CNAM

Intelligence Artificielle

Intelligence Artificielle

Plan du cours

Introduction + Systèmes à base de règles	(1)
Parcours de graphes	(1)
Introduction à la logique (d'ordre 0 et 1)	(4/5)
Introduction à PROLOG	(1)
Logique à description	(3)
Satisfaction de contraintes	(1) + (1)
Apprentissage symbolique (introduction, classification)	(2)
Introduction à la fouille de données	
Analogie	

Introduction à l'Intelligence Artificielle

1 - Introduction

L'Intelligence Artificielle est une branche de l'Informatique qui consiste à concevoir des "systèmes intelligents" (dont le comportement peut être qualifié d'intelligent) = être capable de **résoudre des problèmes en raisonnant** et en s'appuyant sur des **connaissances** générales ou spécifiques au domaine du problème.

- ➔ Résolution de problèmes
- ➔ Raisonnement
- ➔ Représentation de connaissances

Résolution de problèmes :

Un problème peut se formaliser comme un triplet (I, F, O) ou :

- I = état initial
- F = état final
- O = ensemble d'opérateurs qui permettent de changer d'état

Avec ce point de vue, résoudre 1 problème consiste à trouver un chemin d'extrèmités données un graphe (en l'occurrence, les extrèmités I et F).

Raisonnement

Une forme standard et bien connue de raisonnement est le syllogisme (déjà étudié par Aristote)

Syllogisme: Tout ce qui est vert a peur
La bouteille est verte
Donc la bouteille a peur

La formalisation Modus ponens

$$\frac{p \rightarrow q \text{ et } p}{q}$$

schéma d'inférence

Sur la forme, la conclusion est vraie, le "raisonnement est correct".

Sur le fond, on peut ne pas être d'accord ➔ faire intervenir le fait que seuls les êtres vivants puissent avoir peur ➔ **connaissance**

Lorsqu'on donne un sens aux éléments du syllogisme, la conclusion peut s'avérer erronnée.

Ainsi la connaissance possède 2 facettes qui sont interreliées = forme et fond
Syntaxe et sémentique (sens)

2 - Formalisation des problèmes et résolution de problèmes

Généralement, un problème a un énoncé en langage naturel et une solution (ici un problème a toujours une solution)

Il est possible de classer les problèmes en problèmes **algorithmiques**, il existe un algorithme qui permet de résoudre le problème et problème non algorithmique.

L'Intelligence Artificielle s'intéresse à la classe des problèmes non algorithmiques.
Un problème de base = satisfaction d'une formule booléenne

Etant donné une formule booléenne φ, trouver l'ensemble des points tels que $\varphi = 1$ (rapport de φ).

La complexité du problème est exponnentielle (en 2^N où N est le nombre de variables de la formule)
Un problème peut se formaliser sous la forme d'un triplet (I, O, F) ou

I = état initial
F = état final
O = un ensemble d'opérateurs

Un état est une configuration du problème à un instant donné.
Il faut savoir décrire un état = il existe plusieurs possibilités pour cela (analogie avec l'algorithme)

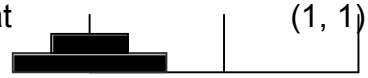
Un opérateur permet de changer d'état et il possède des **conditions d'applicabilité** (il n'est pas applicable dans tous les états)

Les états spéciaux = I, F et états qui symbolisent des **Impasses**.
L'ensemble des états possibles obtenus par application des opérateurs est appelé **l'espace d'états**

3 - 5 Exemple: Tours de Hanoï, 3 piles et 2 disques

Etat initial  Etat final 

Opérateurs = déplacer 1 disque d'un pilier à l'autre applicabilité (condition), ne pas déplacer le grand disque pour le mettre sur le petit disque

Espace d'état (1, 1) desciption d'un état = (pg, pd)
Etat initial 

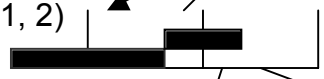
pilier où est le grand disque — pilier où est le petit disque

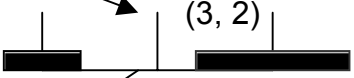
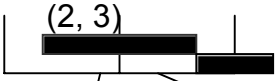
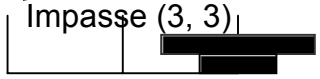
Remarque: la notation est ambiguë

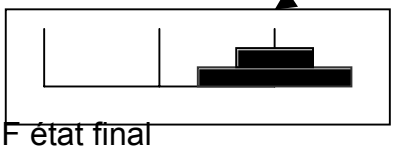
(2, 2)  (2, 2) 

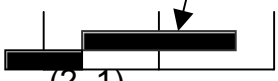
On ne prend en compte que les états légaux

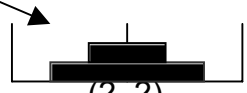

(1, 1) = Etat initial = I

(1, 2)  (1, 3) 

(2, 2) Impasse  (3, 2)  (2, 3)  Impasse (3, 3) 


F état final


(2, 1)


(2, 2)

…….

