s'il existe un terme E' en forme normale tel que $E \to_{\beta}^* E'$;
- Un terme *E* est fortement normalisable si toutes les réductions à partir de *E* sont finies.

Exemples

- $(\lambda x.f \ x)$ y est fortement normalisable, la forme normale est f y;
- Si $\Delta = \lambda x.x \ x$, $\Omega = \Delta \Delta$ n'est pas normalisable, $(\lambda x.x \ x) \ (\lambda x.x \ x) \rightarrow_{\beta} (\lambda x.x \ x) \ (\lambda x.x \ x) \rightarrow_{\beta} ...;$
- $(\lambda x.y)$ Ω est normalisable, la forme normale est y, mais il n'est pas fortement normalisable car $\Omega \to_{\beta} \Omega \to_{\beta} \ldots$, donc $(\lambda x.y)$ $\Omega \to_{\beta} (\lambda x.y)$ $\Omega \to_{\beta} \ldots$

< □ > < □ > 9 Q (P

Théorèmes fondamentaux

- Théorème de Church-Rosser : si E_1 et E_2 sont des termes tels que $E_1 =_{\beta} E_2$, alors il existe un terme E tel que $E_1 \to_{\beta}^* E$ et $E_2 \to_{\beta}^* E$.
- Théorème de confluence (ou du losange) : Si E, E_1 et E_2 sont des termes tels que $E \to_{\beta}^* E_1$ et $E \to_{\beta}^* E_2$, alors il existe un terme E' te que $E_1 \to_{\beta}^* E'$ et $E_2 \to_{\beta}^* E'$;
- Le théorème de confluence implique l'unicité de la forme normale.

Théorèmes fondamentaux

- Théorème de Church-Rosser : si E_1 et E_2 sont des termes tels que $E_1 =_{\beta} E_2$, alors il existe un terme E tel que $E_1 \to_{\beta}^* E$ et $E_2 \to_{\beta}^* E$.
- Théorème de confluence (ou du losange) : Si E, E_1 et E_2 sont des termes tels que $E \to_{\beta}^* E_1$ et $E \to_{\beta}^* E_2$, alors il existe un terme E' tel que $E_1 \to_{\beta}^* E'$ et $E_2 \to_{\beta}^* E'$;
- Le théorème de confluence implique l'unicité de la forme normale.

Théorèmes fondamentaux

- Théorème de Church-Rosser : si E_1 et E_2 sont des termes tels que $E_1 =_{\beta} E_2$, alors il existe un terme E tel que $E_1 \to_{\beta}^* E$ et $E_2 \to_{\beta}^* E$.
- Théorème de confluence (ou du losange) : Si E, E_1 et E_2 sont des termes tels que $E \to_{\beta}^* E_1$ et $E \to_{\beta}^* E_2$, alors il existe un terme E' tel que $E_1 \to_{\beta}^* E'$ et $E_2 \to_{\beta}^* E'$;
- Le théorème de confluence implique l'unicité de la forme normale.

- $true = \lambda a, b.a \text{ et } false = \lambda a, b.b;$
- On a : $true \times y \rightarrow^*_{\beta} x$ et $false \times y \rightarrow^*_{\beta} y$;
- Définition de l'alternative ifthenelse = λb , u, v.b u v:
 - ifthenelse true x $y = (\lambda b, u, v.b u v)$ true x $y \to_{\beta}^*$ true x $y \to_{\beta}^* x$
 - ifthenelse false x $y = (\lambda b, u, v.b \ u \ v)$ false x $y \to_{\beta}^*$ false x $y \to_{\beta}^*$ y

- true = λa , b.a et false = λa , b.b;
- On a : true x $y \to_{\beta}^* x$ et false x $y \to_{\beta}^* y$;

- $true = \lambda a, b.a$ et $false = \lambda a, b.b$;
- On a : true x $y \rightarrow_{\beta}^* x$ et false x $y \rightarrow_{\beta}^* y$;
- Définition de l'alternative ifthenelse = λb , u, v.b u v:

```
ifthenelse true x y=(\lambda b,u,v.b\;u\;v) true x y\to_{\beta}^* true x y\to_{\beta}^*x; ifthenelse false x y=(\lambda b,u,v.b\;u\;v) false x y\to_{\beta}^* false x y\to_{\beta}^*y
```

- $true = \lambda a, b.a$ et $false = \lambda a, b.b$;
- On a : true x $y \rightarrow_{\beta}^* x$ et false x $y \rightarrow_{\beta}^* y$;
- Définition de l'alternative ifthenelse = λb , u, v.b u v:
 - ifthenelse true x $y = (\lambda b, u, v.b u v)$ true x $y \to_{\beta}^*$ true x $y \to_{\beta}^* x$; ifthenelse false x $y = (\lambda b, u, v.b u v)$ false x $y \to_{\beta}^*$ false x $y \to_{\beta}^*$ y

- $true = \lambda a, b.a$ et $false = \lambda a, b.b$;
- On a : true $x \ y \to_{\beta}^* x$ et false $x \ y \to_{\beta}^* y$;
- Définition de l'alternative ifthenelse = λb , u, v.b u v:
 - if then else true x $y = (\lambda b, u, v.b u v)$ true x $y \to_{\beta}^*$ true x $y \to_{\beta}^* x$;
 - if then else false x $y = (\lambda b, u, v.b u v)$ false x $y \to_{\beta}^*$ false x $y \to_{\beta}^* y$.

Structures de données : les entiers ou itérateurs de Church

• Itérer une fonction f, n fois :

```
0 = \lambda f, x.x;
1 = \lambda f, x.f x;
2 = \lambda f, x.f (f x);
\dots
n = \lambda f, x.f (f (...(f x))) = \lambda f, x.f^n x, \text{ avec } f \text{ itérée } n \text{ fois.}
```

- Fonction successeur, deux solutions :
 - Ajouter une itération de f en tête : $\lambda n, f, x.f$ (n f x);
 - Ajouter une itération de f en queue : $\lambda n, f, x.n f$ (f x)

Structures de données : les entiers ou itérateurs de Church

• Itérer une fonction f, n fois :

```
0 = \lambda f, x.x;
1 = \lambda f, x.f x;
2 = \lambda f, x.f (f x);
...
n = \lambda f, x.f (f (...(f x))) = \lambda f, x.f^n x, \text{ avec } f \text{ itérée } n \text{ fois.}
```

- Fonction successeur, deux solutions :
 - Ajouter une itération de f en tête : $\lambda n, f, x.f$ (n f x);
 - Ajouter une itération de f en queue : $\lambda n, f, x.n f$ (f x).

