λ-calcul et théorie des types

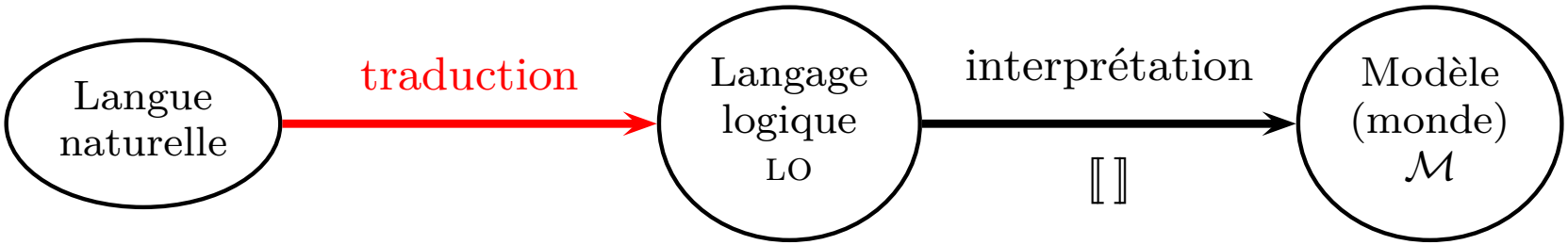
Vers une interface syntaxe-sémantique

Laurent Roussarie

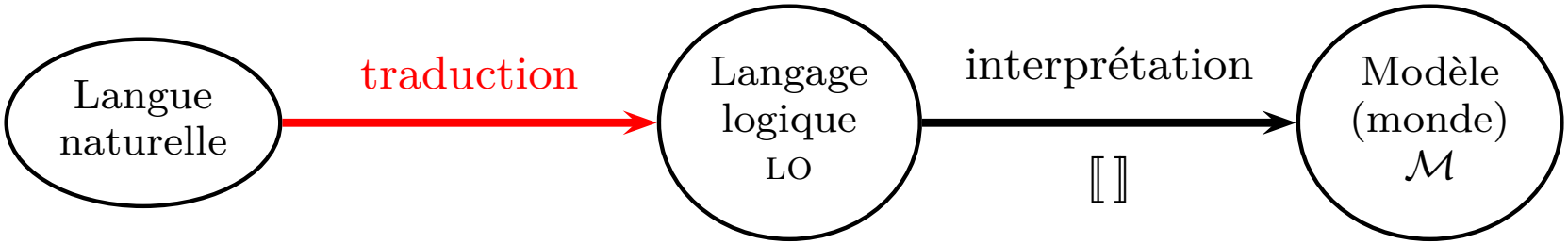
Sémantique, M1 LTD

2010

Introduction



Introduction



- **Objectif :** Se donner une système des règles qui permette d'obtenir "automatiquement" la représentation sémantique d'une phrase tout en restant compositionnel

Construction du sens

Décorer les arbres

Principe de compositionnalité

Le sens d'une expression est une fonction du sens de ses parties et de sa structure syntaxique.

☞ Notre système de règles va opérer sur des analyses syntaxique, c'est-à-dire des arbres.

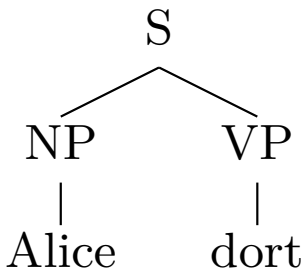
Construction du sens

Décorer les arbres

Principe de compositionnalité

Le sens d'une expression est une fonction du sens de ses parties et de sa structure syntaxique.

☞ Notre système de règles va opérer sur des analyses syntaxique, c'est-à-dire des arbres.



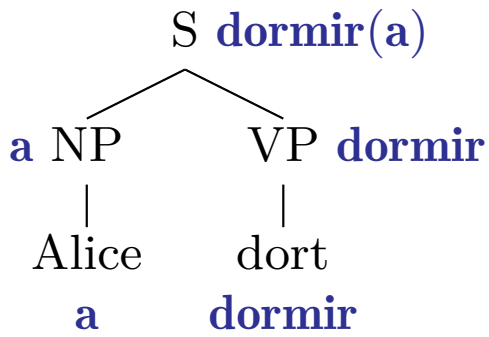
Construction du sens

Décorer les arbres

Principe de compositionnalité

Le sens d'une expression est une fonction du sens de ses parties et de sa structure syntaxique.

☞ Notre système de règles va opérer sur des analyses syntaxique, c'est-à-dire des arbres.

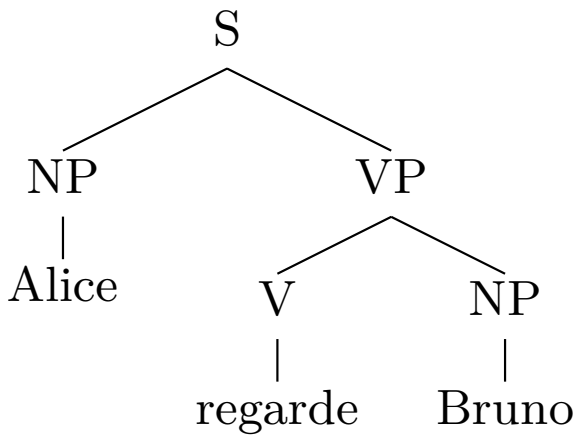


Enjeu : Comment passer "mécaniquement" des ingrédients (nœuds) au plat final (racine) ?

Construction du sens

Expressions non saturées

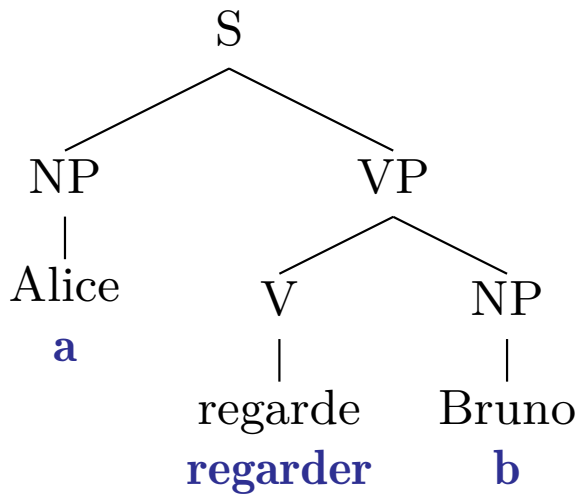
Par compositionnalité, tout constituant interprétable doit avoir une représentation sémantique valide.



Construction du sens

Expressions non saturées

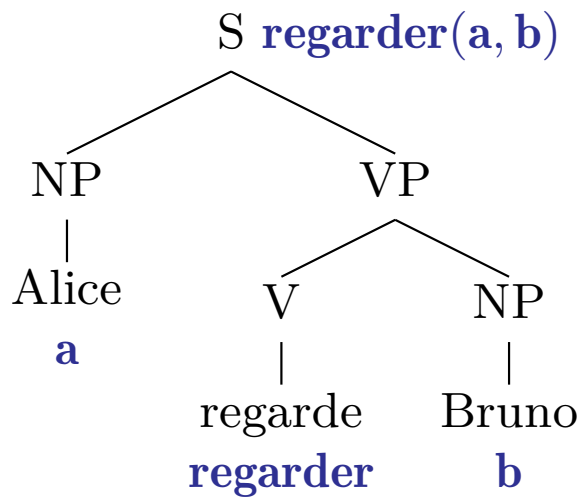
Par compositionnalité, tout constituant interprétable doit avoir une représentation sémantique valide.



Construction du sens

Expressions non saturées

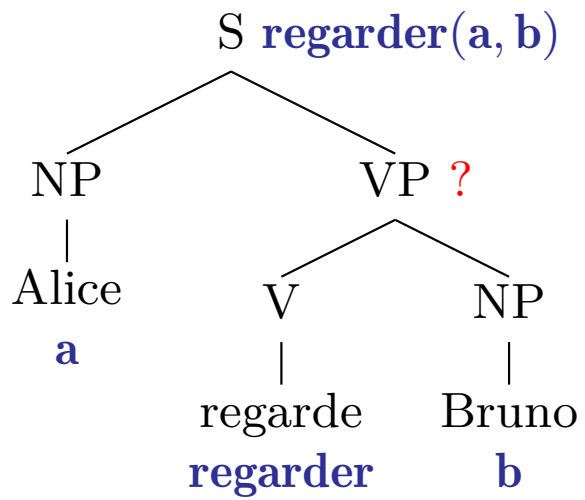
Par compositionnalité, tout constituant interprétable doit avoir une représentation sémantique valide.



Construction du sens

Expressions non saturées

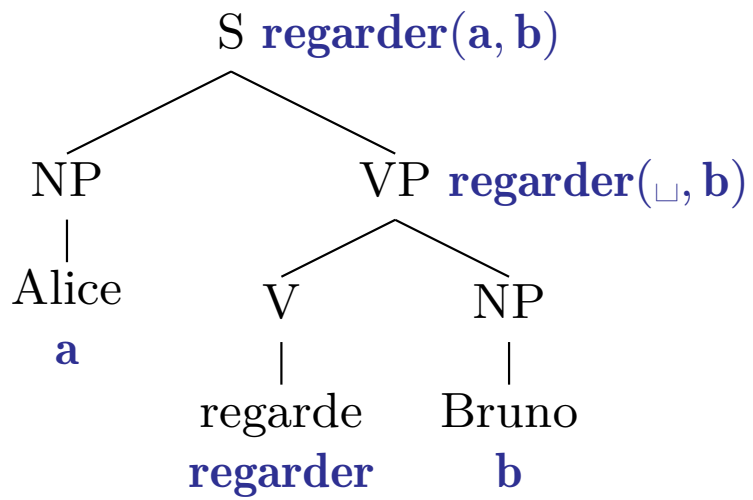
Par compositionnalité, tout constituant interprétable doit avoir une représentation sémantique valide.



Construction du sens

Expressions non saturées

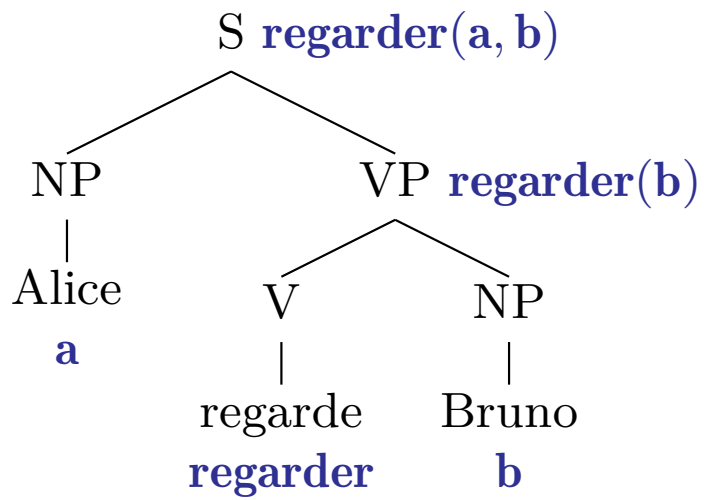
Par compositionnalité, tout constituant interprétable doit avoir une représentation sémantique valide.



Construction du sens

Expressions non saturées

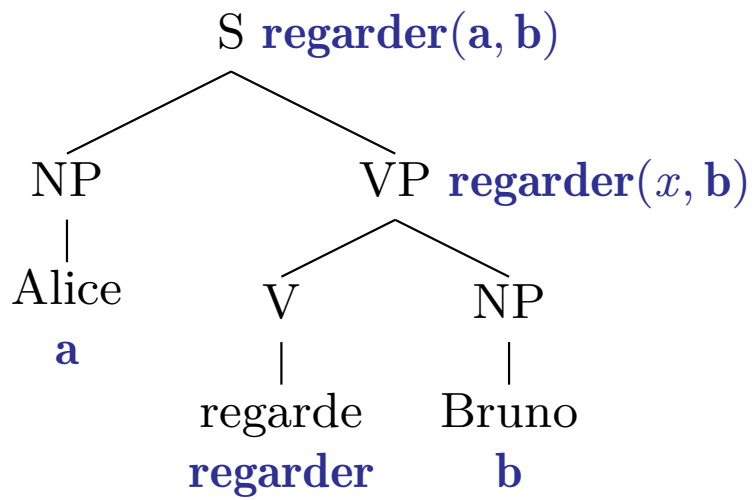
Par compositionnalité, tout constituant interprétable doit avoir une représentation sémantique valide.



Construction du sens

Expressions non saturées

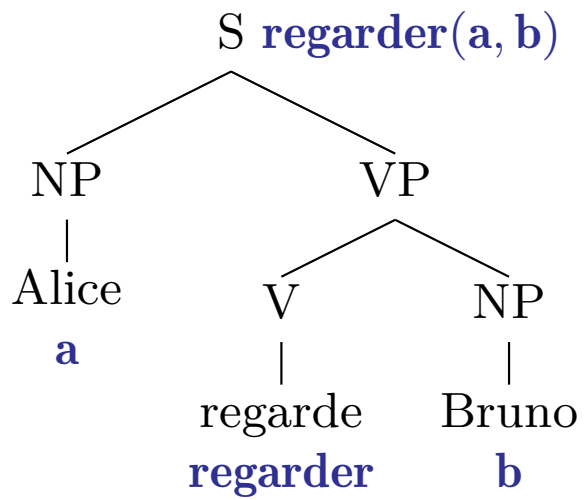
Par compositionnalité, tout constituant interprétable doit avoir une représentation sémantique valide.



Construction du sens

Expressions non saturées

Par compositionnalité, tout constituant interprétable doit avoir une représentation sémantique valide.



Allons voir ce que peut nous dire la sémantique de LO.

Les prédicats dénotent des fonctions

Prédicats à une place

$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) =$ l'**ensemble** de tous les dormeurs de w.

Les prédicats dénotent des fonctions

Prédicats à une place

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) = \text{l'}\textbf{ensemble} \text{ de tous les dormeurs de } w.$$

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) = \begin{array}{ccc} \mathcal{A} & \longrightarrow & \{0\,;1\} \\ \text{x} & \longmapsto & \begin{cases} 1 \text{ si x dort dans } w \\ 0 \text{ sinon} \end{cases} \end{array}$$

Les prédicats dénotent des fonctions

Prédicats à une place

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) = \text{l'}\textbf{ensemble}\text{ de tous les dormeurs de } w.$$

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) = \begin{array}{ccc} \mathcal{A} & \longrightarrow & \{0\,;\,1\} \\ \text{x} & \longmapsto & \begin{cases} 1 \text{ si x dort dans } w \\ 0 \text{ sinon} \end{cases} \end{array}$$

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = \left[\begin{array}{ll} \textsc{Alice} & \\ \textsc{Bruno} & 1 \\ \textsc{Charles} & \\ \textsc{Dina} & 0 \end{array}\right]$$

Les prédicats dénotent des fonctions

Prédicats à une place

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) = \text{l'}\textbf{ensemble}\text{ de tous les dormeurs de } w.$$

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) = \mathcal{A} \longrightarrow \{0\ ;\ 1\}$$

$$\text{x} \longmapsto \begin{cases} 1 \text{ si x dort dans } w \\ 0 \text{ sinon} \end{cases}$$

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = \left[\begin{array}{ll} \textsc{Alice} & \\ \textsc{Bruno} & 1 \\ \textsc{Charles} & \\ \textsc{Dina} & 0 \end{array}\right] \equiv \left\{\begin{array}{l} \textsc{Alice}\ ; \\ \textsc{Bruno}\ ; \\ \textsc{Dina} \end{array}\right\}$$

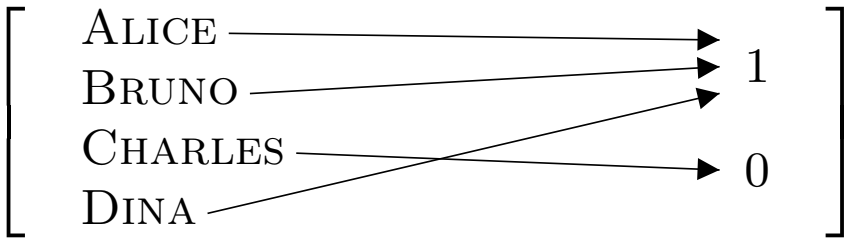
Les prédicats dénotent des fonctions

Prédicats à une place

$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) =$ l'**ensemble** de tous les dormeurs de w.

