Logique du premier ordre

Sylvain Salvati

Plan

La logique propositionnelle

De la logique propositionnelle, nous avons vu :

- sa syntaxe : la façon de représenter des formules, la notion de terme,

La logique propositionnelle

De la logique propositionnelle, nous avons vu :

- sa syntaxe : la façon de représenter des formules, la notion de terme,
- sa sémantique :
  - les modèles des formules : les valuations,

La logique propositionnelle

De la logique propositionnelle, nous avons vu :

- sa syntaxe : la façon de représenter des formules, la notion de terme,
- sa sémantique :
  - les modèles des formules : les valuations,
  - l'évaluation d'une formule avec une valuation : vérification de modèle

La logique propositionnelle

De la logique propositionnelle, nous avons vu :

- sa syntaxe : la façon de représenter des formules, la notion de terme,
- sa sémantique :
  - les modèles des formules : les valuations,
  - l'évaluation d'une formule avec une valuation : vérification de modèle
  - la sémantique d'une formule étant la valeur qu'elle prend pour chaque modèle/valuation, autrement dit sa table de vérité.

La logique propositionnelle

De la logique propositionnelle, nous avons vu :

- sa syntaxe : la façon de représenter des formules, la notion de terme,
- sa sémantique :
  - les modèles des formules : les valuations,
  - l'évaluation d'une formule avec une valuation : vérification de modèle
  - la sémantique d'une formule étant la valeur qu'elle prend pour chaque modèle/valuation, autrement dit sa table de vérité.
- son rôle en programmation :
  - l'utilisation des booléens en programmation et la possibilité de raisonner sur le flot du programme,

La logique propositionnelle

De la logique propositionnelle, nous avons vu :

- sa syntaxe : la façon de représenter des formules, la notion de terme,
- sa sémantique :
  - les modèles des formules : les valuations,
  - l'évaluation d'une formule avec une valuation : vérification de modèle
  - la sémantique d'une formule étant la valeur qu'elle prend pour chaque modèle/valuation, autrement dit sa table de vérité.
- son rôle en programmation :
  - l'utilisation des booléens en programmation et la possibilité de raisonner sur le flot du programme,
  - la résolution du problème de satisfiabilité,

La logique propositionnelle

De la logique propositionnelle, nous avons vu :

- sa syntaxe : la façon de représenter des formules, la notion de terme,
- sa sémantique :
  - les modèles des formules : les valuations,
  - l'évaluation d'une formule avec une valuation : vérification de modèle
  - la sémantique d'une formule étant la valeur qu'elle prend pour chaque modèle/valuation, autrement dit sa table de vérité.
- son rôle en programmation :
  - l'utilisation des booléens en programmation et la possibilité de raisonner sur le flot du programme,
  - la résolution du problème de satisfiabilité,
  - le codage de problèmes à contraintes dans SAT pour les résoudre rapidement.

Les limites de la logique propositionnelle

La logique propositionnelle ne parle que de vérité :

- elle ne permet pas de faire référence à des objets, ou à des notions,
- elle ne permet pas de mettre objets ou notions en rapport.

Les limites de la logique propositionnelle

La logique propositionnelle ne parle que de vérité :

- elle ne permet pas de faire référence à des objets, ou à des notions,
- elle ne permet pas de mettre objets ou notions en rapport.

La logique du premier ordre est un langage qui permet de dépasser ces limites.

Plan

La modélisation des objets par des termes

En logique du premier ordre, on utilise des termes pour modéliser les objets et les concepts.

La modélisation des objets par des termes

En logique du premier ordre, on utilise des termes pour modéliser les objets et les concepts.
Voici quelques exemples :

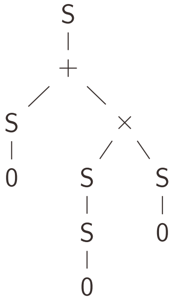
- les termes de la logique propositionnelle,

La modélisation des objets par des termes

En logique du premier ordre, on utilise des termes pour modéliser les objets et les concepts.
Voici quelques exemples :

- les termes de la logique propositionnelle,
- les termes construits avec les opérateurs suivants pour représenter les nombres et les calculs associés :
    - constante : 0, zéro
    - opérateur unaire : S, le successeur
    - opérateurs binaires : $+$ et $\times$, l'addition et la multiplication

$$S(+(S(0), \times(S(S(0)), S(0))))$$
$$S(S(0) + S(S(0)) \times S(0))$$



Termes typés : motivation et exemple

Lorsque l'on modélise, il arrive que l'on manipule des concepts distincts.
Au niveau des termes cela se traduit par l'utilisation de typage.

Termes typés : motivation et exemple

Lorsque l'on modélise, il arrive que l'on manipule des concepts distincts. Au niveau des termes cela se traduit par l'utilisation de typage.
Supposons que l'on souhaite modéliser un petit langage de programmation sur les entiers :

- Des variables contenant des entiers de type Var,
- Des entiers et des expressions de type Int :

Expression	Type	Description
i	Int	entier en écriture décimale
$+$	Int $\times$ Int $\rightarrow$ Int	opérateur d'addition
$-$	Int $\times$ Int $\rightarrow$ Int	opérateur de soustraction
$\times$	Int $\times$ Int $\rightarrow$ Int	opérateur de multiplication
!	Var $\rightarrow$ Int	récupération de la valeur associée à une variable

- Des expressions booléennes de type Bool :

Expression	Type	Description
$<, >, \leq, \geq, =$	Int $\times$ Int $\rightarrow$ Bool	comparaison d'entier
&&, $\vert\vert$, not	Bool $\times$ Bool $\rightarrow$ Bool	opérateurs booléens

Termes typés : motivation et exemple

Lorsque l'on modélise, il arrive que l'on manipule des concepts distincts.
Au niveau des termes cela se traduit par l'utilisation de typage.
Supposons que l'on souhaite modéliser un petit langage de programmation sur les entiers :

- Des variables contenant des entiers de type Var,
- Des entiers et des expressions de type Int :

Expression	Type	Description
i	Int	entier en écriture décimale
$+$	Int $\times$ Int $\rightarrow$ Int	opérateur d'addition
$-$	Int $\times$ Int $\rightarrow$ Int	opérateur de soustraction
$\times$	Int $\times$ Int $\rightarrow$ Int	opérateur de multiplication
!	Var $\rightarrow$ Int	récupération de la valeur associée à une variable

- Des expressions booléennes de type Bool :

Expression	Type	Description
$<, >, \leq, \geq, =$	Int $\times$ Int $\rightarrow$ Bool	comparaison d'entier
&&, $\|\|$, not	Bool $\times$ Bool $\rightarrow$ Bool	opérateurs booléens

- Des commandes de type Command :

Expression	Type	Description
ite	Bool $\times$ Command $\times$ Command $\rightarrow$ Command	branchement conditionnel
while	Bool $\times$ Command $\rightarrow$ Command	boucle *while*
:=	Var $\times$ Int $\rightarrow$ Command	l'affectation de variable
;	Command $\times$ Command $\rightarrow$ Command	la composition de commandes

Termes bien typé

Si on dispose d'opérateurs typé, on ne travaille plus qu'avec des termes bien typés, c'est-à-dire avec des termes auxquels ont peut associer un type :

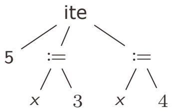
- Soit t_1, ..., t_n sont des termes bien typés de types respectifs α_1, ..., α_n, ce que l'on note $t_1 : \alpha_1$,..., $t_n : \alpha_n$,
- soit f est un opérateur de type $\alpha_1 \times \cdots \times \alpha_n \rightarrow \alpha$,
- le terme $f(t_1, \ldots, t_n)$ est un terme bien typé de type α

Termes bien typé

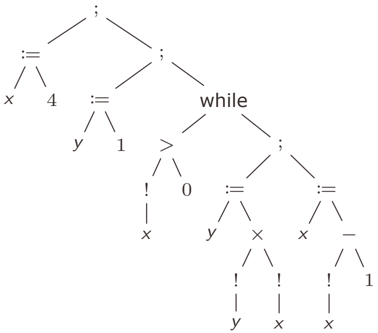
Si on dispose d'opérateurs typé, on ne travaille plus qu'avec des termes bien typés, c'est-à-dire avec des termes auxquels ont peut associer un type :

- Soit t_1, ..., t_n sont des termes bien typés de types respectifs α_1, ..., α_n, ce que l'on note $t_1 : \alpha_1, \ldots, t_n : \alpha_n$,
- soit f est un opérateur de type $\alpha_1 \times \cdots \times \alpha_n \to \alpha$,
- le terme $f(t_1, \ldots, t_n)$ est un terme bien typé de type α

Mal typé



Bien typé



Plan

Termes et variables

On utilise des variables pour construire des termes (celles-ci sont typées lors que l'on travaille avec des termes typés) :

- C'est ce que nous avons fait avec les variables propositionnelles.

Termes et variables

On utilise des variables pour construire des termes (celles-ci sont typées lors que l'on travaille avec des termes typés) :

- C'est ce que nous avons fait avec les variables propositionnelles.
- Cela nous permet d'avoir des termes paramétrés par d'autres termes :

Termes et variables

On utilise des variables pour construire des termes (celles-ci sont typées lors que l'on travaille avec des termes typés) :

- C'est ce que nous avons fait avec les variables propositionnelles.
- Cela nous permet d'avoir des termes paramétrés par d'autres termes :
  - De pouvoir substituer des termes aux variables.

Termes et variables

On utilise des variables pour construire des termes (celles-ci sont typées lors que l'on travaille avec des termes typés) :

- C'est ce que nous avons fait avec les variables propositionnelles.
- Cela nous permet d'avoir des termes paramétrés par d'autres termes :
  - De pouvoir substituer des termes aux variables.
  - De voir les termes un peu comme des fonctions.

Termes et variables

On utilise des variables pour construire des termes (celles-ci sont typées lors que l'on travaille avec des termes typés) :

- C'est ce que nous avons fait avec les variables propositionnelles.
- Cela nous permet d'avoir des termes paramétrés par d'autres termes :
  - De pouvoir substituer des termes aux variables.
  - De voir les termes un peu comme des fonctions.

Variables dans un terme

On note $var(t)$ l'ensemble des variables qui apparaissent dans t.

Substitutions : définition

Une substitution est une fonction partielle des variables dans les termes avec un domaine fini.
On note $x_1 = t_1, x_2 = t_2, \ldots, x_n = t_n$ la substitution σ telle que

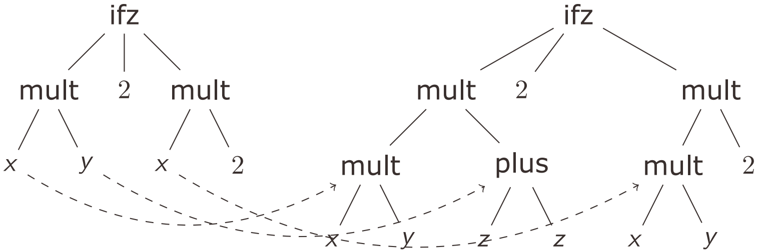
$$\sigma(y) = \begin{cases} t_i \text{ lorsque } y = x_i \\ \perp \text{ sinon} \end{cases}$$

Application d'une substitution à un terme

Appliquer une substitution à un terme revient à remplacer les variables du termes par le terme que la substitution leur associe. Pour une substitution σ on note $t.\sigma$ ou $\sigma(t)$ le résultat de son application à t.

Exemple

Considérons la substitution $x = \mathsf{mult}(x, y), y = \mathsf{plus}(z, z)$. On l'applique au terme $\mathsf{ifz}(\mathsf{mult}(x, y), 2, \mathsf{mult}(x, 2))$ et on obtient :
$\mathsf{ifz}(\mathsf{mult}(\mathsf{mult}(x, y), \mathsf{plus}(z, z)), 2, \mathsf{mult}(\mathsf{mult}(x, y)), 2)$



Le problème d'unification

Unification

Étant donné deux termes t et u, le problème de trouver une substitution σ telle $\sigma(t) = \sigma(u)$ est un problème d'unification noté $t = u$.

Le problème d'unification

Unification

Étant donné deux termes t et u, le problème de trouver une substitution σ telle $\sigma(t) = \sigma(u)$ est un problème d'unification noté $t = u$.

- Il existe des algorithmes efficaces pour résoudre ce problème,

Le problème d'unification

Unification

Étant donné deux termes t et u, le problème de trouver une substitution σ telle $\sigma(t) = \sigma(u)$ est un problème d'unification noté $t = u$.

- Il existe des algorithmes efficaces pour résoudre ce problème,
- L'unification est le problème au coeur de l'exécution du langage de programmation `PROLOG`,

Le problème d'unification

Unification

Étant donné deux termes t et u, le problème de trouver une substitution σ telle $\sigma(t) = \sigma(u)$ est un problème d'unification noté $t = u$.

- Il existe des algorithmes efficaces pour résoudre ce problème,
- L'unification est le problème au coeur de l'exécution du langage de programmation `PROLOG`,
- Des restrictions particulières de ce problème sont utilisées dans de nombreux langage de programmation : le filtrage.

Le problème d'unification

Unification

Étant donné deux termes t et u, le problème de trouver une substitution σ telle $\sigma(t) = \sigma(u)$ est un problème d'unification noté $t = u$.

- Il existe des algorithmes efficaces pour résoudre ce problème,
- L'unification est le problème au coeur de l'exécution du langage de programmation `PROLOG`,
- Des restrictions particulières de ce problème sont utilisées dans de nombreux langage de programmation : le filtrage.

Filtrage

Un problème d'unification $t = u$ est :

- un problème de filtrage lorsque *u ne contient pas de variables*,
- un problème de filtrage linéaire lorsque :
  - *u ne contient pas de variables*,
  - chaque variable a *au plus une occurrence* dans t

Filtrage linéaire ou filtrage structurel et programmation

Le filtrage linéaire ou filtrage structurel est de plus en plus utilisé en programmation (Python, Java, Rust, Haskell, Ocaml...)

```
x=level[0]
y=level[1]
if x==0 && y==0:
    print("Origin")
elif x==0:
    print(f"Y={y}")
elif y==0:
    print(f"X={x}")
else:
    print(f"X={x}, Y={y}")
```

```
match level:
    case (0, 0):
        print("Origin")
    case (0, y):
        print(f"Y={y}")
    case (x, 0):
        print(f"X={x}")
    case (x, y):
        print(f"X={x}, Y={y}")
```

Filtrage linéaire ou filtrage structurel et programmation

Le filtrage linéaire ou filtrage structurel est de plus en plus utilisé en programmation (Python, Java, Rust, Haskell, Ocaml. . .)

```
x=level[0]
y=level[1]
if x==0 && y==0:
    print("Origin")
elif x==0:
    print(f"Y={y}")
elif y==0:
    print(f"X={x}")
else:
    print(f"X={x}, Y={y}")
```

```
match level:
    case (0, 0):
        print("Origin")
    case (0, y):
        print(f"Y={y}")
    case (x, 0):
        print(f"X={x}")
    case (x, y):
        print(f"X={x}, Y={y}")
```

Il rend le code plus simple et plus lisible.