Récursion

- Théorème : tout terme F a un point fixe X, autrement dit un terme X tel que $X =_{\beta} F$ X. En fait, il existe un terme Y sans variable libre tel que pour tout F, Y F soit un point fixe de F. Un tel terme est appelé un combinateur de point fixe ;
- Combinateur de point fixe de Church/Curry :
- On a bien $Y F =_{\beta} F (Y F)$, mais pas $Y F \rightarrow_{\beta}^{*} F (Y F)$;
- Combinateur de point fixe de Turing : $\Theta = (\lambda g, h.h (g g h)) (\lambda g, h.h (g g h))$
- On a bien Θ $F =_{\beta} F$ (ΘF) et Θ $F \to_{\beta}^* F$ (ΘF) .

Récursion

- Théorème: tout terme F a un point fixe X, autrement dit un terme X tel que X =_β F X. En fait, il existe un terme Y sans variable libre tel que pour tout F, Y F soit un point fixe de F. Un tel terme est appelé un combinateur de point fixe;
- Combinateur de point fixe de Church/Curry : $Y = \lambda f.(\lambda x.f.(x.x))(\lambda x.f.(x.x))$;
- On a bien $Y F =_{\beta} F (Y F)$, mais pas $Y F \rightarrow_{\beta}^{*} F (Y F)$;
- Combinateur de point fixe de Turing : $\Theta = (\lambda g, h.h (g g h)) (\lambda g, h.h (g g h))$
- On a bien Θ $F =_{\beta} F$ (ΘF) et Θ $F \to_{\beta}^* F$ (ΘF) .

Récursion

- Théorème : tout terme F a un point fixe X, autrement dit un terme X tel que $X =_{\beta} F$ X. En fait, il existe un terme Y sans variable libre tel que pour tout F, Y F soit un point fixe de F. Un tel terme est appelé un combinateur de point fixe ;
- Combinateur de point fixe de Church/Curry : $Y = \lambda f.(\lambda x.f.(x.x))(\lambda x.f.(x.x))$;
- On a bien $Y F =_{\beta} F (Y F)$, mais pas $Y F \rightarrow_{\beta}^{*} F (Y F)$;
- Combinateur de point fixe de Turing : $\Theta = (\lambda g, h.h (g g h)) (\lambda g, h.h (g g h))$;
- On a bien $\Theta F =_{\beta} F (\Theta F)$ et $\Theta F \rightarrow_{\beta}^{*} F (\Theta F)$.

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0 thenelse, pred et plus ;
- $F = \lambda f$, n.if 0 then else n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta \ F \ 2) \rightarrow_{\beta}^* F \ (\Theta \ F) \ 2 \rightarrow_{\beta}^* \\ if 0 thenelse \ 2 \ 0 \ (plus \ 2 \ ((\Theta \ F) \ (pred \ 2))) \rightarrow_{\beta}^* \\ plus \ 2 \ ((\Theta \ F) \ (pred \ 2)) \rightarrow_{\beta}^* plus \ 2 \ ((\Theta \ F) \ 1) \rightarrow_{\beta}^* \\ plus \ 2 \ (F \ (\Theta \ F) \ 1) \rightarrow_{\beta}^* \\ plus \ 2 \ (if 0 thenelse \ 1 \ 0 \ (plus \ 1 \ ((\Theta \ F) \ (pred \ 1)))) \rightarrow_{\beta}^* \\ plus \ 2 \ (plus \ 1 \ ((\Theta \ F) \ (pred \ 1))) \rightarrow_{\beta}^* \\ plus \ 2 \ (plus \ 1 \ (if 0 thenelse \ 0 \ 0 \ (plus \ 0 \ ((\Theta \ F) \ (pred \ 0))))) \rightarrow_{\beta}^* \\ plus \ 2 \ (plus \ 1 \ 0) \rightarrow_{\beta}^* \ 3 \ ;$
- Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0thenelse, pred et plus;
- $F = \lambda f$, n.if 0 then else n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta F 2) \rightarrow_{\beta}^{*} F (\Theta F) 2 \rightarrow_{\beta}^{*}$ if 0 thenelse 2 0 (plus 2 ((\Omega F) (pred 2))) \(\rightarrow_{\beta}^{*} \) plus 2 ((\Omega F) (pred 2)) \(\rightarrow_{\beta}^{*} \) plus 2 ((\Omega F) 1) \(\rightarrow_{\beta}^{*} \) plus 2 (F (\Omega F) 1) \(\rightarrow_{\beta}^{*} \) plus 2 (if 0 thenelse 1 0 (plus 1 ((\Omega F) (pred 1)))) \(\rightarrow_{\beta}^{*} \) plus 2 (plus 1 ((\Omega F) (pred 1))) \(\rightarrow_{\beta}^{*} \) plus 2 (plus 1 (if 0 thenelse 0 0 (plus 0 ((\Omega F) (pred 0))))) \(\rightarrow_{\beta}^{*} \) plus 2 (plus 1 0) \(\rightarrow_{\beta}^{*} 3 \);
- Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0thenelse, pred et plus;
- $F = \lambda f$, n.if 0 then else n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta F 2) \rightarrow_{\beta}^{*} F (\Theta F) 2 \rightarrow_{\beta}^{*}$ if 0 thenelse 2 0 (plus 2 ((\Omega F) (pred 2))) \rightarrow_{β}^{*} plus 2 ((\Omega F) (pred 2)) \rightarrow_{β}^{*} plus 2 ((\Omega F) 1) \rightarrow_{β}^{*} plus 2 (F (\Omega F) 1) \rightarrow_{β}^{*} plus 2 (if 0 thenelse 1 0 (plus 1 ((\Omega F) (pred 1)))) \rightarrow_{β}^{*} plus 2 (plus 1 ((\Omega F) (pred 1))) \rightarrow_{β}^{*} plus 2 (plus 1 (if 0 thenelse 0 0 (plus 0 ((\Omega F) (pred 0))))) \rightarrow_{β}^{*} plus 2 (plus 1 0) \rightarrow_{β}^{*} 3;
- Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

Récursion : exemple

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0 thenelse, pred et plus ;
- $F = \lambda f$, n.if 0 thenelse n 0 (plus n (f (pred n)));
- Exemple de calcul :

```
 \begin{array}{l} (\Theta \ F \ 2) \rightarrow_{\beta}^{*} F \ (\Theta \ F) \ 2 \rightarrow_{\beta}^{*} \\ if 0 thenelse \ 2 \ 0 \ (plus \ 2 \ ((\Theta \ F) \ (pred \ 2))) \rightarrow_{\beta}^{*} \\ plus \ 2 \ ((\Theta \ F) \ (pred \ 2)) \rightarrow_{\beta}^{*} plus \ 2 \ ((\Theta \ F) \ 1) \rightarrow_{\beta}^{*} \\ plus \ 2 \ (F \ (\Theta \ F) \ 1) \rightarrow_{\beta}^{*} \\ plus \ 2 \ (if 0 thenelse \ 1 \ 0 \ (plus \ 1 \ ((\Theta \ F) \ (pred \ 1)))) \rightarrow_{\beta}^{*} \\ plus \ 2 \ (plus \ 1 \ (if 0 thenelse \ 0 \ 0 \ (plus \ 0 \ ((\Theta \ F) \ (pred \ 0))))) \rightarrow_{\beta}^{*} \\ plus \ 2 \ (plus \ 1 \ (if 0 thenelse \ 0 \ 0 \ (plus \ 0 \ ((\Theta \ F) \ (pred \ 0))))) \rightarrow_{\beta}^{*} \\ plus \ 2 \ (plus \ 1 \ 0) \rightarrow_{\beta}^{*} 3 \ ; \end{array}
```

• Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

Récursion : exemple

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0thenelse, pred et plus;
- $F = \lambda f$, n.if 0 then else n 0 (plus n (f (pred n)));
- Exemple de calcul :

```
\begin{array}{l} (\Theta \ F \ 2) \rightarrow_{\beta}^{*} F \ (\Theta \ F) \ 2 \rightarrow_{\beta}^{*} \\ \text{if 0thenelse 2 0 (plus 2 ((\Theta \ F) \ (pred \ 2)))} \rightarrow_{\beta}^{*} \\ \text{plus 2 ((\Theta \ F) \ (pred \ 2))} \rightarrow_{\beta}^{*} \\ \text{plus 2 (F (\Theta \ F) \ 1)} \rightarrow_{\beta}^{*} \\ \text{plus 2 (if 0thenelse 1 0 (plus 1 ((\Theta \ F) \ (pred \ 1))))} \rightarrow_{\beta}^{*} \\ \text{plus 2 (plus 1 ((\Theta \ F) \ (pred \ 1)))} \rightarrow_{\beta}^{*} \\ \text{plus 2 (plus 1 (if 0thenelse 0 0 (plus 0 ((\Theta \ F) \ (pred \ 0)))))} \rightarrow_{\beta}^{*} \\ \text{plus 2 (plus 1 0)} \rightarrow_{\beta}^{*} 3; \end{array}
```

• Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

Récursion : exemple

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0 thenelse, pred et plus ;
- $F = \lambda f$, n.if 0 then else n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta F 2) \rightarrow_{\beta}^{*} F (\Theta F) 2 \rightarrow_{\beta}^{*}$

```
if 0 thenelse 2 0 (plus 2 ((\Theta F) (pred 2))) \rightarrow_{\beta}^{*} plus 2 ((\Theta F) (pred 2)) \rightarrow_{\beta}^{*} plus 2 ((\Theta F) 1) \rightarrow_{\beta}^{*} plus 2 (F (\Theta F) 1) \rightarrow_{\beta}^{*} plus 2 (if 0 thenelse 1 0 (plus 1 ((\Theta F) (pred 1)))) \rightarrow_{\beta}^{*} plus 2 (plus 1 ((\Theta F) (pred 1))) \rightarrow_{\beta}^{*} plus 2 (plus 1 (if 0 thenelse 0 0 (plus 0 ((\Theta F) (pred 0))))) \rightarrow_{\beta}^{*} plus 2 (plus 1 0) \rightarrow_{\beta}^{*} 3;
```

• Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0 thenelse, pred et plus ;
- $F = \lambda f$, n.if0thenelse n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta F 2) \rightarrow_{\beta}^{*} F (\Theta F) 2 \rightarrow_{\beta}^{*}$ if 0thenelse 2 0 (plus 2 ((\Omega F) (pred 2))) \(\rightarrow_{\beta}^{*}\)
 plus 2 ((\Omega F) (pred 2)) \(\rightarrow_{\beta}^{*}\)
 plus 2 (F (\Omega F) 1) \(\rightarrow_{\beta}^{*}\)
 plus 2 (if 0thenelse 1 0 (plus 1 ((\Omega F) (pred 1)))) \(\rightarrow_{\beta}^{*}\)
 plus 2 (plus 1 ((\Omega F) (pred 1))) \(\rightarrow_{\beta}^{*}\)
 plus 2 (plus 1 (if 0thenelse 0 0 (plus 0 ((\Omega F) (pred 0))))) \(\rightarrow_{\beta}^{*}\)
 plus 2 (plus 1 0) \(\rightarrow_{\beta}^{*}\) 3:
- Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0 thenelse, pred et plus ;
- $F = \lambda f$, n.if0thenelse n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta F 2) \rightarrow_{\beta}^{*} F (\Theta F) 2 \rightarrow_{\beta}^{*}$ $if 0 thenelse 2 0 (plus 2 ((\Theta F) (pred 2))) \rightarrow_{\beta}^{*}$ $plus 2 ((\Theta F) (pred 2)) \rightarrow_{\beta}^{*} plus 2 ((\Theta F) 1) \rightarrow_{\beta}^{*}$ $plus 2 (F (\Theta F) 1) \rightarrow_{\beta}^{*}$ $plus 2 (if 0 thenelse 1 0 (plus 1 ((\Theta F) (pred 1)))) \rightarrow_{\beta}^{*}$ $plus 2 (plus 1 ((\Theta F) (pred 1))) \rightarrow_{\beta}^{*} plus 2 (plus 1 (F (\Theta F) 0)) \rightarrow_{\beta}^{*}$ $plus 2 (plus 1 (if 0 thenelse 0 0 (plus 0 ((\Theta F) (pred 0))))) \rightarrow_{\beta}^{*}$ $plus 2 (plus 1 0) \rightarrow_{\beta}^{*} 3 :$
- Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0 thenelse, pred et plus ;
- $F = \lambda f$, n.if 0 then else n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta F 2) \rightarrow_{\beta}^{*} F (\Theta F) 2 \rightarrow_{\beta}^{*}$ if 0 thenelse 2 0 (plus 2 ((ΘF) (pred 2))) \rightarrow_{β}^{*} plus 2 ((ΘF) (pred 2)) \rightarrow_{β}^{*} plus 2 ((ΘF) 1) \rightarrow_{β}^{*} plus 2 ($F (\Theta F) P$) \rightarrow_{β}^{*} plus 2 (if 0 thenelse 1 0 (plus 1 ((ΘF) (pred 1)))) \rightarrow_{β}^{*} plus 2 (plus 1 ((ΘF) (pred 1))) \rightarrow_{β}^{*} plus 2 (plus 1 (if 0 thenelse 0 0 (plus 0 ((ΘF) (pred 0))))) \rightarrow_{β}^{*} plus 2 (plus 1 0) \rightarrow_{β}^{*} 3;
- Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0 thenelse, pred et plus ;
- $F = \lambda f$, n.if 0 then else n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta F 2) \rightarrow_{\beta}^{*} F (\Theta F) 2 \rightarrow_{\beta}^{*}$ $if 0 thenelse 2 0 (plus 2 ((\Theta F) (pred 2))) \rightarrow_{\beta}^{*}$ $plus 2 ((\Theta F) (pred 2)) \rightarrow_{\beta}^{*} plus 2 ((\Theta F) 1) \rightarrow_{\beta}^{*}$ $plus 2 (F (\Theta F) 1) \rightarrow_{\beta}^{*}$ $plus 2 (if 0 thenelse 1 0 (plus 1 ((\Theta F) (pred 1)))) \rightarrow_{\beta}^{*}$ $plus 2 (plus 1 ((\Theta F) (pred 1))) \rightarrow_{\beta}^{*} plus 2 (plus 1 (F (\Theta F) 0)) \rightarrow_{\beta}^{*}$ $plus 2 (plus 1 (if 0 thenelse 0 0 (plus 0 ((\Theta F) (pred 0))))) \rightarrow_{\beta}^{*}$ $plus 2 (plus 1 0) \rightarrow_{\beta}^{*} 3;$
- Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0 thenelse, pred et plus;
- $F = \lambda f$, n.if 0 then else n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta F 2) \rightarrow_{\beta}^{*} F (\Theta F) 2 \rightarrow_{\beta}^{*}$ if 0thenelse 2 0 (plus 2 ((ΘF) (pred 2))) \rightarrow_{β}^{*} plus 2 ((ΘF) (pred 2)) \rightarrow_{β}^{*} plus 2 ((ΘF) 1) \rightarrow_{β}^{*} plus 2 ($F (\Theta F) P$) \rightarrow_{β}^{*} plus 2 (if 0thenelse 1 0 (plus 1 ((P P) (pred 1)))) \rightarrow_{β}^{*} plus 2 (plus 1 ((P P) (pred 1))) \rightarrow_{β}^{*} plus 2 (plus 1 (if 0thenelse 0 0 (plus 0 ((P P) (pred 0))))) \rightarrow_{β}^{*} plus 2 (plus 1 0) \rightarrow_{β}^{*} 3;
- Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0 thenelse, pred et plus ;
- $F = \lambda f$, n.if 0 then else n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta F 2) \rightarrow_{\beta}^{*} F (\Theta F) 2 \rightarrow_{\beta}^{*}$ if 0 thenelse 2 0 (plus 2 ((ΘF) (pred 2))) \rightarrow_{β}^{*} plus 2 ((ΘF) (pred 2)) \rightarrow_{β}^{*} plus 2 ((ΘF) 1) \rightarrow_{β}^{*} plus 2 ($F (\Theta F) P$) \rightarrow_{β}^{*} plus 2 (if 0 thenelse 1 0 (plus 1 ((P P) (pred 1)))) \rightarrow_{β}^{*} plus 2 (plus 1 ((P P) (pred 1))) \rightarrow_{β}^{*} plus 2 (plus 1 (if 0 thenelse 0 0 (plus 0 ((P P) (pred 0))))) \rightarrow_{β}^{*} plus 2 (plus 1 0) \rightarrow_{β}^{*} 3;
- Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0thenelse, pred et plus;
- $F = \lambda f$, n.if 0 then else n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta F 2) \rightarrow_{\beta}^{*} F (\Theta F) 2 \rightarrow_{\beta}^{*}$ if 0 thenelse 2 0 (plus 2 ((ΘF) (pred 2))) \rightarrow_{β}^{*} plus 2 ((ΘF) (pred 2)) \rightarrow_{β}^{*} plus 2 ((ΘF) 1) \rightarrow_{β}^{*} plus 2 ($F (\Theta F) P$) \rightarrow_{β}^{*} plus 2 (if 0 thenelse 1 0 (plus 1 ((P P) (pred 1)))) \rightarrow_{β}^{*} plus 2 (plus 1 ((P P) (pred 1))) \rightarrow_{β}^{*} plus 2 (plus 1 (if 0 thenelse 0 0 (plus 0 ((P P) (pred 0))))) \rightarrow_{β}^{*} plus 2 (plus 1 0) \rightarrow_{β}^{*} 3;
- Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0 thenelse, pred et plus ;
- $F = \lambda f$, n.if 0 then else n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta F 2) \rightarrow_{\beta}^{*} F (\Theta F) 2 \rightarrow_{\beta}^{*}$ if 0 thenelse 2 0 (plus 2 ((ΘF) (pred 2))) \rightarrow_{β}^{*} plus 2 ((ΘF) (pred 2)) \rightarrow_{β}^{*} plus 2 ((ΘF) 1) \rightarrow_{β}^{*} plus 2 ($F (\Theta F) P$) \rightarrow_{β}^{*} plus 2 (if 0 thenelse 1 0 (plus 1 ((P P) (pred 1)))) \rightarrow_{β}^{*} plus 2 (plus 1 ((P P) (pred 1))) \rightarrow_{β}^{*} plus 2 (plus 1 (if 0 thenelse 0 0 (plus 0 ((P P) (pred 0))))) \rightarrow_{β}^{*} plus 2 (plus 1 0) \rightarrow_{β}^{*} 3
- Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

λ -calcul de Church

Récursion : exemple

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0thenelse, pred et plus;
- $F = \lambda f$, n.if 0 then else n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta F 2) \rightarrow_{\beta}^{*} F (\Theta F) 2 \rightarrow_{\beta}^{*}$ if 0 thenelse 2 0 (plus 2 ((ΘF) (pred 2))) \rightarrow_{β}^{*} plus 2 ((ΘF) (pred 2)) \rightarrow_{β}^{*} plus 2 ((ΘF) 1) \rightarrow_{β}^{*} plus 2 ($F (\Theta F) P$) \rightarrow_{β}^{*} plus 2 (if 0 thenelse 1 0 (plus 1 ((P P) (pred 1)))) \rightarrow_{β}^{*} plus 2 (plus 1 ((P P) (pred 1))) \rightarrow_{β}^{*} plus 2 (plus 1 (if 0 thenelse 0 0 (plus 0 ((P P) (pred 0))))) \rightarrow_{β}^{*} plus 2 (plus 1 0) \rightarrow_{β}^{*} 3;
- Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

λ -calcul de Church

Récursion : exemple

- Écrire la fonction qui calcule la somme des *n* premiers entiers;
- On suppose qu'on a les fonctions if 0 thenelse, pred et plus;
- $F = \lambda f$, n.if 0 then else n 0 (plus n (f (pred n)));
- Exemple de calcul : $(\Theta F 2) \rightarrow_{\beta}^{*} F (\Theta F) 2 \rightarrow_{\beta}^{*}$ if 0 thenelse 2 0 (plus 2 ((ΘF) (pred 2))) \rightarrow_{β}^{*} plus 2 ((ΘF) (pred 2)) \rightarrow_{β}^{*} plus 2 ((ΘF) 1) \rightarrow_{β}^{*} plus 2 ($F (\Theta F) P$) \rightarrow_{β}^{*} plus 2 (if 0 thenelse 1 0 (plus 1 ((P P) (pred 1)))) \rightarrow_{β}^{*} plus 2 (plus 1 ((P P) (pred 1))) \rightarrow_{β}^{*} plus 2 (plus 1 (if 0 thenelse 0 0 (plus 0 ((P P) (pred 0))))) \rightarrow_{β}^{*} plus 2 (plus 1 0) \rightarrow_{β}^{*} 3;
- Il faut une stratégie de réduction car on peut toujours réduire Θ F et partir sur une réduction infinie.