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) = \begin{array}{ccc} \mathcal{A} & \longrightarrow & \{0\ ;\ 1\} \\ \text{x} & \longmapsto & \begin{cases} 1 \text{ si x dort dans } w \\ 0 \text{ sinon} \end{cases} \end{array}$$

$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} =$



$[\![\textit{Alice dort}]\!]^{\mathcal{M},w,g} = ?$

Les prédicats dénotent des fonctions

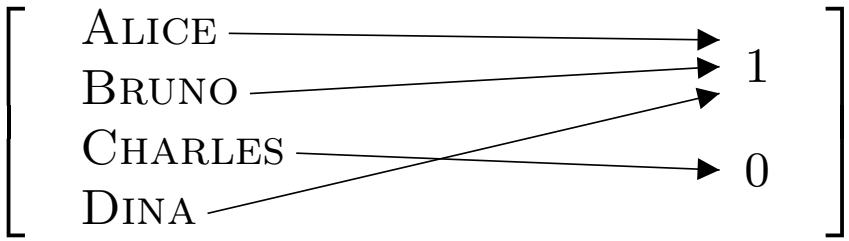
Prédicats à une place

$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) =$ l'**ensemble** de tous les dormeurs de w.

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) = \mathcal{A} \longrightarrow \{0\ ;\ 1\}$$

$$\text{x} \longmapsto \begin{cases} 1 \text{ si x dort dans } w \\ 0 \text{ sinon} \end{cases}$$

$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} =$



$[\![\textit{Alice dort}]\!]^{\mathcal{M},w,g} = ?$

$[\![\textbf{dormir(a)}]\!]^{\mathcal{M},w,g} =$

Les prédicats dénotent des fonctions

Prédicats à une place

$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) =$ l'**ensemble** de tous les dormeurs de w.

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) = \begin{array}{ccc} \mathcal{A} & \longrightarrow & \{0\ ;\ 1\} \\ \text{x} & \longmapsto & \begin{cases} 1 \text{ si x dort dans } w \\ 0 \text{ sinon} \end{cases} \end{array}$$

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = \left[\begin{array}{ll} \textsc{Alice} & \\ \textsc{Bruno} & 1 \\ \textsc{Charles} & \\ \textsc{Dina} & 0 \end{array}\right]$$

$[\![\textit{Alice dort}]\!]^{\mathcal{M},w,g} = ?$

$[\![\textbf{dormir(a)}]\!]^{\mathcal{M},w,g} =$

$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g}$

Les prédicats dénotent des fonctions

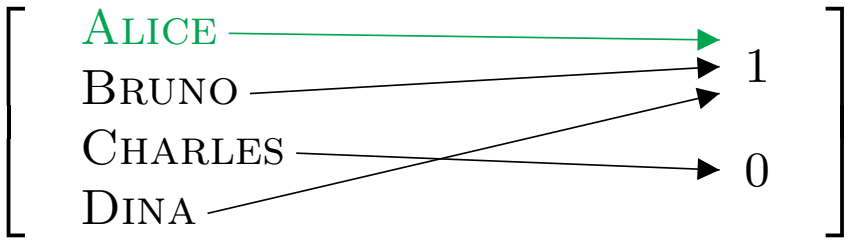
Prédicats à une place

$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) =$ l'**ensemble** de tous les dormeurs de w.

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) = \mathcal{A} \longrightarrow \{0\ ;\ 1\}$$

$$\text{x} \longmapsto \begin{cases} 1 \text{ si x dort dans } w \\ 0 \text{ sinon} \end{cases}$$

$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} =$



$[\![\textit{Alice dort}]\!]^{\mathcal{M},w,g} = ?$

$[\![\textbf{dormir(a)}]\!]^{\mathcal{M},w,g} =$

$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g}([\![\textbf{a}]\!]^{\mathcal{M},w,g})$

Les prédicats dénotent des fonctions

Prédicats à une place

$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) =$ l'**ensemble** de tous les dormeurs de w.

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = F(w, \textbf{dormir}) = \begin{array}{ccc} \mathcal{A} & \longrightarrow & \{0\,;\,1\} \\ \text{x} & \longmapsto & \begin{cases} 1 \text{ si x dort dans } w \\ 0 \text{ sinon} \end{cases} \end{array}$$

$$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g} = \left[\begin{array}{ll} \textsc{Alice} & \\ \textsc{Bruno} & 1 \\ \textsc{Charles} & \\ \textsc{Dina} & 0 \end{array}\right]$$

$[\![\textit{Alice dort}]\!]^{\mathcal{M},w,g} = 1$

$[\![\textbf{dormir(a)}]\!]^{\mathcal{M},w,g} = 1$

$[\![\textbf{dormir}]\!]^{\mathcal{M},w,g}([\![\textbf{a}]\!]^{\mathcal{M},w,g}) = 1$

Les prédicats dénotent des fonctions

Prédicats à deux arguments

Les prédicats dénotent des fonctions

Prédicats à deux arguments

... regarde Bruno

$$[\![\textbf{regarder}(\sqcup, \textbf{b})]\!]^{\mathcal{M},w,g} = \left[\begin{array}{ll} \textsc{Alice} & \\ \textsc{Bruno} & 1 \\ \textsc{Charles} & \\ \textsc{Dina} & 0 \end{array}\right]$$

Les prédicats dénotent des fonctions

Prédicats à deux arguments

... regarde Alice

$$[\![\textbf{regarder}(\sqcup, \textbf{a})]\!]^{\mathcal{M},w,g} = \left[\begin{array}{ll} \textsc{Alice} & \\ \textsc{Bruno} & 1 \\ \textsc{Charles} & \\ \textsc{Dina} & 0 \end{array}\right]$$

Les prédicats dénotent des fonctions

Prédicats à deux arguments

... regarde Charles

$$[\![\textbf{regarder}(\sqcup, \textbf{c})]\!]^{\mathcal{M},w,g} = \left[\begin{array}{ll} \textsc{Alice} & \\ \textsc{Bruno} & 1 \\ \textsc{Charles} & \\ \textsc{Dina} & 0 \end{array}\right]$$

Les prédicats dénotent des fonctions

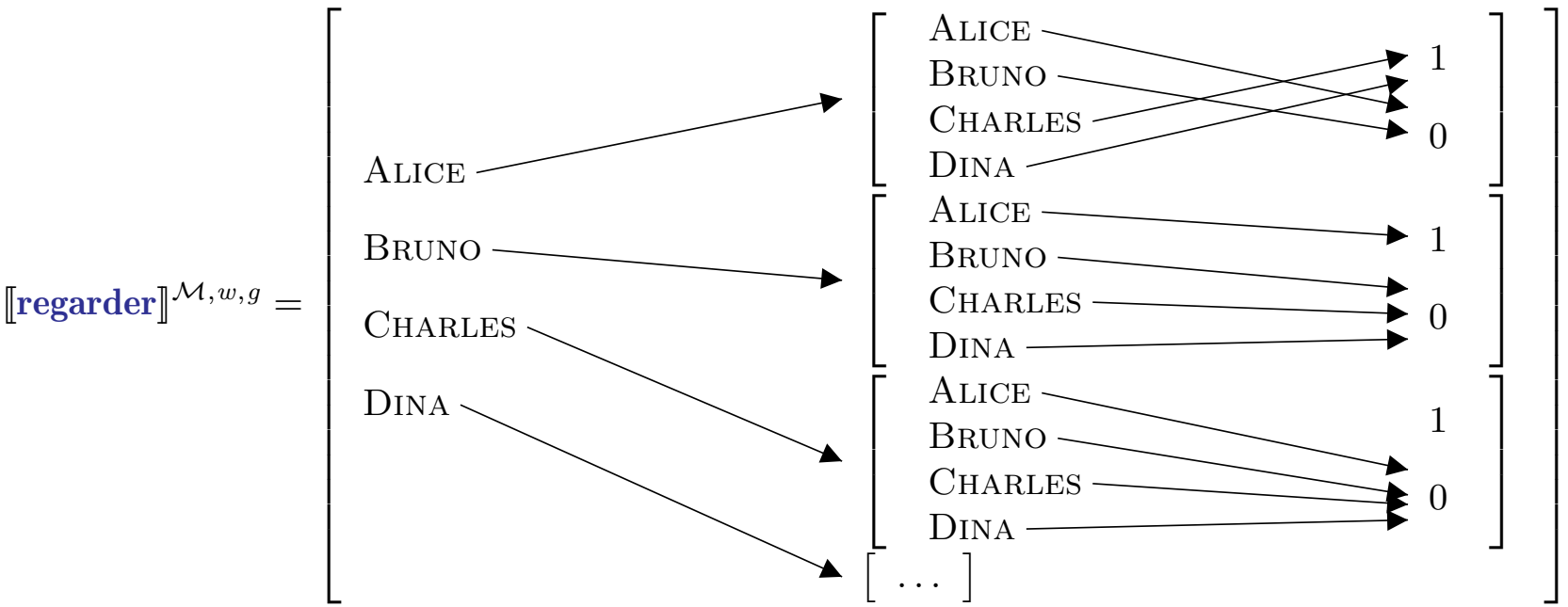
Prédicats à deux arguments

... regarde ...

$$[\![\textbf{regarder}]\!]^{\mathcal{M},w,g} = \left[\begin{array}{ll} \textsc{Alice} & \longrightarrow \text{« qui regarde } \textsc{Alice} \text{ ? »} \\ \textsc{Bruno} & \longrightarrow \text{« qui regarde } \textsc{Bruno} \text{ ? »} \\ \textsc{Charles} & \longrightarrow \text{« qui regarde } \textsc{Charles} \text{ ? »} \\ \textsc{Dina} & \longrightarrow \text{« qui regarde } \textsc{Dina} \text{ ? »} \end{array}\right]$$

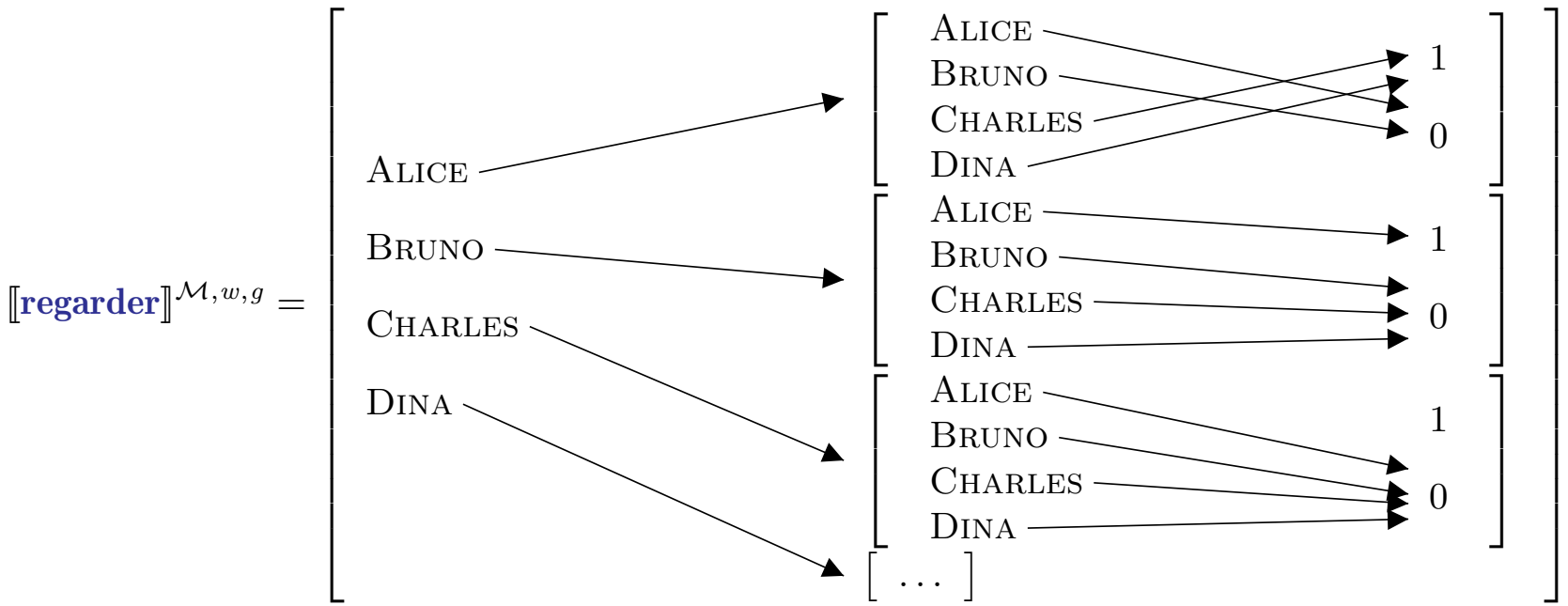
Les prédicats dénotent des fonctions

Prédicats à deux arguments



Les prédicats dénotent des fonctions

Prédicats à deux arguments

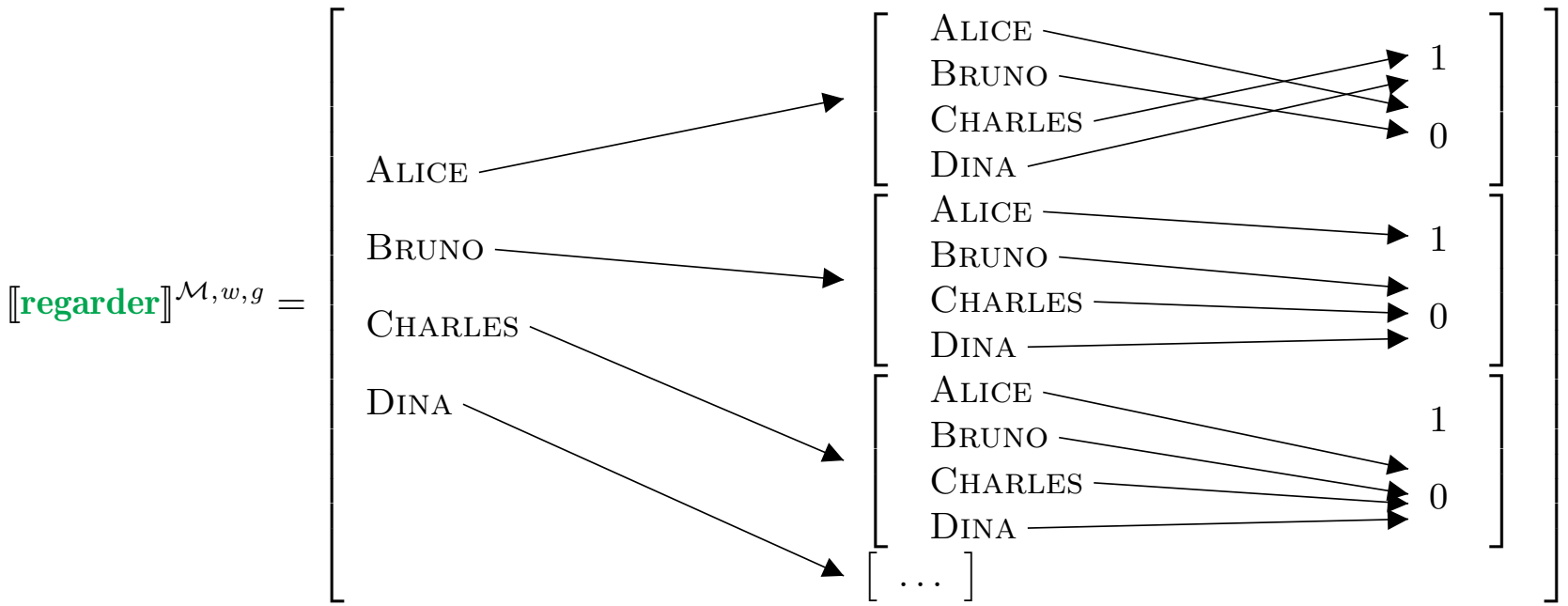


$[\![\textit{Dina regarde Alice}]\!]^{\mathcal{M},w,g} = ?$

$[\![\textbf{regarder}(\textbf{d},\textbf{a})]\!]^{\mathcal{M},w,g} = ?$

Les prédicats dénotent des fonctions

Prédicats à deux arguments



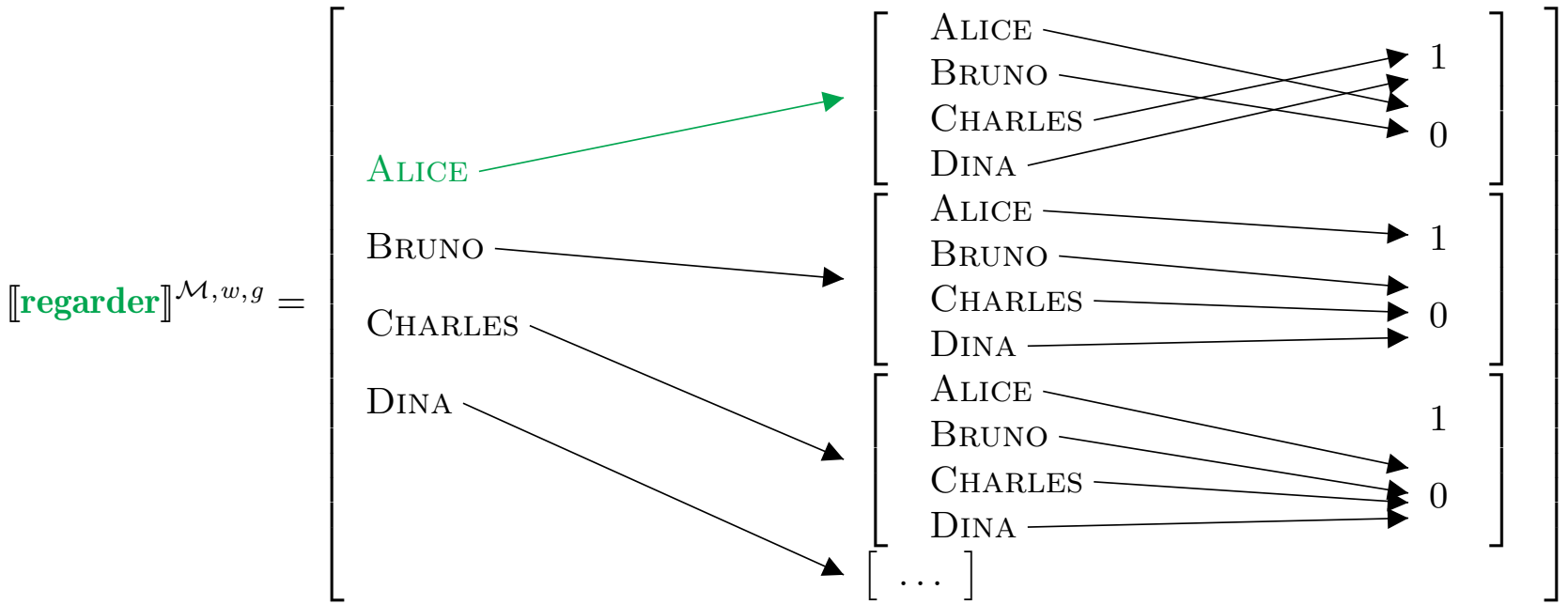
$[\![\textit{Dina regarde Alice}]\!]^{\mathcal{M},w,g} = ?$