Plan

Interprétation des termes : le cas non typé

Associer une *sémantique* aux termes revient à :

- fixer un ensemble $\mathcal{D}$, le domaine d'interprétation,
- pour chaque opération f d'arité n, on associe une fonction $[\![f]\!]$ de $\mathcal{D}^n \mapsto \mathcal{D}$, (**NB** : quand $n = 0$, f est interprété comme un élément de $\mathcal{D}$).

On appelle modèle la paire $\mathcal{M} = (\mathcal{D}, [\![\cdot]\!])$.

Interprétation des termes : le cas non typé

Associer une *sémantique* aux termes revient à :

- fixer un ensemble $\mathcal{D}$, le domaine d'interprétation,
- pour chaque opération f d'arité n, on associe une fonction $[\![f]\!]$ de $\mathcal{D}^n \mapsto \mathcal{D}$, (**NB** : quand $n = 0$, f est interprété comme un élément de $\mathcal{D}$).

On appelle modèle la paire $\mathcal{M} = (\mathcal{D}, [\![\cdot]\!])$.
Pour interpréter un terme contenant des variables, il nous faut une valuation qui associe à chaque variable du terme un élément de $\mathcal{D}$. Étant donnée une valuation ν, la sémantique $[\![t, \nu]\!]_{\mathcal{M}}$ d'un terme t est définie par :

$$\begin{array}{rcl} [\![x, \nu]\!]_{\mathcal{M}} & = & \nu(x) \\ [\![f(t_1, \ldots, t_n), \nu]\!]_{\mathcal{M}} & = & [\![f]\!]([\![t_1, \nu]\!]_{\mathcal{M}}, \ldots, [\![t_n, \nu]\!]_{\mathcal{M}}) \end{array}$$

Interprétation des termes : le cas typé

En présence de types $A_1, \ldots, A_m$, associer une *sémantique* aux termes revient à :

- fixer une famille d'ensembles $(\mathcal{D}_i)_{i\in[1,m]}$, pour tout i dans $[1, m]$, $\mathcal{D}_i$ est le domaines d'interprétation de A_i
- pour chaque opération f de type $A_{i_1} \times \cdots \times A_{i_n} \to A_i$, on associe une fonction $[\![f]\!]$ de $\mathcal{D}_{i_1} \times \cdots \times \mathcal{D}_{i_n} \mapsto \mathcal{D}_i$.

On appelle modèle la paire $\mathcal{M} = ((\mathcal{D}_i)_{i\in[1,m]}, [\![\cdot]\!])$.

Interprétation des termes : le cas typé

En présence de types $A_1, \ldots, A_m$, associer une *sémantique* aux termes revient à :

- fixer une famille d'ensembles $(\mathcal{D}_i)_{i\in[1,m]}$, pour tout i dans $[1, m]$, $\mathcal{D}_i$ est le domaines d'interprétation de A_i
- pour chaque opération f de type $A_{i_1} \times \cdots \times A_{i_n} \to A_i$, on associe une fonction $[\![f]\!]$ de $\mathcal{D}_{i_1} \times \cdots \times \mathcal{D}_{i_n} \mapsto \mathcal{D}_i$.

On appelle modèle la paire $\mathcal{M} = ((\mathcal{D}_i)_{i\in[1,m]}, [\![\cdot]\!])$.
Pour interpréter un terme contenant des variables, il nous faut une valuation qui associe à chaque variable du terme un élément de $\mathcal{D}_i$ suivant le type de la variable. Étant donnée une valuation ν, la sémantique $[\![t, \nu]\!]_{\mathcal{M}}$ d'un terme t est définie par :

$$
\begin{array}{rcl}
[\![x, \nu]\!]_{\mathcal{M}} & = & \nu(x) \\
[\![f(t_1, \ldots, t_n), \nu]\!]_{\mathcal{M}} & = & [\![f]\!]([\![t_1, \nu]\!], \ldots, [\![t_n, \nu]\!]_{\mathcal{M}})
\end{array}
$$

Interprétation des termes : le cas typé

En présence de types $A_1, \ldots, A_m$, associer une *sémantique* aux termes revient à :

- fixer une famille d'ensembles $(\mathcal{D}_i)_{i \in [1,m]}$, pour tout i dans $[1, m]$, $\mathcal{D}_i$ est le domaines d'interprétation de A_i
- pour chaque opération f de type $A_{i_1} \times \cdots \times A_{i_n} \to A_i$, on associe une fonction $[\![f]\!]$ de $\mathcal{D}_{i_1} \times \cdots \times \mathcal{D}_{i_n} \mapsto \mathcal{D}_i$.

On appelle modèle la paire $\mathcal{M} = ((\mathcal{D}_i)_{i \in [1,m]}, [\![\cdot]\!])$.
Pour interpréter un terme contenant des variables, il nous faut une valuation qui associe à chaque variable du terme un élément de $\mathcal{D}_i$ suivant le type de la variable. Étant donnée une valuation ν, la sémantique $[\![t, \nu]\!]_{\mathcal{M}}$ d'un terme t est définie par :

$$\begin{array}{rcl} [\![x, \nu]\!]_{\mathcal{M}} & = & \nu(x) \\ [\![f(t_1, \ldots, t_n), \nu]\!]_{\mathcal{M}} & = & [\![f]\!]([\![t_1, \nu]\!], \ldots, [\![t_n, \nu]\!]_{\mathcal{M}}) \end{array}$$

Si t est un terme de type A_i, alors $[\![t, \nu]\!]$ est un élément de $\mathcal{D}_i$.

Plan

Vers la logique

Pour le moment, nous avons vu comment modéliser les objets et les concepts sur lesquelles nous voulons asserter des propriétés ou raisonner.

Vers la logique

Pour le moment, nous avons vu comment modéliser les objets et les concepts sur lesquelles nous voulons asserter des propriétés ou raisonner. Il nous faut maintenant nous donner les moyens de dire de choses sur ces objets. Pour cela, on se dote d'un ensemble de prédicats.

Vers la logique

Pour le moment, nous avons vu comment modéliser les objets et les concepts sur lesquelles nous voulons asserter des propriétés ou raisonner.
Il nous faut maintenant nous donner les moyens de dire de choses sur ces objets. Pour cela, on se dote d'un ensemble de prédicats.
Chaque prédicats a une arité (un type si on travaille avec des termes typés).

Vers la logique

Pour le moment, nous avons vu comment modéliser les objets et les concepts sur lesquelles nous voulons asserter des propriétés ou raisonner.
Il nous faut maintenant nous donner les moyens de dire de choses sur ces objets. Pour cela, on se dote d'un ensemble de prédicats.
Chaque prédicats a une arité (un type si on travaille avec des termes typés).
Cela revient à enrichir les opérateurs sur les termes de la manière suivante :

- on rajoute un nouveau type au(x) type(s) des termes (on appelle T le type des termes) : Bool,
- pour chaque prédicat on ajoute un opérateur de type $T \times \cdots \times T \rightarrow \mathsf{Bool}$ (on adapte au typage pour le cas typé),
- on ajoute les connecteurs logique :

Interprétation des formules

Afin de pouvoir interpréter les formules, il nous faut donner une interprétation aux termes composés avec les prédicats telle que :

- Bool est interpréter dans l'ensemble $\{0, 1\}$,
- les connecteurs logiques ont l'interprétation que nous avons vue pour la logique propositionnelle.

NB : On appelle modèle une telle interprétation (il s'agit toujours d'une paire $\mathcal{M} = ((\mathcal{D}_i)_{i \in [1,m]}, [\![\cdot]\!])$

Interprétation des formules

Afin de pouvoir interpréter les formules, il nous faut donner une interprétation aux termes composés avec les prédicats telle que :

- Bool est interpréter dans l'ensemble $\{0, 1\}$,
- les connecteurs logiques ont l'interprétation que nous avons vue pour la logique propositionnelle.

NB : On appelle modèle une telle interprétation (il s'agit toujours d'une paire $\mathcal{M} = ((\mathcal{D}_i)_{i \in [1,m]}, [\![\cdot]\!])$

Les prédicats représentent ainsi des relations entre les éléments du domaine d'interprétation : c'est une représentation par fonction caractéristique.

Interprétation des formules

Afin de pouvoir interpréter les formules, il nous faut donner une interprétation aux termes composés avec les prédicats telle que :

- Bool est interpréter dans l'ensemble $\{0, 1\}$,
- les connecteurs logiques ont l'interprétation que nous avons vue pour la logique propositionnelle.

NB : On appelle modèle une telle interprétation (il s'agit toujours d'une paire $\mathcal{M} = ((\mathcal{D}_i)_{i \in [1,m]}, [\![\cdot]\!])$

Les prédicats représentent ainsi des relations entre les éléments du domaine d'interprétation : c'est une représentation par fonction caractéristique.

Cette interprétation relationnelle des prédicats est très proche de ce que vous avez vu en base de données sur l'interprétation des tables par des relations.

Se passer des termes

Il est également possible de se passer tout simplement de terme en n'utilisant que des prédicats :

- pour chaque symbole de fonction f d'arité n, on crée un prédicat p_f d'arité $n+1$ et on l'interprète de la façon suivante :
  $[\![p_f]\!](u_1, \ldots, u_n, v) = \begin{cases} 1 \text{ si } v = [\![f]\!](u_1, \ldots, u_n) \\ 0 \text{ sinon} \end{cases}$

Se passer des termes

Il est également possible de se passer tout simplement de terme en n'utilisant que des prédicats :

- pour chaque symbole de fonction f d'arité n, on crée un prédicat p_f d'arité $n+1$ et on l'interprète de la façon suivante :
  $[\![p_f]\!](u_1, \ldots, u_n, v) = \begin{cases} 1 \text{ si } v = [\![f]\!](u_1, \ldots, u_n) \\ 0 \text{ sinon} \end{cases}$

En ajoutant une variable par sous-terme, on peut ainsi transformer toute formule en une formule ayant la même interprétation et qui ne contient plus de termes.

$$\leq (S(x), S(S(y))) \mapsto p_S(x, sx) \wedge p_S(y, sy) \wedge p_S(sy, ssy) \wedge \leq (sx, ssy)$$

Plan

Les quantificateurs

En logique du premier ordre, il y a deux quantificateurs :

- $\forall$: *pour tout*,
- $\exists$: *il existe*.

Les quantificateurs

En logique du premier ordre, il y a deux quantificateurs :

- $\forall$: *pour tout*,
- $\exists$: *il existe*.

Les quantificateurs s'utilisent de la façon suivante :

$$\forall x.\varphi \qquad\qquad \exists x.\varphi$$

$$\begin{tikzcd} \forall x. \arrow[d, dash] \\ \varphi \end{tikzcd} \qquad\qquad \begin{tikzcd} \exists x. \arrow[d, dash] \\ \varphi \end{tikzcd}$$

Les quantificateurs

En logique du premier ordre, il y a deux quantificateurs :

- $\forall$: *pour tout*,
- $\exists$: *il existe*.

Les quantificateurs s'utilisent de la façon suivante :

$$\forall x.\varphi \qquad\qquad \exists x.\varphi$$

$$\begin{array}{c} \forall x. \\ | \\ \varphi \end{array} \qquad\qquad \begin{array}{c} \exists x. \\ | \\ \varphi \end{array}$$

Les quantificateurs de variables sont des lieurs : les formules contiennent alors deux type de variables, les variables libres et les variables liées :

$$\begin{array}{rcl} VL(\forall x.\varphi) & = & VL(\varphi) - \{x\} \\ VL(\exists x.\varphi) & = & VL(\varphi) - \{x\} \\ VL(f(t_1, \ldots, t_n)) & = & \bigcup_{i=1}^{n} VL(t_i) \\ VL(x) & = & \{x\} \end{array}$$

Une formule φ est close si $VL(\varphi) = \emptyset$.

Interprétation des quantificateurs

Si $\mathcal{M}$ est un modèle des opérateurs et des prédicats alors, étant donnée une valuation :

$$
\begin{array}{rcl}
[\![\forall x.\varphi, \nu]\!]_{\mathcal{M}} & = & \min\{[\![\varphi, \nu[x = d]]\!]_{\mathcal{M}} \mid d \in \mathcal{D}\} \\
[\![\exists x.\varphi, \nu]\!]_{\mathcal{M}} & = & \max\{[\![\varphi, \nu[x = d]]\!]_{\mathcal{M}} \mid d \in \mathcal{D}\}
\end{array}
$$

Interprétation des quantificateurs

Si $\mathcal{M}$ est un modèle des opérateurs et des prédicats alors, étant donnée une valuation :

$$\begin{array}{rcl} [\![\forall x.\varphi, \nu]\!]_{\mathcal{M}} & = & \min\{[\![\varphi, \nu[x = d]]\!]_{\mathcal{M}} \mid d \in \mathcal{D}\} \\ [\![\exists x.\varphi, \nu]\!]_{\mathcal{M}} & = & \max\{[\![\varphi, \nu[x = d]]\!]_{\mathcal{M}} \mid d \in \mathcal{D}\} \end{array}$$

Satisfiabilité

Une fomule φ est *satifaite* par un modèle $\mathcal{M}$ et une valuation ν si $[\![\varphi, \nu]\!]_{\mathcal{M}} = 1$: $\mathcal{M}, \nu \models \varphi$.

Interprétation des quantificateurs

Si $\mathcal{M}$ est un modèle des opérateurs et des prédicats alors, étant donnée une valuation :

$$\begin{array}{rcl} [\![\forall x.\varphi,\nu]\!]_{\mathcal{M}} & = & \min\{[\![\varphi,\nu[x=d]]\!]_{\mathcal{M}} \mid d \in \mathcal{D}\} \\ [\![\exists x.\varphi,\nu]\!]_{\mathcal{M}} & = & \max\{[\![\varphi,\nu[x=d]]\!]_{\mathcal{M}} \mid d \in \mathcal{D}\} \end{array}$$

Satisfiabilité

Une fomule φ est *satifaite* par un modèle $\mathcal{M}$ et une valuation ν si $[\![\varphi,\nu]\!]_{\mathcal{M}} = 1$: $\mathcal{M},\nu \models \varphi$.

