# Logique propositionnelle :

* ^ et v ainsi que ⇔ associent à gauche : a^b^c donne (a^b)^c
* => associe à droite : a => b => c donne a => (b => c)

On à la précédence suivante : nég > ^> v > => > ⇔

## **Sémantique :**

Affectation (ou interprétation) de ⍴ : application de l'ensemble des variables de propositions V vers ß.

La sémantique [[phi]]⍴ d’une formule phi dans l’affectation ⍴ est définie par récurrence structurelle sur phi par :

Si A appartient à V alors [[A]]⍴ = ⍴(A), [[T]]⍴ = T et [[Bot]] = F

Si phi appartient à l’ensemble des formules, alors [[non(phi)]]⍴ = non(béta)[[phi]]⍴

sinon pour tout connecteur binaire c : [[phi c phi’]]⍴ = [[phi]]⍴ c béta[[phi’]]⍴

Soit phi une formule et ⍴ une affectation :

* ⍴ est un modèle de phi ou ⍴ satisfait phi noté ⍴|= phi ssi [[phi]]⍴ = T
* phi est valide ssi phi est vraie dans toutes les affectations, elle est invalide sinon
* Une formule valide est aussi appelée une tautologie
* phi est satisfiable ssi elle est vraie dans au moins une affectation

Toutes les formules valides sont satisfaisables et toutes les formules insatisfiables sont invalides.

Avec Gentzen ne jamais oublier de bien parenthésée pour éviter les erreurs.

### Les quantificateurs :

La portée d’un quantificateur va jusqu’à la parenthèse fermante de la formule du quantificateur.

Vx,y.phi == Vx.Vy.phi

### Variables libres et liées :

* Une variable x est libre dans une formule phi ssi il existe une occurrence de x dans phi qui n’est pas sous la portée d’aucun quantificateurs.
* Une variable x est liée dans une formule phi ssi il existe une occurrence de x dans phi qui est sous la portée d’un quantificateur.
* FV() est l’ensemble des variables libres, tandis que BV() est l’ensemble des variables liées.

Quelques règles de la récurrence structurelle :

* Si x e V alors FV(x) = {x}, BV(x) = ensemble vide
* FV(T)=FV(bot)=BV(T)=BV(Bot) = ensemble vide
* Si phi e F alors FV(non(phi)) = FV(phi), BV(non(phi) = BV(phi)
* Pour tout connecteur binaire c , FV(phi c phi’) = FV(phi) U FV(phi’)
* Pour tout connecteur binaire c , BV(phi c phi’) = BV(phi) U BV(phi’)
* Si x e V et phi e F alors FV(Vx.phi) = FV(Ǝx.phi) = FV(phi\{x}) comme ce n’est pas une variable libre, alors on refait la même chose sur le reste mais sans la variable x qui est liée, et BV(Vx.phi) = BV(Ǝx.phi) = BV(phi) U {x} ici elle est liée et entre donc dans le sous-ensemble on réitère donc l’opération sur le reste union de cet ensemble qui contient déjà la variable liée x.

Exemples :

* y est libre dans Vx.P(x,y)
* x est liée dans Vx.P(x,y)
* x est libre et liée dans (Vx.P(x,y)) ^ Q(x)

### Formules Polies (Propres) :

Une formule est polie ou propre si aucune variable n’est libre est liée en même temps et si aucune variable n’est sous la portée de plus d’un quantificateur.

* (Vx.P(x,y)) ^ (Ǝz.Q(z)) v R(t)) est une formule polie.
* (Vx.P(x,y)) ^ Ǝx.Q(x) n’est pas une formule polie.

### Alpha conversion d’une formule non polie :

Il est possible de rendre une formule non polie propre en effectuant une alpha conversion, cad en renommant certaines variables liées. Ainsi :

(Vx.P(x,y)) ^ Q(x) devient (Vz.P(z,y)) ^ Q(x) qui maintenant est une formule polie.

### Formule close ou fermée :

Une formule est close si aucune variable n’est libre dans la formule. Un énoncé est une formule close.