$[\![\textbf{regarder}(\textbf{d},\textbf{a})]\!]^{\mathcal{M},w,g} = ?$

$[\![\textbf{regarder}]\!]^{\mathcal{M},w,g}$

Les prédicats dénotent des fonctions

Prédicats à deux arguments



$[\![\textit{Dina regarde Alice}]\!]^{\mathcal{M},w,g} = ?$

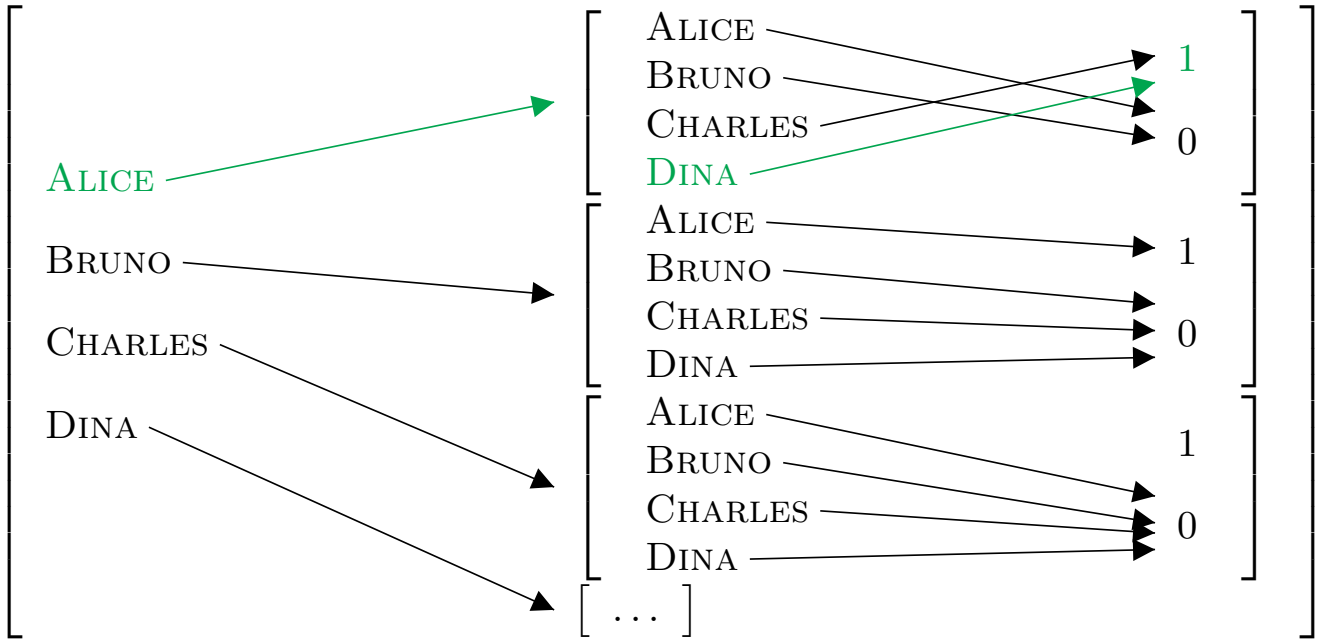
$[\![\textbf{regarder}(\textbf{d},\textbf{a})]\!]^{\mathcal{M},w,g} = ?$

$[\![\textbf{regarder}]\!]^{\mathcal{M},w,g}([\![\textbf{a}]\!]^{\mathcal{M},w,g})$

Les prédicats dénotent des fonctions

Prédicats à deux arguments

$[\![\textbf{regarder}]\!]^{\mathcal{M},w,g} =$



$[\![\textit{Dina regarde Alice}]\!]^{\mathcal{M},w,g} = 1$

$[\![\textbf{regarder}(\textbf{d},\textbf{a})]\!]^{\mathcal{M},w,g} = 1$

$[\![\textbf{regarder}]\!]^{\mathcal{M},w,g}([\![\textbf{a}]\!]^{\mathcal{M},w,g})([\![\textbf{d}]\!]^{\mathcal{M},w,g}) = 1$

Retour à la syntaxe de LO

Des trous dans les formules

Les prédicats = des formules incomplètes, non saturées

☞ On va écrire des formules “inachevées” en indiquant quels éléments manquent.

$\mathbf{dormir}(x)$	saturée

Retour à la syntaxe de LO

Des trous dans les formules

Les prédicats = des formules incomplètes, non saturées

☞ On va écrire des formules “inachevées” en indiquant quels éléments manquent.

$\lambda x\, \mathbf{dormir}(x)$ désaturée

$\lambda x \approx$ “dans ce qui suit, il manque x”

Retour à la syntaxe de LO

Des trous dans les formules

Les prédicats = des formules incomplètes, non saturées

☞ On va écrire des formules “inachevées” en indiquant quels éléments manquent.

$\lambda x\, \mathbf{dormir}(x)$	désaturée

$\lambda x \approx$ “dans ce qui suit, il manque une valeur pour x”

Retour à la syntaxe de LO

Des trous dans les formules

Les prédicats = des formules incomplètes, non saturées

☞ On va écrire des formules “inachevées” en indiquant quels éléments manquent.

$\lambda x\, \mathbf{dormir}(x)$ désaturée

$\lambda x \approx$ “dans ce qui suit, il manque une valeur pour x”

- λ : opérateur d’abstraction
- λ s’applique à une variable (c’est un lieur)
- On appellera une expression qui commence par λ un λ-terme.

Retour à la syntaxe de LO

Des trous dans les formules

Les prédicats = des formules incomplètes, non saturées

☞ On va écrire des formules “inachevées” en indiquant quels éléments manquent.

$\lambda x\, \mathbf{dormir}(x)$ désaturée

$\lambda x \approx$ “dans ce qui suit, il manque une valeur pour x”

- λ : opérateur d’abstraction
- λ s’applique à une variable (c’est un lieur)
- On appellera une expression qui commence par λ un λ-terme.

$\lambda y \lambda x\, \mathbf{regarder}(x, y)$

Retour à la syntaxe de LO

Des trous dans les formules

Les prédicats = des formules incomplètes, non saturées

☞ On va écrire des formules “inachevées” en indiquant quels éléments manquent.

$\lambda x\, \mathbf{dormir}(x)$ désaturée

$\lambda x \approx$ “dans ce qui suit, il manque une valeur pour x”

- λ : opérateur d’abstraction
- λ s’applique à une variable (c’est un lieur)
- On appellera une expression qui commence par λ un λ-terme.

$\lambda y \lambda x\, \mathbf{regarder}(x, y)$

$[_{\text{VP}}\ \textit{regarde}\ \ \textit{Bruno}] \rightsquigarrow \lambda x\, \mathbf{regarder}(x, \mathbf{b})$

Retour à la syntaxe de LO

Des trous dans les formules

Les prédicats = des formules incomplètes, non saturées

☞ On va écrire des formules “inachevées” en indiquant quels éléments manquent.

$\lambda x\, \mathbf{dormir}(x)$ désaturée

$\lambda x \approx$ “dans ce qui suit, il manque une valeur pour x”

- λ : opérateur d’abstraction
- λ s’applique à une variable (c’est un lieur)
- On appellera une expression qui commence par λ un λ-terme.

$\lambda y \lambda x\, \mathbf{regarder}(x, y)$

$[_{\text{VP}}\ \textit{regarde}\ \textit{Bruno}] \rightsquigarrow \lambda x\, \mathbf{regarder}(x, \mathbf{b})$
Remarque : $\lambda x\, \mathbf{dormir}(x) \equiv \mathbf{dormir}$ et $\lambda y \lambda x\, \mathbf{regarder}(x, y) \equiv \mathbf{regarder}$

λ-abstraction

Une nouvelle règle syntaxique de LO

$\lambda x\, \mathbf{dormir}(x)$, $\lambda y \lambda x\, \mathbf{regarder}(x, y)$, $\lambda x\, \mathbf{regarder}(x, \mathbf{b})$ sont des expressions bien formées de LO (mais pas des formules !)

λ-abstraction

Une nouvelle règle syntaxique de LO

$\lambda x\, \mathbf{dormir}(x)$, $\lambda y \lambda x\, \mathbf{regarder}(x, y)$, $\lambda x\, \mathbf{regarder}(x, \mathbf{b})$ sont des expressions bien formées de LO (mais pas des formules !)

λ-abstraction

Si α est une expression bien formée de LO et si v est une variable de LO, alors $\lambda v \alpha$ est une expression bien formée de LO.

λ-abstraction

Une nouvelle règle syntaxique de LO

$\lambda x\,\mathbf{dormir}(x)$, $\lambda y \lambda x\,\mathbf{regarder}(x, y)$, $\lambda x\,\mathbf{regarder}(x, \mathbf{b})$ sont des expressions bien formées de LO (mais pas des formules !)

λ-abstraction

Si α est une expression bien formée de LO et si v est une variable de LO, alors $\lambda v \alpha$ est une expression bien formée de LO.

Un $\lambda \longrightarrow$ prédicat à un argument,
Deux $\lambda \longrightarrow$ prédicat à deux arguments, etc.
donc $\lambda x\,\mathbf{regarder}(x, \mathbf{b})$ est un prédicat à un argument (normal, c'est un VP)

Types : catégories d'expressions de LO

Types de base

Nous allons catégoriser les expressions de LO selon ce qu'elles dénotent dans le modèle.
Ces catégories s'appellent des types.

Types : catégories d'expressions de LO

Types de base

Nous allons catégoriser les expressions de LO selon ce qu'elles dénotent dans le modèle.
Ces catégories s'appellent des types.

- Les expressions qui dénotent des individus de $\mathcal{A}$ sont de type e.
  Ce sont les termes.

Types : catégories d'expressions de LO

Types de base

Nous allons catégoriser les expressions de LO selon ce qu'elles dénotent dans le modèle.
Ces catégories s'appellent des types.

- Les expressions qui dénotent des individus de $\mathcal{A}$ sont de type e.
  Ce sont les termes.
- Les expressions qui dénotent une valeur de vérité (1/0) sont de type t.
  Ce sont les formules.

Types : catégories d'expressions de LO

Types de base

Nous allons catégoriser les expressions de LO selon ce qu'elles dénotent dans le modèle.
Ces catégories s'appellent des types.

- Les expressions qui dénotent des individus de $\mathcal{A}$ sont de type e.
  Ce sont les termes.
- Les expressions qui dénotent une valeur de vérité (1/0) sont de type t.
  Ce sont les formules.
- Et les autres expressions ? Les prédicats ?

Types : catégories d'expressions de LO

Types de base

Nous allons catégoriser les expressions de LO selon ce qu'elles dénotent dans le modèle.
Ces catégories s'appellent des types.

- Les expressions qui dénotent des individus de $\mathcal{A}$ sont de type e.
  Ce sont les termes.
- Les expressions qui dénotent une valeur de vérité (1/0) sont de type t.
  Ce sont les formules.
- Et les autres expressions ? Les prédicats ?
  Ils dénotent des fonctions.

Interprétation des λ-termes

Les prédicats à un argument, comme **dormir** ou $\lambda x\,\textbf{dormir}(x)$ dénotent des fonctions de $\mathcal{A}$ vers $\{0\ ;\ 1\}$. $\mathcal{A} \longrightarrow \{0\ ;\ 1\}$

$$[\![\lambda x\,\textbf{dormir}(x)]\!]^{\mathcal{M},w,g} = \left[\begin{array}{ll} \textsc{Alice} & \\ \textsc{Bruno} & 1 \\ \textsc{Charles} & \\ \textsc{Dina} & 0 \end{array}\right]$$

- $[\![\lambda x\,\textbf{dormir}(x)]\!]^{\mathcal{M},w,g}$ attend un argument qui est un individu et retourne comme résultat une valeur de vérité.
- $\lambda x\,\textbf{dormir}(x)$ attend un argument de type e et donne comme résultat qui est de type t, i.e. une formule.

Interprétation des λ-termes

Les prédicats à un argument, comme **dormir** ou $\lambda x\,\textbf{dormir}(x)$ dénotent des fonctions de $\mathcal{A}$ vers $\{0\ ;\ 1\}$. $\mathcal{A} \longrightarrow \{0\ ;\ 1\}$

$$[\![\lambda x\,\textbf{dormir}(x)]\!]^{\mathcal{M},w,g} = \left[\begin{array}{ll} \textsc{Alice} & \\ \textsc{Bruno} & 1 \\ \textsc{Charles} & \\ \textsc{Dina} & 0 \end{array}\right]$$

- $[\![\lambda x\,\textbf{dormir}(x)]\!]^{\mathcal{M},w,g}$ attend un argument qui est un individu et retourne comme résultat une valeur de vérité.
- $\lambda x\,\textbf{dormir}(x)$ attend un argument de type e et donne comme résultat qui est de type t, i.e. une formule.

C'est une fonction de type e → t

Interprétation des λ-termes

Les prédicats à un argument, comme **dormir** ou $\lambda x\,\textbf{dormir}(x)$ dénotent des fonctions de $\mathcal{A}$ vers $\{0\,;\,1\}$. $\mathcal{A} \longrightarrow \{0\,;\,1\}$

$$[\![\lambda x\,\textbf{dormir}(x)]\!]^{\mathcal{M},w,g} = \left[\begin{array}{ll} \textsc{Alice} & \\ \textsc{Bruno} & 1 \\ \textsc{Charles} & \\ \textsc{Dina} & 0 \end{array}\right]$$

- $[\![\lambda x\,\textbf{dormir}(x)]\!]^{\mathcal{M},w,g}$ attend un argument qui est un individu et retourne comme résultat une valeur de vérité.
- $\lambda x\,\textbf{dormir}(x)$ attend un argument de type e et donne comme résultat qui est de type t, i.e. une formule.

C'est une fonction de type $\langle \mathrm{e}, \mathrm{t} \rangle$

Interprétation des λ-termes

Les prédicats à un argument, comme **dormir** ou $\lambda x\,\textbf{dormir}(x)$ dénotent des fonctions de $\mathcal{A}$ vers $\{0\,;\,1\}$. $\mathcal{A} \longrightarrow \{0\,;\,1\}$

$$[\![\lambda x\,\textbf{dormir}(x)]\!]^{\mathcal{M},w,g} = \left[\begin{array}{ll} \textsc{Alice} & \\ \textsc{Bruno} & 1 \\ \textsc{Charles} & \\ \textsc{Dina} & 0 \end{array}\right]$$

- $[\![\lambda x\,\textbf{dormir}(x)]\!]^{\mathcal{M},w,g}$ attend un argument qui est un individu et retourne comme résultat une valeur de vérité.
- $\lambda x\,\textbf{dormir}(x)$ attend un argument de type e et donne comme résultat qui est de type t, i.e. une formule.