Tautologie

Une fomule φ est une *tautologie* si pour tout modèle $\mathcal{M}$ et toute valuation ν : $\mathcal{M},\nu \models \varphi$.

Résumé

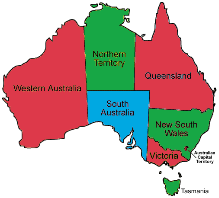
Pour représenter des propriétés sur des objets on utilise :

- des opérateurs (e.g. opérateurs binaires $+$ et $*$ pour les entiers et opérateur unaire du successeur S, constante 0),
- des prédicats (e.g. prédicat d'égalité, $=$ pour les entiers).

On parle alors de langage une fois que tout cela est fixé.

Retour en Autralie

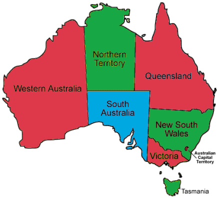
$$\mathcal{D} = \{r, v, b, WA, NT, Q, SA, NSW, V, T\}$$



On se donne les prédicats suivants :

- les prédicats unaires : `couleur` et `etat`,

Retour en Autralie



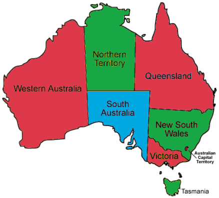
$\mathcal{D} = \{r, v, b, WA, NT, Q, SA, NSW, V, T\}$

$\texttt{couleur} = \{r, v, b\}$

On se donne les prédicats suivants :

- les prédicats unaires : `couleur` et `etat`,

Retour en Autralie



$\mathcal{D} = \{r, v, b, WA, NT, Q, SA, NSW, V, T\}$

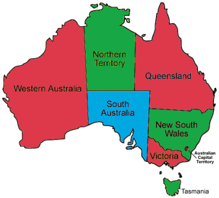
$\texttt{couleur} = \{r, v, b\}$

$\texttt{etat} = \{WA, NT, Q, SA, NSW, V, T\}$

On se donne les prédicats suivants :

- les prédicats unaires : `couleur` et `etat`,

Retour en Autralie



$\mathcal{D} = \{r, v, b, WA, NT, Q, SA, NSW, V, T\}$

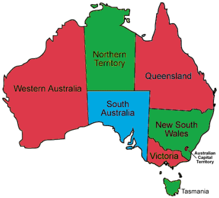
$\texttt{couleur} = \{r, v, b\}$

$\texttt{etat} = \{WA, NT, Q, SA, NSW, V, T\}$

On se donne les prédicats suivants :

- les prédicats unaires : `couleur` et `etat`,
- les predicats binaires : `voisin` et `colorie`.

Retour en Autralie



$$\mathcal{D} = \{r, v, b, WA, NT, Q, SA, NSW, V, T\}$$

$$\texttt{couleur} = \{r, v, b\}$$

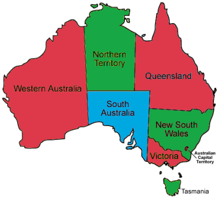
$$\texttt{etat} = \{WA, NT, Q, SA, NSW, V, T\}$$

$$\texttt{voisin} = \{\{WA, NT\}, \{WA, SA\}, \{NT, Q\}, \{NT, SA\}, \{SA, Q\}, \{SA, NSW\}, \{SA, V\}, \{Q, NSW\}, \{NSW, V\}, \{V, T\}\}$$

On se donne les prédicats suivants :

- les prédicats unaires : `couleur` et `etat`,
- les predicats binaires : `voisin` et `colorie`.

Retour en Autralie



$\mathcal{D} = \{r, v, b, WA, NT, Q, SA, NSW, V, T\}$

$\texttt{couleur} = \{r, v, b\}$

$\texttt{etat} = \{WA, NT, Q, SA, NSW, V, T\}$

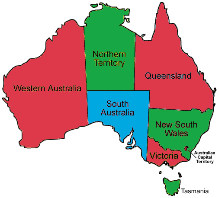
$$\begin{aligned}\texttt{voisin} = \{&\{WA, NT\}, \{WA, SA\}, \{NT, Q\},\\ &\{NT, SA\}, \{SA, Q\}, \{SA, NSW\},\\ &\{SA, V\}, \{Q, NSW\}, \{NSW, V\},\\ &\{V, T\}\}\end{aligned}$$

$$\begin{aligned}\texttt{colorie} = \{&(r, WA), (v, NT), (b, SO), (r, Q),\\ &(v, NSW), (r, V), (v, T)\}\end{aligned}$$

On se donne les prédicats suivants :

- les prédicats unaires : `couleur` et `etat`,
- les predicats binaires : `voisin` et `colorie`.

Retour en Autralie



$$\mathcal{D} = \{r, v, b, WA, NT, Q, SA, NSW, V, T\}$$

$$\mathtt{couleur} = \{r, v, b\}$$

$$\mathtt{etat} = \{WA, NT, Q, SA, NSW, V, T\}$$

$$\begin{aligned}\mathtt{voisin} = \{&\{WA, NT\}, \{WA, SA\}, \{NT, Q\},\\ &\{NT, SA\}, \{SA, Q\}, \{SA, NSW\},\\ &\{SA, V\}, \{Q, NSW\}, \{NSW, V\},\\ &\{V, T\}\}\end{aligned}$$

$$\begin{aligned}\mathtt{colorie} = \{&(r, WA), (v, NT), (b, SO), (r, Q),\\ &(v, NSW), (r, V), (v, T)\}\end{aligned}$$

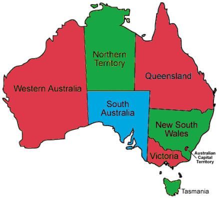
On se donne les prédicats suivants :

- les prédicats unaires : $\mathtt{couleur}$ et $\mathtt{etat}$,
- les predicats binaires : $\mathtt{voisin}$ et $\mathtt{colorie}$.

Tout état est colorié par une couleur :

$$\forall x.\mathtt{etat}(x) \Rightarrow \exists y.\mathtt{couleur}(y) \wedge \mathtt{colorie}(y, x)$$

Retour en Autralie



$$\mathcal{D} = \{r, v, b, WA, NT, Q, SA, NSW, V, T\}$$

$$\texttt{couleur} = \{r, v, b\}$$

$$\texttt{etat} = \{WA, NT, Q, SA, NSW, V, T\}$$

$$\texttt{voisin} = \{\{WA, NT\}, \{WA, SA\}, \{NT, Q\}, \{NT, SA\}, \{SA, Q\}, \{SA, NSW\}, \{SA, V\}, \{Q, NSW\}, \{NSW, V\}, \{V, T\}\}$$

$$\texttt{colorie} = \{(r, WA), (v, NT), (b, SO), (r, Q), (v, NSW), (r, V), (v, T)\}$$

On se donne les prédicats suivants :

- les prédicats unaires : couleur et etat,
- les predicats binaires : voisin et colorie.

Si deux états sont voisins ils sont coloriés par des couleurs différentes :

$$\forall x.\forall y.\texttt{etat}(x) \land \texttt{etat}(y) \land \texttt{voisin}(x, y) \Rightarrow \forall c_1.\forall c_2.\texttt{couleur}(c_1) \land \texttt{couleur}(c_2) \land \texttt{colorie}(c_1, x) \land \texttt{colorie}(c_2, y) \Rightarrow \neg c_1 = c_2)$$

Axiomatisation

Pour donner une définition complète des objets, on en donne un axiomatisation : c'est-à-dire l'ensemble des propriétés que nos objets doivent vérifier.

Axiomatisation

Pour donner une définition complète des objets, on en donne un axiomatisation : c'est-à-dire l'ensemble des propriétés que nos objets doivent vérifier.

Axiomatisation des entiers de Peano

- $\forall x.\neg S(x) = 0$,
- $\forall x.x = 0 \vee \exists y.x = S(y)$,
- $\forall x.\forall y.S(x) = S(y) \Rightarrow x = y$
- $\forall x.\forall y.x + S(y) = S(x + y)$
- $\forall x.x * 0 = 0$
- $\forall x.\forall y.x * S(y) = (x * y) + x$
- schéma d'induction, pour toute formule $\varphi(x, x_1, \ldots, x_n)$:

$$\forall x_1. \ldots \forall x_n. \quad (\varphi(0, x_1 \ldots, x_n) \wedge \forall x.\varphi(x, x_1, \ldots, x_n) \Rightarrow \varphi(S(x), x_1, \ldots, x_n)) \Rightarrow \forall x.\varphi(x, x_1, \ldots, x_n)$$

Plan

Traduire du français vers la logique

L'approche de *Sujet/Prédicat*

Afin de traduire un énoncé en français (ou une autre langue) vers la logique il faut y repérer deux éléments :

- Le sujet : *ce dont on parle*. La plupart du temps pour nous ce sera le sujet grammatical de la phrase.
- Le prédicat : *ce qu'on en dit*. La plupart du temps pour nous il s'agira du groupe verbal principal de la phrase.

Traduire du français vers la logique

L'approche de *Sujet/Prédicat*

Afin de traduire un énoncé en français (ou une autre langue) vers la logique il faut y repérer deux éléments :

- Le sujet : *ce dont on parle*. La plupart du temps pour nous ce sera le sujet grammatical de la phrase.
- Le prédicat : *ce qu'on en dit*. La plupart du temps pour nous il s'agira du groupe verbal principal de la phrase.

Quelques phrases

- $\underbrace{\text{Tous les étudiants}}_{\text{sujet}}\ \underbrace{\text{apprennent l'anglais.}}_{\text{prédicat}}$

Traduire du français vers la logique

L'approche de *Sujet/Prédicat*

Afin de traduire un énoncé en français (ou une autre langue) vers la logique il faut y repérer deux éléments :

- Le sujet : *ce dont on parle*. La plupart du temps pour nous ce sera le sujet grammatical de la phrase.
- Le prédicat : *ce qu'on en dit*. La plupart du temps pour nous il s'agira du groupe verbal principal de la phrase.

Quelques phrases

- $\underbrace{\text{Tous les étudiants}}_{\text{sujet}}\ \underbrace{\text{apprennent l'anglais}}_{\text{prédicat}}.$
- $$\underbrace{\text{Tous les étudiants qui apprennent l'anglais}}_{\text{sujet}}\ \underbrace{\text{n'apprennent pas l'allemand}}_{\text{prédicat}}.$$

Traduire du français vers la logique

L'approche de *Sujet/Prédicat*

Afin de traduire un énoncé en français (ou une autre langue) vers la logique il faut y repérer deux éléments :

- Le sujet : *ce dont on parle*. La plupart du temps pour nous ce sera le sujet grammatical de la phrase.
- Le prédicat : *ce qu'on en dit*. La plupart du temps pour nous il s'agira du groupe verbal principal de la phrase.

Quelques phrases

- $\underbrace{\text{Tous les étudiants}}_{\text{sujet}}$ $\underbrace{\text{apprennent l'anglais}}_{\text{prédicat}}$.
- $\underbrace{\text{Tous les étudiants qui apprennent l'anglais}}_{\text{sujet}}$ $\underbrace{\text{n'apprennent pas l'allemand}}_{\text{prédicat}}$.
- $\underbrace{\text{Certains étudiant qui apprennent l'anglais}}_{\text{sujet}}$ $\underbrace{\text{n'apprennent pas l'allemand}}_{\text{prédicat}}$.

Traduction de la quantification universelle

Tous les...

En français, la quantification universelle s'exprime par *tous les*/*aucun*/... et se traduit en logique par :

$$\forall x.\mathtt{sujet}(x) \Rightarrow \mathtt{predicat}(x)$$

lorsque :

- $\mathtt{sujet}(x)$ représente la propriété exprimée par le sujet,
- $\mathtt{predicat}(x)$ représente la propriété exprimée par le prédicat.

Traduction de la quantification universelle

Tous les...

En français, la quantification universelle s'exprime par *tous les*/*aucun*/... et se traduit en logique par :

$$\forall x.\mathtt{sujet}(x) \Rightarrow \mathtt{predicat}(x)$$

lorsque :

- $\mathtt{sujet}(x)$ représente la propriété exprimée par le sujet,
- $\mathtt{predicat}(x)$ représente la propriété exprimée par le prédicat.

Quelques exemples

Nous nous donnons le langage logique suivant :

- Des constantes : `english`, `german`,
- Des prédicats : `student` (unaire), `learn` (binaire), $\mathtt{learn}(x, y)$ signifiant x *apprend* y.

Traduction de la quantification universelle

Tous les...

En français, la quantification universelle s'exprime par *tous les*/*aucun*/... et se traduit en logique par :

$$\forall x.\mathtt{sujet}(x) \Rightarrow \mathtt{predicat}(x)$$

lorsque :

- $\mathtt{sujet}(x)$ représente la propriété exprimée par le sujet,
- $\mathtt{predicat}(x)$ représente la propriété exprimée par le prédicat.

Quelques exemples

Nous nous donnons le langage logique suivant :

- Des constantes : $\mathtt{english}$, $\mathtt{german}$,
- Des prédicats : $\mathtt{student}$ (unaire), $\mathtt{learn}$ (binaire), $\mathtt{learn}(x, y)$ signifiant *x apprend y*.

$$\underbrace{\text{Tous les étudiants}}_{\text{sujet}}\ \underbrace{\text{apprennent l'anglais}}_{\text{prédicat}}$$

$$\downarrow$$

$$\forall x.\,\underbrace{\mathtt{student}(x)}_{\text{sujet}} \Rightarrow \underbrace{\mathtt{learn}(x, \mathtt{english})}_{\text{prédicat}}$$

Traduction de la quantification universelle

Tous les...

En français, la quantification universelle s'exprime par *tous les*/*aucun*/... et se traduit en logique par :

$$\forall x.\mathtt{sujet}(x) \Rightarrow \mathtt{predicat}(x)$$

lorsque :

- $\mathtt{sujet}(x)$ représente la propriété exprimée par le sujet,
- $\mathtt{predicat}(x)$ représente la propriété exprimée par le prédicat.

Quelques exemples

Nous nous donnons le langage logique suivant :

- Des constantes : $\mathtt{english}$, $\mathtt{german}$,
- Des prédicats : $\mathtt{student}$ (unaire), $\mathtt{learn}$ (binaire), $\mathtt{learn}(x, y)$ signifiant *x apprend y*.

$$\underbrace{\text{Tous les étudiants qui apprennent l'anglais}}_{\text{sujet}}\ \underbrace{\text{n'apprennent pas l'allemand}}_{\text{prédicat}}$$

$$\downarrow$$

$$\forall x.\, \underbrace{\mathtt{student}(x) \wedge \mathtt{learn}(x, \mathtt{english})}_{\text{sujet}} \Rightarrow \underbrace{\neg\mathtt{learn}(x, \mathtt{german})}_{\text{prédicat}}$$

Traduction de la quantification universelle

Tous les...