### Modélisation :

* A est une condition nécessaire pour B (A doit être réalisé pour que B le soit aussi)

= B -> A

* A est une condition suffisante pour B (si A est réalisé alors B le sera) = A->B
* A est une condition nécessaire suffisante pour B (A et B sont réalisés en même temps = A ⇔ B

Il est nécessaire d’avoir le permis de conduire pour conduire une voiture :

* C(x) = x conduit une voiture
* P(x) = x a le permis
* Vx.C(x) -> P(x)

Il suffit qu’il neige à Montpellier pour qu’il neige à Oslo :

* N(x) : il neige à x
* m : Montpellier
* O : Oslo
* N(m)->N(O)

Valide = La proposition donne T dans tous les cas

Invalide = La proposition donne F dans certains cas

Satisfiable = La proposition donne T dans certains cas

Insatisfiable = La proposition donne F dans tous les cas

• **Idempotence** de ^ et v

(P^P) ≡ P (PvP) ≡ P

• **Associativité** de ^ et v :

((P^Q)^R) ≡ (P^(Q^R)) ((PvQ)vR) ≡ (Pv(QvR))

• **Commutativité** de ^ et v

(P^Q) ≡ (Q^P) (PvQ) ≡ (QvP)

• **Distributivité** du ^ par rapport à v (et vice versa)

((P^(QvR) ≡ ((P^Q)v(P^R))

((Pv(Q^R) ≡ ((PvQ)^(PvR))

*• Double négation*

¬¬P ≡ P

*• Lois de De Morgan*

¬(P^Q) ≡ (¬Pv¬Q) ¬(PvQ) ≡ (¬P^¬Q)

*• Implication et équivalent*

(P→Q) ≡ (¬PvQ) (P↔Q) ≡ ((P→Q)^(Q→P))

*• Négation et True et Absurde*

¬P ≡ (P→**⊥**)

(P v ¬P) ≡ T (T^P) ≡ P (TvP) ≡ T

(P^¬P) ≡ **⊥** (**⊥**^P) ≡ **⊥** (**⊥**vP) ≡ P

*Les macros :*

* ⊥ pour (P ^ ¬P)
* T pour ¬⊥
* (P v Q) pour ¬(¬P ^ ¬Q)
* (P → Q) pour (¬P v Q)

Pour une formule close phi, sa sémantique sera donc notée : [[phi]]’

## **Fonctions et égalités :**

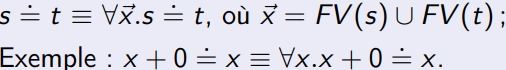
L’égalité : C’est un prédicat binaire, noté de manière infixe

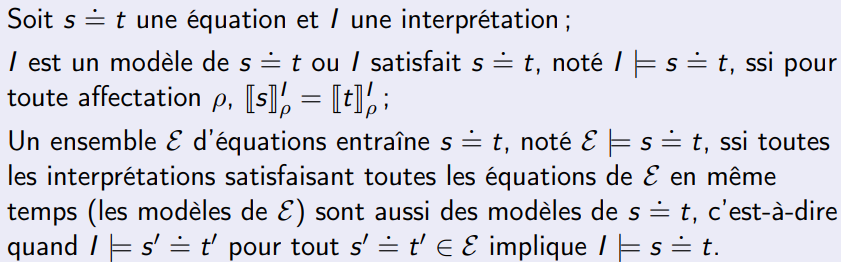
s ≗ t, où s et t sont des termes.

* [[s ≗ t]]Iρ = ([[s]]Iρ ≗ [[t]]Iρ)
* Où « = » est l’égalité sur DI
* Ne pas confondre la syntaxe et la sémantique :
* « ≗ » ≡ égalité syntaxique
* « = » ≡ égalité sémantique.

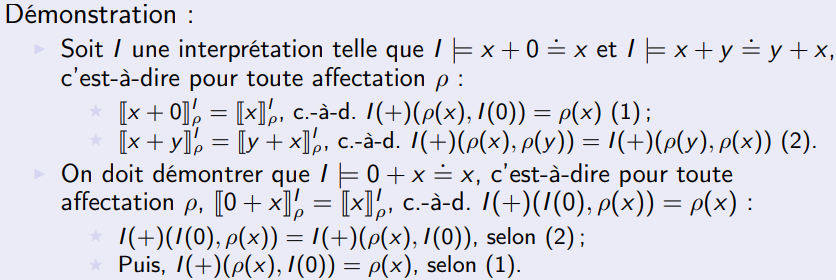
Equation = paire de termes notée s ≗ t

Les termes s et t ne sont pas forcément clos. Mais la quantification sur les variables libres est implicite.