C'est une fonction de type $\langle \mathrm{e}, \mathrm{t}\rangle$

$\langle \mathrm{e}, \mathrm{t}\rangle$ = type des expressions qui dénotent une fonction dont l'argument est de type e et le résultat est de type t.

Type fonctionnel

Type fonctionnel

Si a et b sont deux types, alors $\langle a, b \rangle$ est aussi un type.

Type fonctionnel

Type fonctionnel

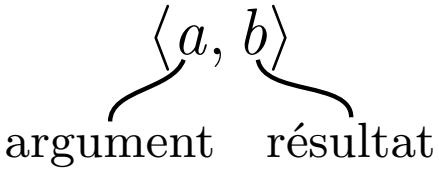
Si a et b sont deux types, alors $\langle a, b\rangle$ est aussi un type. C'est le type des expressions qui dénotent des fonctions dont l'argument est de type a et le résultat de type b.

Type fonctionnel

Type fonctionnel

Si a et b sont deux types, alors $\langle a, b\rangle$ est aussi un type. C'est le type des expressions qui dénotent des fonctions dont l'argument est de type a et le résultat de type b.

Les types fonctionnels sont toujours binaires (malgré les apparences).

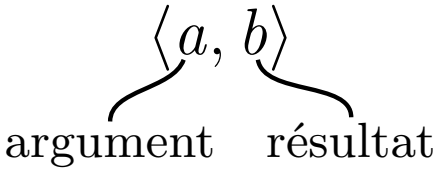


Type fonctionnel

Type fonctionnel

Si a et b sont deux types, alors $\langle a, b\rangle$ est aussi un type. C'est le type des expressions qui dénotent des fonctions dont l'argument est de type a et le résultat de type b.

Les types fonctionnels sont toujours binaires (malgré les apparences).



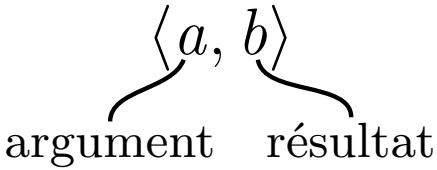
Donc techniquement les expressions fonctionnelles (i.e. les prédicats) de LO dénotent toujours des fonctions à 1 argument

Type fonctionnel

Type fonctionnel

Si a et b sont deux types, alors $\langle a, b\rangle$ est aussi un type. C'est le type des expressions qui dénotent des fonctions dont l'argument est de type a et le résultat de type b.

Les types fonctionnels sont toujours binaires (malgré les apparences).



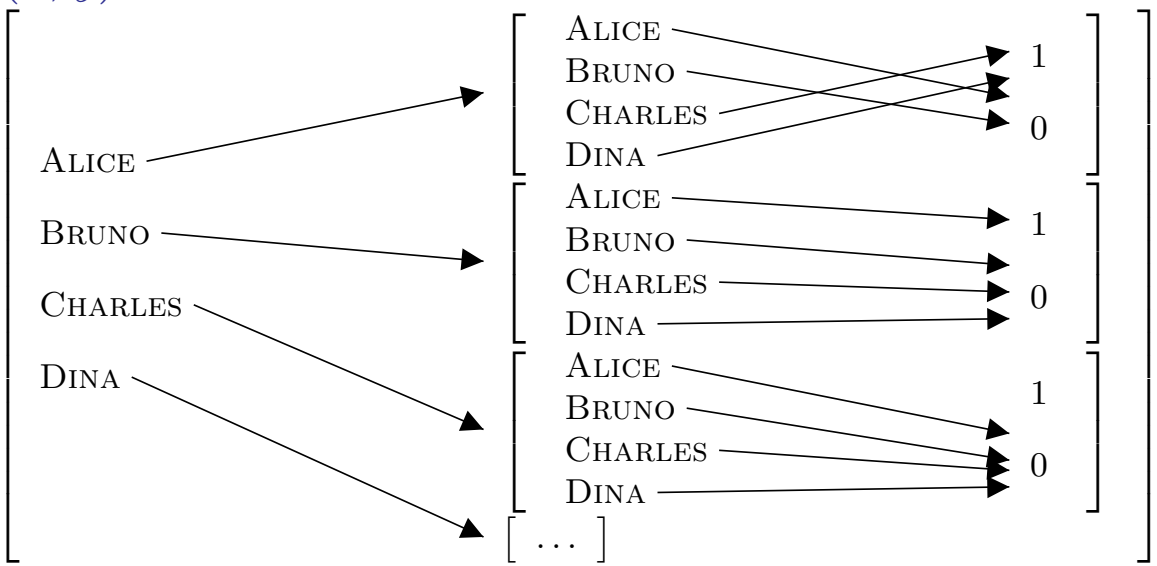
Donc techniquement les expressions fonctionnelles (i.e. les prédicats) de LO dénotent toujours des fonctions à 1 argument (... à la fois).

Prédicats à deux arguments

Un prédicat à deux arguments dénote une fonction qui lorsqu'on lui donne un argument de type e, renvoie une fonction de type $\langle \mathrm{e}, \mathrm{t} \rangle$.

$\lambda y \lambda x \, \mathbf{regarder}(x, y)$

$[\![\mathbf{regarder}]\!]^{\mathcal{M},w,g} =$

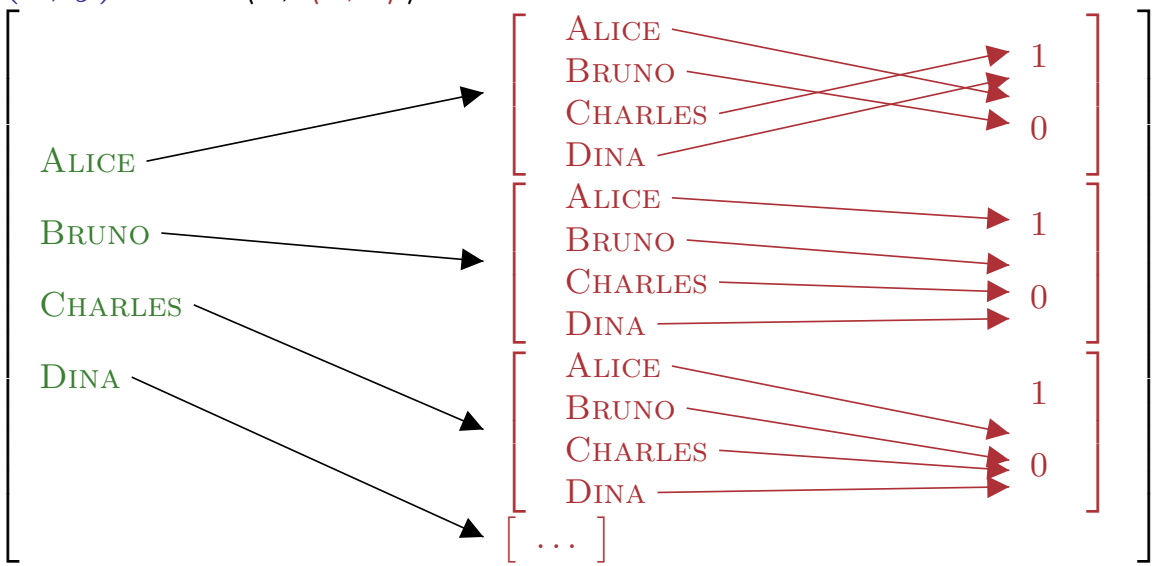


Prédicats à deux arguments

Un prédicat à deux arguments dénote une fonction qui lorsqu'on lui donne un argument de type e, renvoie une fonction de type $\langle e, t\rangle$.

$\lambda y \lambda x\, \textbf{regarder}(x, y)$ $\langle e, \langle e, t\rangle\rangle$

$[\![\textbf{regarder}]\!]^{\mathcal{M},w,g} =$



Des fonctions dans le modèle

On conçoit une grande variété de fonctions dans un modèle donné.

type	dénotation : fonction	expressions de LO
$\langle e, t\rangle$	$\mathcal{A} \longrightarrow \{0\ ;\ 1\}$	prédicat à 1 place
$\langle e, \langle e, t\rangle\rangle$	$\mathcal{A} \longrightarrow (\mathcal{A} \longrightarrow \{0\ ;\ 1\})$	prédicat à 2 places
$\langle e, \langle e, \langle e, t\rangle\rangle\rangle$	$\mathcal{A} \longrightarrow (\mathcal{A} \longrightarrow (\mathcal{A} \longrightarrow \{0\ ;\ 1\}))$	prédicat à 3 places
$\langle t, t\rangle$	$\{0\ ;\ 1\} \longrightarrow \{0\ ;\ 1\}$	opérateur logique (comme $\neg$)
$\langle\langle e, t\rangle, t\rangle$	$(\mathcal{A} \longrightarrow \{0\ ;\ 1\}) \longrightarrow \{0\ ;\ 1\}$	ensemble de prédicats à 1 place

Des fonctions dans le modèle

On conçoit une grande variété de fonctions dans un modèle donné.

type	dénotation : fonction	expressions de LO
$\langle e, t\rangle$	$\mathcal{A} \longrightarrow \{0\ ;\ 1\}$	prédicat à 1 place
$\langle e, \langle e, t\rangle\rangle$	$\mathcal{A} \longrightarrow (\mathcal{A} \longrightarrow \{0\ ;\ 1\})$	prédicat à 2 places
$\langle e, \langle e, \langle e, t\rangle\rangle\rangle$	$\mathcal{A} \longrightarrow (\mathcal{A} \longrightarrow (\mathcal{A} \longrightarrow \{0\ ;\ 1\}))$	prédicat à 3 places
$\langle t, t\rangle$	$\{0\ ;\ 1\} \longrightarrow \{0\ ;\ 1\}$	opérateur logique (comme $\neg$)
$\langle\langle e, t\rangle, t\rangle$	$(\mathcal{A} \longrightarrow \{0\ ;\ 1\}) \longrightarrow \{0\ ;\ 1\}$	ensemble de prédicats à 1 place

Rappel : la dénotation d'une expression de type $\langle e, t\rangle$ peut aussi être vue comme un **ensemble** d'individus.

Des fonctions dans le modèle

On conçoit une grande variété de fonctions dans un modèle donné.

type	dénotation : fonction	expressions de LO
$\langle e, t\rangle$	$\mathcal{A} \longrightarrow \{0\ ;\ 1\}$	prédicat à 1 place
$\langle e, \langle e, t\rangle\rangle$	$\mathcal{A} \longrightarrow (\mathcal{A} \longrightarrow \{0\ ;\ 1\})$	prédicat à 2 places
$\langle e, \langle e, \langle e, t\rangle\rangle\rangle$	$\mathcal{A} \longrightarrow (\mathcal{A} \longrightarrow (\mathcal{A} \longrightarrow \{0\ ;\ 1\}))$	prédicat à 3 places
$\langle t, t\rangle$	$\{0\ ;\ 1\} \longrightarrow \{0\ ;\ 1\}$	opérateur logique (comme $\neg$)
$\langle\langle e, t\rangle, t\rangle$	$(\mathcal{A} \longrightarrow \{0\ ;\ 1\}) \longrightarrow \{0\ ;\ 1\}$	ensemble de prédicats à 1 place

Rappel : la dénotation d'une expression de type $\langle e, t\rangle$ peut aussi être vue comme un **ensemble** d'individus.

De manière générale : on assimile la dénotation de toute expression de type $\langle a, t\rangle$ à un ensemble d'objets qui sont de type a.

La fabrique de fonctions

Anatomie d'un λ-terme

Révision de la règle syntaxique

λ-abstraction

Si α est une expression de type a dans LO et si v est une variable de type b, alors $\lambda v \alpha$ est une expression bien formée de type $\langle b, a \rangle$.

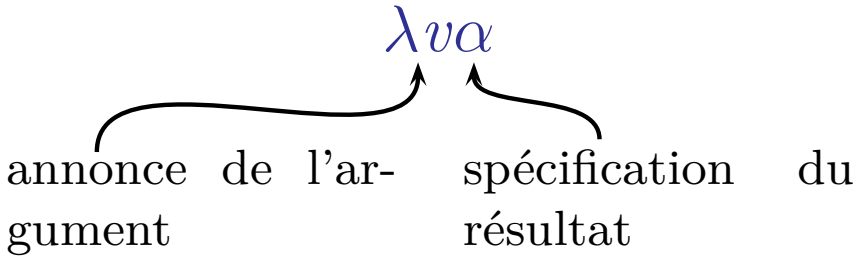
La fabrique de fonctions

Anatomie d'un λ-terme

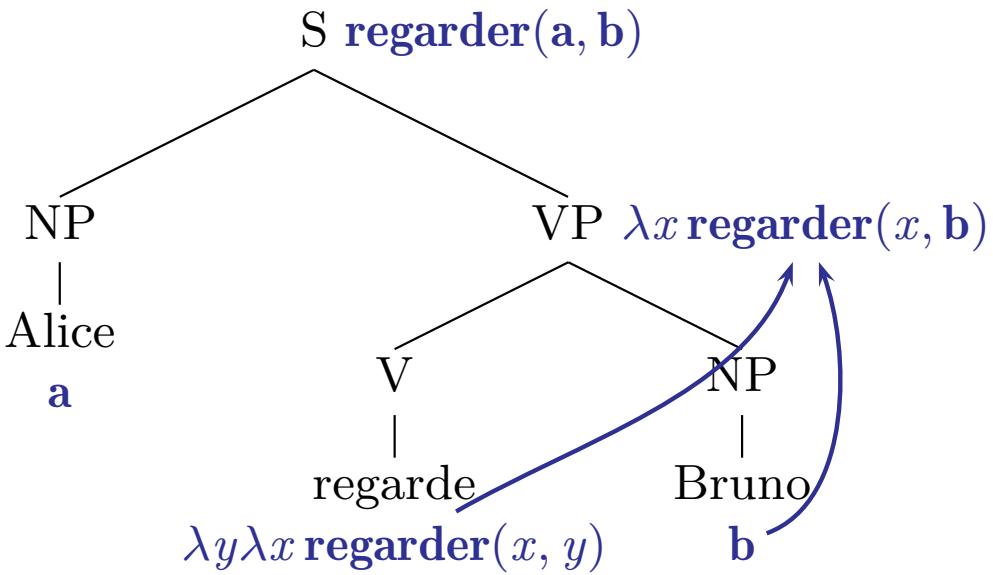
Révision de la règle syntaxique

λ-abstraction

Si α est une expression de type a dans LO et si v est une variable de type b, alors $\lambda v \alpha$ est une expression bien formée de type $\langle b, a \rangle$.



Composition des formules



Méthode : on construit la représentation sémantique d'un nœud à partir de la représentation des nœuds qu'il domine directement.

Application fonctionnelle

Nourrir les fonctions

Une fonction, par essence, attend qu'on lui donne un argument.
On va lui en donner.

Application fonctionnelle

Nourrir les fonctions

Une fonction, par essence, attend qu'on lui donne un argument.
On va lui en donner.

γ

fonction γ

Application fonctionnelle

Nourrir les fonctions

Une fonction, par essence, attend qu'on lui donne un argument.
On va lui en donner.

$$[\gamma(\beta)]$$

On fournit β comme argument à la fonction γ

Application fonctionnelle

Nourrir les fonctions

Une fonction, par essence, attend qu'on lui donne un argument.
On va lui en donner.

Application fonctionnelle (dans LO)

Si γ est de type $\langle b, a\rangle$ et si β est de type b, alors $[\gamma(\beta)]$ est de type a.

$$[\gamma(\beta)]$$

On fournit β comme argument à la fonction γ

Je ne peux pas donner n'importe quel type d'argument à n'importe quel type de fonction.

Application fonctionnelle

Plusieurs arguments

Si γ attend plusieurs arguments :

Application fonctionnelle

Plusieurs arguments

Si γ attend plusieurs arguments :

$$\gamma$$

Fonction à 2 arguments

Application fonctionnelle

Plusieurs arguments

Si γ attend plusieurs arguments :

$$[\gamma(\beta)]$$

Fonction à 1 argument

Application fonctionnelle

Plusieurs arguments

Si γ attend plusieurs arguments :

$$[[\gamma(\beta)](\delta)]$$

Fonction à 0 argument (saturée)

Application fonctionnelle

Plusieurs arguments

Si γ attend plusieurs arguments :

$$[[\gamma(\beta)](\delta)]$$

Fonction à 0 argument (saturée)

Simplification d'écriture

On s'autorisera à simplifier toute notation $[[\gamma(\beta)](\delta)]$ en $\gamma(\delta, \beta)$.

Application fonctionnelle

Plusieurs arguments

Si γ attend plusieurs arguments :

$$[[\gamma(\beta)](\delta)]$$

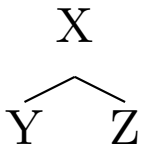
Fonction à 0 argument (saturée)

Simplification d'écriture

On s'autorisera à simplifier toute notation $[[\gamma(\beta)](\delta)]$ en $\gamma(\delta, \beta)$.
De manière générale, on simplifie $[[[[\gamma(\beta_1)](\beta_2)]\dots](\beta_n)]$ en $\gamma(\beta_n, \dots, \beta_2, \beta_1)$.