En français, la quantification universelle s'exprime par *tous les*/*aucun*/... et se traduit en logique par :

$$\forall x.\mathtt{sujet}(x) \Rightarrow \mathtt{predicat}(x)$$

lorsque :

- $\mathtt{sujet}(x)$ représente la propriété exprimée par le sujet,
- $\mathtt{predicat}(x)$ représente la propriété exprimée par le prédicat.

Quelques exemples

Nous nous donnons le langage logique suivant :

- Des constantes : $\mathtt{english}$, $\mathtt{german}$,
- Des prédicats : $\mathtt{student}$ (unaire), $\mathtt{learn}$ (binaire), $\mathtt{learn}(x, y)$ signifiant *x apprend y*.

$$\underbrace{\text{Aucun étudiants}}_{\text{sujet}}\ \underbrace{\text{n'apprend l'anglais et l'allemand}}_{\text{prédicat}}$$

$$\downarrow$$

$$\forall x.\,\underbrace{\mathtt{student}(x)}_{\text{sujet}} \Rightarrow \underbrace{\neg(\mathtt{learn}(x, \mathtt{german}) \land \mathtt{learn}(x, \mathtt{english}))}_{\text{prédicat}}$$

$$\equiv$$

$$\forall x.\mathtt{student}(x) \Rightarrow (\neg\mathtt{learn}(x, \mathtt{german}) \lor \neg\mathtt{learn}(x, \mathtt{english}))$$

Traduction de la quantification existentielle

Il existe. . .

En français, la quantification existentielle s'exprime par *il y a*/*certains*/*il existe*. . . et se traduit en logique par :

$$\exists x.\mathtt{sujet}(x) \wedge \mathtt{predicat}(x)$$

lorsque :

- $\mathtt{sujet}(x)$ représente la propriété exprimée par le sujet,
- $\mathtt{predicat}(x)$ représente la propriété exprimée par le prédicat.

Traduction de la quantification existentielle

Il existe. . .

En français, la quantification existentielle s'exprime par *il y a*/*certains*/*il existe*. . . et se traduit en logique par :

$$\exists x.\texttt{sujet}(x) \land \texttt{predicat}(x)$$

lorsque :

- $\texttt{sujet}(x)$ représente la propriété exprimée par le sujet,
- $\texttt{predicat}(x)$ représente la propriété exprimée par le prédicat.

Quelques exemples

Nous reprenons le langage logique :

- Des constantes : `english`, `german`,
- Des prédicats : `student` (unaire), `learn` (binaire), $\texttt{learn}(x, y)$ signifiant x *apprend* y.

Traduction de la quantification existentielle

Il existe. . .

En français, la quantification existentielle s'exprime par *il y a*/*certains*/*il existe*. . . et se traduit en logique par :

$$\exists x.\mathtt{sujet}(x) \land \mathtt{predicat}(x)$$

lorsque :

- $\mathtt{sujet}(x)$ représente la propriété exprimée par le sujet,
- $\mathtt{predicat}(x)$ représente la propriété exprimée par le prédicat.

Quelques exemples

Nous reprenons le langage logique :

- Des constantes : $\mathtt{english}$, $\mathtt{german}$,
- Des prédicats : $\mathtt{student}$ (unaire), $\mathtt{learn}$ (binaire), $\mathtt{learn}(x, y)$ signifiant x *apprend* y.

$$\underbrace{\text{Certains étudiants}}_{\text{sujet}}\ \underbrace{\text{apprennent l'anglais}}_{\text{prédicat}}$$

$$\downarrow$$

$$\exists x.\ \underbrace{\mathtt{student}(x)}_{\text{sujet}} \land \underbrace{\mathtt{learn}(x, \mathtt{english})}_{\text{prédicat}}$$

Traduction de la quantification existentielle

Il existe. . .

En français, la quantification existentielle s'exprime par *il y a*/*certains*/*il existe*. . . et se traduit en logique par :

$$\exists x.\mathtt{sujet}(x) \wedge \mathtt{predicat}(x)$$

lorsque :

- $\mathtt{sujet}(x)$ représente la propriété exprimée par le sujet,
- $\mathtt{predicat}(x)$ représente la propriété exprimée par le prédicat.

Quelques exemples

Nous reprenons le langage logique :

- Des constantes : `english`, `german`,
- Des prédicats : `student` (unaire), `learn` (binaire), $\mathtt{learn}(x, y)$ signifiant *x apprend y*.

$$\underbrace{\text{Certains étudiants qui apprennent l'anglais}}_{\text{sujet}}\ \underbrace{\text{n'apprennent pas l'allemand}}_{\text{prédicat}}$$

$$\downarrow$$

$$\exists x.\,\underbrace{\mathtt{student}(x) \wedge \mathtt{learn}(x, \mathtt{english})}_{\text{sujet}} \wedge \underbrace{\neg\mathtt{learn}(x, \mathtt{german})}_{\text{prédicat}}$$

Traduction de la quantification existentielle

Il existe...

En français, la quantification existentielle s'exprime par *il y a/certains/il existe...* et se traduit en logique par :

$$\exists x.\mathtt{sujet}(x) \wedge \mathtt{predicat}(x)$$

lorsque :

- $\mathtt{sujet}(x)$ représente la propriété exprimée par le sujet,
- $\mathtt{predicat}(x)$ représente la propriété exprimée par le prédicat.

Quelques exemples

Nous reprenons le langage logique :

- Des constantes : $\mathtt{english}$, $\mathtt{german}$,
- Des prédicats : $\mathtt{student}$ (unaire), $\mathtt{learn}$ (binaire), $\mathtt{learn}(x, y)$ signifiant *x apprend y*.

$$\underbrace{\text{Certains étudiants}}_{\text{sujet}}\ \underbrace{\text{n'apprennent pas l'anglais et l'allemand}}_{\text{prédicat}}$$

$$\downarrow$$

$$\exists x.\,\underbrace{\mathtt{student}(x)}_{\text{sujet}} \wedge \underbrace{\neg(\mathtt{learn}(x, \mathtt{german}) \wedge \mathtt{learn}(x, \mathtt{english}))}_{\text{prédicat}}$$

$$\equiv$$

$$\exists x.\mathtt{student}(x) \wedge (\neg\mathtt{learn}(x, \mathtt{german}) \vee \neg\mathtt{learn}(x, \mathtt{english}))$$

Dualité de la traduction des quantificateurs

$$
\begin{aligned}
\neg(\forall x.\mathtt{sujet}(x) \Rightarrow \mathtt{predicat}(x)) &\equiv \exists x.\neg(\mathtt{sujet}(x) \Rightarrow \mathtt{predicat}(x)) \\
&\equiv \exists x.\neg(\neg\mathtt{sujet}(x) \vee \mathtt{predicat}(x)) \\
&\equiv \exists x.\mathtt{sujet}(x) \wedge \neg\mathtt{predicat}(x)
\end{aligned}
$$

Dualité de la traduction des quantificateurs

$$\begin{aligned}\neg(\forall x.\mathtt{sujet}(x) \Rightarrow \mathtt{predicat}(x)) &\equiv \exists x.\neg(\mathtt{sujet}(x) \Rightarrow \mathtt{predicat}(x))\\ &\equiv \exists x.\neg(\neg\mathtt{sujet}(x) \vee \mathtt{predicat}(x))\\ &\equiv \exists x.\mathtt{sujet}(x) \wedge \neg\mathtt{predicat}(x)\end{aligned}$$

Exemple

Il n'est pas vrai que tous les étudiants apprennent l'anglais

Dualité de la traduction des quantificateurs

$$\begin{aligned}
\neg(\forall x.\mathtt{sujet}(x) \Rightarrow \mathtt{predicat}(x)) &\equiv \exists x.\neg(\mathtt{sujet}(x) \Rightarrow \mathtt{predicat}(x)) \\
&\equiv \exists x.\neg(\neg\mathtt{sujet}(x) \vee \mathtt{predicat}(x)) \\
&\equiv \exists x.\mathtt{sujet}(x) \wedge \neg\mathtt{predicat}(x)
\end{aligned}$$

Exemple

Il n'est pas vrai que tous les étudiants apprennent l'anglais

$$\neg(\forall x.\mathtt{student}(x) \Rightarrow \mathtt{learn}(x, \mathtt{english}))$$
$$\equiv$$
$$\exists x.\mathtt{student}(x) \wedge \neg\mathtt{learn}(x, \mathtt{english})$$

Dualité de la traduction des quantificateurs

$$\begin{aligned}
\neg(\forall x.\mathtt{sujet}(x) \Rightarrow \mathtt{predicat}(x)) &\equiv \exists x.\neg(\mathtt{sujet}(x) \Rightarrow \mathtt{predicat}(x)) \\
&\equiv \exists x.\neg(\neg\mathtt{sujet}(x) \vee \mathtt{predicat}(x)) \\
&\equiv \exists x.\mathtt{sujet}(x) \wedge \neg\mathtt{predicat}(x)
\end{aligned}$$

Exemple

Il n'est pas vrai que tous les étudiants apprennent l'anglais

$$\begin{gathered}
\neg(\forall x.\mathtt{student}(x) \Rightarrow \mathtt{learn}(x, \mathtt{english})) \\
\equiv \\
\exists x.\mathtt{student}(x) \wedge \neg\mathtt{learn}(x, \mathtt{english})
\end{gathered}$$

Un étudiant n'apprend pas l'anglais

Dualité de la traduction des quantificateurs

$$
\begin{aligned}
\neg(\exists x.\texttt{sujet}(x) \wedge \texttt{predicat}(x)) &\equiv \forall x.\neg(\texttt{sujet}(x) \wedge \texttt{predicat}(x)) \\
&\equiv \forall x.\neg\texttt{sujet}(x) \vee \neg\texttt{predicat}(x) \\
&\equiv \forall x.\texttt{sujet}(x) \Rightarrow \neg\texttt{predicat}(x)
\end{aligned}
$$

Dualité de la traduction des quantificateurs

$$\begin{aligned}
\neg(\exists x.\mathtt{sujet}(x) \wedge \mathtt{predicat}(x)) &\equiv \forall x.\neg(\mathtt{sujet}(x) \wedge \mathtt{predicat}(x)) \\
&\equiv \forall x.\neg\mathtt{sujet}(x) \vee \neg\mathtt{predicat}(x) \\
&\equiv \forall x.\mathtt{sujet}(x) \Rightarrow \neg\mathtt{predicat}(x)
\end{aligned}$$

Exemple

Il n'est pas vrai qu'il y a étudiant qui apprend l'anglais et l 'allemand

Dualité de la traduction des quantificateurs

$$\begin{aligned}\neg(\exists x.\mathtt{sujet}(x) \wedge \mathtt{predicat}(x)) &\equiv \forall x.\neg(\mathtt{sujet}(x) \wedge \mathtt{predicat}(x)) \\ &\equiv \forall x.\neg\mathtt{sujet}(x) \vee \neg\mathtt{predicat}(x) \\ &\equiv \forall x.\mathtt{sujet}(x) \Rightarrow \neg\mathtt{predicat}(x)\end{aligned}$$

Exemple

Il n'est pas vrai qu'il y a étudiant qui apprend l'anglais et l 'allemand

$$\begin{gathered}\neg(\exists x.\mathtt{student}(x) \wedge \mathtt{learn}(x, \mathtt{english}) \wedge \mathtt{learn}(x, \mathtt{allemand})) \\ \equiv \\ \forall x.\mathtt{student}(x) \Rightarrow (\neg\mathtt{learn}(x, \mathtt{english}) \vee \neg\mathtt{learn}(x, \mathtt{allemand}))\end{gathered}$$

Dualité de la traduction des quantificateurs

$$\begin{aligned}
\neg(\exists x.\mathtt{sujet}(x) \wedge \mathtt{predicat}(x)) &\equiv \forall x.\neg(\mathtt{sujet}(x) \wedge \mathtt{predicat}(x)) \\
&\equiv \forall x.\neg\mathtt{sujet}(x) \vee \neg\mathtt{predicat}(x) \\
&\equiv \forall x.\mathtt{sujet}(x) \Rightarrow \neg\mathtt{predicat}(x)
\end{aligned}$$

Exemple

Il n'est pas vrai qu'il y a étudiant qui apprend l'anglais et l 'allemand

$$\neg(\exists x.\mathtt{student}(x) \wedge \mathtt{learn}(x, \mathtt{english}) \wedge \mathtt{learn}(x, \mathtt{allemand}))$$
$$\equiv$$
$$\forall x.\mathtt{student}(x) \Rightarrow (\neg\mathtt{learn}(x, \mathtt{english}) \vee \neg\mathtt{learn}(x, \mathtt{allemand}))$$

Un étudiant n'apprend pas l'anglais ou n'apprend pas l'allemand

Pour aller plus loin : la sémantique de Montague

- La linguistique et la philosophie se sont intéressés à la relation entre texte et sens.

Pour aller plus loin : la sémantique de Montague

- La linguistique et la philosophie se sont intéressés à la relation entre texte et sens.
- Au début des années 70, Richard Montague publie une série d'articles :
  - English as a formal Language (1970),
  - Universal Grammar (1970),
  - Pragmatics and intentional logic (1970),
  - The proper treatment of quantification in ordinary English (1974)

  qui jettent les bases de la *sémantique formelle* : une théorie qui cherche à associer aux énoncés en *langage naturel* leurs *conditions de vérité*, i.e. une formule qui contraint *les mondes possibles* dans laquelle ces énoncés sont vrais.

