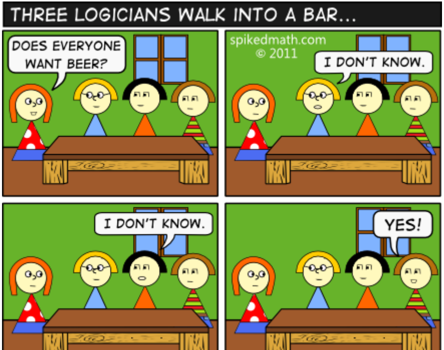
Formule logique : Syntaxe

Quentin Fortier

April 21, 2022



Formule logique : Définition

Définition

Soit V un ensemble (de **variables**).
L'ensemble des **formules logiques** sur V est défini inductivement :

- T et F sont des formules (Vrai et Faux)
- Toute variable $x \in V$ est une formule
- Si φ est une formule alors $\neg\varphi$ est une formule
- Si φ, ψ sont des formules alors $\varphi \wedge \psi$ (conjonction) et $\varphi \vee \psi$ (disjonction) sont des formules

Formule logique : Définition

Définition

Soit V un ensemble (de **variables**).
L'ensemble des **formules logiques** sur V est défini inductivement :

- T et F sont des formules (Vrai et Faux)
- Toute variable $x \in V$ est une formule
- Si φ est une formule alors $\neg\varphi$ est une formule
- Si φ, ψ sont des formules alors $\varphi \wedge \psi$ (conjonction) et $\varphi \vee \psi$ (disjonction) sont des formules

Ceci définit uniquement la **syntaxe** des formules logiques, sans leur donner de sens (ce qu'on appelle la **sémantique**).

Formule logique : Définition

Définition

Soit V un ensemble (de **variables**).
L'ensemble des **formules logiques** sur V est défini inductivement :

- T et F sont des formules (Vrai et Faux)
- Toute variable $x \in V$ est une formule
- Si φ est une formule alors $\neg\varphi$ est une formule
- Si φ, ψ sont des formules alors $\varphi \wedge \psi$ (conjonction) et $\varphi \vee \psi$ (disjonction) sont des formules

Ceci définit uniquement la **syntaxe** des formules logiques, sans leur donner de sens (ce qu'on appelle la **sémantique**).

Exemple : si x_1, $x_2 \in V$, $\neg(x_1 \vee x_2)$ et $\neg x_2 \wedge \neg x_2$ sont deux formules différentes.

Formule logique : En OCaml

```ocaml
type 'a formula =
    | T | F (* true, false *)
    | Var of 'a (* variable *)
    | Not of 'a formula
    | And of 'a formula * 'a formula
    | Or of 'a formula * 'a formula
```

Remarque : l'égalité (avec =) est automatiquement définie en OCaml.

Exercice

Écrire une fonction pour obtenir la liste des variables dans une formule logique.

Formule logique : BNF

On peut aussi utiliser une grammaire décrivant les formules logiques (BNF pour *Backus-Naur Form*) :

$$\begin{aligned}
<\text{formule}> ::= \; & T \mid F \mid <\text{variable}> \\
& \mid \neg <\text{formule}> \\
& \mid <\text{formule}> \vee <\text{formule}> \\
& \mid <\text{formule}> \wedge <\text{formule}>
\end{aligned}$$

Formule logique : BNF

On peut aussi utiliser une grammaire décrivant les formules logiques (BNF pour *Backus-Naur Form*) :

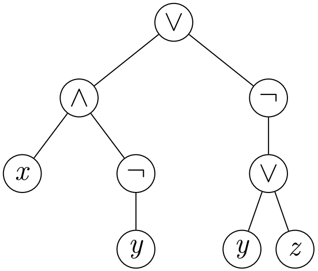
$$\begin{aligned}
<\text{formule}> ::= \; & T \mid F \mid <\text{variable}> \\
& \mid \neg <\text{formule}> \\
& \mid <\text{formule}> \vee <\text{formule}> \\
& \mid <\text{formule}> \wedge <\text{formule}>
\end{aligned}$$

Cette notation est très utilisée pour décrire la syntaxe d'un langage de programmation.

Exemple : OCaml, C, Python

Formule logique : Représentation par un arbre

On peut représenter une formule logique sous forme d'un arbre.
Par exemple, $(x \wedge \neg y) \vee \neg(y \vee z)$ est représenté par :

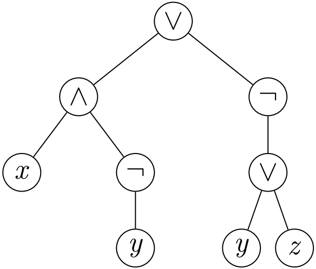



```
Or(
    And(x, Not y),
    Not (Or(y, z))
)
```

L'**arité** d'un connecteur logique est son nombre d'arguments (= nombre de fils dans l'arbre).
$\neg$ est d'arité 1 (unaire) et $\wedge, \vee$ sont d'arités 2 (binaire).

Formule logique : Représentation par un arbre

On peut représenter une formule logique sous forme d'un arbre.
Par exemple, $(x \wedge \neg y) \vee \neg(y \vee z)$ est représenté par :



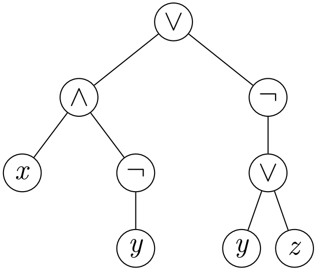

```
Or(
    And(x, Not y),
    Not (Or(y, z))
)
```

Exercice

Écrire des fonctions pour obtenir la taille (nombre de symboles) et la hauteur (de l'arbre associé) d'une formule logique.