Application fonctionnelle

Opération à tout faire

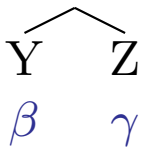
Hypothèse : pour tout sous-arbre 

un des constituants est une fonction et l'autre son argument

Application fonctionnelle

Opération à tout faire

Hypothèse : pour tout sous-arbre

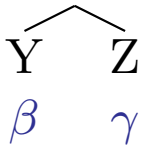


un des constituants est une fonction et l'autre son argument

Application fonctionnelle

Opération à tout faire

Hypothèse : pour tout sous-arbre

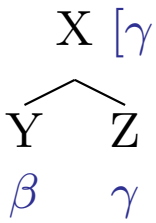


un des constituants est une fonction et l'autre son argument

Application fonctionnelle

Opération à tout faire

Hypothèse : pour tout sous-arbre



un des constituants est une fonction et l'autre son argument

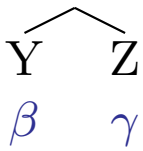
Schéma général

Règle syntaxique :	X	→	Y	Z
Règle sémantique :	$[\beta(\gamma)]$	←	β	γ
ou	$[\gamma(\beta)]$	←	β	γ

Application fonctionnelle

Opération à tout faire

Hypothèse : pour tout sous-arbre



un des constituants est une fonction et l'autre son argument

Schéma général

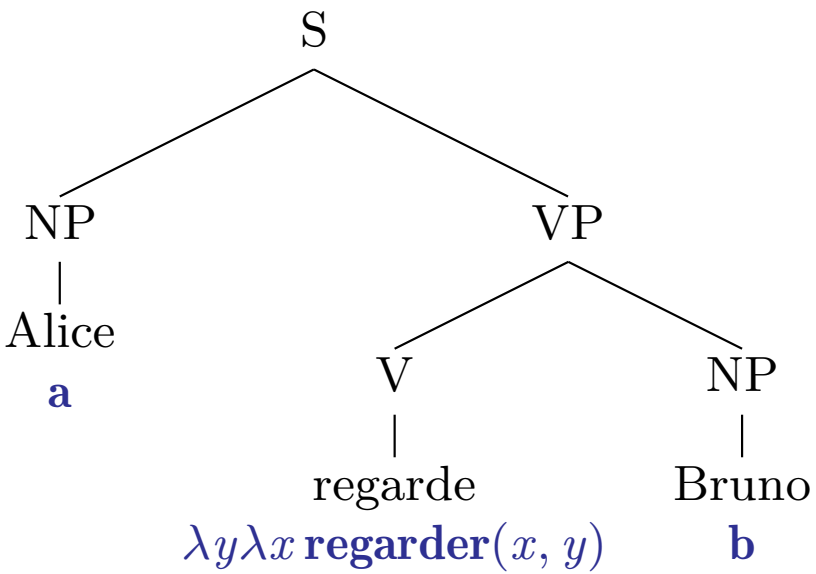
Règle syntaxique :	X	→	Y	Z
Règle sémantique :	$[\beta(\gamma)]$	←	β	γ
ou	$[\gamma(\beta)]$	←	β	γ

Exemples:

VP	→	V	NP
$[\beta(\gamma)]$	←	β	γ
S	→	NP	VP
$[\gamma(\beta)]$	←	β	γ

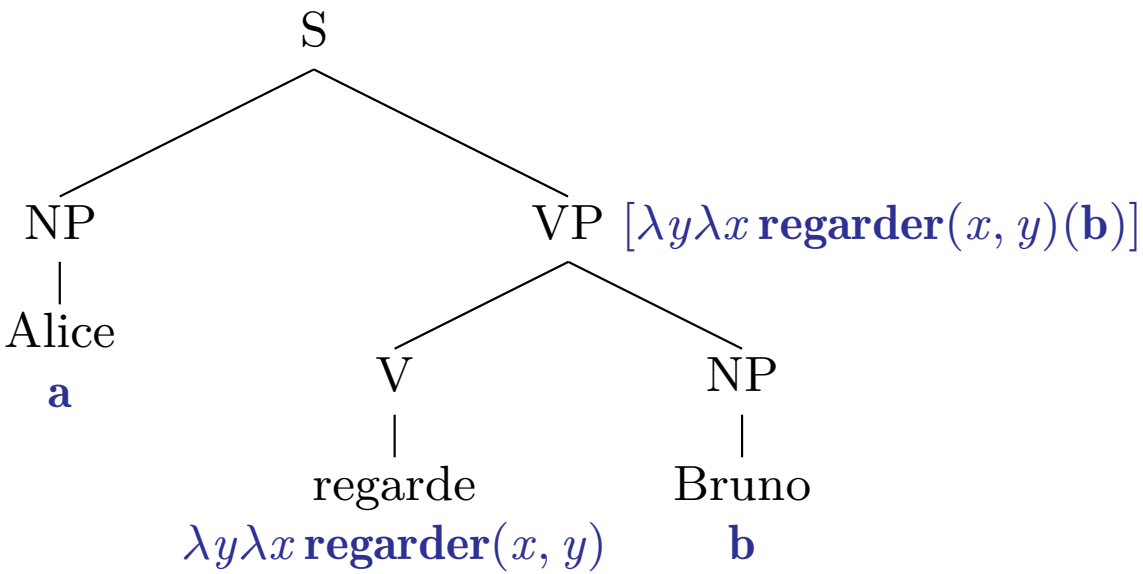
Exemple

Retour à notre arbre



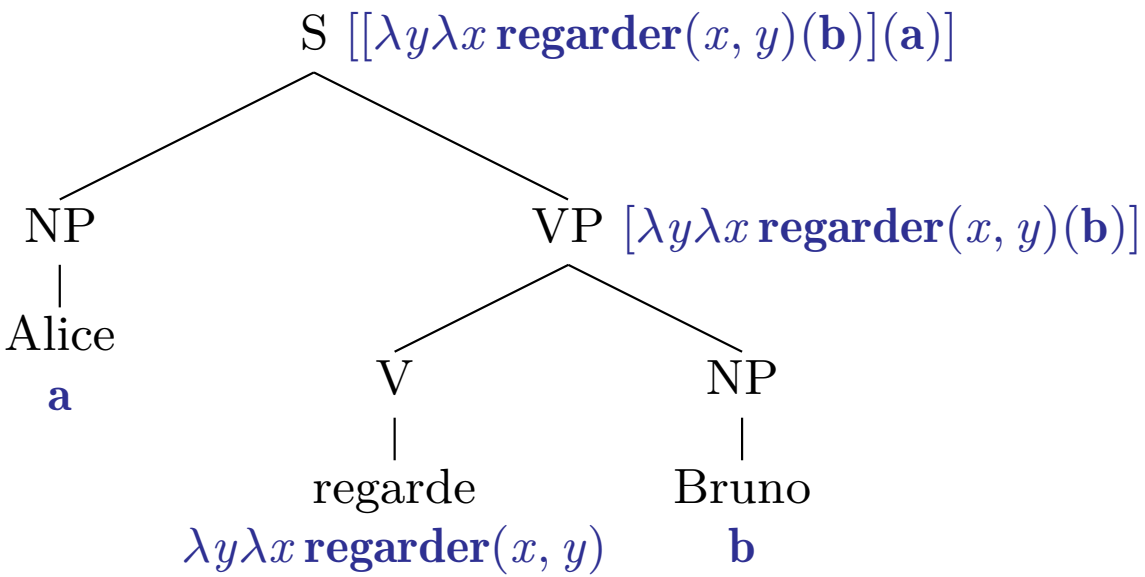
Exemple

Retour à notre arbre



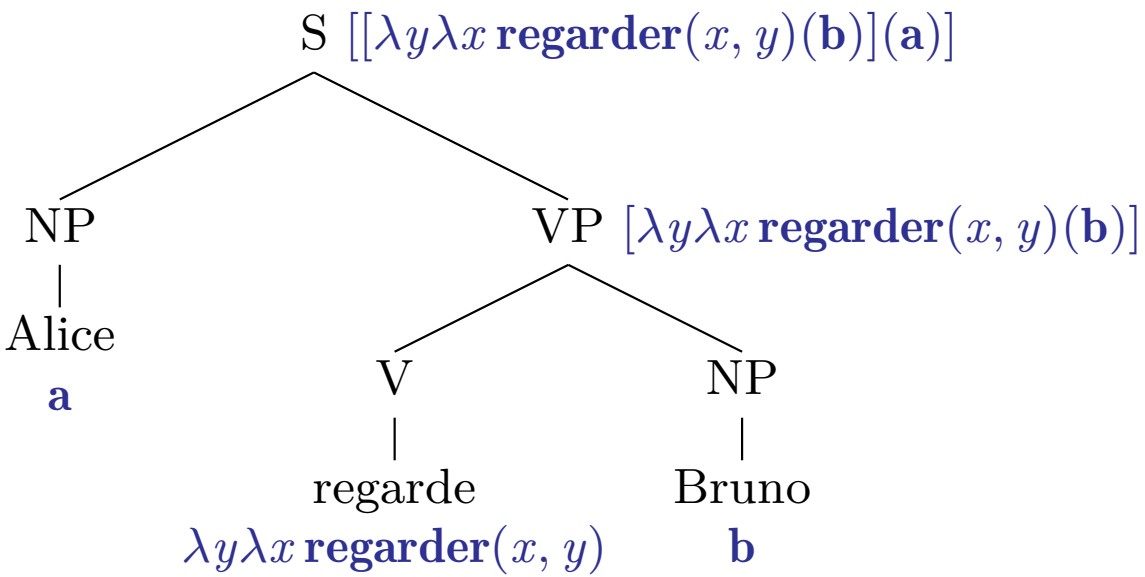
Exemple

Retour à notre arbre



Exemple

Retour à notre arbre



Pas exactement ce qu'on attendait, mais c'est l'analyse correcte.

λ-conversion / β-réduction

Une règle de simplification sémantique et graphique

β-réduction

$[\lambda v \alpha(\beta)]$ est équivalent à $[\beta/v]\alpha$
où $[\beta/v]\alpha$ est l'expression α où on a remplacé toutes les occurences libres de v par β.

λ-conversion / β-réduction

Une règle de simplification sémantique et graphique

β-réduction

$[\lambda v \alpha(\beta)]$ est équivalent à $[\beta/v]\alpha$
où $[\beta/v]\alpha$ est l'expression α où on a remplacé toutes les occurences libres de v par β.

En pratique :

$$[\lambda v \alpha(\beta)]$$

λ-conversion / β-réduction

Une règle de simplification sémantique et graphique

β-réduction

$[\lambda v \alpha(\beta)]$ est équivalent à $[\beta/v]\alpha$
où $[\beta/v]\alpha$ est l'expression α où on a remplacé toutes les occurences libres de v par β.

En pratique :

$$\lambda v \alpha(\beta)$$

λ-conversion / β-réduction

Une règle de simplification sémantique et graphique

β-réduction

$[\lambda v \alpha(\beta)]$ est équivalent à $[\beta/v]\alpha$
où $[\beta/v]\alpha$ est l'expression α où on a remplacé toutes les occurences libres de v par β.

En pratique :

$$\alpha(\beta)$$

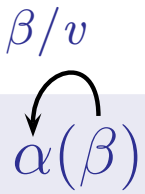
λ-conversion / β-réduction

Une règle de simplification sémantique et graphique

β-réduction

$[\lambda v \alpha(\beta)]$ est équivalent à $[\beta/v]\alpha$
où $[\beta/v]\alpha$ est l'expression α où on a remplacé toutes les occurences libres de v par β.

En pratique :



λ-conversion / β-réduction

Une règle de simplification sémantique et graphique

β-réduction

$[\lambda v \alpha(\beta)]$ est équivalent à $[\beta/v]\alpha$
où $[\beta/v]\alpha$ est l'expression α où on a remplacé toutes les occurences libres de v par β.

En pratique :

λ-conversion / β-réduction

Une règle de simplification sémantique et graphique

β-réduction

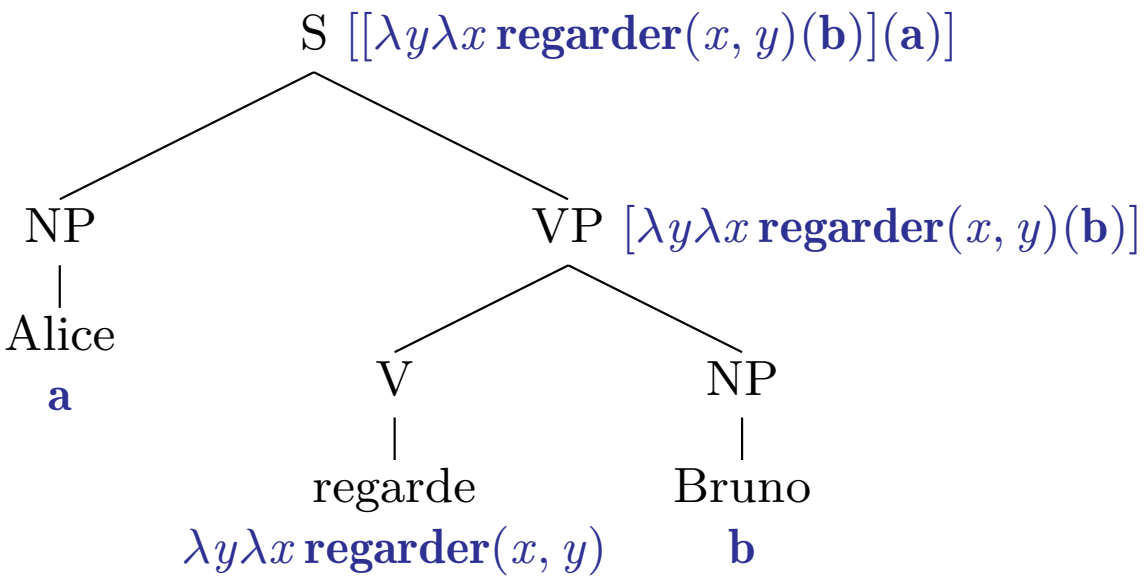
$[\lambda v \alpha(\beta)]$ est équivalent à $[\beta/v]\alpha$
où $[\beta/v]\alpha$ est l'expression α où on a remplacé toutes les occurences libres de v par β.