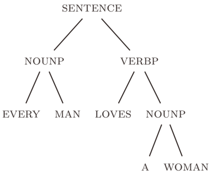
Pour aller plus loin : la sémantique de Montague

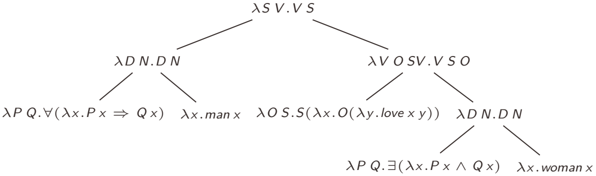
- La linguistique et la philosophie se sont intéressés à la relation entre texte et sens.
- Au début des années 70, Richard Montague publie une série d'articles :
  - English as a formal Language (1970),
  - Universal Grammar (1970),
  - Pragmatics and intentional logic (1970),
  - The proper treatment of quantification in ordinary English (1974)

  qui jettent les bases de la *sémantique formelle* : une théorie qui cherche à associer aux énoncés en *langage naturel* leurs *conditions de vérité*, i.e. une formule qui contraint *les mondes possibles* dans laquelle ces énoncés sont vrais.
- Ces méthodes permettent une traduction systématique de fragments importants de toutes les langues vers la logique. C'est un domaine de recherche actif qui étend sans cesse ces fragments. La mise en pratique se heurte néanmoins à deux grandes difficulté :
  - la très grande ambiguïté du langage,
  - la complexité de la manipulation des énoncés logiques,

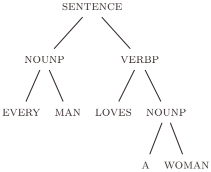
  L'utilisation pratique de ces méthodes se fait désormais en lien étroit avec les méthodes d'apprentissage automatique.

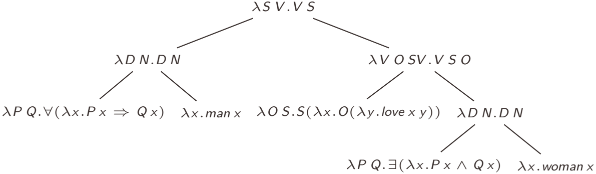
Interprétation sémantique d'une structure grammaticale





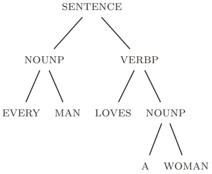
Interprétation sémantique d'une structure grammaticale

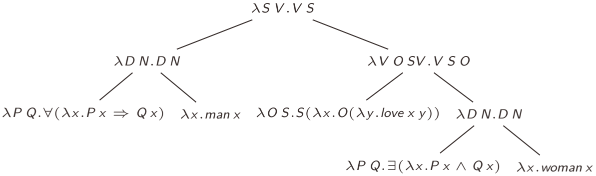




$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$

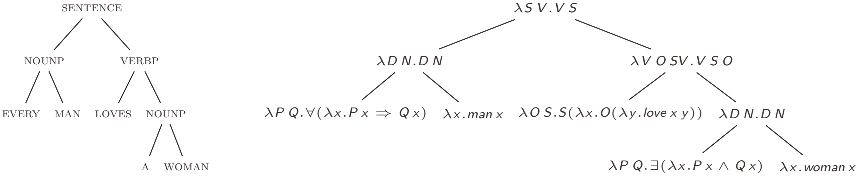
Interprétation sémantique d'une structure grammaticale





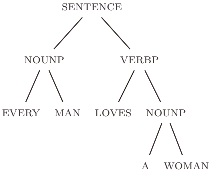
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda P\,Q.\exists(\lambda z.P\,z \wedge Q\,z)(\lambda x.woman\,x)$

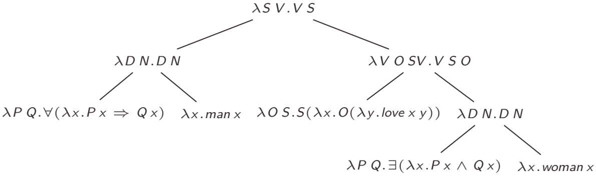
Interprétation sémantique d'une structure grammaticale



$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda P\, Q. \exists(\lambda z. P\, z \wedge Q\, z)(\lambda x. woman\, x)$

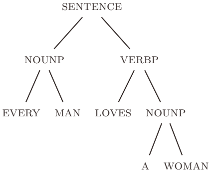
Interprétation sémantique d'une structure grammaticale

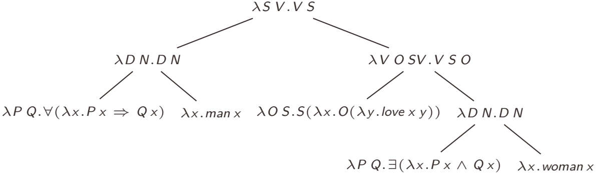




$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.(\lambda x.woman\,x)\,z \wedge Q\,z)$

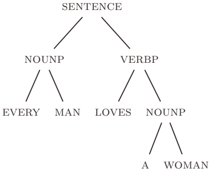
Interprétation sémantique d'une structure grammaticale

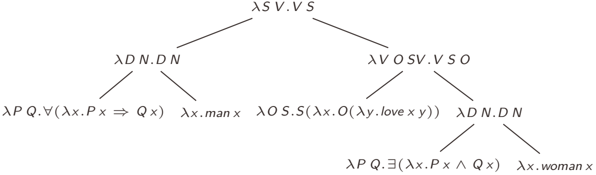




$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.(\lambda x.\mathit{woman}\,x)\,z \wedge Q\,z)$

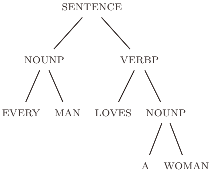
Interprétation sémantique d'une structure grammaticale

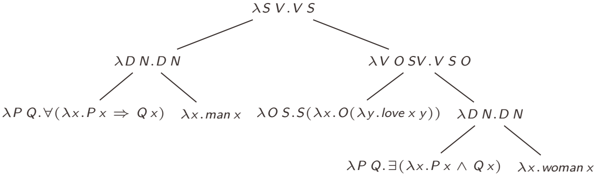




$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q. \exists(\lambda z. woman\, z \wedge Q\, z)$

Interprétation sémantique d'une structure grammaticale

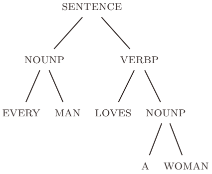


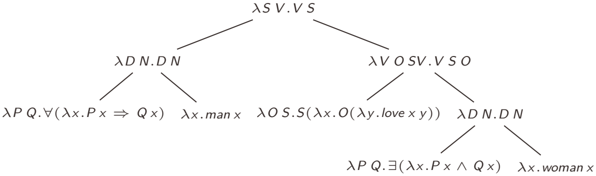


$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q. \exists(\lambda z. woman\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q. \forall(\lambda u. man\, u \Rightarrow Q\, u)$

Interprétation sémantique d'une structure grammaticale



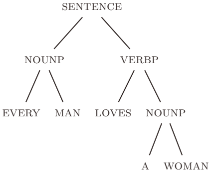


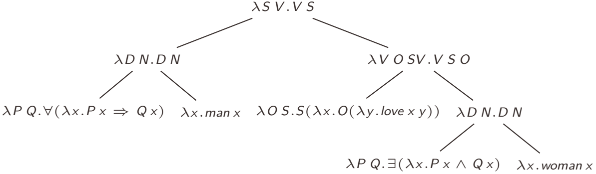
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\,z \wedge Q\,z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\,u \Rightarrow Q\,u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$

Interprétation sémantique d'une structure grammaticale



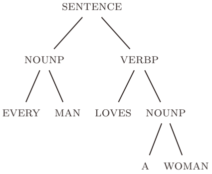


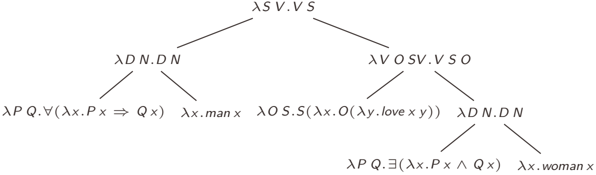
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\,z \wedge Q\,z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\,u \Rightarrow Q\,u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda O\,S.S(\lambda x.O(\lambda y.love\,x\,y))(\lambda P.\exists(\lambda z.woman\,z \wedge P\,z))$

Interprétation sémantique d'une structure grammaticale



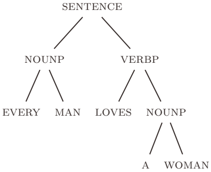


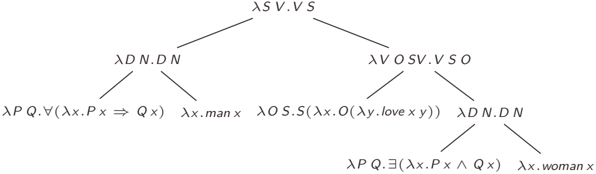
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda O\, S.S(\lambda x.O(\lambda y.love\, x\, y))(\lambda P.\exists(\lambda z.woman\, z \wedge P\, z))$

Interprétation sémantique d'une structure grammaticale



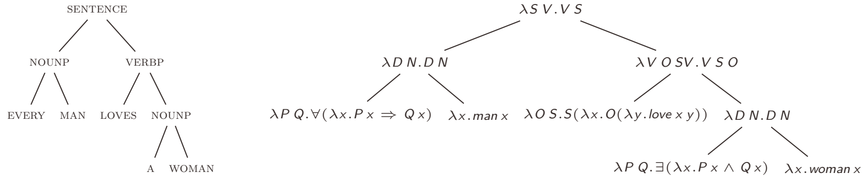


$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\,z \wedge Q\,z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\,u \Rightarrow Q\,u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda S.S(\lambda x.(\lambda P.\exists(\lambda z.(\lambda z.woman\,z \wedge P\,z)))(\lambda y.love\,x\,y))$

Interprétation sémantique d'une structure grammaticale

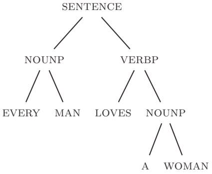


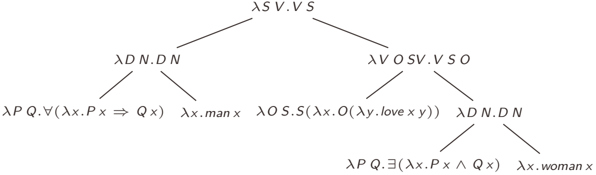
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda S.S(\lambda x.(\lambda P.\exists(\lambda z.(\lambda z.woman\, z \wedge P\, z)))(\lambda y.love\, x\, y))$

Interprétation sémantique d'une structure grammaticale



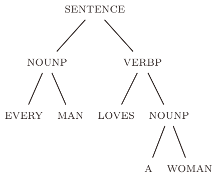


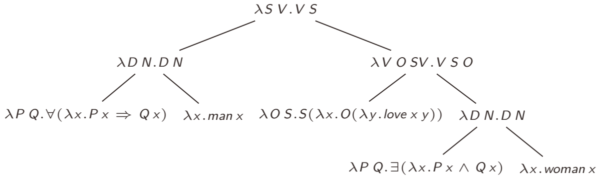
$$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) = \\ \lambda Q.\exists(\lambda z.woman\,z \wedge Q\,z)$$

$$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) = \\ \lambda Q.\forall(\lambda u.man\,u \Rightarrow Q\,u)$$

$$\mathcal{L}_{sem}(\text{VERBP LOVES }(\text{NOUNP A WOMAN})) = \\ \lambda S.S(\lambda x.\exists(\lambda z.woman\,z \wedge (\lambda y.love\,x\,y)\,z))$$

Interprétation sémantique d'une structure grammaticale



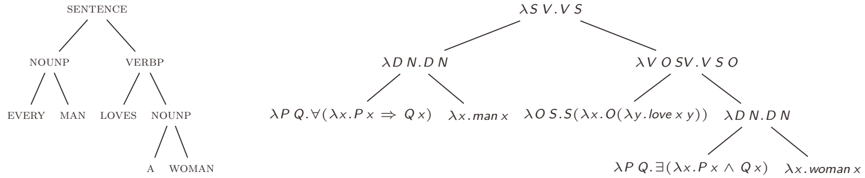


$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q. \exists(\lambda z. woman\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q. \forall(\lambda u. man\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda S. S(\lambda x. \exists(\lambda z. woman\, z \wedge (\lambda y. love\, x\, y)\, z))$

Interprétation sémantique d'une structure grammaticale

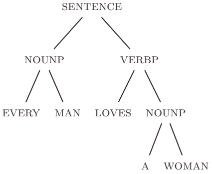


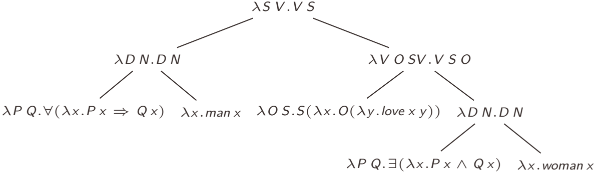
$$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) = \lambda Q.\exists(\lambda z.woman\ z \wedge Q\ z)$$

$$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) = \lambda Q.\forall(\lambda u.man\ u \Rightarrow Q\ u)$$

$$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) = \lambda S.S(\lambda x.\exists(\lambda z.woman\ z \wedge love\ x\ z))$$

Interprétation sémantique d'une structure grammaticale





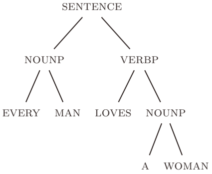
$\mathcal{L}_{sem}(\textsc{nounp a woman}) =$
$\lambda Q.\exists(\lambda z.woman\, z \wedge Q\, z)$

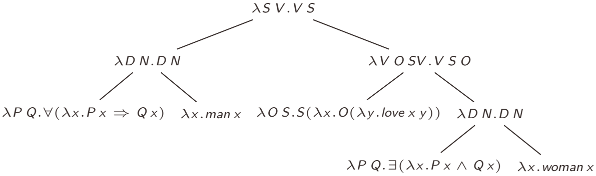
$\mathcal{L}_{sem}(\textsc{nounp every man}) =$
$\lambda Q.\forall(\lambda u.man\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\textsc{verbp loves}\ (\textsc{nounp a woman})) =$
$\lambda S.S(\lambda x.\exists(\lambda z.woman\, z \wedge love\, x\, z))$

$\mathcal{L}_{sem}(\textsc{sentence}(\textsc{nounp every man})(\textsc{verbp loves}\ (\textsc{nounp a woman}))) =$

Interprétation sémantique d'une structure grammaticale





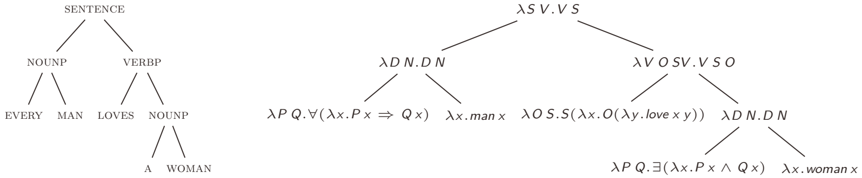
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q . \exists(\lambda z . woman\, z \wedge Q\, z)$

$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q . \forall(\lambda u . man\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda S . S(\lambda x . \exists(\lambda z . woman\, z \wedge love\, x\, z))$