Démontrer que : x + 0 ≗ x, x + y ≗ y + x |= 0 + x ≗ x, où 0 est une constante et « + » est un symbole de fonction binaire noté de manière infixe



**Une substitution σ** est une application de V vers T. Elle se définit par récurrence structurelle comme suit :

* Si x ∈ V alors σ(x) = σ(x);
* Si f ∈ SF d’arité n et t1, . . . ,tn ∈ T alors σ(f (t1, . . . ,tn)) = f (σ(t1), . . . , σ(tn)).

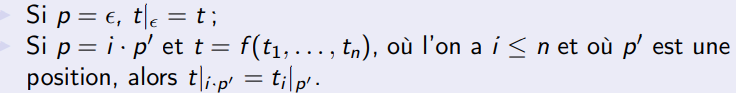
Exemple :

Soit la substitution σ telle que σ(x) = a et σ(y) = f (b), où a et b sont des constantes, et f un symbole de fonction unaire. σ(f (x, y)) = f (a, f (b)).

**Position :**

Étant donné un terme t, le terme t|p désigne le terme à la position p

et se définit par récurrence structurelle sur les positions :



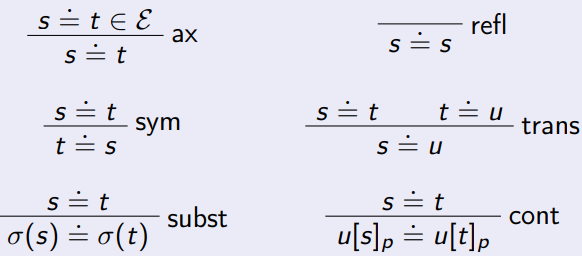
Exemples : si t = f (x, g(y, z)), t| = f (x, g(y, z)), t|1 = x,

t|2 = g(y, z), t|21 = y, t|22 = z.

La notation t[u]p désigne la substitution de u au terme t|p dans t ;

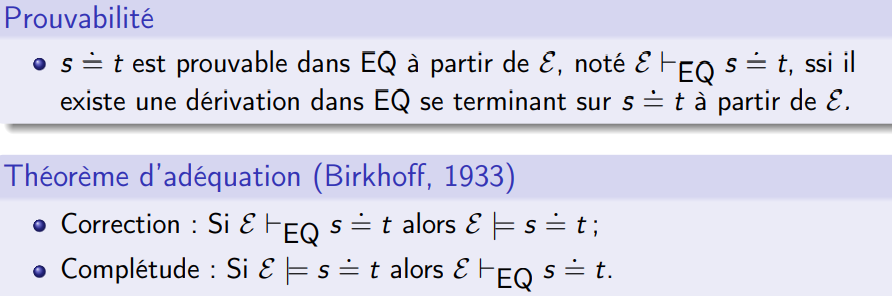
Exemple : si t = f (x, g(y, z)), t[h(a)]21 = f (x, g(h(a), z)).

Système de preuves logique équationnelle :

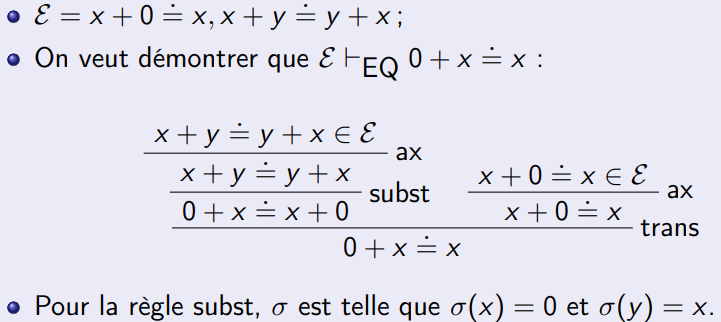


Arbre de preuve (arbre de dérivation)

* On part de l’équation initiale à démontrer ;
* On applique les règles en raisonnement abductif (raisonnement arrière), c’est-à-dire que l’on part de ce qu’on veut montrer pour aller vers les hypothèses/axiomes) ;
* On construit ainsi un arbre dont le séquent est la racine et les branches sont créées par les différentes prémisses des règles de déduction ;
* Dans une branche, on s’arrête lorsqu’on atteint une règle axiomatique, qui devient ainsi une feuille de l’arbre ;
* Un arbre de preuve est un arbre dont toutes les branches se terminent par une règle axiomatique (on parle alors de branche close).



Exemple :



## **Syntaxe :**

V = ensemble de variables d’individus x, y etc

Sf = ensemble de symboles de fonctions f, g etc

Sp = ensemble de symboles de prédicats P, Q etc

Sf ∩ Sp = Ensemble vide

Arité = nombre d’arguments m : Sf U Sp = N

* f(x,y) avec f e Sf, m(f) = 2;
* f(x, y, z) avec f e Sp m(p) = 3;

Les constantes sont des fonctions d’arité 0.Une interprétation i est un ensemble non vide Di.