Langage

Règles

- Jugements de typage :
 - De la forme : $x_1 : A_1, \dots, x_n : A_n \vdash t : B$, où les x_i sont des variables, et les A_i et B des termes.
 - Si les variables x_i ont respectivement les types A_i alors t a le type B.
- Notations : Γ sera une séquence $x_1:A_1,\ldots,x_n:A_n$; A,B,C,\ldots seront des termes ; K,L seront soit Prop, soit Type.

L3 Info. 2024-2025

Règles

Autres opérateurs logiques

- $A \Rightarrow B \equiv \Pi x : A.B \ (x \notin B).$
- $A \wedge B \equiv \Pi C : \text{Prop.}(A \Rightarrow B \Rightarrow C) \Rightarrow C$.
- $A \lor B \equiv \Pi C : \mathsf{Prop.}(A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow C$.
- $\neg A \equiv \Pi C : \mathsf{Prop}.A \Rightarrow C$.
- $\top \equiv \Pi C : \mathsf{Prop.} C \Rightarrow C$.
- $\perp \equiv \Pi C$: Prop. C
- $\exists x : A.B \equiv \Pi C : \mathsf{Prop.}(\Pi x : A.B \Rightarrow C) \Rightarrow C$



Fonctions

• On peut écrire des fonctions et les typer.

```
\Gamma_1 \vdash \mathsf{Prop} : \mathsf{Type} \qquad \qquad \Gamma_2 = \Gamma_1, B : \mathsf{Prop} \vdash \begin{array}{c} \lambda x : A.\lambda y : B.x : \\ \Pi x : A.\Pi y : B.A \end{array} \vdash \mathsf{Prop} : \mathsf{Type} \qquad \qquad \Gamma_1 = A : \mathsf{Prop} \vdash \begin{array}{c} \lambda B : \mathsf{Prop}.\lambda x : A.\lambda y : B.x : \\ \Pi B : \mathsf{Prop}.\lambda x : A.\lambda y : B.x : \\ \Pi B : \mathsf{Prop}.\Pi x : A.\Pi y : B.A \end{array} \vdash \begin{array}{c} \lambda A, B : \mathsf{Prop}.\lambda x : A.\lambda y : B.x : \\ \Pi A, B : \mathsf{Prop}.\Pi x : A.\Pi y : B.A \end{array} \vdash \begin{array}{c} \Gamma_3 \vdash A : \mathsf{Prop} \\ \Gamma_3 \vdash A : \mathsf{Prop} \end{array} \qquad \begin{array}{c} \Gamma_3 \vdash A : \mathsf{Prop} \\ \Gamma_3 \vdash \Gamma_2 \vdash \Gamma_3 : \Gamma_3 \Gamma_3 :
```

Fonctions

On peut écrire des fonctions et les typer.

Fonctions

• On peut écrire des fonctions et les typer.

Fonctions

• On peut écrire des fonctions et les typer.

Fonctions

On peut écrire des fonctions et les typer.

```
\Gamma_2 = \Gamma_1, B : \mathsf{Prop} \vdash \quad \lambda x : A.\lambda y : B.x :
                                                  Γ<sub>1</sub> ⊢ Prop : Type
                                                                                                                                                                                            Lam
                                                                                                          \lambda B : \mathsf{Prop}.\lambda x : A.\lambda y : B.x :
⊢ Prop : Type
                                                                        \Gamma_1 = A : \mathsf{Prop} \vdash
                                                                                                                                                                      Lam
                                                       \lambda A, B : \mathsf{Prop}.\lambda x : A.\lambda y : B.x :
```

Fonctions

On peut écrire des fonctions et les typer.

```
Пι
                                                \Gamma_1 \vdash \mathsf{Prop} : \mathsf{Type}
                                                                                                                                                  \lambda x : A.\lambda y : B.x :
                                                                                                         \Gamma_2 = \Gamma_1, B : \mathsf{Prop} \vdash
                                                                                                                                                                                         I am
                                                                                                         \lambda B : \mathsf{Prop}.\lambda x : A.\lambda y : B.x :
⊢ Prop : Type
                                                                      \Gamma_1 = A : \mathsf{Prop} \vdash
                                                                                                                                                                    Lam
                                                     \lambda A, B : \mathsf{Prop.} \lambda x : A.\lambda y : B.x :
```

Fonctions

On peut écrire des fonctions et les typer.

```
Пι
                                                  \Gamma_1 \vdash \mathsf{Prop} : \mathsf{Type}
                                                                                                                                                        \lambda x : A.\lambda y : B.x :
                                                                                                              \Gamma_2 = \Gamma_1, B : \mathsf{Prop} \vdash
                                                                                                                                                                                                  L<sub>am</sub>
                                                                                                              \lambda B : \mathsf{Prop}.\lambda x : A.\lambda y : B.x :
⊢ Prop : Type
                                                                          \Gamma_1 = A : \mathsf{Prop} \vdash
                                                                                                                                                                           Lam
                                                        \lambda A, B : \mathsf{Prop.} \lambda x : A.\lambda y : B.x :
  Preuve II1:
                                                  \Gamma_2 \vdash \lambda x : A \cdot \lambda y : B \cdot x : \Pi x : A \cdot \Pi y : B \cdot A
```

Fonctions

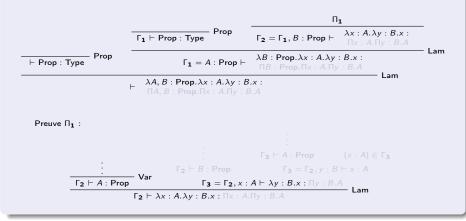
On peut écrire des fonctions et les typer.

```
Пι
                                                                                                                                                       \lambda x : A.\lambda y : B.x :
                                                   \Gamma_1 \vdash \mathsf{Prop} : \mathsf{Type}
                                                                                                            \Gamma_2 = \Gamma_1, B : \mathsf{Prop} \vdash
                                                                                                                                                                                               Lam
                                                                                                            \lambda B : \mathsf{Prop}.\lambda x : A.\lambda y : B.x :
⊢ Prop : Type
                                                                         \Gamma_1 = A : \mathsf{Prop} \vdash
                                                                                                                                                                         Lam
                                                       \lambda A, B : \mathsf{Prop}.\lambda x : A.\lambda y : B.x :
  Preuve \Pi_1:
                         \Gamma_2 \vdash A : \mathsf{Prop}
                                                               \Gamma_3 = \Gamma_2, x : A \vdash \lambda y : B.x : \Pi y : B.A
                                                  \Gamma_2 \vdash \lambda x : A \cdot \lambda y : B \cdot x : \Pi x : A \cdot \Pi y : B \cdot A
```

Fonctions

On peut écrire des fonctions et les typer.

