Substitution de termes La substitution d'un terme à une variable est définie inductivement comme suit (on prend les précautions d'usage concernant les variables libres ou liées : aucune variable libre dans le terme que l'on substitue ne doit devenir liée dans le terme résultant) :

- $_{[x:=t]}x \rightsquigarrow t$
- $_{[x:=t]}y \rightsquigarrow y$ si $y \neq x$
- $_{[x:=t]}(M)N \rightsquigarrow (_{[x:=t]}M)_{[x:=t]}N$
- $_{[x:=t]}\lambda y.M \rightsquigarrow \lambda y._{[x:=t]}M$ si y n'est pas libre dans t.

Ce sont les seuls cas de substitution admissibles.

α-conversion Il est naturel, comme on le fait pour les formules quantifiées en logique du premier ordre, de considérer des termes qui ne diffèrent que par le nom des variables liées comme équivalents. C'est l'α-équivalence :

$$\lambda x.\varphi \equiv \lambda z._{[x:=z]}\varphi$$

Beta-conversion (ou β-équivalence) :

$$(\lambda x.M)N \equiv {}_{[x:=N]}M$$

La β-conversion est une règle de **calcul** (on parle de β-réduction lorsque l'on "réduit" $(\lambda x.M)N$ en $_{[x:=N]}M$) ; c'est même la **seule** règle primitive de calcul du λ-calcul (modulo l'α-conversion).

Combinateurs Un **combinateur** est un λ-terme clos, i.e. sans variable libre.

Remarque : un combinateur se comporte de façon uniforme (indépendante du contexte), et ainsi peut être assimilié à une *constante* du langage. À noter qu'un combinateur est défini (comme tout λ-terme) à une α-conversion près.

Quelques combinateurs et leurs propriétés

Identité $\mathsf{I} =_{\text{def}} \lambda x.x$
Pour tout terme t, on a $(\mathsf{I})t \equiv t$.

Entiers Il existe plusieurs façons de représenter les entiers par des λ-termes ; la plus naturelle est celle de Church, où les entiers sont conçus comme des itérateurs :
$\mathsf{0} =_{\text{def}} \lambda f.\lambda x.x$
$\mathsf{1} =_{\text{def}} \lambda f.\lambda x.(f)x$
$\mathsf{n} =_{\text{def}} \lambda f.\lambda x.(f)(f)\dots(f)x$, avec f n fois.
On peut alors représenter la fonction successeur, l'addition et la multiplication :
$\mathsf{Succ} =_{\text{def}} \lambda n.\lambda f.\lambda x.(f)((n)f)x$
$+ \equiv \lambda m.\lambda n.\lambda f.\lambda x.((m)f)((n)f)x$
$* \equiv \lambda m.\lambda n.\lambda f.(m)(n)f$

Booléens $\mathsf{T} =_{\text{def}} \lambda x.\lambda y.x$
$\mathsf{F} =_{\text{def}} \lambda x.\lambda y.y$
Ces définitions, conventionnelles, s'expliquent par la forme très simple que reçoit alors la définition par cas : `if` P `then` Q `else` $R =_{\text{def}} PQR$ $(=((P)Q)R)$.

En effet, si P se réduit en T (ie $P \equiv \mathsf{T}$), alors $\texttt{if}\ P\ \texttt{then}\ Q\ \texttt{else}\ R \equiv \mathsf{T}QR \equiv Q$. De même, si $P \equiv \mathsf{F}$, alors on obtient R.

On peut alors définir relativement aisément les combinateurs booléens, en passant par la définition précédente : $\neg P = \texttt{if}\ P\ \texttt{then}\ \mathsf{F}\ \texttt{else}\ \mathsf{T} = P\mathsf{FT}$ (ou $((P)\mathsf{F})\mathsf{T}$). D'où par λ-abstraction : $\mathsf{NOT} =_{\text{def}} \lambda x.((x)\mathsf{F})\mathsf{T}$.
Alors, on peut vérifier que si $P \equiv \mathsf{T}$, alors $\mathsf{NOT}P \equiv ((P)\mathsf{F})\mathsf{T} \equiv ((\mathsf{T})\mathsf{F})\mathsf{T} \equiv \mathsf{F}$.

En exercice, on peut définir les opérateurs $\wedge$, $\vee$, et vérifier les propriétés usuelles.

Langage typé

Théorie des types L'ensemble des types est défini par induction :

1. e est un type
2. t est un type
3. Si a et b sont des types, alors $\langle a, b\rangle$ est un type

Un langage typé est un langage dont chaque *fbf* se voit assigner un type par une syntaxe compositionnelle dont la sémantique sera appuyée sur les domaines suivants (où D_a dénote l'ensemble des dénotations possibles des expressions de type a, où A est un domaine d'entités) :

— $D_e = A$
— $D_t = \{0, 1\}$
— $D_{\langle a,b\rangle}$ = l'ensemble des fonctions de D_a dans D_b.

Langage de Montague (non intentionel) L'ensemble des expressions interprétables (*meaningful expressions*) de type a, ME_a, est défini inductivement :

— Pour chaque type a, les variables et les constantes de type a sont dans ME_a.
— Pour tous types a et b, si $\alpha \in ME_{\langle a,b\rangle}$ et $\beta \in ME_a$ alors $(\alpha)\beta \in ME_b$.
— Pour tous types a et b, si u est une variable de type a et $\alpha \in ME_b$, alors $\lambda u.\alpha$ est dans $ME_{\langle a,b\rangle}$.
— Si φ et ψ sont dans ME_t, alors les expressions suivantes sont aussi dans ME_t : $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$.
— Pour tout type a, si φ est dans ME_t et u est une variable de type a, alors $\forall u\varphi$ et $\exists u\varphi$ sont dans ME_t.