Remarques: - Impasses
- Etats non accessibles

Les choses ne sont pas toujours aussi simples :

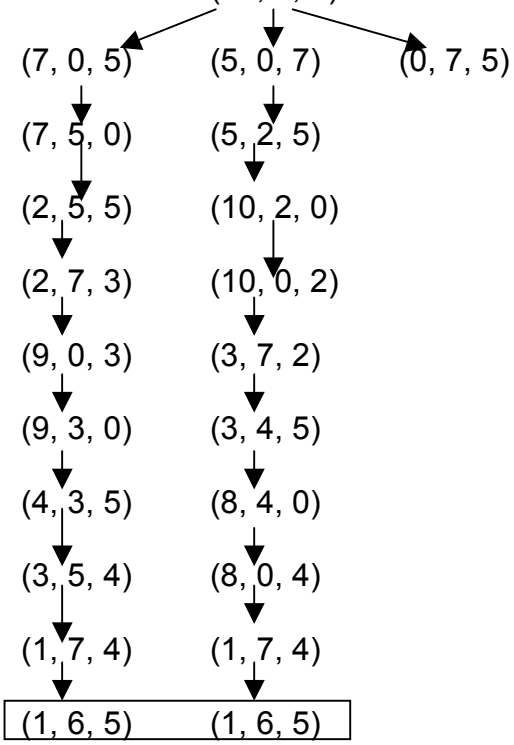
- Problème mal spécifié = on ne connaît pas l'état final
- L'ensemble des opérations peut être infini
- …..

2 - 2 Exercice:

Une personne dispose d'une bonbonne de 12l dans laquelle elle veut mesurer 6 litres. Pour cela elle dispose de 2 bouteilles de 7 et 5 litres.

1) Formaliser le problème en terme (I, F, O) donner 1 représentation possible d'un état
2) Développer l'espace d'état et donner 1 solution

Un état triplet (C_{12}, C_7, C_5)
Etat initial (12, 0, 0) Pas de duplication d'état



2 - 3 Types de problèmes :

- Problèmes de contraintes Cryptarithmétique SEND 1 lettre
- Problèmes logiques MORE ⇕
  MONEY 1 chiffre

SEND E ≠ 0 Conjoncture M = 1
MORE S = 9 (retenue pour avoir M de money)
MONEY O = 0
E + 1 = N (présence nécessaire d'une retenue)
N + R = "1 E"
D + E = "1 Y"

E = 8 Impasse
Conjoncture E = 7 ➔ N = 8 ➔ R = 8 Impasse

Conjoncture	E = 6	967X	Impasse
	N = 7	1086	
	R = 8	1070X	
Conjoncture	E = 5	9567	D = 7
	N = 6	1085	Y = 2
	R = 8	10652	

2 - Systèmes à base de règles

Une première famille de systèmes intelligents = les systèmes à base de règles (SBR)

3 - 5 Introduction

Définition: Une règle de production se présente sous la forme prémisse (s)

Si condition (s) alors Conclusion (s)

partie gauche partie droite

Une telle règle peur être utilisée pour représenter des connaissances dans de nombreux domaines :

(1) Si l'animal a des plumes alors c'est un oiseau

(2) Si l'animal est bipède et qu'il n'a pas de plumes alors c'est un homidé

(3) Si le patient a de la fièvre et s'il présente des pistules alors il faut envisager la varicelle

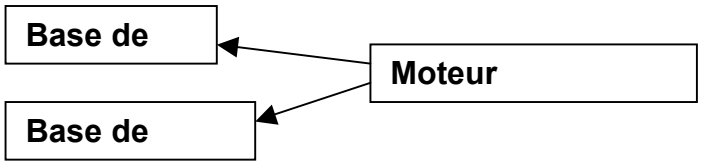
Remarque:

- Les règles sont bien adaptées pour représenter des connaissances liées à des diagnostics ou des classiffications
- Les règles peuvent être très complexes = elles peuvent utiliser des connecteurs booléens et, ou et non
- Un coefficient de **plausibilité** peut être associé aux éléments d'une règle

3 - 5 Architecture d'un SBR

Définition: Un SBR se compose de 3 modules :

- Une **base de faits** qui contient les données relatives au problème à résoudre (mémoire "à court terme")
- Une **base de règles** qui contient les règles représentant la connaissance du domaine du problème
- Un **moteur d'inférences** chargé d'appliquer les règles en fonction de la base de faits pour résoudre le problème



Remarque fondamentale :

Il y a **séparation** entre les connaissances - ici les règles - et les procédures de manipulation des connaissances du moteur d'inférences.

On dit qu'on a une représentation déclarative des connaissances, par opposition aux représentations procédurales.

Représentation procédurale		Base de Règles
Programme 1	Si A alors B	(R1) Si A alors B
Programme 2	Si A et C alors B	(R2) Si A et C alors Z

Les règles sont **autonomes et indépendantes**

Le moteur se charge d'appliquer les règles

Principe de Base

Aucune règle n'influe sur les autres règles

Mais les **métarègles** peuvent être utilisées pour :

(1) Contrôler le fonctionnement du moteur d'inférence

(2) Mettre en œuvre des stratégies (dépendant du domaine du problème)

"Déclaratif" = Représenter des connaissances indépendemment de leur utilisation

3 - 5 Le fonctionnement du moteur d'inférences

Un moteur d'inférences fonctionne selon 1 cycle à 3 temps :

(1) Sélection des règles applicables (une règle est applicable si toutes ses conditions sont satisfaites)

(2) Choix d'une règle = résolution des conflits.

L'ensemble des règles applicables s'appelle aussi l'ensemble des conflits, et il y a effectivement dès qu'il existe plus d'une règle applicable.

Méta - règles qui **permettent de choisir une règle**

(MR1) Ne pas choisir une règle qui conclut sur un fait déjà connu

(MR2) Choisir la règle qui a le "coefficient d'importance" le plus élevé (lorsque ce coefficient existe) Pour nous, ce coefficient sera symbolisépar le **rang** de la règle (son numéro)

(MR3) Choisir la règle qui a le plus de conclusions

(MR4) Choisir la règle qui a le plus de conditions

(MR5) arrêter le moteur d'inférences lorsque l'ensemble des conflits est vide

(3) Appliquer la règle choisie et prendre en compte les conclusions de la règle

Un exemple

BF = {G, H, K}

BR = 1. A ➔ E Si A alors E
2. B ➔ D
3. H ➔ A
4. A et G ➔ C
5. E et K ➔ B
6. D et E et K ➔ C
7. G et K et F ➔ A

Un exemple de fonctionnement : déduire tous les faits qui peuvent l'être (chaînage avant)

Cycle 1 BF = {G, H, K} Applicables = {R3} BF = {A, G, H, K}

Cycle 2 Applicables = {R1, ~~R3~~, R4} choisi R1 BF = {A, E, G, H, K}

Cycle 3 Applicables = ~~{~~R1, ~~R3~~, R4, R5} choisi R4 BF = {A, C, E, G, H, K}

Cycle 4 Applicables = ~~{~~R1, ~~R3~~, ~~R4~~, R5} choisi R5 BF = {A, B, C, E, G, H, K}

Cycle 5 Applicables = ~~{~~R1, R2, ~~R3~~, ~~R4~~, R5} choisi R2 BF = {A, B, C, D, E, G, H, K}

Cycle 6 Applicables = {0} arrêt du moteur

Rappel Introduction Résolution de problèmes
Espace d'Etat (I, O, F)
SBR BR Règles Si alors
BF Faits
MI + Sélection
+ Choix

+ Application

Stratégies d'Application

3 - 5 Le chaînage avant ou la résolution dirigée par les données

Un moteur d'inférences se caractérise par la façon dont il applique les règles et la façon de mémoriser les résultats intermédiaires.

Le chaînage avant se décrit de la façon suivante (il correspond intuitivement aux 3 étapes d'un cycle de moteur) = étant donné un certain nombre de faits déjà établis, le moteur sélectionne les règles applicables, en choisi une et la déclanche.