En pratique :

Attention : pour appliquer la β-réduction, il est indispensable de reconnaître **exactement** le schéma $[\lambda v. . . (. . .)]$

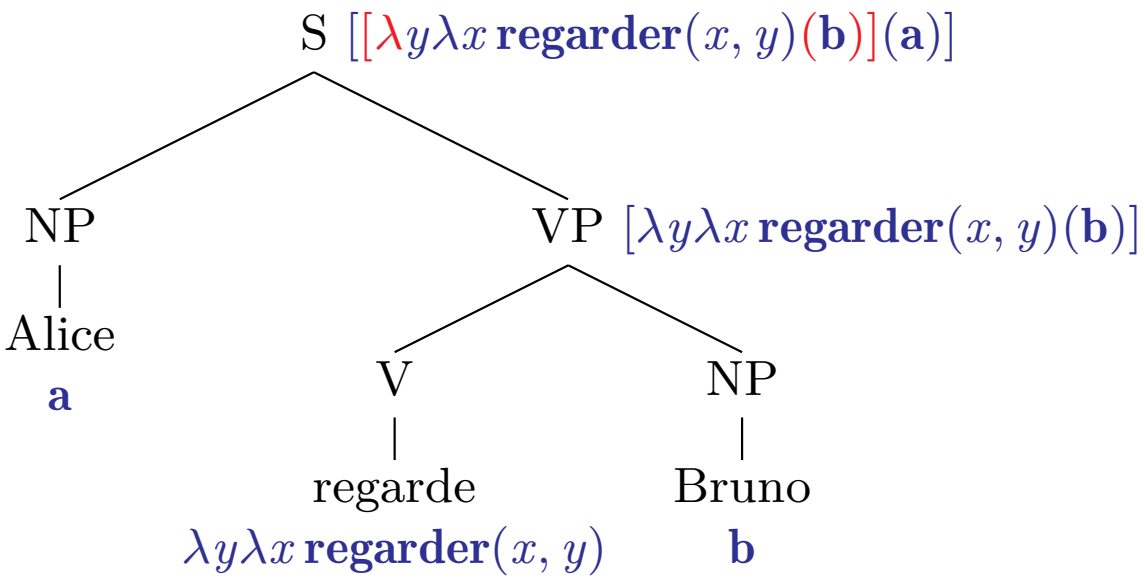
Exemple

Pour en finir avec notre arbre



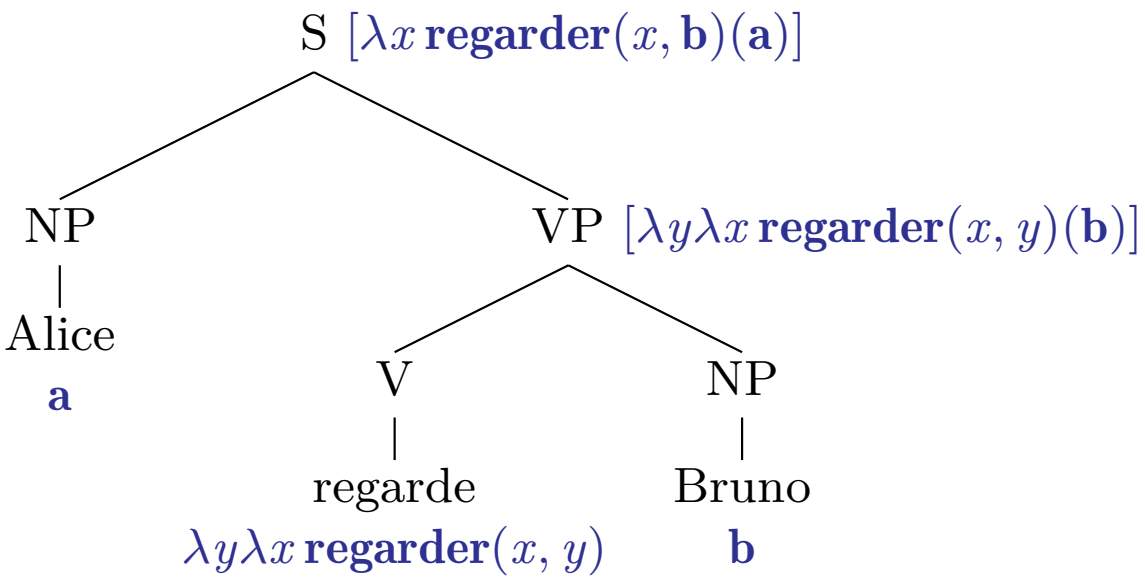
Exemple

Pour en finir avec notre arbre



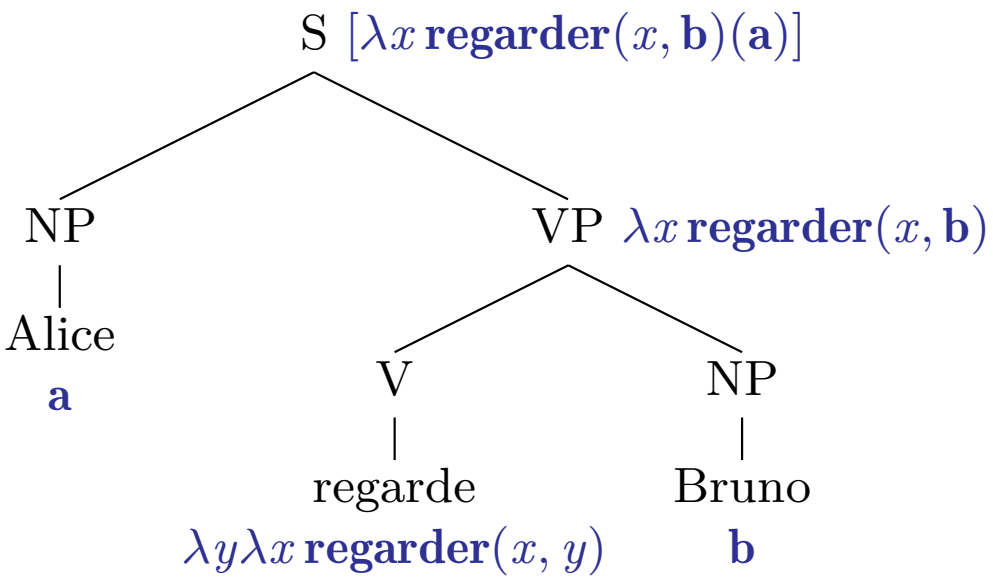
Exemple

Pour en finir avec notre arbre



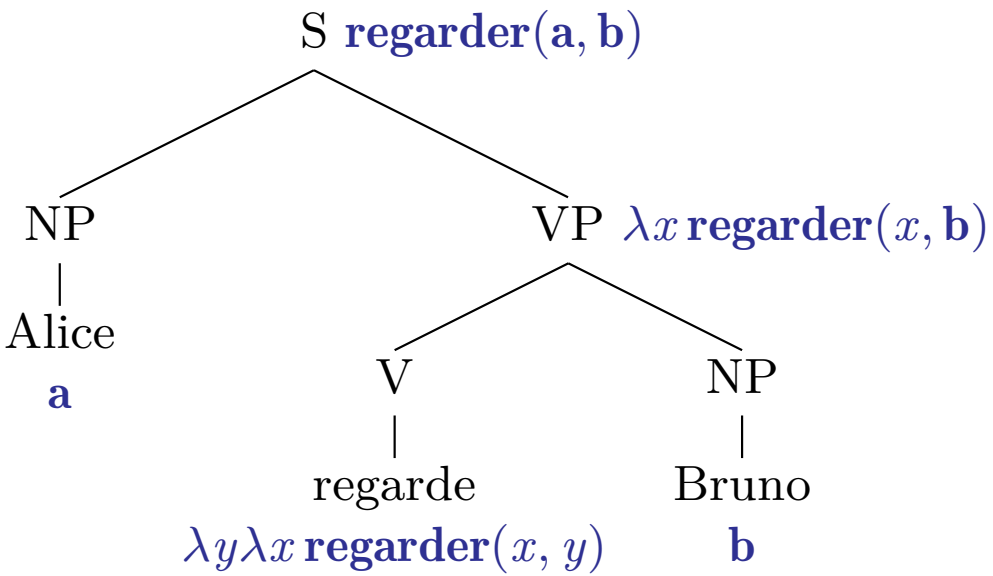
Exemple

Pour en finir avec notre arbre



Exemple

Pour en finir avec notre arbre



Limitation de la β-réduction

Un (des deux) cas de figures où la réduction est interdite

Limitation de la β-réduction

Un (des deux) cas de figures où la réduction est interdite

- $[\![\lambda x \exists y\, \mathbf{regarder}(x, y)]\!]^{\mathcal{M},w,g} =$

Limitation de la β-réduction

Un (des deux) cas de figures où la réduction est interdite

- $[\![\lambda x \exists y \, \textbf{regarder}(x, y)]\!]^{\mathcal{M},w,g}$ = la fonction qui pour toute valeur de x nous renvoie la dénotation de $\exists y \, \textbf{regarder}(x, y)$ par rapport à $\mathcal{M}$ et w

Limitation de la β-réduction

Un (des deux) cas de figures où la réduction est interdite

- $[\![\lambda x \exists y \, \mathbf{regarder}(x, y)]\!]^{\mathcal{M},w,g}$ = la fonction qui pour toute valeur de x nous renvoie la dénotation de $\exists y \, \mathbf{regarder}(x, y)$ par rapport à $\mathcal{M}$ et w (elle retourne 1 ssi x regarde quelqu'un)

Limitation de la β-réduction

Un (des deux) cas de figures où la réduction est interdite

- $[\![\lambda x \exists y \, \textbf{regarder}(x, y)]\!]^{\mathcal{M},w,g}$ = la fonction qui pour toute valeur de x nous renvoie la dénotation de $\exists y \, \textbf{regarder}(x, y)$ par rapport à $\mathcal{M}$ et w (elle retourne 1 ssi x regarde quelqu'un)
- $[\![[\lambda x \exists y \, \textbf{regarder}(x, y)(z)]]\!]^{\mathcal{M},w,g}$ =

Limitation de la β-réduction

Un (des deux) cas de figures où la réduction est interdite

- $[\![\lambda x \exists y \, \textbf{regarder}(x, y)]\!]^{\mathcal{M},w,g}$ = la fonction qui pour toute valeur de x nous renvoie la dénotation de $\exists y \, \textbf{regarder}(x, y)$ par rapport à $\mathcal{M}$ et w (elle retourne 1 ssi x regarde quelqu'un)
- $[\![[\lambda x \exists y \, \textbf{regarder}(x, y)(z)]]\!]^{\mathcal{M},w,g} = 1$ ssi la dénotation de z est un individu qui regarde quelqu'un dans w.

Limitation de la β-réduction

Un (des deux) cas de figures où la réduction est interdite

- $[\![\lambda x \exists y \, \mathbf{regarder}(x, y)]\!]^{\mathcal{M},w,g}$ = la fonction qui pour toute valeur de x nous renvoie la dénotation de $\exists y \, \mathbf{regarder}(x, y)$ par rapport à $\mathcal{M}$ et w (elle retourne 1 ssi x regarde quelqu'un)
- $[\![[\lambda x \exists y \, \mathbf{regarder}(x, y)(z)]]\!]^{\mathcal{M},w,g} = 1$ ssi la dénotation de z est un individu qui regarde quelqu'un dans w.
- Confirmé par β-réduction : $\exists y \, \mathbf{regarder}(z, y)$

Limitation de la β-réduction

Un (des deux) cas de figures où la réduction est interdite

- $[\![\lambda x \exists y\, \textbf{regarder}(x, y)]\!]^{\mathcal{M},w,g}$ = la fonction qui pour toute valeur de x nous renvoie la dénotation de $\exists y\, \textbf{regarder}(x, y)$ par rapport à $\mathcal{M}$ et w (elle retourne 1 ssi x regarde quelqu'un)
- $[\![[\lambda x \exists y\, \textbf{regarder}(x, y)(z)]]\!]^{\mathcal{M},w,g} = 1$ ssi la dénotation de z est un individu qui regarde quelqu'un dans w.
- Confirmé par β-réduction : $\exists y\, \textbf{regarder}(z, y)$
- Cela vaut pour n'importe quel argument donné au λ-terme : $[\![[\lambda x \exists y\, \textbf{regarder}(x, y)(y)]]\!]^{\mathcal{M},w,g} = 1$ ssi la dénotation de y est un individu qui regarde quelqu'un dans w.

Limitation de la β-réduction

Un (des deux) cas de figures où la réduction est interdite

- $[\![\lambda x \exists y\, \textbf{regarder}(x, y)]\!]^{\mathcal{M},w,g}$ = la fonction qui pour toute valeur de x nous renvoie la dénotation de $\exists y\, \textbf{regarder}(x, y)$ par rapport à $\mathcal{M}$ et w (elle retourne 1 ssi x regarde quelqu'un)
- $[\![[\lambda x \exists y\, \textbf{regarder}(x, y)(z)]]\!]^{\mathcal{M},w,g} = 1$ ssi la dénotation de z est un individu qui regarde quelqu'un dans w.
- Confirmé par β-réduction : $\exists y\, \textbf{regarder}(z, y)$
- Cela vaut pour n'importe quel argument donné au λ-terme : $[\![[\lambda x \exists y\, \textbf{regarder}(x, y)(y)]]\!]^{\mathcal{M},w,g} = 1$ ssi la dénotation de y est un individu qui regarde quelqu'un dans w.
- **Mais** par β-réduction : $\exists y\, \textbf{regarder}(y, y)$

Limitation de la β-réduction

Un (des deux) cas de figures où la réduction est interdite

- $[\![\lambda x \exists y\, \textbf{regarder}(x, y)]\!]^{\mathcal{M},w,g}$ = la fonction qui pour toute valeur de x nous renvoie la dénotation de $\exists y\, \textbf{regarder}(x, y)$ par rapport à $\mathcal{M}$ et w (elle retourne 1 ssi x regarde quelqu'un)
- $[\![[\lambda x \exists y\, \textbf{regarder}(x, y)(z)]]\!]^{\mathcal{M},w,g} = 1$ ssi la dénotation de z est un individu qui regarde quelqu'un dans w.
- Confirmé par β-réduction : $\exists y\, \textbf{regarder}(z, y)$
- Cela vaut pour n'importe quel argument donné au λ-terme : $[\![[\lambda x \exists y\, \textbf{regarder}(x, y)(y)]]\!]^{\mathcal{M},w,g} = 1$ ssi la dénotation de y est un individu qui regarde quelqu'un dans w.
- **Mais** par β-réduction : $\exists y\, \textbf{regarder}(y, y)$
  ce qui signifie que *quelqu'un se regarde lui-même*!

Limitation de la β-réduction

Un (des deux) cas de figures où la réduction est interdite

- $[\![\lambda x \exists y\, \textbf{regarder}(x, y)]\!]^{\mathcal{M},w,g}$ = la fonction qui pour toute valeur de x nous renvoie la dénotation de $\exists y\, \textbf{regarder}(x, y)$ par rapport à $\mathcal{M}$ et w (elle retourne 1 ssi x regarde quelqu'un)
- $[\![[\lambda x \exists y\, \textbf{regarder}(x, y)(z)]]\!]^{\mathcal{M},w,g} = 1$ ssi la dénotation de z est un individu qui regarde quelqu'un dans w.
- Confirmé par β-réduction : $\exists y\, \textbf{regarder}(z, y)$
- Cela vaut pour n'importe quel argument donné au λ-terme : $[\![[\lambda x \exists y\, \textbf{regarder}(x, y)(y)]]\!]^{\mathcal{M},w,g} = 1$ ssi la dénotation de y est un individu qui regarde quelqu'un dans w.
- **Mais** par β-réduction : $\exists y\, \textbf{regarder}(y, y)$
  ce qui signifie que *quelqu'un se regarde lui-même*!

Contrainte sur la β-réduction

Soit $[\alpha(\gamma)]$ une application fonctionnelle. Une variable libre dans γ ne doit pas se retrouver liée dans α après β-réduction.

Limitation de la β-réduction

Un (des deux) cas de figures où la réduction est interdite

- $[\![\lambda x \exists y\, \mathbf{regarder}(x, y)]\!]^{\mathcal{M},w,g}$ = la fonction qui pour toute valeur de x nous renvoie la dénotation de $\exists y\, \mathbf{regarder}(x, y)$ par rapport à $\mathcal{M}$ et w (elle retourne 1 ssi x regarde quelqu'un)
- $[\![[\lambda x \exists y\, \mathbf{regarder}(x, y)(z)]]\!]^{\mathcal{M},w,g} = 1$ ssi la dénotation de z est un individu qui regarde quelqu'un dans w.
- Confirmé par β-réduction : $\exists y\, \mathbf{regarder}(z, y)$
- Cela vaut pour n'importe quel argument donné au λ-terme : $[\![[\lambda x \exists y\, \mathbf{regarder}(x, y)(y)]]\!]^{\mathcal{M},w,g} = 1$ ssi la dénotation de y est un individu qui regarde quelqu'un dans w.
- **Mais** par β-réduction : $\exists y\, \mathbf{regarder}(y, y)$
  ce qui signifie que *quelqu'un se regarde lui-même* !

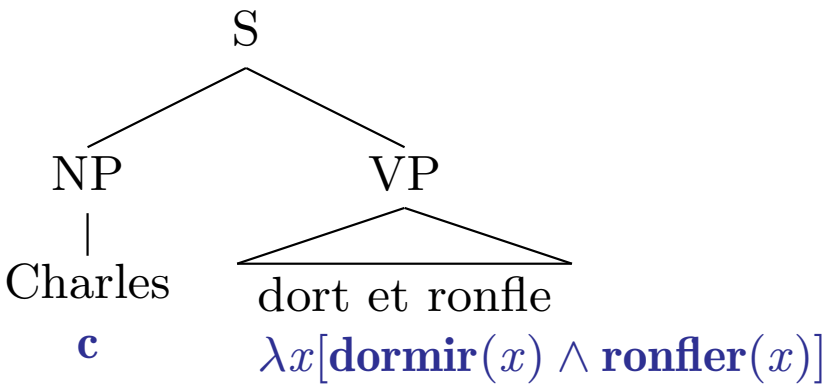
Contrainte sur la β-réduction

Soit $[\alpha(\gamma)]$ une application fonctionnelle. Une variable libre dans γ ne doit pas se retrouver liée dans α après β-réduction.

Heureusement : $\lambda x \exists y\, \mathbf{regarder}(x, y) \equiv \lambda x \exists u\, \mathbf{regarder}(x, u)$

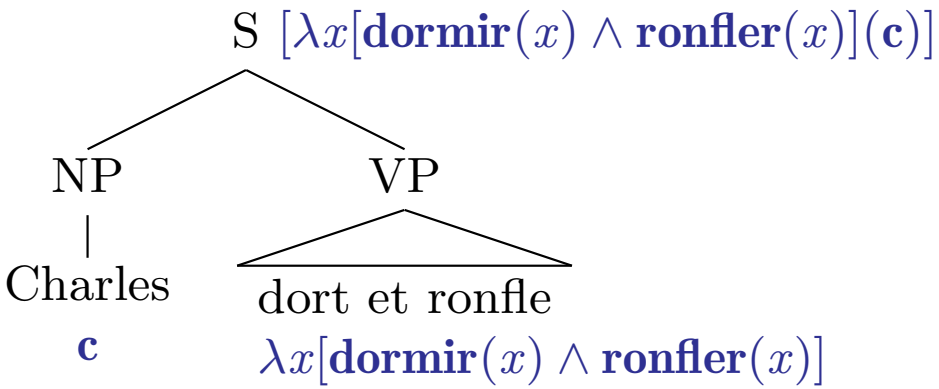
Application à la coordination de VP

(1) Charles dort et ronfle.