Une affectation ⍴ est une application de V dans Di. Pour toute affectation ⍴, ⍴[v/x] est l’affectation envoyant chaque variable y autre que x vers ⍴(y), et x vers v.

Si x e V alors [[x]]i⍴ = ⍴(x).

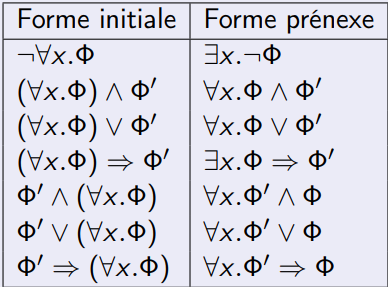
* [[Vx.phi]]i⍴ = ^ v e Di [[phi]]i⍴ [v/x]
* [[Ǝx.phi]]i⍴ = V v e Di [[phi]]i⍴ [v/x]

## Manipulations syntaxiques :

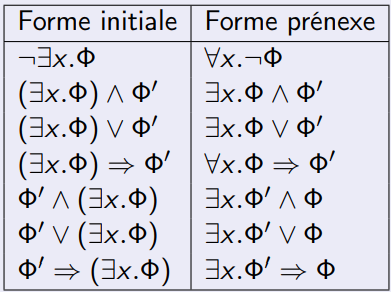
Une formule est dite en forme prénexe si et seulement si elle est de la forme Q1x1. . . . .QnXn.Φ, où Q1, . . . , Qn sont des quantificateurs et Φ est une formule sans quantificateur.

**Théorème** : Pour toute formule Φ, il existe une formule en forme prénexe Φ’ calculable à partir de Φ, telle que Φ et Φ’ sont (sémantiquement, resp. prouvablement) équivalent.

Transformation en prénexe pour **(Pour tout)** :

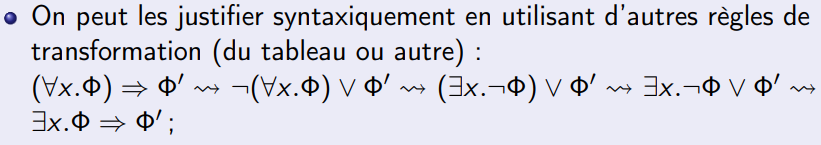


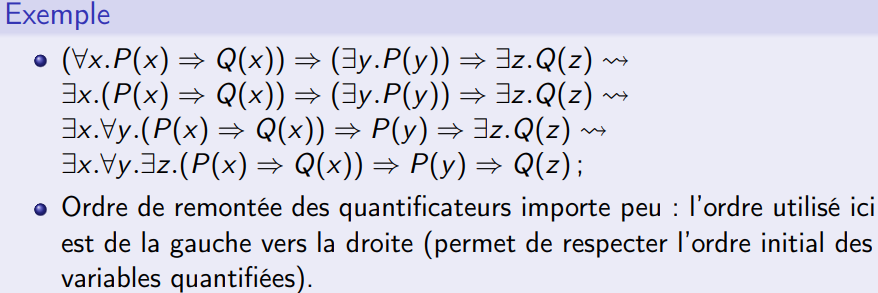
Transformation en prénexe pour **(Il existe)** :



**Règles de transformation**

* x ne doit pas être libre dans Φ’
* C’est le cas si la formule initiale est polie (propre) ;
* Cette transformation respecte les connecteurs de la formule initiale, sauf pour ⇔, où il faut transformer Φ ⇔ Φ’ en (Φ ⇒ Φ’) ∧ (Φ’ ⇒ Φ)





