

```

-- Exemple : 2012
g_gagne_region = VAR "g_gagne_region" :: Form
rigueur = VAR "rigueur" :: Form
- ...
g1 = IMP g_gagne_region (IMP rigueur crise)
g2 = IMP unite_ps_ecolos g_credible
g3 = IMP g_credible unite_ps_ecolos
g4 = IMP (ET crise g_credible) d_perd_2012
hyps = [g1,g2,g3,g4]
hyps2 = hyps ++ [g_gagne_region, rigueur]
-- sats hyps
-- True
-- *Main> conseq hyps2 d_perd_2012
-- False
-- *Main> modeless (hyps2 ++ [NON d_perd_2012])
--[[("unite_ps_ecolos",False),("g_credible",False),
("crise",True),("d_perd_2012",False),("g_gagne_region",
True),("rigueur",True)]]

```

3. Forme normale disjonctive et méthode de décision

- Forme normale disjonctive (et conjonctive)
 - Un *littéral* est une variable propositionnelle ou la négation d'une variable propositionnelle
 - Une formule est en *forme normale conjonctive* (FNC) si elle est une conjonction de disjonctions de littéraux.
 - Une formule est en *forme normale disjonctive* (FND) si elle est une disjonction de conjonctions de littéraux.

Exemple:

Avec nos conventions d'écriture:

$p, \neg q$ sont des littéraux

$q \wedge (p \vee \neg q \vee r) \wedge (\neg q \vee \neg r)$ est en FNC

$q \vee (p \wedge \neg q \wedge r) \vee (\neg q \wedge \neg r)$ est en FND.

- Mise sous FND

Proposition : pour toute formule A il existe A' équivalente, en FND

Méthode

- 1) Elimination des $\rightarrow$ et mise en forme normale négative (FNN)
- 2) On applique autant que possible les règles de réécriture (distributivité du $\wedge$ sur le $\vee$)

$$A \wedge (B \vee C) \text{ ---> } (A \wedge B) \vee (A \wedge C)$$

$$(B \vee C) \wedge A \text{ ---> } (B \wedge A) \vee (C \wedge A)$$

Exemple

$$\mathbf{A} = (p \rightarrow q) \wedge \neg (q \wedge \neg r)$$

$$1) \text{ FNN } (\neg p \vee q) \wedge (\neg q \vee r)$$

$$2) \text{ FND } (\neg p \wedge \neg q) \vee (\neg p \wedge r) \vee (q \wedge \neg q) \vee (q \wedge r)$$

NB. Procédure similaire pour la FNC. Distributivité « symétrique »:

$$A \vee (B \wedge C) \text{ ---> } (A \vee B) \wedge (A \vee C)$$

$$(B \wedge C) \vee A \text{ ---> } (B \vee A) \wedge (C \vee A)$$

- Interprétation d'une FND

Fait : Chaque conjonction donne un ensemble de valuation possible et on couvre ainsi toutes les possibilités de rendre A vraie.

Exemple

$$A = (p \rightarrow q) \wedge \neg (q \wedge \neg r)$$

$$\text{FND} \quad (\neg p \wedge \neg q) \vee (\neg p \wedge r) \vee (q \wedge \neg q) \vee (q \wedge r)$$

$$v0 \left| \begin{array}{l} p \rightarrow 0 \\ q \rightarrow 0 \\ r \rightarrow ? \end{array} \right.$$

$$v1 \left| \begin{array}{l} p \rightarrow 0 \\ q \rightarrow ? \\ r \rightarrow 1 \end{array} \right.$$

$$v2 \left| \begin{array}{l} p \rightarrow ? \\ q \rightarrow 1 \\ r \rightarrow 1 \end{array} \right.$$

Simplifications de la FND

- Supprimer les conjonctions incohérentes (présence de q et $\neg q$)
- Autres : voir TD

- Application : preuve de théorème (Cf. définitions diapos 6 et 7)

1. Test de satisfiabilité

- A est SAT ssi sa FND contient une conjonction non contradictoire
- Si c'est le cas on obtient toutes la valuations qui satisfont A par le procédé précédent
- Pour tester $E = \{A_1, A_2, \dots, A_n\}$ regarder la FND de $A_1 \wedge A_2 \wedge \dots \wedge A_n$

Exemple

$$E = \{p \vee q \rightarrow r, p\} \quad \text{« équivalent à »} \quad A = (p \vee q \rightarrow r) \wedge p$$

$$\text{FND « brute »} \quad (\neg p \wedge \neg q \wedge p) \vee (r \wedge p)$$

$$\text{Simplifiée} \quad r \wedge p$$

Valuations :

$$\begin{array}{l|l} v_0 & p \rightarrow 1 \\ & q \rightarrow ? \\ & r \rightarrow 1 \end{array}$$

E est donc SAT

2. Conséquence logique

$\{A_1, A_2, \dots, A_n\}$ a pour conséquence B si : La FND de $A_1 \wedge A_2 \wedge \dots \wedge A_n \neg B$ est vide

Addendum arbres généraux (n-aires)

```
data Arbre a = Arbre a [Arbre a]
```

```
    deriving (Show, Eq)
```

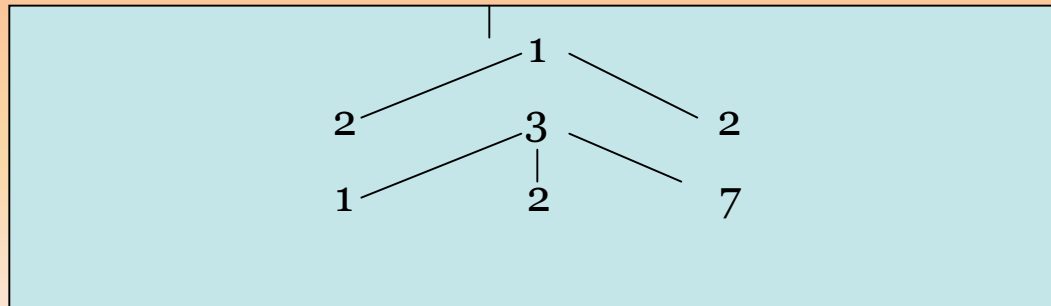
```
type Foret a = [Arbre a]
```

```
t2 = Arbre 1
```

```
    [(Arbre 2 []),
```

```
     (Arbre 3 [(Arbre 1 []), (Arbre 2 []), (Arbre 7 [])]),
```

```
     (Arbre 2 [])]
```



```

data Arbre a = Arbre a [Arbre a]
    deriving (Show, Eq)
type Foret a = [Arbre a]

t1 = Arbre 1 [(Arbre 2 []), (Arbre 3 [(Arbre 5 []), (Arbre 6
    []), (Arbre 7 [])]), (Arbre 4 [])]
t2 = Arbre 1 [(Arbre 2 []), (Arbre 3 [(Arbre 1 []), (Arbre 2
    []), (Arbre 7 [])]), (Arbre 2 [])]

count :: (Eq a) => a -> Arbre a -> Int
countForest :: (Eq a) => a -> Foret a -> Int

count x (Arbre r f) = (if x == r then 1 else 0) + countForest
    x f
countForest x f = sum (map (count x) f)

memberTree :: (Eq a) => a -> Arbre a -> Bool
memberForest :: (Eq a) => a -> Foret a -> Bool
memberTree x (Arbre r f) = x == r || memberForest x f
memberForest x f = or (map (memberTree x) f)

```