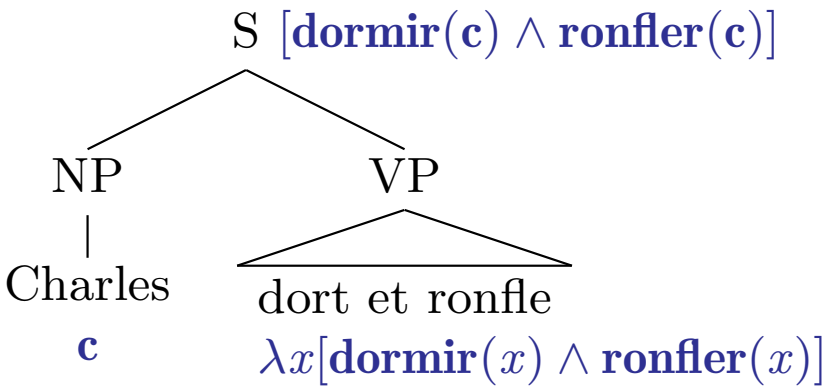
Application à la coordination de VP

(1) Charles dort et ronfle.



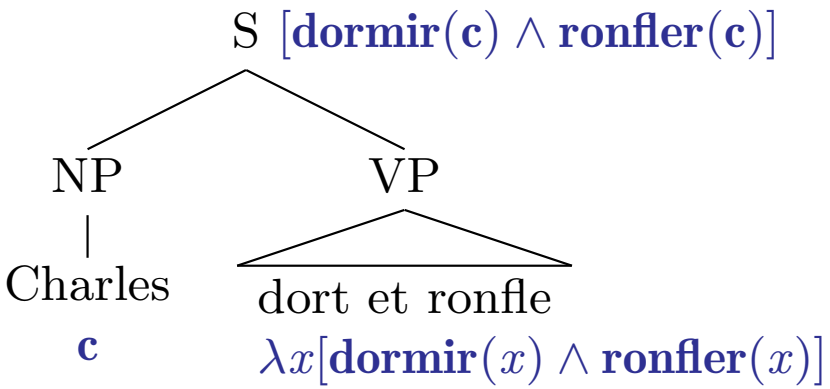
Application à la coordination de VP

(1) Charles dort et ronfle.



Application à la coordination de VP

(1) Charles dort et ronfle.



On a introduit **c** une seule fois.

Variables pour tous les types

Vers un ordre supérieur

- On va se donner des variables de n'importe quel type.

Variables pour tous les types

Vers un ordre supérieur

- On va se donner des variables de n'importe quel type.
- Ex : soit P une variable de type $\langle \mathrm{e}, \mathrm{t} \rangle$. C'est une variable de prédicat.

Variables pour tous les types

Vers un ordre supérieur

- On va se donner des variables de n'importe quel type.
- Ex : soit P une variable de type $\langle \mathrm{e}, \mathrm{t} \rangle$. C'est une variable de prédicat.
- Pour n'importe quel type, on sait quel genre d'*objet* dénote toute expression de ce type (une entité, une valeur de vérité, ou une fonction).
- Donc pour toute variable d'un type donné, g lui assignera une valeur appropriée.
  $g(P)$ = une fonction de $\mathcal{A}$ vers $\{0 ; 1\}$, ou, plus simplement, un sous-ensemble de $\mathcal{A}$.

Variables pour tous les types

Vers un ordre supérieur

- On va se donner des variables de n'importe quel type.
- Ex : soit P une variable de type $\langle \mathrm{e}, \mathrm{t} \rangle$. C'est une variable de prédicat.
- Pour n'importe quel type, on sait quel genre d'*objet* dénote toute expression de ce type (une entité, une valeur de vérité, ou une fonction).
- Donc pour toute variable d'un type donné, g lui assignera une valeur appropriée.
  $g(P)$ = une fonction de $\mathcal{A}$ vers $\{0 ; 1\}$, ou, plus simplement, un sous-ensemble de $\mathcal{A}$.
- Autre exemple :
  Si p et q sont des variables de type t, $g(p) = 0$ ou 1; idem pour $g(q)$.

Variables pour tous les types

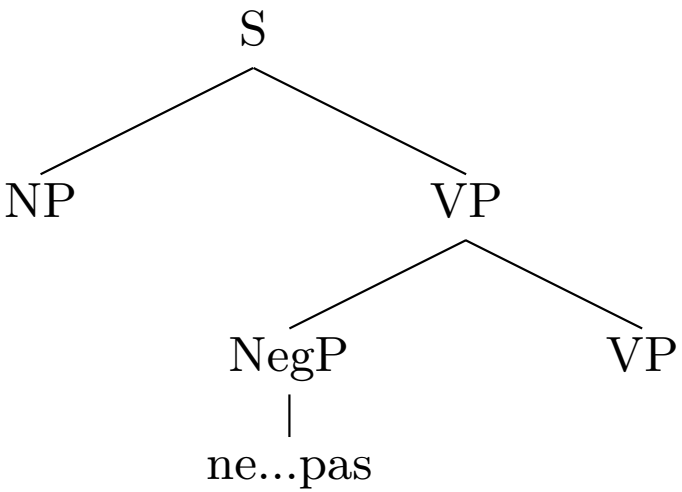
Vers un ordre supérieur

- On va se donner des variables de n'importe quel type.
- Ex : soit P une variable de type $\langle \mathrm{e}, \mathrm{t} \rangle$. C'est une variable de prédicat.
- Pour n'importe quel type, on sait quel genre d'*objet* dénote toute expression de ce type (une entité, une valeur de vérité, ou une fonction).
- Donc pour toute variable d'un type donné, g lui assignera une valeur appropriée.
  $g(P) =$ une fonction de $\mathcal{A}$ vers $\{0 ; 1\}$, ou, plus simplement, un sous-ensemble de $\mathcal{A}$.
- Autre exemple :
  Si p et q sont des variables de type t, $g(p) = 0$ ou 1; idem pour $g(q)$.
  Et on peut écrire le λ-terme :
  $\lambda p \lambda q[p \wedge q]$

Exemples

La négation

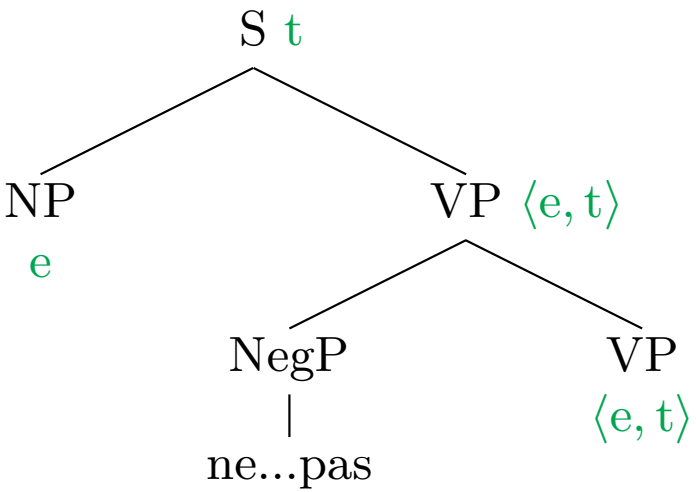
(2) Alice ne dort pas.



Exemples

La négation

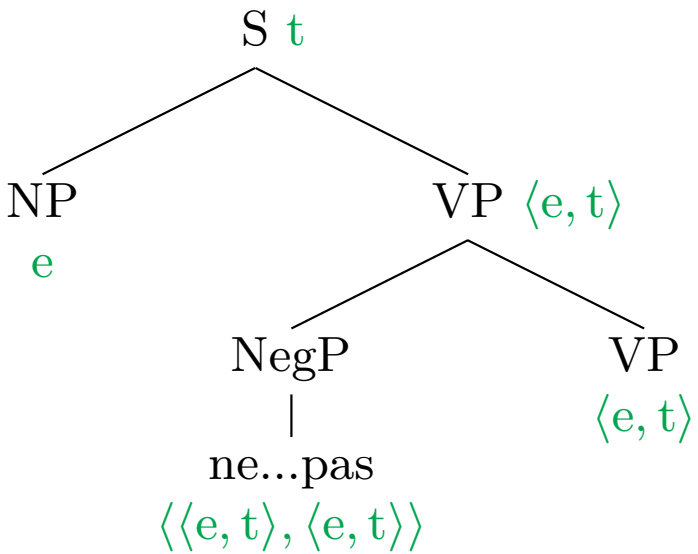
(2) Alice ne dort pas.



Exemples

La négation

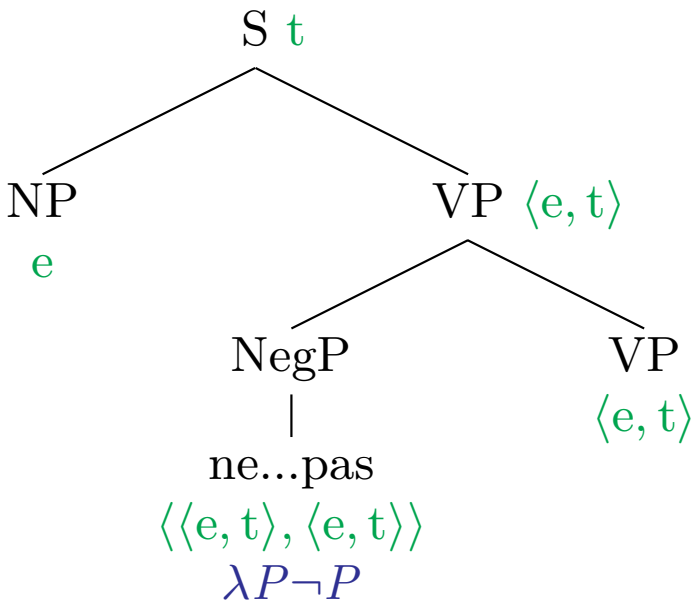
(2) Alice ne dort pas.



Exemples

La négation

(2) Alice ne dort pas.

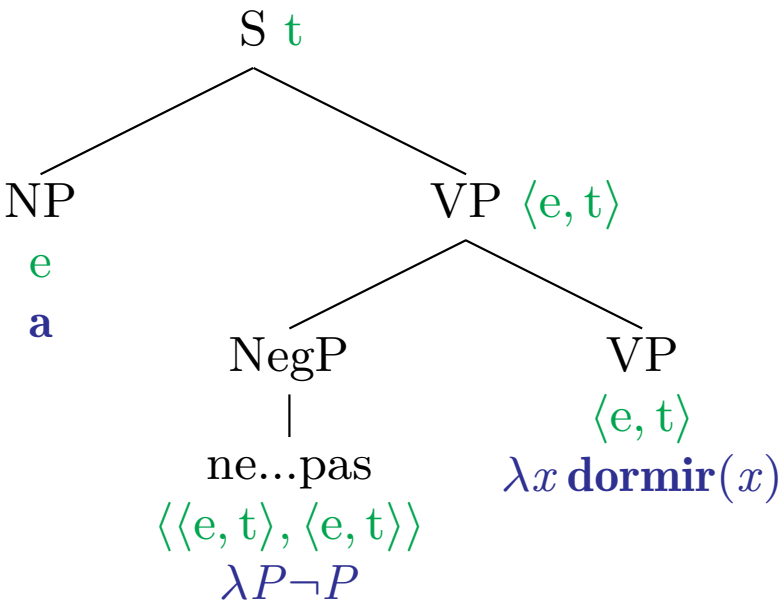


1er essai : NegP $\rightsquigarrow \lambda P \neg P$

Exemples

La négation

(2) Alice ne dort pas.

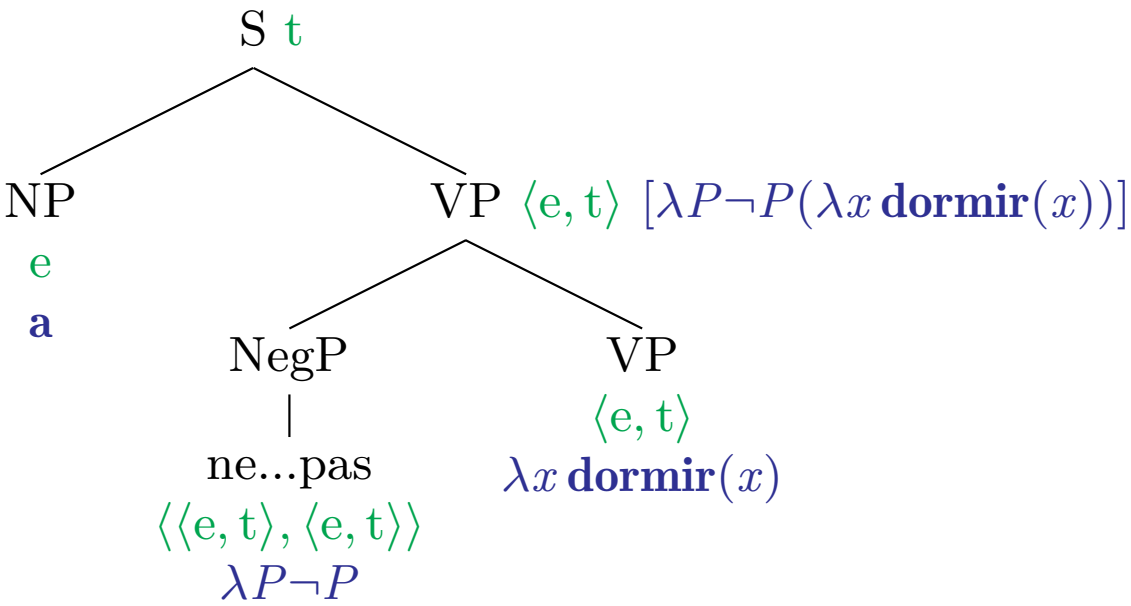


1er essai : NegP $\rightsquigarrow \lambda P \neg P$

Exemples

La négation

(2) Alice ne dort pas.

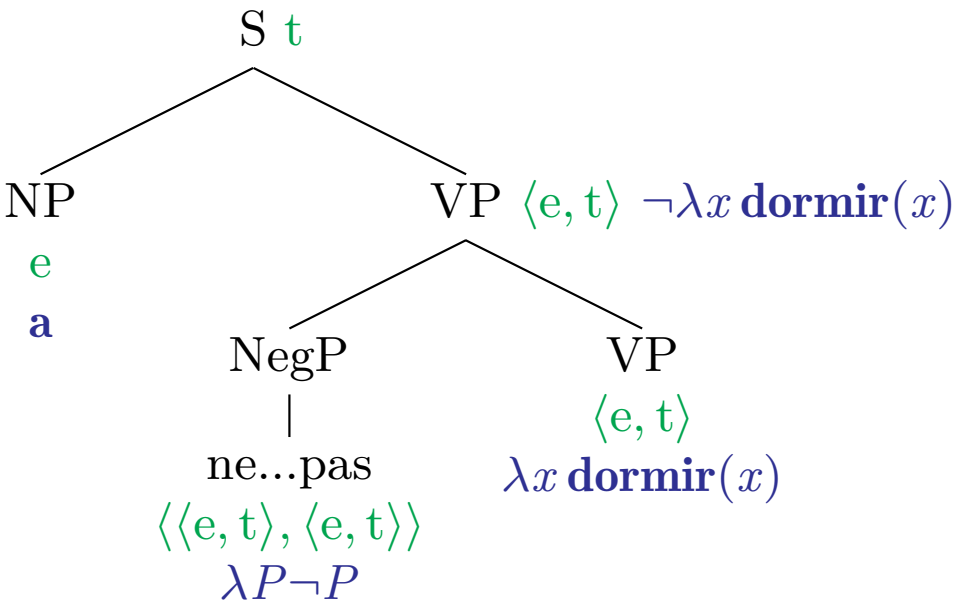


1er essai : NegP $\rightsquigarrow \lambda P \neg P$

Exemples

La négation

(2) Alice ne dort pas.