**Skolémisation et Herbrandisation :**

**Formule existentielle/universelle**

**Une formule existentielle** est une formule de la forme ∃x1. . . . .∃xn.Φ, où Φ est une formule sans quantificateur ;

**Une formule universelle** est une formule de la forme ∀x1. . . . .∀xn.Φ, où Φ est une formule sans quantificateur.

Théorème de Herbrand-Skolem

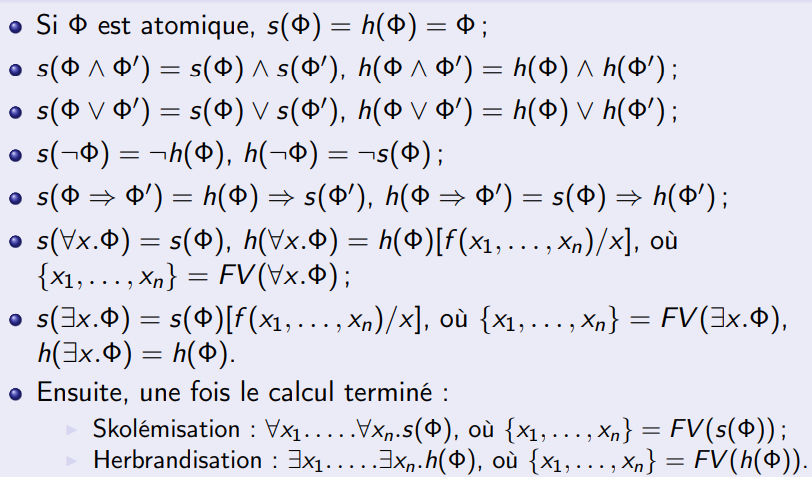
Pour toute formule Φ :

* Il existe une **formule existentielle Φ’** telle que Φ’ est valide si et seulement si Φ est valide (Φ’ est une forme de Herbrand de Φ) ;
* Il existe une **formule universelle Φ’** telle que Φ’ est insatisfiable si et seulement si Φ est insatisfiable (Φ’ est une forme de Skolem de Φ).

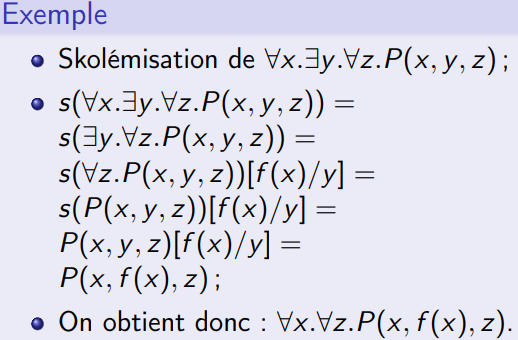
Si on a une formule ∀x.∃y.P(x, y) vraie dans une certaine interprétation, on peut se dire que pour chaque x, il existe (au moins) un y tel que P(x, y) soit vrai. On peut donc supposer qu’il y a une **fonction f (x)** qui nous fournit un « bon » y pour chaque x. Ceci nous permet de réécrire la formule comme ∀x.P(x, f (x)), où f est un nouveau symbole de fonction (le langage est étendu).

Il est plus simple de décrire la skolémisation pour une formule sous forme prénexe.S’il n’y a aucun quantificateur universel avant celui de y, f sera une fonction d’arité 0, c’est-à-dire une constante.

Règles de skolémisation et d’Herbrandisation :

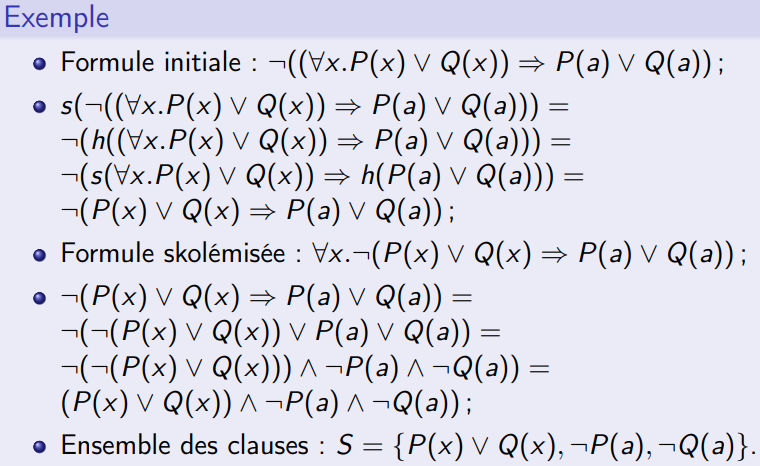


* **La formule de départ doit être polie (propre)** ;
* La formule de départ **n’est pas forcément en forme prénexe** ;
* Après le calcul, on obtient une forme prénexe (tous les quantificateurs universels ou existentiels sont en tête), mais la formule obtenue n’est pas logiquement équivalente à la formule de départ (voir le théorème de Herbrand-Skolem)



* On skolémise : on obtient une formule universelle ∀~x.Φ ;
* On élimine les quantificateurs, puis on met Φ en forme cnf ;
* Les clauses seront les membres de la cnf (quantification implicite) ;

Pour la clausification voir les règles et les macros plus haut dans le doc.