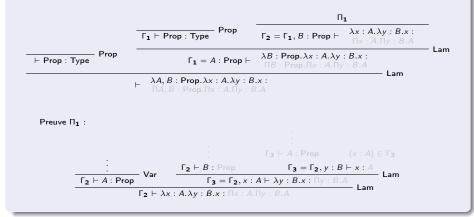
Par exemple : λA , B : Prop. λx : A. λy : B.x.



L3 Info. 2024-2025

Fonctions

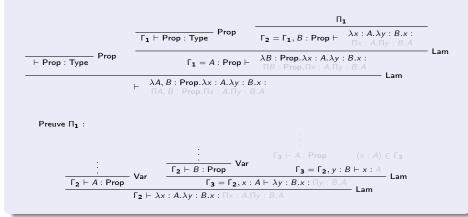
On peut écrire des fonctions et les typer.



Fonctions

On peut écrire des fonctions et les typer.

Par exemple : λA , B : Prop. λx : A. λy : B.x.

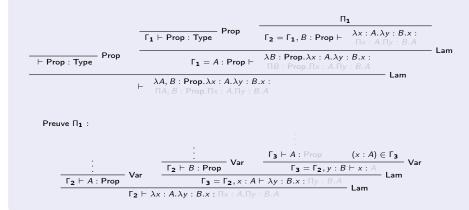


L3 Info. 2024-2025

Fonctions

On peut écrire des fonctions et les typer.

Par exemple : λA , B : Prop. λx : A. λy : B.x.

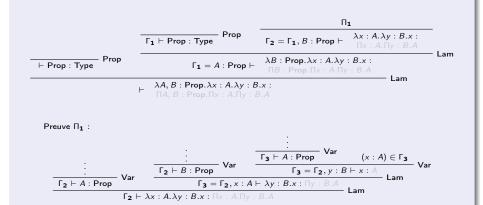


L3 Info. 2024-2025

Fonctions

On peut écrire des fonctions et les typer.

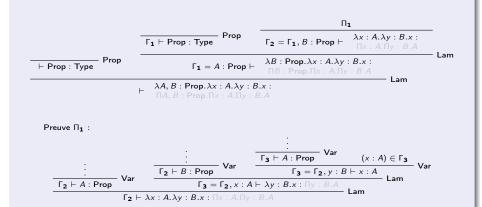
Par exemple : λA , B : Prop. λx : $A \cdot \lambda y$: $B \cdot x$.



Fonctions

On peut écrire des fonctions et les typer.

Par exemple : λA , B : Prop. λx : A. λy : B.x.

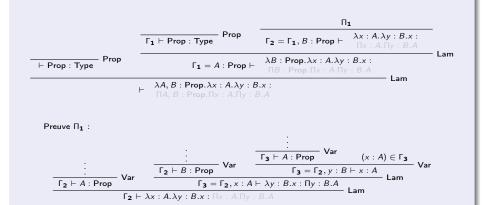


28 / 37

Fonctions

On peut écrire des fonctions et les typer.

Par exemple : λA , B : Prop. λx : $A \cdot \lambda y$: $B \cdot x$.

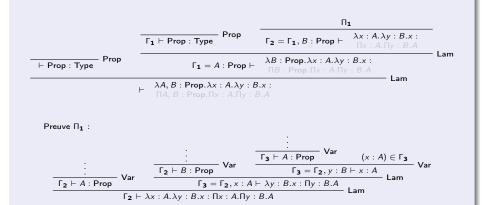


L3 Info. 2024-2025

Fonctions

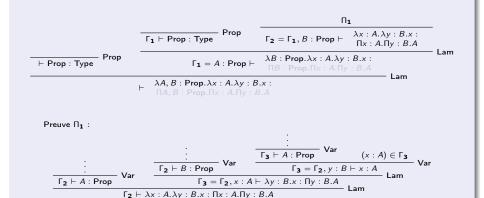
On peut écrire des fonctions et les typer.

Par exemple : λA , B : Prop. λx : $A \cdot \lambda y$: $B \cdot x$.



Fonctions

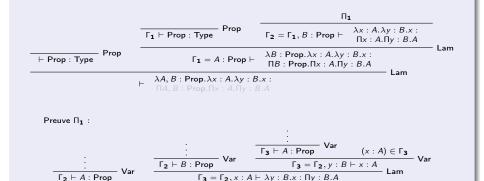
On peut écrire des fonctions et les typer.



Fonctions

On peut écrire des fonctions et les typer.

Par exemple : λA , B : Prop. λx : A. λy : B.x.

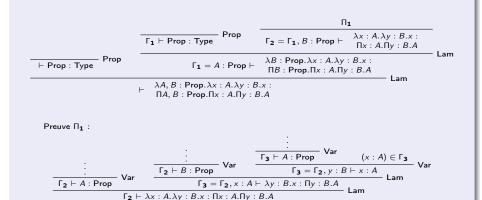


L3 Info. 2024-2025

 $\Gamma_2 \vdash \lambda x : A.\lambda y : B.x : \Pi x : A.\Pi y : B.A$

Fonctions

On peut écrire des fonctions et les typer.



Fonctions

- Cette fonction est polymorphe (paramétrée par A et B de type Prop).
- Prop est le type des propositions mais on s'aperçoit qu'il est également habité par les types de données (Coq utilise plutôt la sorte Set pour les types de données).
- On pourrait rajouter la sorte Set : elle serait au même niveau que Prop, à savoir de type Type, et Prop et Set seraient incomparables.

Preuves

• On peut écrire des preuves de propositions.

```
\Gamma_1 \vdash \mathsf{Prop} : \mathsf{Type} \qquad \Gamma_2 = \Gamma_1, B : \mathsf{Prop} \vdash \frac{\lambda x : A.\lambda y : B.x :}{\Gamma x : A.\Pi y : B.A} \vdash \mathsf{Prop} : \mathsf{Type} \qquad \Gamma_1 = A : \mathsf{Prop} \vdash \frac{\lambda B : \mathsf{Prop}.\lambda x : A.\lambda y : B.x :}{\Pi B : \mathsf{Prop}.\Pi x : A.\Pi y : B.A} \vdash \frac{\lambda A, B : \mathsf{Prop}.\lambda x : A.\lambda y : B.x :}{\Pi A, B : \mathsf{Prop}.\Pi x : A.\Pi y : B.A} \vdash \frac{\lambda A, B : \mathsf{Prop}.\Pi x : A.\Pi y : B.A}{\Pi A : \mathsf{Prop} \quad (x : A) \in \Gamma_3} \vdash \Gamma_1 : \qquad \qquad \vdots \qquad \qquad \qquad \vdots \qquad \qquad \vdots \qquad \qquad \qquad \vdots \qquad \qquad \qquad \qquad \qquad \vdots \qquad
```

Preuves

On peut écrire des preuves de propositions.
 Par exemple : ΠΑ, Β : Prop.Πx : Α.Πy : Β.Α.