Formule logique : Représentation par un arbre

On peut représenter une formule logique sous forme d'un arbre.
Par exemple, $(x \wedge \neg y) \vee \neg(y \vee z)$ est représenté par :



```
Or(
    And(x, Not y),
    Not (Or(y, z))
)
```

Exercice

Quelle est la taille d'une formule contenant b connecteurs binaires et n symboles de négations ?

Formule logique : Preuve par induction structurelle

Soit $P(\varphi)$ une propriété sur les formules φ (en fixant l'ensemble V des variables).
On peut montrer $\forall \varphi, P(\varphi)$:

1. Par récurrence sur la taille/hauteur de φ

Formule logique : Preuve par induction structurelle

Soit $P(\varphi)$ une propriété sur les formules φ (en fixant l'ensemble V des variables).
On peut montrer $\forall \varphi, P(\varphi)$:

1. Par récurrence sur la taille/hauteur de φ
2. Par **induction structurelle**

Formule logique : Preuve par induction structurelle

Pour montrer $\forall\varphi, P(\varphi)$ par induction structurelle, il faut montrer :

1. $P(T)$, $P(F)$, et $\forall x \in V,\ P(x)$
2. $P(\varphi) \implies P(\neg\varphi)$
3. $P(\varphi_1)$ et $P(\varphi_2) \implies P(\varphi_1 \vee \varphi_2)$
4. $P(\varphi_1)$ et $P(\varphi_2) \implies P(\varphi_1 \wedge \varphi_2)$

Formule logique : Preuve par induction structurelle

Pour montrer $\forall \varphi, P(\varphi)$ par induction structurelle, il faut montrer :

1. $P(T)$, $P(F)$, et $\forall x \in V,\ P(x)$
2. $P(\varphi) \implies P(\neg\varphi)$
3. $P(\varphi_1)$ et $P(\varphi_2) \implies P(\varphi_1 \vee \varphi_2)$
4. $P(\varphi_1)$ et $P(\varphi_2) \implies P(\varphi_1 \wedge \varphi_2)$

Remarque : On a un schéma de preuve similaire pour les arbres binaires, et toutes les structures définies récursivement.

Formule logique : Preuve par induction structurelle

Pour montrer $\forall\varphi, P(\varphi)$ par induction structurelle, il faut montrer :

1. $P(T)$, $P(F)$, et $\forall x \in V,\ P(x)$
2. $P(\varphi) \implies P(\neg\varphi)$
3. $P(\varphi_1)$ et $P(\varphi_2) \implies P(\varphi_1 \vee \varphi_2)$
4. $P(\varphi_1)$ et $P(\varphi_2) \implies P(\varphi_1 \wedge \varphi_2)$

Remarque : On a un schéma de preuve similaire pour les arbres binaires, et toutes les structures définies récursivement.

Exemples :

- $P(\varphi) =$ « Si φ possède n opérateurs binaires alors son nombre de terminaux est $n+1$ » (TD)
- $P(\varphi) =$ « φ est équivalence à une formule où toutes les négations sont sur les variables »

Formule logique : Sous-formule

Si φ est représenté par un arbre A, une **sous-formule** de φ est un sous-arbre de A.

Formule logique : Sous-formule

Si φ est représenté par un arbre A, une **sous-formule** de φ est un sous-arbre de A.

Dit autrement, on associe à chaque formule φ l'**ensemble des sous-formules** $F(\varphi)$ inductivement :

$$\begin{aligned} \forall x \in V : \ F(x) &= \{x\} \\ F(\neg\varphi) &= \{\neg\varphi\} \cup F(\varphi) \\ \forall * \in \{\vee, \wedge\} : \ F(\varphi * \psi) &= \{\varphi * \psi\} \cup F(\varphi) \cup F(\psi) \end{aligned}$$

Formule logique : Autres opérateurs

Définition

- On définit $\varphi \longrightarrow \psi$ par $\neg\varphi \vee \psi$.
- On définit $\varphi \longleftrightarrow \psi$ par $\varphi \longrightarrow \psi \wedge \psi \longrightarrow \varphi$.

Formule logique : Autres opérateurs

Définition

- On définit $\varphi \longrightarrow \psi$ par $\neg\varphi \vee \psi$.
- On définit $\varphi \longleftrightarrow \psi$ par $\varphi \longrightarrow \psi \wedge \psi \longrightarrow \varphi$.

```
let implies p q = Or(Not p, q)

let equiv p q = And(implies p q, implies q p)
```

Formule logique : Substitution

Si φ, ψ sont des formules et x une variable, on note $\varphi[x \leftarrow \psi]$ la substitution de x par ψ, définie par :

$$\begin{aligned}
\forall x \in V,\ x[x \leftarrow \psi] &= \psi \\
T[x \leftarrow \psi] &= T \\
F[x \leftarrow \psi] &= F \\
\forall * \in \{\vee, \wedge\},\ (\varphi_1 * \varphi_2)[x \leftarrow \psi] &= \varphi_1[x \leftarrow \psi] * \varphi_2[x \leftarrow \psi]
\end{aligned}$$

Exercice

Écrire une fonction OCaml effectuant une substitution.

Formule logique : Quantificateurs

Si φ est une formule, on peut définir $\forall$ et $\exists$ par :

$$\forall x, \varphi = \varphi[x \leftarrow T] \wedge \varphi[x \leftarrow F]$$

$$\exists x, \varphi = \varphi[x \leftarrow T] \vee \varphi[x \leftarrow F]$$