

Lambda-calcul

Introduction à la sémantique formelle

David Blunier · Université de Poitiers L3 · Printemps 2025



Vers un système compositionnel

- Rappelez-vous de notre introduction que notre but est de développer une sémantique compositionnelle pour les langues naturelles.
- Une sémantique compositionnelle est un système formel permettant d'attribuer une signification à chaque constituant atomique de la phrase, qui reste la même quelle que soit la complexité de la structure dans laquelle il est utilisé.

Vers un système compositionnel

- Aujourd'hui nous allons apprendre à maîtriser l'outil formel le plus puissant à cette fin: le **lambda-calcul**.
- Le lambda-calcul va nous permettre de développer un nouveau langage, L-lambda (LL), basé sur LPred.
- Ce nouveau langage nous permettra de créer un système (quasi-)complètement compositionnel.

Le problème avec LPred

- Considérons une phrase simple du français:

$[_{NP} \text{ Selina } [_{VP} [_{V} \text{ aime}] [_{NP} \text{ Fido}]]]$

- Grâce à LPred, nous pouvons assigner une dénotation à **presque tous** les constituants syntaxiques de cette phrase:

$$[s]^{M,g} = \text{Selina}$$

$$[f]^{M,g} = \text{Fido}$$

$$[aime(s, f)]^{M,g} = \text{Selina aime Fido}$$

Le problème avec LPred

- Problème: comment faisons-nous pour représenter le syntagme verbal VP *aime Fido*?

$[_{NP} \text{ Selina } [_{VP} [_{V} \text{ aime}] [_{NP} \text{ Fido}]]]$

Le problème avec LPred

- Nous ne pouvons pas écrire la chose suivante, car ce n'est pas une formule bien formée de LPred:

☠ $\llbracket aime(x, f) \rrbracket^{M,g} = ?$

- Nous ne pouvons pas non plus écrire ceci:

☠ $\llbracket \exists x. aime(x, f) \rrbracket^{M,g}$

Car la signification de ceci est *Quelqu'un aime Fido!*

Le problème avec LPred

- Intuitivement, on aimerait représenter *aime Fido* d'une façon analogue à *aime*, i.e. comme un prédicat binaire dont seul un argument a été saturé:

$$\textit{aime}(\quad, f)$$

- Or, c'est exactement ce que le lambda-calcul va nous permettre de faire grâce à l'introduction d'un nouvel outil: **la fonction lambda**.

La fonction lambda

- La **fonction lambda**, également appelée **abstracteur lambda** ou **lambda-abstracteur**, est un élément nous permettant d'introduire une variable en tant qu'argument d'un prédicat "non-saturé", i.e. vide, afin de la lier.

$$\lambda x. aime(x, f)$$

- Comme toutes les variables doivent être liées (dans LL comme dans LPred), le résultat est une structure interprétable!
- La formule finale ici peut se lire "la fonction caractéristique de l'ensemble des éléments (dans M) qui aiment Fido", i.e., n'importe quel x tel que x aime Fido, ce qui est (intuitivement) exactement la signification de notre VP!

La fonction lambda

$$\lambda x. aime(x, f)$$

- Plus techniquement, cette formule dénote une **fonction d'un individu à une valeur de vérité**, une fonction dont la valeur sera de 1 (vraie) ssi cet individu aime Fido.
- C'est ce que l'on appelle la "fonction caractéristique de l'ensemble" de tous les individus qui aiment Fido.

La fonction lambda (λ)

- À l'aide de λ , nous pouvons également exprimer la signification d'éléments qui semblent ne pas avoir de signification autre que logique (i.e., ne renvoient ni à des individus, ni à des prédicats), comme *tout*.
- Intuitivement, la dénotation de *tout* semble être ceci:

$$\llbracket tout \rrbracket^{M,g} = \forall x. \quad (x)$$

Où l'espace blanc indique que cette position doit être remplie par un prédicat.

La fonction lambda (λ)

- Nous pouvons capturer la signification de *tout* en effectuant une lambda-abstraction sur des **prédicats**.
- Voici par exemple l'entrée lexicale (= la dénotation) de *tout* (adverbe):

$$\llbracket tout \rrbracket^{M,g} = \lambda P. \forall (x). P(x)$$

"La fonction caractéristique de l'ensemble des prédicats P tels que pour tout x , $P(x)$ est vrai".

- Il s'agit ici d'une fonction demandant un prédicat, et délivrant une valeur de vérité (vraie si le prédicat est satisfait; fausse autrement).

La fonction lambda (λ)

- Ceci est une nouveauté de LL par rapport à LPred:
- Dans LPred, nous n'avons que des **variables d'individus** $x, y, z...$
- Dans LL, nous avons également des **variables de prédicats** $P, Q...$
- Ceci nous permet d'identifier LL comme une logique **d'ordre supérieur** (*higher-order logic*).

Quelques applications

- Grâce à notre nouvel outil, nous pouvons représenter beaucoup d'expressions complexes des langues naturelles. Voici quelques exemples:

$\llbracket \text{vapoteur} \rrbracket^{M,g} = \lambda x. \text{vapote}(x)$ = l'ensemble des individus qui vapotent.

$\llbracket \text{non-vapoteur} \rrbracket^{M,g} = \lambda x. \neg \text{vapote}(x)$ = les non-vapoteurs.

$\llbracket \text{tous} \rrbracket^{M,g} = \lambda P. \lambda Q. \forall x. P(x) \rightarrow Q(x)$ = l'ensemble des choses possédant les propriétés P et Q .

$\llbracket \text{tous les vapoteurs} \rrbracket^{M,g} = \lambda Q. \forall x. \text{vapote}(x) \rightarrow Q(x)$ = tous les vapoteurs.