Preuves

• On peut écrire des preuves de propositions.

Preuves

• On peut écrire des preuves de propositions.

Preuves

• On peut écrire des preuves de propositions.

```
\Gamma_2 = \Gamma_1, B : \mathsf{Prop} \vdash \begin{array}{c} \lambda x : A \cdot \lambda y : B \cdot x : \\ B \cdot A \cdot \lambda y : B \cdot x : \\ B \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y : B \cdot x : \\ C \cdot A \cdot \lambda y 
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              Γ<sub>1</sub> ⊢ Prop : Type
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            \Pi x : A.\Pi y : B.A
⊢ Prop : Type
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   \Gamma_1 = A : \mathsf{Prop} \vdash
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      \Pi B : \mathsf{Prop}.\Pi x : A.\Pi y : B.A
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    \Pi A. B : \mathsf{Prop}.\Pi x : A.\Pi v : B.A
```

Preuves

• On peut écrire des preuves de propositions.

```
Пι
                                              \Gamma_1 \vdash \mathsf{Prop} : \mathsf{Type}
                                                                                                     \Gamma_2 = \Gamma_1, B : \mathsf{Prop} \vdash \lambda x : A \cdot \lambda y : B \cdot x :
                                                                                                                                              \Pi x : A.\Pi y : B.A
                                                                                                                                                                                  L<sub>am</sub>
⊢ Prop : Type
                                                                    \Gamma_1 = A : \mathsf{Prop} \vdash
                                                                                                      \Pi B : \mathsf{Prop}.\Pi x : A.\Pi y : B.A
                                                                                                                                                             Lam
                                                    \Pi A, B : \mathsf{Prop}.\Pi x : A.\Pi y : B.A
```

Preuves

• On peut écrire des preuves de propositions.

```
Пι
                                             \Gamma_1 \vdash \mathsf{Prop} : \mathsf{Type}
                                                                                                   \Gamma_2 = \Gamma_1, B : \mathsf{Prop} \vdash \lambda x : A \cdot \lambda y : B \cdot x :
                                                                                                                                           \Pi x : A.\Pi y : B.A
                                                                                                                                                                              L<sub>am</sub>
⊢ Prop : Type
                                                                  \Gamma_1 = A : \mathsf{Prop} \vdash
                                                                                                    \Pi B : \mathsf{Prop}.\Pi x : A.\Pi y : B.A
                                                                                                                                                          Lam
                                                   \Pi A, B : \mathsf{Prop}.\Pi x : A.\Pi y : B.A
  Preuve II1:
                                             \Gamma_2 \vdash \lambda_X : A.\lambda_Y : B.X : \Pi_X : A.\Pi_Y : B.A
```

Preuves

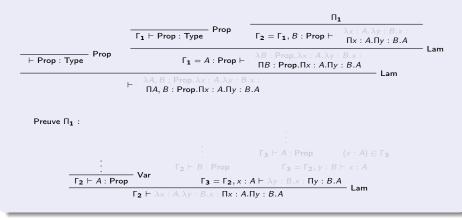
• On peut écrire des preuves de propositions.

```
\Pi_1
                                               \Gamma_1 \vdash \mathsf{Prop} : \mathsf{Type}
                                                                                                    \Gamma_2 = \Gamma_1, B : \mathsf{Prop} \vdash
                                                                                                                                             \Pi x : A.\Pi v : B.A
⊢ Prop : Type
                                                                   \Gamma_1 = A : \mathsf{Prop} \vdash
                                                                                                      \Pi B : \mathsf{Prop}.\Pi x : A.\Pi y : B.A
                                                                                                                                                             Lam
                                                    \Pi A, B : \mathsf{Prop}.\Pi x : A.\Pi y : B.A
  Preuve \Pi_1:
                       \Gamma_2 \vdash A : \mathsf{Prop} \Gamma_3 = \Gamma_2, x : A \vdash \lambda y : B.x : \Pi y : B.A
                                              \Gamma_2 \vdash \lambda x : A \cdot \lambda y : B \cdot x : \Pi x : A \cdot \Pi y : B \cdot A
```

Preuves

• On peut écrire des preuves de propositions.

Par exemple : $\Pi A, B$: Prop. $\Pi x : A.\Pi y : B.A$.

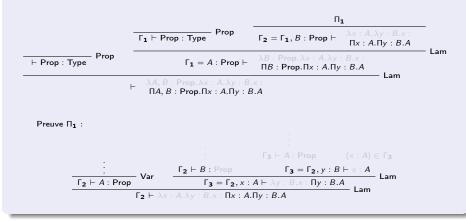


L3 Info. 2024-2025

Preuves

• On peut écrire des preuves de propositions.

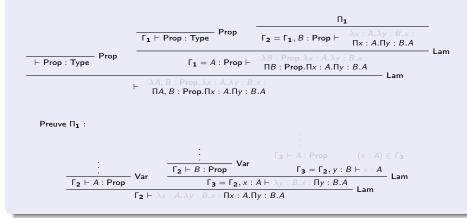
Par exemple : $\Pi A, B$: Prop. $\Pi x : A.\Pi y : B.A$.



L3 Info. 2024-2025

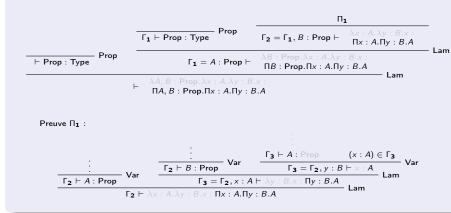
Preuves

• On peut écrire des preuves de propositions.



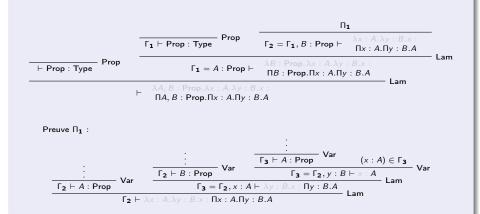
Preuves

• On peut écrire des preuves de propositions.



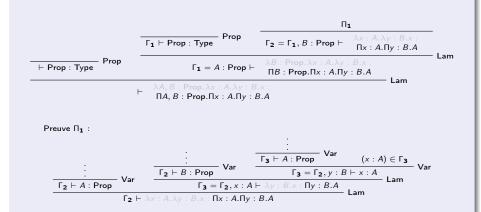
Preuves

• On peut écrire des preuves de propositions.



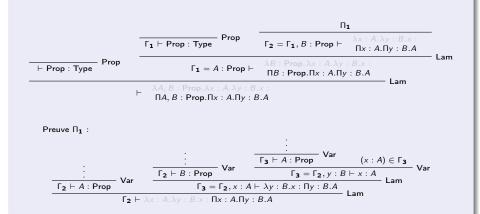
Preuves

• On peut écrire des preuves de propositions.