$\mathcal{L}_{sem}(\text{SENTENCE(NOUNP EVERY MAN)(VERBP LOVES (NOUNP A WOMAN)))} =$
$(\lambda S . S(\lambda x . \exists(\lambda z . woman\, z \wedge love\, x\, z)))(\lambda Q . \forall(\lambda u . man\, u \Rightarrow Q\, u))$

Interprétation sémantique d'une structure grammaticale



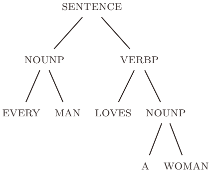
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q. \exists(\lambda z. woman\, z \wedge Q\, z)$

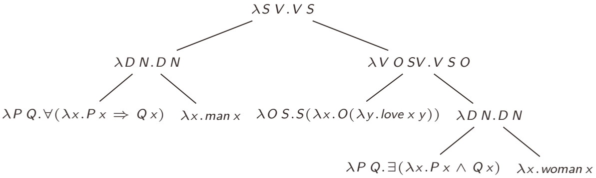
$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q. \forall(\lambda u. man\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda S. S(\lambda x. \exists(\lambda z. woman\, z \wedge love\, x\, z))$

$\mathcal{L}_{sem}(\text{SENTENCE(NOUNP EVERY MAN)(VERBP LOVES (NOUNP A WOMAN)))} =$
$(\lambda S. S(\lambda x. \exists(\lambda z. woman\, z \wedge love\, x\, z)))(\lambda Q. \forall(\lambda u. man\, u \Rightarrow Q\, u))$

Interprétation sémantique d'une structure grammaticale





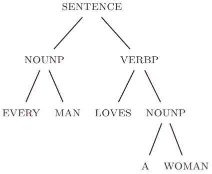
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\, z \wedge Q\, z)$

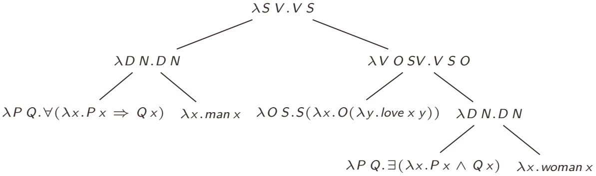
$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda S.S(\lambda x.\exists(\lambda z.woman\, z \wedge love\, x\, z))$

$\mathcal{L}_{sem}(\text{SENTENCE(NOUNP EVERY MAN)(VERBP LOVES (NOUNP A WOMAN))}) =$
$(\lambda Q.\forall(\lambda u.man\, u \Rightarrow Q\, u))(\lambda x.(\exists(\lambda z.woman\, z \wedge love\, x\, z)))$

Interprétation sémantique d'une structure grammaticale





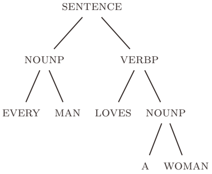
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\,z \wedge Q\,z)$

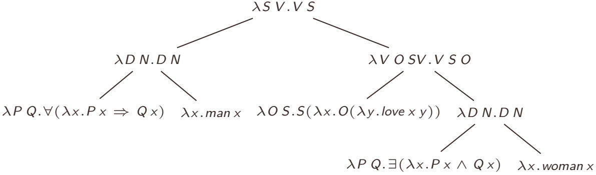
$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\,u \Rightarrow Q\,u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda S.S(\lambda x.\exists(\lambda z.woman\,z \wedge love\,x\,z))$

$\mathcal{L}_{sem}(\text{SENTENCE(NOUNP EVERY MAN)(VERBP LOVES (NOUNP A WOMAN)))} =$
$(\lambda Q.\forall(\lambda u.man\,u \Rightarrow Q\,u))(\lambda x.(\exists(\lambda z.woman\,z \wedge love\,x\,z)))$

Interprétation sémantique d'une structure grammaticale





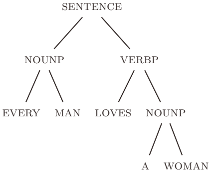
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q.\exists(\lambda z.woman\, z \wedge Q\, z)$

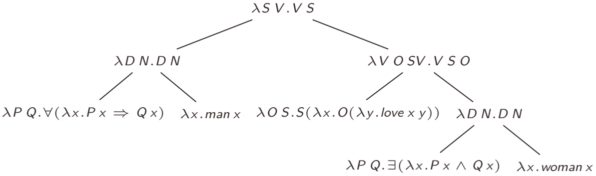
$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q.\forall(\lambda u.man\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda S.S(\lambda x.\exists(\lambda z.woman\, z \wedge love\, x\, z))$

$\mathcal{L}_{sem}(\text{SENTENCE(NOUNP EVERY MAN)(VERBP LOVES (NOUNP A WOMAN)))} =$
$\forall(\lambda u.man\, u \Rightarrow (\lambda x.\exists(\lambda z.woman\, z \wedge love\, x\, z))\, u)$

Interprétation sémantique d'une structure grammaticale





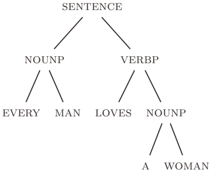
$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) =$
$\lambda Q. \exists(\lambda z. woman\, z \wedge Q\, z)$

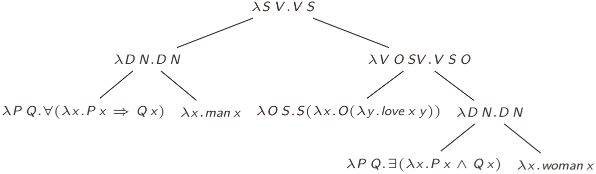
$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) =$
$\lambda Q. \forall(\lambda u. man\, u \Rightarrow Q\, u)$

$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) =$
$\lambda S. S(\lambda x. \exists(\lambda z. woman\, z \wedge love\, x\, z))$

$\mathcal{L}_{sem}(\text{SENTENCE(NOUNP EVERY MAN)(VERBP LOVES (NOUNP A WOMAN)))} =$
$\forall(\lambda u. man\, u \Rightarrow (\lambda x. \exists(\lambda z. woman\, z \wedge love\, x\, z))\, u)$

Interprétation sémantique d'une structure grammaticale





$$\mathcal{L}_{sem}(\text{NOUNP A WOMAN}) = \lambda Q.\exists(\lambda z.woman\, z \wedge Q\, z)$$

$$\mathcal{L}_{sem}(\text{NOUNP EVERY MAN}) = \lambda Q.\forall(\lambda u.man\, u \Rightarrow Q\, u)$$

$$\mathcal{L}_{sem}(\text{VERBP LOVES (NOUNP A WOMAN)}) = \lambda S.S(\lambda x.\exists(\lambda z.woman\, z \wedge love\, x\, z))$$

$$\mathcal{L}_{sem}(\text{SENTENCE(NOUNP EVERY MAN)(VERBP LOVES (NOUNP A WOMAN))}) = \forall(\lambda u.man\, u \Rightarrow \exists(\lambda z.woman\, z \wedge love\, u\, z))$$

Portée des quantificateurs : sujet ou objet

La phrase *every man loves a woman* a deux sens possibles :

- $\forall(\lambda u.\mathit{man}\, u \Rightarrow \exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, u\, z))$

Portée des quantificateurs : sujet ou objet

La phrase *every man loves a woman* a deux sens possibles :

- $\forall(\lambda u.\mathit{man}\, u \Rightarrow \exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, u\, z))$
- $\exists(\lambda z.\mathit{woman}\, z \wedge \forall(\lambda u.\mathit{man}\, u \Rightarrow \mathit{love}\, u\, z))$

Portée des quantificateurs : sujet ou objet

La phrase *every man loves a woman* a deux sens possibles :

- $\forall(\lambda u.\mathit{man}\, u \Rightarrow \exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, u\, z))$
- $\exists(\lambda z.\mathit{woman}\, z \wedge \forall(\lambda u.\mathit{man}\, u \Rightarrow \mathit{love}\, u\, z))$

Le second sens peut être obtenu en utilisant :

$$\lambda O\, S.O(\lambda y.S(\lambda x.\mathit{love}\, x\, y))$$

comme interprétation de LOVE au lieu de :

$$\lambda O\, S.S(\lambda x.O(\lambda y.\mathit{love}\, x\, y))$$

Portée des quantificateurs : sujet ou objet

La phrase *every man loves a woman* a deux sens possibles :

- $\forall(\lambda u.\mathit{man}\, u \Rightarrow \exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, u\, z))$
- $\exists(\lambda z.\mathit{woman}\, z \wedge \forall(\lambda u.\mathit{man}\, u \Rightarrow \mathit{love}\, u\, z))$

Le second sens peut être obtenu en utilisant :

$$\lambda O\, S.O(\lambda y.S(\lambda x.\mathit{love}\, x\, y))$$

comme interprétation de LOVE au lieu de :

$$\lambda O\, S.S(\lambda x.O(\lambda y.\mathit{love}\, x\, y))$$

Cette technique permet également de modéliser d'autres phénomènes (modalité, temps,...). Par exemple, l'ambiguïté de Re/de Dicto :

- *Jean croit que quelqu'un manipule les médias.* :

Portée des quantificateurs : sujet ou objet

La phrase *every man loves a woman* a deux sens possibles :

- $\forall(\lambda u.\mathit{man}\, u \Rightarrow \exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, u\, z))$
- $\exists(\lambda z.\mathit{woman}\, z \wedge \forall(\lambda u.\mathit{man}\, u \Rightarrow \mathit{love}\, u\, z))$

Le second sens peut être obtenu en utilisant :

$$\lambda O\, S.O(\lambda y.S(\lambda x.\mathit{love}\, x\, y))$$

comme interprétation de LOVE au lieu de :

$$\lambda O\, S.S(\lambda x.O(\lambda y.\mathit{love}\, x\, y))$$

Cette technique permet également de modéliser d'autres phénomènes (modalité, temps,...). Par exemple, l'ambiguïté de Re/de Dicto :

- *Jean croit que quelqu'un manipule les médias.* :
  - Jean croit qu'il existe une personne qui manipule les médias (il ne sait pas de qui il s'agit),

Portée des quantificateurs : sujet ou objet

La phrase *every man loves a woman* a deux sens possibles :

- $\forall(\lambda u.\mathit{man}\, u \Rightarrow \exists(\lambda z.\mathit{woman}\, z \wedge \mathit{love}\, u\, z))$
- $\exists(\lambda z.\mathit{woman}\, z \wedge \forall(\lambda u.\mathit{man}\, u \Rightarrow \mathit{love}\, u\, z))$

Le second sens peut être obtenu en utilisant :

$$\lambda O\, S.O(\lambda y.S(\lambda x.\mathit{love}\, x\, y))$$

comme interprétation de LOVE au lieu de :

$$\lambda O\, S.S(\lambda x.O(\lambda y.\mathit{love}\, x\, y))$$

Cette technique permet également de modéliser d'autres phénomènes (modalité, temps,. . .). Par exemple, l'ambiguïté de Re/de Dicto :

- *Jean croit que quelqu'un manipule les médias.* :
  - Jean croit qu'il existe une personne qui manipule les médias (il ne sait pas de qui il s'agit),
  - Il existe quelqu'un (e.g. Bob) dont Jean croit qu'il manipule les médias.

Plan

Modèle et implémentation

$$\begin{array}{c}\text{Modèle du problème}\\ \Big\downarrow\ \text{implémentation}\\ \text{code du logiciel}\end{array}$$

Modèle et implémentation

$$\begin{array}{c}\text{Modèle du problème}\\ \Big\downarrow \text{implémentation}\\ \text{code du logiciel}\end{array}$$

Objectif

On a du code censé implémenter notre modèle et on veut montrer qu'il fonctionne correctement.

Modèle et implémentation

$$\begin{array}{c}\text{Modèle du problème}\\ \Big\downarrow{\scriptstyle\text{implémentation}}\\ \text{code du logiciel}\end{array}$$

Objectif

On a du code censé implémenter notre modèle et on veut montrer qu'il fonctionne correctement.

Principe

Nous allons voir le code comme définissant une fonction sur des configurations. (C'est en général un relation afin de laisser la possibilité au compilateur de réaliser des optimisation.)

Le langage while

Nous allons travailler sur un sous-ensemble de python n'utilisant que :

- pour valeurs :
  - les booléens,
  - les nombres,
- pour constructions :
  - les affectations de variables,
  - les sequences d'instructions,
  - les conditionnelles,
  - les boucles while.

Le langage while

Nous allons travailler sur un sous-ensemble de python n'utilisant que :

- pour valeurs :
  - les booléens,
  - les nombres,
- pour constructions :
  - les affectations de variables,
  - les sequences d'instructions,
  - les conditionnelles,
  - les boucles while.

Nous nous interdisons les listes, les objets, les fonctions, ...

Les environnements : une notion de configuration

Environnements et commandes

- Un environnement (même notion que la valuation) est une fonction partielle à support fini des variables dans les valeurs. Il représentent un *état de la mémoire d'un programme*.
- Les commandes transforment les environnements.

Les environnements : une notion de configuration

Environnements et commandes

- Un environnement (même notion que la valuation) est une fonction partielle à support fini des variables dans les valeurs. Il représentent un *état de la mémoire d'un programme*.
- Les commandes transforment les environnements.

$$\begin{array}{l} x \leftarrow \text{True} \\ y \leftarrow 4 \\ z \leftarrow \text{-45} \end{array} \xrightarrow{\quad y=z+y*2 \quad} \begin{array}{l} x \leftarrow \text{True} \\ y \leftarrow 37 \\ z \leftarrow \text{-45} \end{array}$$

Les environnements : une notion de configuration

Environnements et commandes

- Un environnement (même notion que la valuation) est une fonction partielle à support fini des variables dans les valeurs. Il représentent un *état de la mémoire d'un programme*.
- Les commandes transforment les environnements.

$$\begin{array}{l} \mathrm{x} \leftarrow \mathrm{True} \\ \mathrm{y} \leftarrow 4 \\ \mathrm{z} \leftarrow \text{-45} \end{array} \xrightarrow{\quad \text{y=z+y*2} \quad} \begin{array}{l} \mathrm{x} \leftarrow \mathrm{True} \\ \mathrm{y} \leftarrow 37 \\ \mathrm{z} \leftarrow \text{-45} \end{array}$$

Remarque

Le debugger de Thonny permet d'observer facilement l'action des commandes sur les environnements.