En particulier, [] est la substitution vide (ou identité).

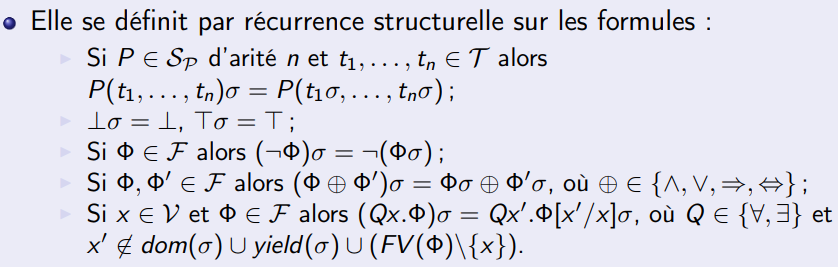
**La notion de substitutio**n s’étend aux termes et se note tσ (de manière postfixe), où t est un terme et σ une substitution. Elle se définit par récurrence structurelle sur les termes :

* Si x ∈ V alors xσ = σ(x);
* Si f ∈ F et t1, . . . ,tn ∈ T alors f (t1, . . . ,tn)σ = f (t1σ, . . . ,tnσ).

Exemples :

* f (x, g(y, z))[a/x, h(b)/y, c/z] = f (a, g(h(b), c));
* f (x, g(y, z))[a/x, h(b)/y] = f (a, g(h(b), z)).

**La notion de substitution s’étend aux formules** et se note Φσ (de manière postfixe), où Φ est un terme et σ une substitution.



Exemple :

* P(f (x), g(y, z))[a/x, h(b)/y, c/z] = P(f (a), g(h(b), c));
* (∀x.P(f (x), g(y, z)))[a/x, h(b)/y, c/z] = ∀x’ .P(f (x’), g(h(b), c)).

**Unification :**

* Une substitution σ est un unificateur de deux termes s et t si et seulement si sσ = tσ ;
* Une substitution ρ est un renommage si et seulement si elle envoie des variables vers des variables, et elle est bijective.

Exemples :

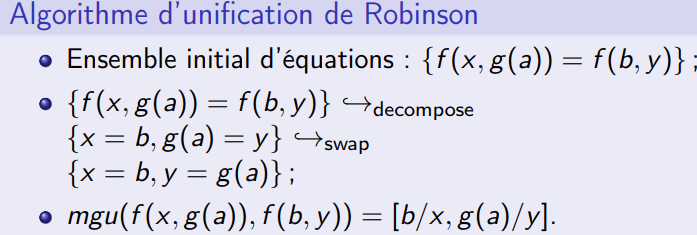
* f (x, g(a)) et f (b, y) sont unifiables, l’unificateur est [b/x, g(a)/y] ;
* ρ = [x’/x, y’/y] est un renommage : f (x, g(y))ρ = f (x’, g(y’)).

**Théorème Unification :**

Étant donnés deux termes s et t, soit il n’existe aucun unificateur de s et t, et nous disons alors que s et t ne sont pas unifiables, ou il existe un unificateur le plus général ou mgu (“most general unifier” en anglais), c’est-à-dire une substitution σ telle que :

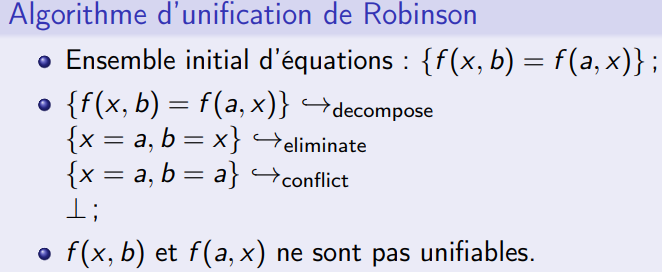
* σ est un unificateur de s et t, autrement dit, sσ = tσ
* Tout unificateur σ’ de s et t est une instance de σ, c’est-à-dire qu’il existe une substitution σ’’ telle que σ’ = σσ’’

**Calculer le MGU de deux termes :**



**Règles de Robinson :**

* G ∪ {t = t} ,→ G **(delete)** ;
* G ∪ {f (s1, . . . ,sn) = f (t1, . . . ,tn)} ,→ G ∪ {s1 = t1, . . . ,sn = tn} **(decompose)** ;
* G ∪ {f (s1, . . . ,sn) = g(t1, . . . ,tm)} ,→ ⊥, si f 6= g ou n 6= m **(conflict)** ;
* G ∪ {f (s1, . . . ,sn) = x} ,→ G ∪ {x = f (s1, . . . ,sn)} **(swap)** ;
* G ∪ {x = t} ,→ G[t/x] ∪ {x = t}, si x 6∈ t et x ∈ G **(eliminate)** ;
* G ∪ {x = f (s1, . . . ,sn)} ,→ ⊥, si x ∈ f (s1, . . . ,sn) **(check)**.