$\llbracket \text{tous les vapoteurs sont dehors} \rrbracket^{M,g} = \forall x. \text{vapote}(x) \rightarrow \text{dehors}(x)$

Application de la fonction λ

- Tout comme dans LPred, nous indiquons l'argument auquel la fonction s'applique entre parenthèses, à droite. Ainsi

$$[\lambda x. aime(x, f)](s)$$

Dans notre modèle équivaut à appliquer la fonction *aime Fido* à l'individu Selina.

- Une fois la fonction appliquée, cette formule devient équivalente à:

$$aime(s, f)$$

- On appelle **lambda-conversion** (λ -conversion) l'application de cette fonction.

Application de la fonction λ

λ -conversion

Pour toute variable x et expression α ,

$$[\lambda x....x...](\alpha) \equiv \alpha$$

- Ainsi, les expressions suivantes sont équivalentes:

$$[\lambda x.heureux(x)](s) \equiv heureux(s)$$

$$[\lambda x.[heureux(x) \wedge chat(x)]](s) \equiv heureux(s) \wedge chat(s)$$

Corps de la fonction λ

- Dans toute expression de la forme $\lambda x.\phi$, la partie ϕ désigne la **portée** de l'expression λ , c'est-à-dire la valeur de la fonction étant donné un argument: c'est ce que l'on appelle le **corps de la fonction**.
- Ainsi, dans l'expression

$$\lambda x. aime(s, x)$$

le corps de la fonction est $aime(s, x)$, l'ensemble des individus aimés par Selina.

Exercice

- Identifiez le corps des fonction suivantes:

$\lambda x. \text{sympa}(x)$

$\lambda x. x$

$\lambda y. \lambda x. [\text{aime}(x, y) \vee \text{aime}(y, x)]$

$\lambda z. \lambda y. \lambda x. \text{entre}(x, y, z)$

Exercice

- Lorsque c'est possible, appliquez une λ -conversion aux fonction suivantes:

$[\lambda x.x](f)$

$[\lambda P.P](homme)$

$[\lambda x.P(x)](f)$

$[\lambda y.\lambda x.R(y, x)](f)$

$[\lambda x.R(y, s)](f)$

$[\lambda P.\exists x.P(x)](homme)$

$[\lambda P.\forall x.P(x)](mortel)$

Exercice

- Lorsque c'est possible, appliquez une λ -conversion aux fonction suivantes:

$$[\lambda x.x](f) \equiv f$$

$$[\lambda P.P](homme) \equiv homme$$

$$[\lambda x.P(x)](f) \equiv P(f)$$

$$[\lambda y.\lambda x.R(y, x)](f) \equiv \lambda x.R(f, x) \text{ (via alpha-conversion)}$$

$$[\lambda x.R(y, s)](f) : \text{pas de r duction possible (la variable } x \text{ n'est pas li e)}$$

$$[\lambda P.\exists x.P(x)](homme) \equiv \exists x.homme(x)$$

$$[\lambda P.\forall x.P(x)](mortel) \equiv \forall x.mortel(x)$$

Types

- Une grande innovation de LL est l'introduction de **types**.
- Les **types** nous permettent de représenter et de définir un nombre infiniment grand d'entités dans LL.
- Un **type** désigne la catégorie syntaxique d'une expression donnée dans LL.
- Ils servent à représenter quel type de dénotation une expression peut avoir, ainsi qu'à contraindre le type d'expressions avec lequel ils peuvent se combiner.

Types

- Nous avons deux types de base dans LL: e et t .

Types

e : le type correspondant aux **entités**, représentant des **individus** (variables, constantes);

t : le type correspondant aux **valeurs de vérité**, représentant des formules.

Types

- À partir de ces types simples, nous pouvons créer des **types complexes**, également appelés des **types-fonctions**:

$$\langle e, t \rangle$$

Ce type correspond à une fonction: une fonction prenant des individus (type e) et renvoyant à une valeur de vérité (type t).

Types

- Il n'y a pas de limites quant à la complexité des types: nous pouvons les créer de **façon réursive**:

$$\langle e, \langle e, t \rangle \rangle$$

Ceci correspond à une fonction demandant un individu, et renvoyant à une fonction demandant un autre individu et renvoyant à une valeur de vérité.

- Il s'agit du type des prédicats transitifs comme *aimer*, *être le frère de*.

Types

- Voici la définition récursive des types:

e est un type;

t est un type;

Si σ est un type et τ est un type, alors $\langle \sigma, \tau \rangle$ est un type;

Rien d'autre n'est un type.

- Comme d'habitude, σ et τ sont des **méta-variables** sur des types.

Types

- Par exemple, les éléments suivants sont des types d'après notre définition:

e

$\langle e, e \rangle$

$\langle e, t \rangle$

$\langle e, \langle e, t \rangle \rangle$

$\langle \langle e, t \rangle, t \rangle$

$\langle \langle e, t \rangle, \langle e, t \rangle \rangle$

$\langle \langle e, t \rangle, \langle e, t \rangle, t \rangle$

...

Exercice

- Identifiez le type de chacune des expressions suivantes:

$\lambda x.x$

$\lambda x.P(x)$

P

db

x

$P(x)$

$[\lambda x.P(x)](s)$

$P(f)$

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

Exercice

- Identifiez le type de chacune des expressions suivantes:

$$\lambda x.x \quad \langle e, e \rangle$$

$$\lambda x.P(x) \quad \langle e, \langle e, t \rangle \rangle$$

$$P \quad \langle e, t \rangle$$

$$db \quad e$$

$$x \quad e$$

$$P(x) \quad t$$

$$[\lambda x.P(x)](s) \quad t$$

$$P(f) \quad t$$

$$\lambda y.\lambda x.R(x, y) \quad \langle e, \langle e, t \rangle \rangle$$