Les environnements : une notion de configuration

Environnements et commandes

- Un environnement (même notion que la valuation) est une fonction partielle à support fini des variables dans les valeurs. Il représentent un *état de la mémoire d'un programme*.
- Les commandes transforment les environnements.

$$\begin{array}{l} \mathrm{x} \leftarrow \mathrm{True} \\ \mathrm{y} \leftarrow 4 \\ \mathrm{z} \leftarrow \text{-}45 \end{array} \xrightarrow{\quad \mathrm{y{=}z{+}y{*}2} \quad} \begin{array}{l} \mathrm{x} \leftarrow \mathrm{True} \\ \mathrm{y} \leftarrow 37 \\ \mathrm{z} \leftarrow \text{-}45 \end{array}$$

Remarque

Le debugger de Thonny permet d'observer facilement l'action des commandes sur les environnements.

Sémantique opérationnelle

C'est la description (formelle) de l'action de chaque commande (du code) sur les configurations.

Logique et environnements

Variables et propriétés sur les environnements

Si on fixe $x_1, \ldots, x_n$ des variables, un formule logique $\varphi(x_1, \ldots, x_n)$, est vue comme une propriété sur les environnements qui assignent des valeurs aux variables x_1, ..., x_n.

- Si E est un environnement qui *satisfait* $\varphi(x_1, \ldots, x_n)$ on note :

$$E \models \varphi(x_1, \ldots, x_n)$$

- Si E est un environnement qui *ne satisfait pas* $\varphi(x_1, \ldots, x_n)$ on note :

$$E \not\models \varphi(x_1, \ldots, x_n)$$

Logique et environnements

Variables et propriétés sur les environnements

Si on fixe $x_1, \ldots, x_n$ des variables, un formule logique $\varphi(x_1, \ldots, x_n)$, est vue comme une propriété sur les environnements qui assignent des valeurs aux variables x_1, ..., x_n.

- Si E est un environnement qui *satisfait* $\varphi(x_1, \ldots, x_n)$ on note :

$$E \models \varphi(x_1, \ldots, x_n)$$

- Si E est un environnement qui *ne satisfait pas* $\varphi(x_1, \ldots, x_n)$ on note :

$$E \not\models \varphi(x_1, \ldots, x_n)$$

x et y ne sont pas premiers entre eux

Par exemple :

$$\varphi(x, y) = \exists z. \neg z = 1 \wedge (\exists d_1 . y = d_1 * z) \wedge (\exists d_2 . x = d_2 * z)$$

désigne les environnements qui assignent à la variable x et à la variable y des nombres qui ont un diviseur commun différent de 1 (i.e. qui ne sont pas premiers entre eux).

Logique et environnements

Variables et propriétés sur les environnements

Si on fixe $x_1, \ldots, x_n$ des variables, un formule logique $\varphi(x_1, \ldots, x_n)$, est vue comme une propriété sur les environnements qui assignent des valeurs aux variables x_1, ..., x_n.

- Si E est un environnement qui *satisfait* $\varphi(x_1, \ldots, x_n)$ on note :

$$E \models \varphi(x_1, \ldots, x_n)$$

- Si E est un environnement qui *ne satisfait pas* $\varphi(x_1, \ldots, x_n)$ on note :

$$E \not\models \varphi(x_1, \ldots, x_n)$$

x et y ne sont pas premiers entre eux

Par exemple :

$$\varphi(x, y) = \exists z. \neg z = 1 \wedge (\exists d_1 . y = d_1 * z) \wedge (\exists d_2 . x = d_2 * z)$$

désigne les environnements qui assignent à la variable x et à la variable y des nombres qui ont un diviseur commun différent de 1 (i.e. qui ne sont pas premiers entre eux).

$$E_1 = x \leftarrow 10, y \leftarrow 14$$

est un tel environnement : $E_1 \models \varphi(x, y)$

$$E_2 = x \leftarrow 32, y \leftarrow 15$$

n'est pas un tel environnement : $E_2 \not\models \varphi(x, y)$

Pré-conditions et post-conditions

Transformation d'un environnement par un programme

Etant donné un programme $\mathbf{p}$, et deux environnements E et E', on note

$$E \triangleright \mathbf{p} \triangleright E'$$

Le fait que $\mathbf{p}$ transforme l'environnement E en l'environnement E'.

Les paires de pré-condition et post-condition

Étant donné un programme $\mathbf{p}$ les formules φ et ψ sont appelées respectivement pré-condition et post-condition de $\mathbf{p}$ lorsque :

Pour tout E et E' tel que $E \triangleright \mathbf{p} \triangleright E'$, si $E \models \varphi$ alors $E' \models \psi$.

Pré-conditions et post-conditions

Transformation d'un environnement par un programme

Etant donné un programme **p**, et deux environnements E et E', on note

$$E \triangleright \mathbf{p} \triangleright E'$$

Le fait que **p** transforme l'environnement E en l'environnement E'.

Les paires de pré-condition et post-condition

Étant donné un programme **p** les formules φ et ψ sont appelées respectivement pré-condition et post-condition de **p** lorsque :

Pour tout E et E' tel que $E \triangleright \mathbf{p} \triangleright E'$, si $E \models \varphi$ alors $E' \models \psi$.

On appelle aussi (φ, ψ) paire de pré/post-conditions de **p** ce que l'on note :

$$\varphi \blacktriangleright \mathbf{p} \blacktriangleright \psi$$

Quelques exemples

Incrémentation

`x=x+1`

Il a pour paires de pré/post-conditions :

- $(x = 1, x = 2)$, i.e. $x = 1 \blacktriangleright x = x + 1 \blacktriangleright x = 2$
- $(x < 1, x < 2)$, i.e. $x < 1 \blacktriangleright x = x + 1 \blacktriangleright x < 2$
- $(x \neq 0, x \neq 1)$, i.e. $x \neq 0 \blacktriangleright x = x + 1 \blacktriangleright x \neq 1$

Quelques exemples

Incrémentation

```
x=x+1
```

Il a pour paires de pré/post-conditions :

- $(x = 1, x = 2)$, i.e. $x = 1 \blacktriangleright x = x + 1 \blacktriangleright x = 2$
- $(x < 1, x < 2)$, i.e. $x < 1 \blacktriangleright x = x + 1 \blacktriangleright x < 2$
- $(x \neq 0, x \neq 1)$, i.e. $x \neq 0 \blacktriangleright x = x + 1 \blacktriangleright x \neq 1$

Échange de valeur

Prenons le programme **p** suivant

```
z=x
x=y
y=z
```

Il a pour paires de pré/post-conditions :

- $(x = 4 \wedge y = 5, x = 5 \wedge y = 4)$, i.e.

$$x = 4 \wedge y = 5 \blacktriangleright \mathbf{p} \blacktriangleright x = 5 \wedge y = 4$$

- $(\exists v. x = v * v \wedge y = v * v * v, \exists v. y = v * v \wedge x = v * v * v \wedge z = x)$

$$\exists v. x = v * v \wedge y = v * v * v \blacktriangleright \mathbf{p} \blacktriangleright \exists v. y = v * v \wedge x = v * v * v \wedge z = x$$

Raisonnement et pré/post-conditions

Clôture par conjonction

Si (φ_1, ψ_1) et (φ_2, ψ_2) sont des paires de pré/post-conditions de **p**, alors $(\varphi_1 \wedge \varphi_2, \psi_1 \wedge \psi_2)$ est une paire de pré/post-conditions.

Raisonnement et pré/post-conditions

Clôture par conjonction

Si (φ_1, ψ_1) et (φ_2, ψ_2) sont des paires de pré/post-conditions de **p**, alors $(\varphi_1 \wedge \varphi_2, \psi_1 \wedge \psi_2)$ est une paire de pré/post-conditions.

Implication

Si (φ_1, ψ_1) est une paire de pré/post-conditions de **p**, et

- $\models \varphi_2 \Rightarrow \varphi_1$,
- $\models \psi_1 \Rightarrow \psi_2$,

alors (φ_2, ψ_2) est une paire de pré/post-conditions.

Invariant de boucle

Invariant

Si (φ, φ) est une paire de pré/post-conditions de **p**, alors φ est appelé invariant de **p**.

Invariant de boucle

Invariant

Si (φ, φ) est une paire de pré/post-conditions de **p**, alors φ est appelé invariant de **p**.

Invariant de boucle

Si $(c \wedge \varphi, \varphi)$ est une paire de pré/post-conditions de **p**, on appelle φ invariant de boucle de

while c : **p**

Démontrer un programme

Étant donné un programme **p**, pour démontrer qu'il calcule ce que l'on souhaite, il faut :

- spécifier avec la logique une propriété ψ de l'environnement que l'on veut obtenir,

Démontrer un programme

Étant donné un programme **p**, pour démontrer qu'il calcule ce que l'on souhaite, il faut :

- spécifier avec la logique une propriété ψ de l'environnement que l'on veut obtenir,
- spécifier avec une formule φ les environnements que le programme est censé transformer,

Nous allons voir comment cela peut être spécifier formellement.

Démontrer un programme

Étant donné un programme **p**, pour démontrer qu'il calcule ce que l'on souhaite, il faut :

- spécifier avec la logique une propriété ψ de l'environnement que l'on veut obtenir,
- spécifier avec une formule φ les environnements que le programme est censé transformer,
- montrer que φ ▶ **p** ▶ ψ.

Nous allons voir comment cela peut être spécifier formellement.

Plan

Sémantique opérationnelle : séquentialité et expression

Nous allons commencer par voir comment on évalue des **expressions**, i.e. constructions qui utilisent :

- des variables,
- des valeurs (nombres ou booléens),
- des opérateurs sur les nombres ($+$,*,/, $==$, $<=$,$<$,...),
- des opérateurs sur les booléens (or, and, not).

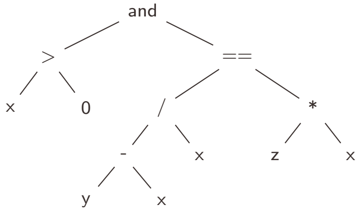
Sémantique opérationnelle : séquentialité et expression

Nous allons commencer par voir comment on évalue des **expressions**, i.e. constructions qui utilisent :

- des variables,
- des valeurs (nombres ou booléens),
- des opérateurs sur les nombres ($+$,*,/, $==$, $<=$,$<$,$\ldots$),
- des opérateurs sur les booléens (or, and, not).

Il faut penser les expressions comme des constructions hiérarchique :

$x > 0$ and $(y - x)/x == z * x$



Elles s'évaluent en descendant de gauche à droite : on dit que les opérations sont **séquentielles à gauche**.

Sémantique court-circuit des opérateurs logiques

Les opérateurs *and* et *or* peuvent n'évaluer que leur argument de gauche :

- si dans l'expression e_1 or e_2, e_1 s'évalue à True alors toute l'expression s'évalue à True quel que soit le comportement de e_2,
- si dans l'expression e_1 and e_2, e_1 s'évalue à False alors toute l'expression s'évalue à False quel que soit le comportement de e_2.

Sémantique court-circuit des opérateurs logiques

Les opérateurs *and* et *or* peuvent n'évaluer que leur argument de gauche :

- si dans l'expression e_1 or e_2, e_1 s'évalue à True alors toute l'expression s'évalue à True quel que soit le comportement de e_2,
- si dans l'expression e_1 and e_2, e_1 s'évalue à False alors toute l'expression s'évalue à False quel que soit le comportement de e_2.

Cela ne poserait pas de problème si les expressions s'évaluaient toujours à une valeur. Mais ce n'est pas toujours le cas :

- une expression peut émettre une exception (une erreur, e.g. division par 0, référence à une variable qui n'est pas affectée dans l'environnement),
- une expression (si on ajoute d'autres constructions que celles auxquelles nous nous sommes restreints) peut effectuer un effet de bord.

Sémantique court-circuit des opérateurs logiques

Les opérateurs *and* et *or* peuvent n'évaluer que leur argument de gauche :

- si dans l'expression e_1 or e_2, e_1 s'évalue à True alors toute l'expression s'évalue à True quel que soit le comportement de e_2,
- si dans l'expression e_1 and e_2, e_1 s'évalue à False alors toute l'expression s'évalue à False quel que soit le comportement de e_2.

Cela ne poserait pas de problème si les expressions s'évaluaient toujours à une valeur. Mais ce n'est pas toujours le cas :

- une expression peut émettre une exception (une erreur, e.g. division par 0, référence à une variable qui n'est pas affectée dans l'environnement),
- une expression (si on ajoute d'autres constructions que celles auxquelles nous nous sommes restreints) peut effectuer un effet de bord.

Remarque

On peut montrer que sans utiliser de parallélisme, il est impossible de construire un opérateur *or* commutatif qui renvoie True ssi l'un de ses arguments s'évalue à True même en présence d'erreur.

Sémantique opérationnelle : l'affectation

Affectation

L'instruction $x = e$ agit sur un environnement E de la façon suivante :

- si e s'évalue à la valeur v dans l'environnement E alors elle produit l'environnement E' tel que :

$$E'(y) = \begin{cases} v \text{ si } y = x \\ E(y) \text{ sinon} \end{cases}$$

- si e retourne une erreur, elle retourne cette erreur.

Sémantique opérationnelle : séquence

Séquence

L'instruction

$$\begin{array}{c}\mathbf{p}\\ \mathbf{q}\end{array}$$

agit sur un environnement E de la façon suivante :

- Si **p** transforme E en E', et **q** transforme E' en E'', alors la *séquence* "**p** suivi de **q**" transforme E en E''.
- Si **p** retourne une erreur à partir de E, la *séquence* "**p** suivi de **q**" retourne cette erreur.
- Si **p** ne retourne pas d'erreur à partir de E, mais que **q** retourne une erreur à partir de E', la *séquence* "**p** suivi de **q**" retourne cette erreur.

Sémantique opérationnelle : séquence

Séquence

L'instruction

$$\begin{array}{c}\mathbf{p}\\ \mathbf{q}\end{array}$$

agit sur un environnement E de la façon suivante :

- Si **p** transforme E en E', et **q** transforme E' en E'', alors la *séquence* "**p** suivi de **q**" transforme E en E''.
- Si **p** retourne une erreur à partir de E, la *séquence* "**p** suivi de **q**" retourne cette erreur.
- Si **p** ne retourne pas d'erreur à partir de E, mais que **q** retourne une erreur à partir de E', la *séquence* "**p** suivi de **q**" retourne cette erreur.

Notation pour la séquence

Dans la suite, par souci de concision, nous noterons $\mathbf{p};\mathbf{q}$ pour la séquence de **p** et de **q**.

Sémantique opérationnelle : branchement conditionnel

Conditionnelle

L'instruction

```
if      c :
        p
else :
        q
```

agit sur un environnement E de la façon suivante :

- si c s'évalue à True elle produit l'action sur E de **p**,
- si c s'évalue à False elle produit l'action sur E de **q**,
- si c retourne une erreur, elle retourne cette erreur.