## Résolution :

La résolution est un mécanisme de démonstration qui n’utilise que la règle de coupure du calcul des séquents et les règles de quantification universelles, et qui lui est équivalent, alors qu’elle procède très différemment du calcul des séquents.

La résolution est le mécanisme d’évaluation du langage PROLOG :

**F est réfutable par résolution**

si et seulement si

**¬F est démontrable dans le calcul des séquents**

si et seulement si

**F est fausse dans toute interprétation**

Pour présenter la méthode de résolution (**une autre technique** de preuve que le calcul des séquents, basée sur la réfutation), nous allons (re)voir :

* mise sous forme prénexe
* Skolemisation
* mise sous forme clausale
* unification
* résolution proprement dite

**Distributivité de ∀ sur & :**

((∀xF) ^ (∀xG)) ≣ (∀x(F ^ G))

((∀xF) v (∀xG)) != (∀x(F v G))

**Distributivité de ∃ sur V :**

((∃xF) ^ (∃xG)) != (∃x(F ^ G))

((∃xF) v (∃xG)) ≣ (∃x(F v G))

**Commutation des formes pré nexes :**

¬(∀xF) ≣ (∃x(¬F))

¬(∃xF) ≣ (∀x(¬F))

Cela se comprend : dire qu’on n’a pas pour tout x la propriété F c’est dire qu’il existe un x qui satisfait (¬F). De même dire qu’il n’existe pas de x satisfaisant F, c’est dire que tout x satisfait (¬F).

**Forme prénexe : remontée des quantificateurs :**

((∀xF) ^ G) ≣ (∀x(F ^ G))

((∃xF) ^ G) ≣ (∃x(F ^ G))

((∀xF) v G) ≣ (∀x(F v G))

((∃xF) v G) ≣ (∃x(F v G))

((∀xF) → G) ≣ (∃x(F → G))

((∃xF) → G) ≣ (∀x(F → G))

(G → (∀xF)) ≣ (∀x(F → G))

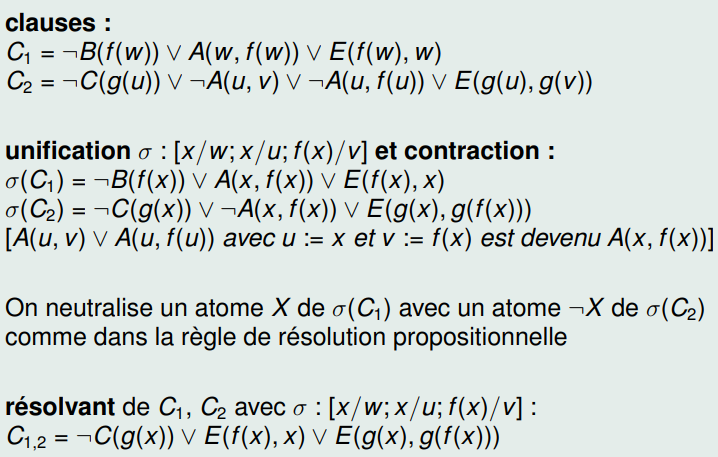
(G → (∃xF)) ≣ (∃x(F → G))

**Forme de Skolem est satisfiabilité :**

Soit G une formule sous forme prénexe et G’ sa forme de skolem, G est satisfiable ssi G’ est satisfiable. Satisfiable : on dit aussi consistante ou non contradictoire : elle est vraie pour une interprétation.

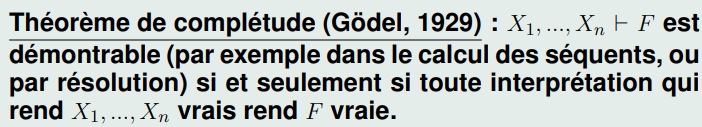
**Théorème de Herbrand** : Supposons qu’on veuille montrer qu’une formule F est insatisfiable. D’après ce qui précède, cela revient à montrer que sa forme clausale est insatisfiable , et pour cela on dispose du théorème de Herbrand : Une forme clausale est insatisfiable si et seulement s’il est possible de remplacer dans chaque clause les variables par des termes de Herbrand de sorte que l’ensemble des clauses (propositionnelles) obtenues soit inconsistant.