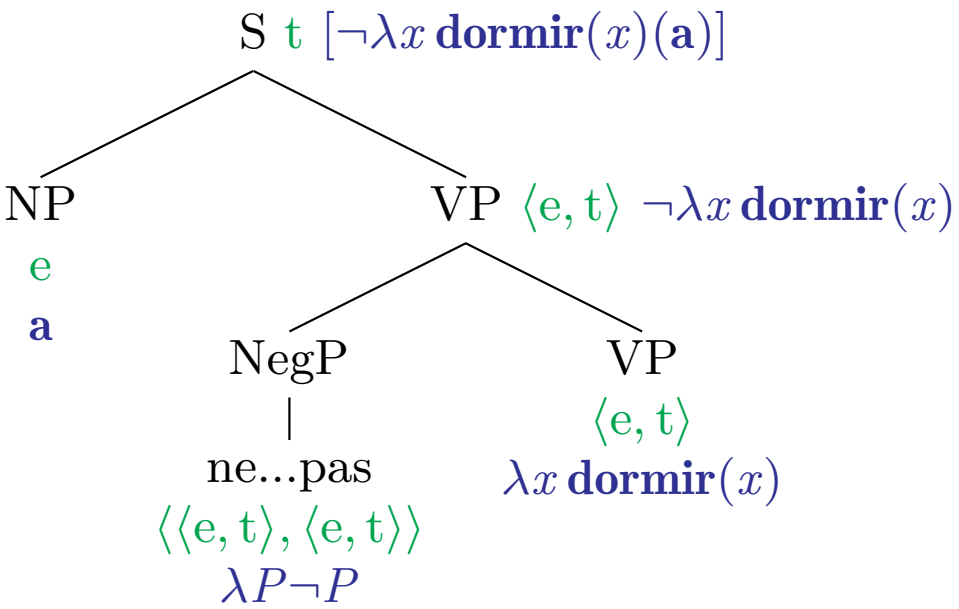


1er essai : NegP $\rightsquigarrow \lambda P \neg P$

Exemples

La négation

(2) Alice ne dort pas.

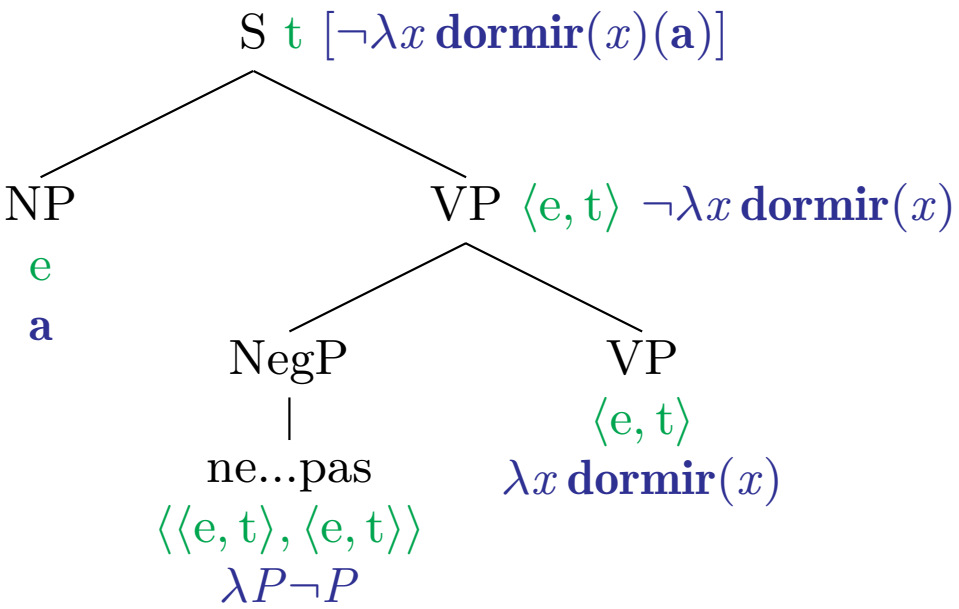


1er essai : NegP $\rightsquigarrow \lambda P \neg P$

Exemples

La négation

(2) Alice ne dort pas.

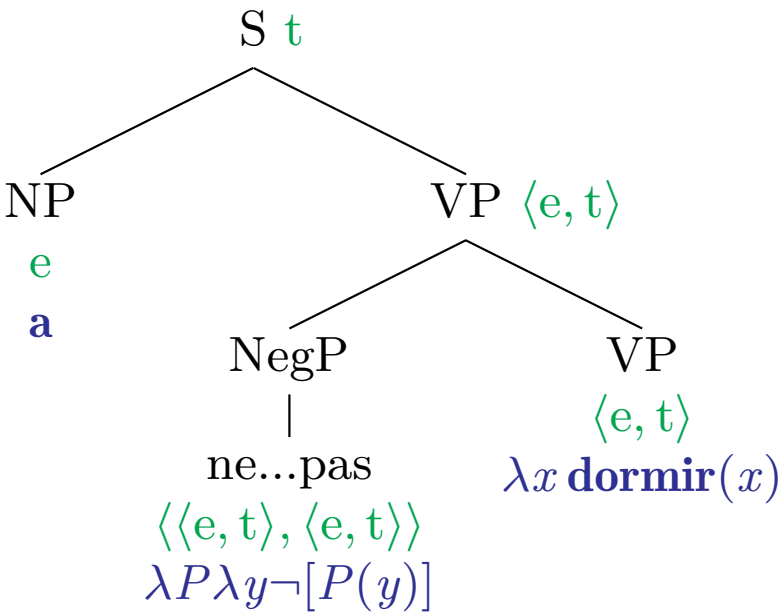


1er essai : NegP $\rightsquigarrow \lambda P \neg P$ P est de type $\langle e, t\rangle$!! Pas le droit d'écrire $\neg P$

Exemples

La négation

(2) Alice ne dort pas.

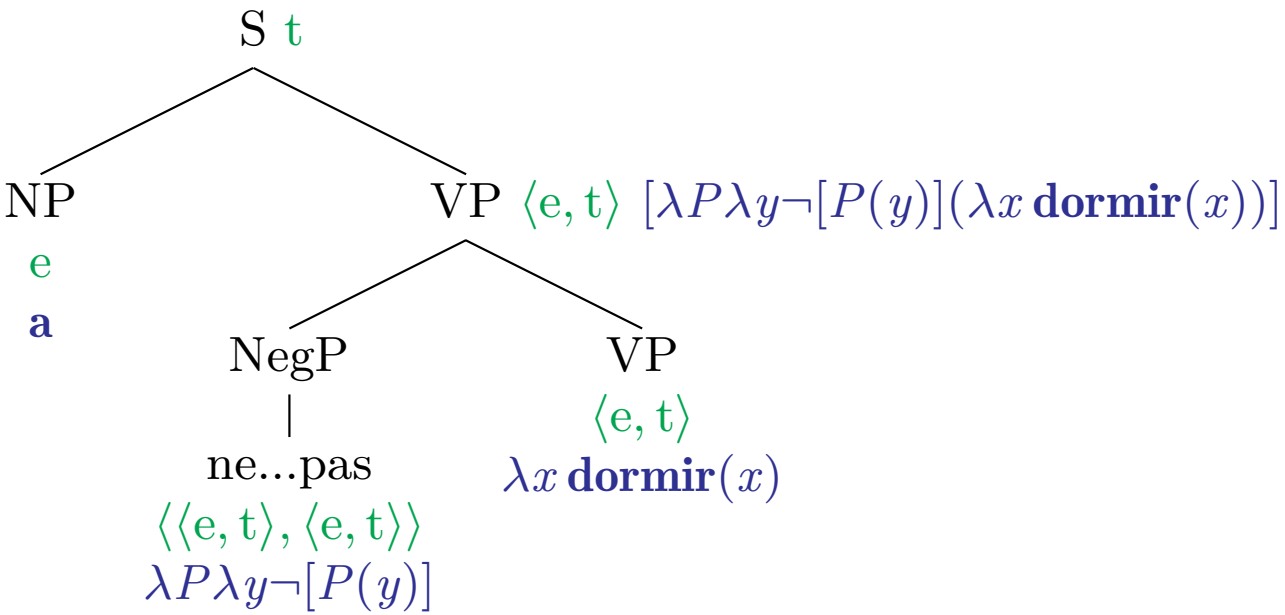


Version correcte : NegP $\rightsquigarrow \lambda P \lambda y \neg [P(y)]$

Exemples

La négation

(2) Alice ne dort pas.

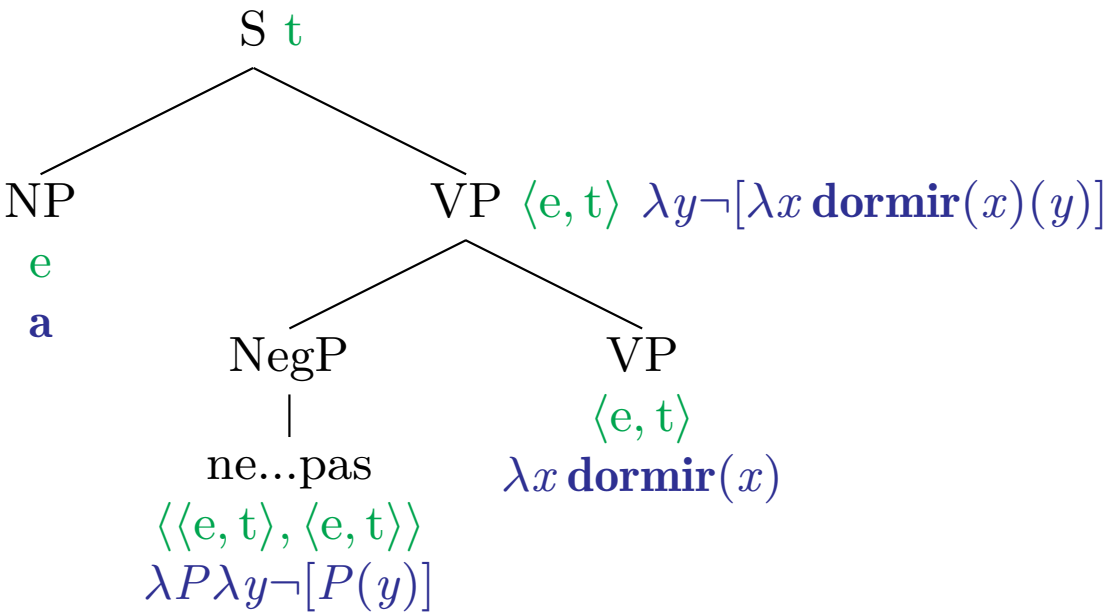


Version correcte : NegP $\rightsquigarrow \lambda P \lambda y \neg [P(y)]$

Exemples

La négation

(2) Alice ne dort pas.

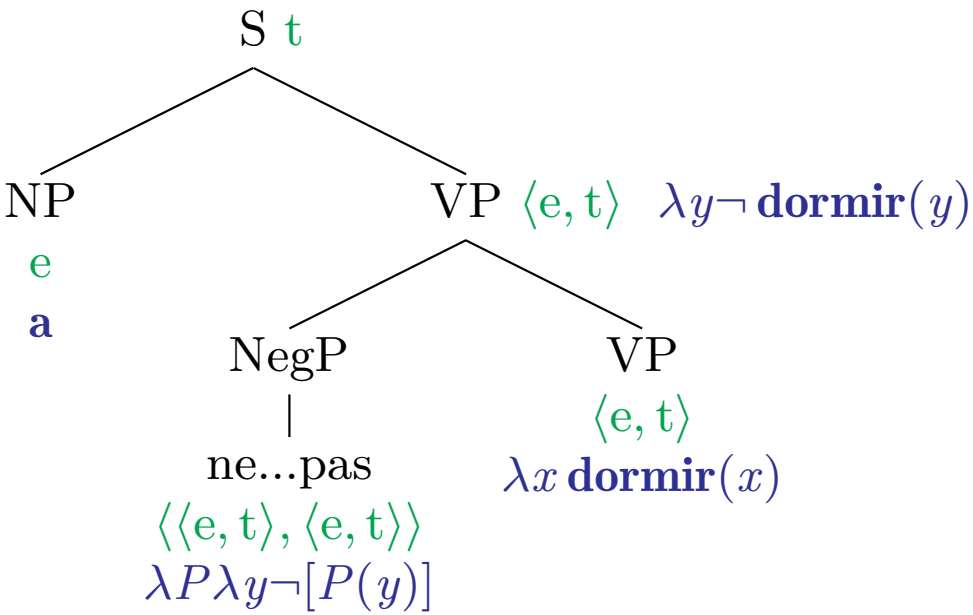


Version correcte : NegP $\rightsquigarrow \lambda P \lambda y \neg [P(y)]$

Exemples

La négation

(2) Alice ne dort pas.

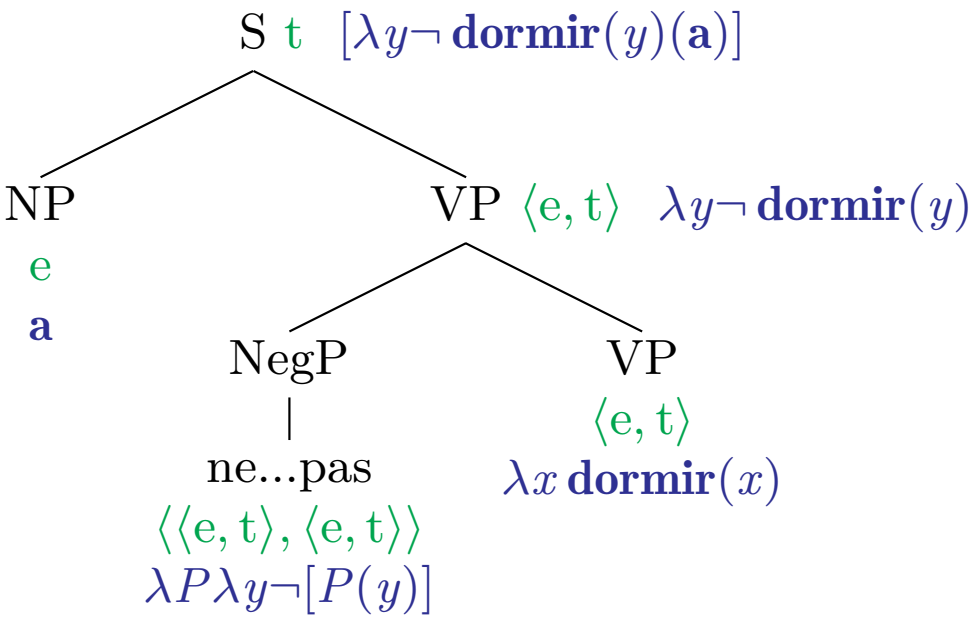


Version correcte : NegP $\rightsquigarrow \lambda P \lambda y \neg [P(y)]$

Exemples

La négation

(2) Alice ne dort pas.

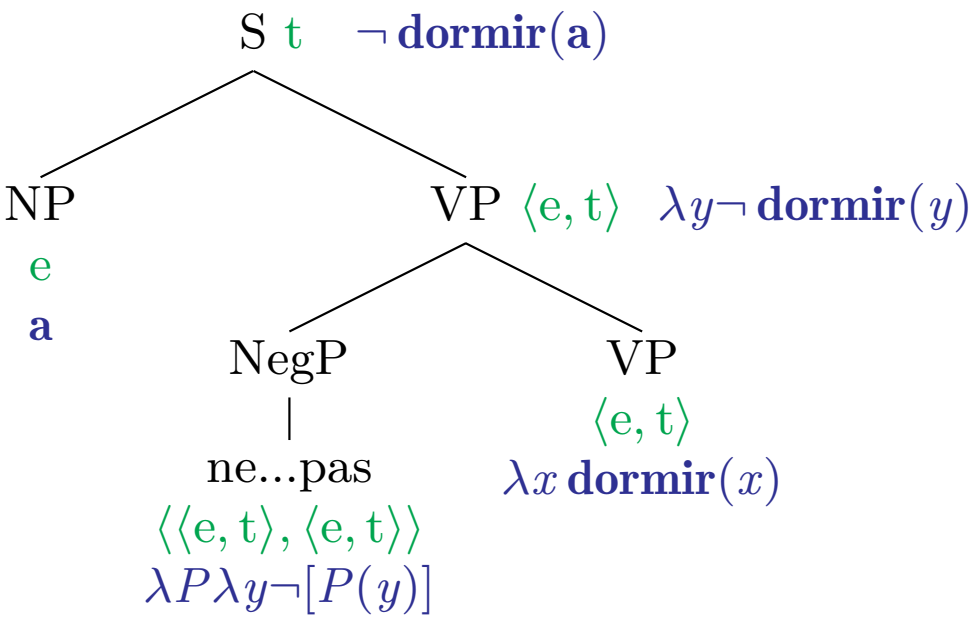


Version correcte : NegP $\rightsquigarrow \lambda P \lambda y \neg [P(y)]$

Exemples

La négation

(2) Alice ne dort pas.



Version correcte : NegP $\rightsquigarrow \lambda P \lambda y \neg [P(y)]$