Sémantique opérationnelle : boucle while

Boucle

L'instruction

```
while c :
      p
```

agit sur un environnement E de la façon suivante :

- si c s'évalue à True elle produit l'action sur E de **p** pour obtenir E', suivie de sa propre action sur E',
- si c s'évalue à False elle laisse E inchangé,
- si c retourne une erreur, elle retourne cette erreur.

Vers un système formel

Evaluation d'une expression

- Si e est une expression, on note $[\![e, E]\!]$, la valeur que prend e dans l'environnement E.
- On note `err` la valeur d'*erreur*, toute autre valeur sera notée $v, v_1, v_2 \ldots$

Rappel : action d'un programme sur un environnement

Etant donné un programme **p**, et deux environnements E et E', on note

$$E \triangleright \mathbf{p} \triangleright E'$$

Le fait que **p** transforme l'environnement E en l'environnement E'.

Vers un système formel

Evaluation d'une expression

- Si e est une expression, on note $\llbracket e, E \rrbracket$, la valeur que prend e dans l'environnement E.
- On note `err` la valeur d'*erreur*, toute autre valeur sera notée $v, v_1, v_2 \ldots$

Rappel : action d'un programme sur un environnement

Etant donné un programme **p**, et deux environnements E et E', on note

$$E \triangleright \mathbf{p} \triangleright E'$$

Le fait que **p** transforme l'environnement E en l'environnement E'.
Si **p** crée une erreur, on écrit $E \triangleright \mathbf{p} \triangleright \mathtt{err}$.

Système de déduction

Pour *calculer* l'action du code sur les environnements on utilise un système formel constitué de règles de la forme :

$$\frac{P_1 \quad P_2 \quad \dots \quad P_n}{C}$$

$P_1, \dots, P_n$ sont appelées prémisses et C est appelée conclusion.

Système de déduction

Pour *calculer* l'action du code sur les environnements on utilise un système formel constitué de règles de la forme :

$$\frac{P_1 \quad P_2 \quad \dots \quad P_n}{C}$$

$P_1, \dots, P_n$ sont appelées prémisses et C est appelée conclusion.
On construit un arbre avec ces règles pour utiliser des conclusions comme prémisses et chaîner les raisonnements.

Sémantique opérationnelle : système formel

$$\frac{[\![e, E]\!] = v}{E \triangleright x = e \triangleright E[x \leftarrow v]}$$

Sémantique opérationnelle : système formel

$$\frac{[\![e, E]\!] = v}{E \triangleright x = e \triangleright E[x \leftarrow v]} \qquad \frac{[\![e, E]\!] = \mathtt{err}}{E \triangleright x = e \triangleright \mathtt{err}}$$

Sémantique opérationnelle : système formel

$$\frac{[\![e, E]\!] = v}{E \triangleright x = e \triangleright E[x \leftarrow v]} \qquad \frac{[\![e, E]\!] = \mathtt{err}}{E \triangleright x = e \triangleright \mathtt{err}}$$

$$\frac{E \triangleright c_1 \triangleright E' \quad E' \triangleright c_2 \triangleright E''}{E \triangleright c_1 ; c_2 \triangleright E''}$$

Sémantique opérationnelle : système formel

$$\frac{[\![e, E]\!] = v}{E \triangleright x = e \triangleright E[x \leftarrow v]} \qquad \frac{[\![e, E]\!] = \texttt{err}}{E \triangleright x = e \triangleright \texttt{err}}$$

$$\frac{E \triangleright c_1 \triangleright E' \quad E' \triangleright c_2 \triangleright E''}{E \triangleright c_1 ; c_2 \triangleright E''} \qquad \frac{E \triangleright c_1 \triangleright \texttt{err}}{E \triangleright c_1 ; c_2 \triangleright \texttt{err}}$$

Sémantique opérationnelle : système formel

$$\frac{[\![e, E]\!] = v}{E \triangleright x = e \triangleright E[x \leftarrow v]} \qquad \frac{[\![e, E]\!] = \mathtt{err}}{E \triangleright x = e \triangleright \mathtt{err}}$$

$$\frac{E \triangleright c_1 \triangleright E' \quad E' \triangleright c_2 \triangleright E''}{E \triangleright c_1 ; c_2 \triangleright E''} \qquad \frac{E \triangleright c_1 \triangleright \mathtt{err}}{E \triangleright c_1 ; c_2 \triangleright \mathtt{err}}$$

$$\frac{[\![e, E]\!] = \mathit{True} \quad E \triangleright ct \triangleright E'}{E \triangleright \text{if } e : \ ct \text{ else} : \ ce \triangleright E'} \qquad \frac{[\![e, E]\!] = \mathit{False} \quad E \triangleright ce \triangleright E'}{E \triangleright \text{if } c : \ ct \text{ else} : \ ce \triangleright E'}$$

$$\frac{[\![e, E]\!] = \mathtt{err}}{E \triangleright \text{if } c : \ ct \text{ else} : \ ce \triangleright \mathtt{err}}$$

Sémantique opérationnelle : système formel

$$\frac{[\![e, E]\!] = v}{E \triangleright x = e \triangleright E[x \leftarrow v]} \qquad \frac{[\![e, E]\!] = \mathtt{err}}{E \triangleright x = e \triangleright \mathtt{err}}$$

$$\frac{E \triangleright c_1 \triangleright E' \quad E' \triangleright c_2 \triangleright E''}{E \triangleright c_1 ; c_2 \triangleright E''} \qquad \frac{E \triangleright c_1 \triangleright \mathtt{err}}{E \triangleright c_1 ; c_2 \triangleright \mathtt{err}}$$

$$\frac{[\![e, E]\!] = \mathit{True} \quad E \triangleright ct \triangleright E'}{E \triangleright \text{if } e : \ ct \text{ else} : \ ce \triangleright E'} \qquad \frac{[\![e, E]\!] = \mathit{False} \quad E \triangleright ce \triangleright E'}{E \triangleright \text{if } c : \ ct \text{ else} : \ ce \triangleright E'}$$

$$\frac{[\![e, E]\!] = \mathtt{err}}{E \triangleright \text{if } c : \ ct \text{ else} : \ ce \triangleright \mathtt{err}}$$

$$\frac{[\![e, E]\!] = \mathit{True} \quad E \triangleright cw \triangleright E' \quad E' \triangleright \text{while } e : \ cw \triangleright E''}{E \triangleright \text{while } e : \ cw \triangleright E''} \qquad \frac{[\![e, E]\!] = \mathit{False}}{E \triangleright \text{while } e : \ cw \triangleright E}$$

$$\frac{[\![e, E]\!] = \mathtt{err}}{E \triangleright \text{while } e : \ cw \triangleright \mathtt{err}}$$

Un exemple

Les règles d'inférence du transparent précédent sont utilisées pour construire des arbres de *dérivation* qui représent l'exécution du programme. On construit un arbre pour les *prémisses* de chaque règles (au-dessus de la barre horizontale), puis on utilise une règle pour déduire sa *conclusion* (sous la barre horizontale).

$$\dfrac{[\![x > 0, [x \leftarrow 5, y \leftarrow 3]]\!] = \mathit{True} \qquad \dfrac{[\![x + y, [x \leftarrow 5, y \leftarrow 3]]\!] = 8}{[x \leftarrow 5, y \leftarrow 3] \triangleright y = x + y \triangleright [x \leftarrow 5, y \leftarrow 8]}}{[x \leftarrow 5, y \leftarrow 3] \triangleright \text{if } x > 0 : y = x + y \text{ else} : y = 0 \triangleright [x \leftarrow 5, y \leftarrow 8]}$$

Plan

Logique de Hoare

Logique de Hoare

La formalisation de la sémantique opérationnelle permet d'associer des propriétés aux environnements et de propager ces propriétés à travers le programme.

Logique de Hoare

Logique de Hoare

La formalisation de la sémantique opérationnelle permet d'associer des propriétés aux environnements et de propager ces propriétés à travers le programme.

L'idée de Hoare est que les opérations sur les environnements peuvent être reflétées sur les propriétés logiques :

- Cela permet d'articuler le code avec les raisonnement,
- en revanche, il faut démontrer deux choses :
  - que les opérations ne produisent pas d'erreur,
  - que les boucles terminent.

Sémantique opérationnelle : système formel

$$\frac{\models \varphi' \Rightarrow \varphi \qquad \models \psi \Rightarrow \psi' \qquad \varphi \blacktriangleright c \blacktriangleright \psi}{\varphi' \blacktriangleright c \blacktriangleright \psi'}$$

Sémantique opérationnelle : système formel

$$\frac{\models \varphi' \Rightarrow \varphi \quad \models \psi \Rightarrow \psi' \quad \varphi \blacktriangleright c \blacktriangleright \psi}{\varphi' \blacktriangleright c \blacktriangleright \psi'}$$

$$\frac{}{\varphi(e/x) \blacktriangleright x = e \blacktriangleright \varphi}$$

Sémantique opérationnelle : système formel

$$\frac{\models \varphi' \Rightarrow \varphi \quad \models \psi \Rightarrow \psi' \quad \varphi \blacktriangleright c \blacktriangleright \psi}{\varphi' \blacktriangleright c \blacktriangleright \psi'}$$

$$\frac{}{\varphi(e/x) \blacktriangleright x = e \blacktriangleright \varphi} \qquad \frac{\varphi \blacktriangleright c_1 \blacktriangleright \psi \quad \psi \blacktriangleright c_2 \blacktriangleright \theta}{\varphi \blacktriangleright c_1 ; c_2 \blacktriangleright \theta}$$

Sémantique opérationnelle : système formel

$$\frac{\models \varphi' \Rightarrow \varphi \quad \models \psi \Rightarrow \psi' \quad \varphi \blacktriangleright c \blacktriangleright \psi}{\varphi' \blacktriangleright c \blacktriangleright \psi'}$$

$$\frac{}{\varphi(e/x) \blacktriangleright x = e \blacktriangleright \varphi} \qquad \frac{\varphi \blacktriangleright c_1 \blacktriangleright \psi \quad \psi \blacktriangleright c_2 \blacktriangleright \theta}{\varphi \blacktriangleright c_1 ; c_2 \blacktriangleright \theta}$$

$$\frac{\varphi \wedge c \blacktriangleright ct \blacktriangleright \psi \quad \varphi \wedge \neg c \blacktriangleright ce \blacktriangleright \psi}{\varphi \blacktriangleright \text{if } c : \ ct \text{ else} : \ ce \blacktriangleright \psi}$$

Sémantique opérationnelle : système formel

$$\frac{\models \varphi' \Rightarrow \varphi \quad \models \psi \Rightarrow \psi' \quad \varphi \blacktriangleright c \blacktriangleright \psi}{\varphi' \blacktriangleright c \blacktriangleright \psi'}$$

$$\frac{}{\varphi(e/x) \blacktriangleright x = e \blacktriangleright \varphi} \qquad \frac{\varphi \blacktriangleright c_1 \blacktriangleright \psi \quad \psi \blacktriangleright c_2 \blacktriangleright \theta}{\varphi \blacktriangleright c_1 ; c_2 \blacktriangleright \theta}$$

$$\frac{\varphi \wedge c \blacktriangleright ct \blacktriangleright \psi \quad \varphi \wedge \neg c \blacktriangleright ce \blacktriangleright \psi}{\varphi \blacktriangleright \text{if } c : \; ct \text{ else} : \; ce \blacktriangleright \psi}$$

$$\frac{\varphi \wedge c \blacktriangleright cw \blacktriangleright \varphi}{\varphi \blacktriangleright \text{while } c : \; cw \blacktriangleright \varphi \wedge \neg c}$$

La notation $\varphi \blacktriangleright c \blacktriangleright \psi$ signifie que (φ, ψ) est une paire de pré-condition et post-condition du code c.
Ces règles s'utilisent de façon similaire à celles de la sémantique opérationnelle.

Sémantique opérationnelle : observation dans Thonny

L'intérêt principal de Thonny est que son debugger est très simple d'utilisation :

- il permet d'aquérir l'intuition de la sémantique opérationnelle, sans la définir formellement.
- Cette compréhension doit mener à des intuitions sur la façon dont les propriétés se propagent dans le programme, i.e. une intuition de la logique de Hoare.

Sémantique opérationnelle : les difficultés

- Nous nous sommes intéressés à un tout petit fragment de python.

Sémantique opérationnelle : les difficultés

- Nous nous sommes intéressés à un tout petit fragment de python.
- Lorsque l'on rajoute les fonctions les objets, les appels systèmes, les fonctions, la notion de configuration devient complexe. Il faut ajouter :
  - des piles (pour l'appel des fonctions),
  - la structure de la mémoire (tas – heap –),
  - une représentation pour les effets de bords sur les périphériques (écrans, réseaux, claviers,. . .)

Sémantique opérationnelle : les difficultés

- Nous nous sommes intéressés à un tout petit fragment de python.
- Lorsque l'on rajoute les fonctions les objets, les appels systèmes, les fonctions, la notion de configuration devient complexe. Il faut ajouter :
  - des piles (pour l'appel des fonctions),
  - la structure de la mémoire (tas – heap –),
  - une représentation pour les effets de bords sur les périphériques (écrans, réseaux, claviers,. . .)
- Le lien entre les abstractions usuelles et le code devient difficile. Les fonctions python (et c'est le cas dans quasiment tous les langages de programmation) ne sont pas des fonctions mathématiques :
  - évaluation stricte,
  - effets de bord.

Bilan

- Les objets mathématiques sont de lointaines approximations des objets informatiques.

Bilan

- Les objets mathématiques sont de lointaines approximations des objets informatiques.
- Pour bien programmer, il est nécessaire de se baser des abstractions mathématiques pour oublier des détails lourds et raisonner sur ce que l'on fait.

Bilan

- Les objets mathématiques sont de lointaines approximations des objets informatiques.
- Pour bien programmer, il est nécessaire de se baser des abstractions mathématiques pour oublier des détails lourds et raisonner sur ce que l'on fait.
- Pour être parfaitement rigoureux, il faut établir un lien entre le code et les abstractions. Le plus généralement, ce lien est laissé à l'intuition.

Bilan

- Les objets mathématiques sont de lointaines approximations des objets informatiques.
- Pour bien programmer, il est nécessaire de se baser des abstractions mathématiques pour oublier des détails lourds et raisonner sur ce que l'on fait.
- Pour être parfaitement rigoureux, il faut établir un lien entre le code et les abstractions. Le plus généralement, ce lien est laissé à l'intuition.
- Pour conserver la qualité de ce lien, il convient de respecter des recommandations stylistiques en programmant qui évitent les comportements étranges.