Résolution : Unifier et simplifier :



une formule (ou un séquent) est démontrable si et seulement si il est vrai pour toute interprétation.

**Complétude :**

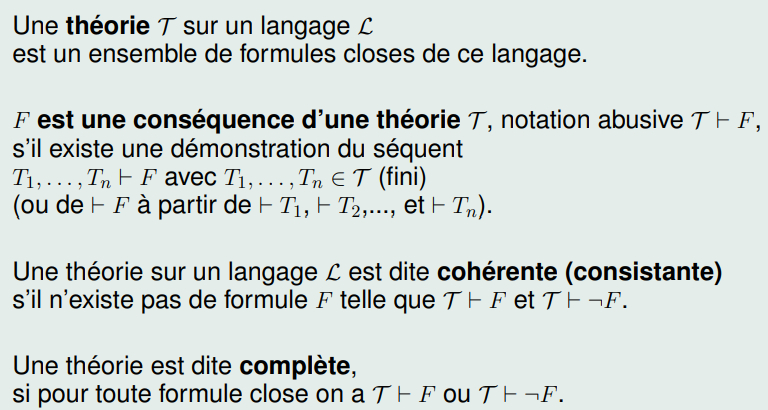


**Model Existence Lemma :**

Si un ensemble de formules est cohérent (c.-à-d.consistent, c.-à-d.s’il ne démontre pas ⊥) alors il admet un modèle.

**Contraposée du MEL** Si un ensemble de formules n’admet pas de modèle (insatisfiable) alors cet ensemble de formules entraîne ⊥ (dans le calcul des séquents, ou par résolution).

Vocabulaire :



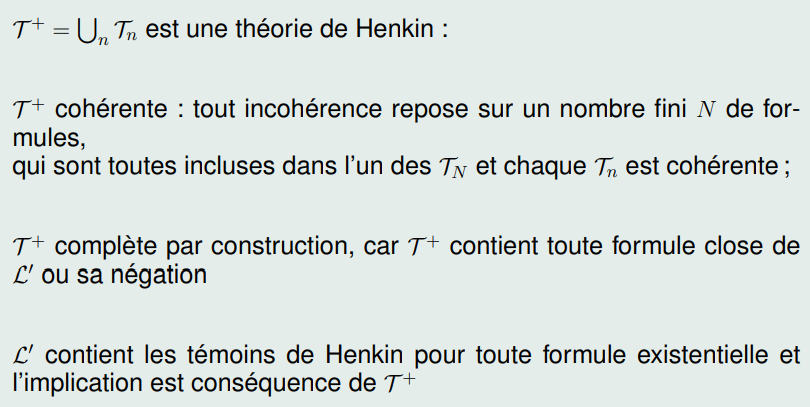
**Témoins de Henkin et théorie de Henkins :**

Une théorie admet des témoins de Henkin si pour toute formule à une variable libre F[v] il existe une constante c telle que (∃v. F[v]) ⇒ F[c].

Une théorie de Henkin est une théorie

* cohérente,
* complète,
* et qui possède des témoins de Henkin

Toute théorie de Henkin admet un modèle.



**Récapitulons ce que nous avons montré :**

Soit une théorie cohérente T sur L’, on peut étendre son langage en L’ et compléter T en une théorie de Henkin T+. Cette théorie T+ admet un modèle I. Ce modèle I satisfait toutes les formules de T+ et donc a fortiori toutes les formules de T.

**Ce qui démontre le Model Existence Lemma** : Toute théorie cohérente T admet au moins un modèle.

**Complétude :**

Si la formule F est vraie dans tout modèle d’une théorie T (cohérente) alors T |- F.

**Validité et complétude :**

On sait que le calcul des prédicats est valide, que les règles ne dérivent que des séquents vrais tant tout modèle. Donc une théorie T qui admet un modèle I est forcément cohérente.

**Capacité du calcul des prédicats :**

Si toute partie finie d’une théorie T admet un modèle. Alors T toute entière admet un modèle.

Toute partie finie de T est cohérente donc T est cohérente (une incohérence est une preuve de ⊥ qui n’utilise qu’un nombre fini de formules de T ).

Comme T est cohérente, elle admet un modèle.