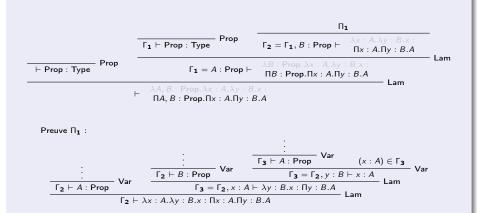
Preuves

• On peut écrire des preuves de propositions.



Preuves

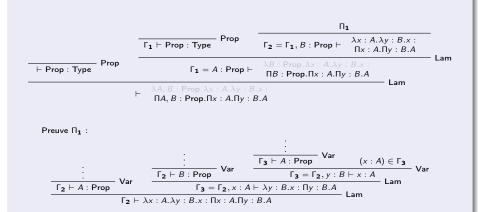
• On peut écrire des preuves de propositions.



Preuves

• On peut écrire des preuves de propositions.

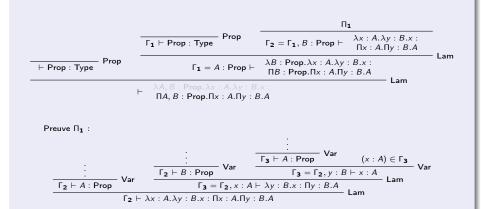
Par exemple : $\Pi A, B$: Prop. $\Pi x : A.\Pi y : B.A$.



L3 Info. 2024-2025

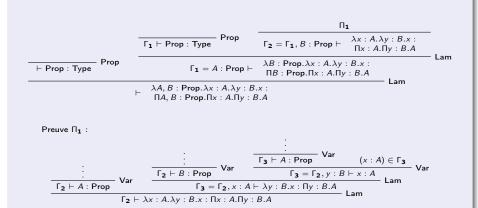
Preuves

• On peut écrire des preuves de propositions.



Preuves

• On peut écrire des preuves de propositions.



- On peut identifier deux modes :
 - ▶ On a programmé t et on recherche son type : $\Gamma \vdash t$:?.
 - On a une proposition A et on recherche sa preuve : $\Gamma \vdash ? : A$.
 - On obtient dans les deux cas un arbre de preuve/typage
 - Dans le cas des preuves, on gagne un objet preuve (un terme).
- Cela montre bien la dualité de l'isomorphisme de Curry-Howard :
 - propositions ≡ types
 - preuves ≡ programmes

- On peut identifier deux modes :
 - ▶ On a programmé t et on recherche son type : $\Gamma \vdash t$:?.
 - ▶ On a une proposition A et on recherche sa preuve : $\Gamma \vdash ? : A$.
 - On obtient dans les deux cas un arbre de preuve/typage
 - Dans le cas des preuves, on gagne un objet preuve (un terme).
- Cela montre bien la dualité de l'isomorphisme de Curry-Howard :
 - propositions ≡ types
 - □ preuves ≡ programmes

- On peut identifier deux modes :
 - ▶ On a programmé t et on recherche son type : $\Gamma \vdash t$:?.
 - ▶ On a une proposition A et on recherche sa preuve : $\Gamma \vdash$? : A.
 - ▶ On obtient dans les deux cas un arbre de preuve/typage.
 - Dans le cas des preuves, on gagne un objet preuve (un terme).
- Cela montre bien la dualité de l'isomorphisme de Curry-Howard :
 - propositions ≡ types
 - preuves ≡ programmes

- On peut identifier deux modes :
 - ▶ On a programmé t et on recherche son type : $\Gamma \vdash t$:?.
 - ▶ On a une proposition A et on recherche sa preuve : $\Gamma \vdash$? : A.
 - ▶ On obtient dans les deux cas un arbre de preuve/typage.
 - Dans le cas des preuves, on gagne un objet preuve (un terme).
- Cela montre bien la dualité de l'isomorphisme de Curry-Howard :
 - propositions ≡ types
 - preuves ≡ programmes

- On peut identifier deux modes :
 - ▶ On a programmé t et on recherche son type : $\Gamma \vdash t$:?.
 - ▶ On a une proposition A et on recherche sa preuve : $\Gamma \vdash$? : A.
 - ▶ On obtient dans les deux cas un arbre de preuve/typage.
 - Dans le cas des preuves, on gagne un objet preuve (un terme).
- Cela montre bien la dualité de l'isomorphisme de Curry-Howard :
 - ▶ propositions ≡ types
 - ▶ preuves ≡ programmes

Syntaxe

CC	Coq
Туре	Туре
Prop	Prop
X	x
Пх : А.В	forall (x : A),B
$\lambda x : A.M$	fun (x : A) => M
M N	M N

Typage

Typage avec la commande Check.

```
Coq < Check (fun (A : Prop) (a : A) \Rightarrow a).

fun (A : Prop) (a : A) \Rightarrow a

: forall A : Prop, A \rightarrow A
```

Environnement

• Le contexte Γ des dérivations peut être enrichi avec la commande Paramater(s) (contexte valable pour tout le reste de la session).

```
Coq < Parameter T : Prop.
T is declared

Coq < Parameter I : T.
I is declared
```

Évaluation

• Évaluation avec la commande Eval compute in.

Définitions

- On peut rajouter des définitions, qui sont constituées d'un nom, d'un type et d'un corps.
- Il faut une règle de réduction supplémentaire pour réduire les définitions (c'est-à-dire expanser le corps des définitions) : c'est la δ -réduction en Coq.
- La δ -réduction est capturée par Eval compute in.

```
Coq < Definition my\_true (A : Prop) (a : A) := a.
my\_true is defined
```

```
Coq < Eval compute in (my_true T I).
= I
: T</pre>
```

Preuves

- On travaille sur la partie des types des jugements de typage.
- La partie terme peut-être :
 - Soit donnée directement par un λ -terme avec la tactique exact.
 - Soit construite progressivement par des tactiques (voir tableau).

Règle CC	Tactique Coq
Var	assumption
Lam	intro
Арр	apply
Conv	compute

Les règles Prop et Prod sont implicites.

Preuves

• Exemple : on veut démontrer ΠA : Prop. $A \Rightarrow A$. On utilise la notation " \Rightarrow " car pas de dépendance.

```
Proof. exact (fun\ (A: Prop)\ (a: A) \Rightarrow a). Qed.

Lemma proof\_2: forall\ (A: Prop),\ A \rightarrow A. Proof. intro. (*\ R\`egle\ Prod\ *) intro. (*\ R\`egle\ Prod\ *) assumption. (*\ R\`egle\ Var\ *) Qed.
```

Lemma proof 1 : forall (A : Prop), $A \rightarrow A$.

Preuves

• On peut afficher le contenu des preuves réalisées :

```
Print proof_1.
Print proof_2.
```

Les deux commandes donnent exactement le même résultat :

```
proof_1 =
fun(A : Prop)(H : A) \Rightarrow H
: forall(A : Prop, A \rightarrow A)
proof_2 =
fun(A : Prop)(H : A) \Rightarrow H
: forall(A : Prop, A \rightarrow A)
```