La mémorisation de nouveaux faits est soit immédiate (une fois la valeur de vérité du fait est établie) ou différée (une fois que l'ensemble des conflits est vide)

Il existe 2 conditions d'arrêt principales =
- la base de faits contient le fait recherché
- la base de faits est saturée = plus aucune règle n'est applicable

Remarque: Un problème correspond en général à vérifier qu'un certain fait est vrai.

Résoudre le problème consiste à trouver une règle déclanchable pour laquelle le fait recherché est en partie conclusion.

p ➔ q "p est vrai" la règle déclanche ➔ "q est vrai" ➔ cas particulier d'une

prémisses conclusions règle d'inférence logique "modus ponens" p, p ➔ q

p est vrai q q est vrai

Exemple:
R1 Si K et L et M Alors I
R2 Si I et L et J Alors Q
R3 Si C et D et E Alors B
R4 Si A et B Alors Q
R5 Si L et N et O et P Alors Q
R6 Si C et H Alors R
R7 Si R et J et M Alors S
R8 Si F et H Alors B
R9 Si G Alors F BF = {A, C, D, E, G, H, K} Problème Q

Rappel (MR1) Ne pas appliquer une règle qui conclut sur un fait déjà connu
(MR2) Choisir la règle de rang minimal si plusieurs règles sont applicables

Chaînage avant avec insertion immédiate des nouveaux faits

Cycle1: Sélection {R3, R6, R9} Choix R3
Application BF = BF U {B} ← insertion immédiate

Espace d'Etat : Etat initial = base de faits initiale
Etat final = Q ∈ base de faits
Ensemble d'opérateurs = ensemble des règles avec leurs conditions d'application

E0 = I = {A, C, D, E, G, H, K} ➔ R3 ➔ E1 = {A, C, D, E, G, H, K} U {B}

Cycle2: Sélection {R4, R6, R9} Choix R4
Application BF = BF U {Q} ➔ R4 ➔ E2 = E1 U {Q}

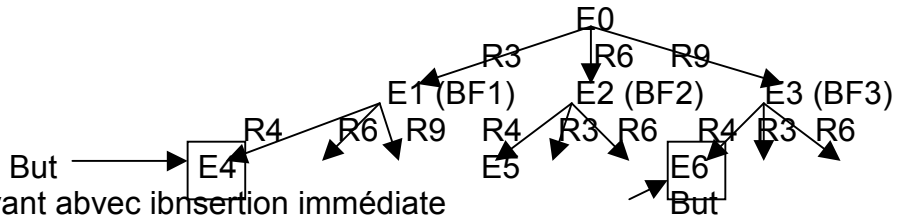
Cycle2: Arrêt du moteur = E2 = Faits Q ∈ E2 Q ∈ BF

Schéma d'algorithme

BF
But
Tant que But ∉ BF et conflit non vide
    Si le But ∈ BF alors ne rien faire
    Sinon | Sélection des règles (conflit)
          | Choix
          | Application

Exercice: Ecrire correctement cet algorithme
Chaînage avant avec une insertion différée des nouveaux faits

Cycle 1.1 : Sélection = {R3, R6, R9} Choix R3 Application B est mis en attente
Cycle 1.2 : Sélection = {R3, R6, R9} Choix R6 Application R est mis en attente
Cycle 1.3 : Sélection = {R3, R6, R9} Choix R9 Application F est mis en attente
Cycle 1.4 : Sélection = { } Choix = { } Application BF = BF U {B, R, F}
Cycle 2.1 : Sélection = {R4, R8} Choix R4 Application Q est mis en attente
Cycle 2.2 : Sélection = { } Choix = { } Application BF = BF U {Q}
Cycle3.1 : Arrêt du moteur = car Q ∈ BF



Chaînage avant abvec ibnsertion immédiate

3 - 5 Le chaînage arrière ou la résolution dirigée par le but

Cette fois la résolution ne s'applique plus directement sur la base de faits, mais sur le but à vérifier.

Le principe est le suivant :

- Le moteur recherche les règles qui concluent sur le but à vérifier, et s'assurent que ces règles sont "déclanchables".
- La règle est déclanchable si ses prémisses sont vérfiées.
- Si parmi les règles sélectionnées, une règle est déclanchable, alors le but est vérifié.
- Si ce n'est pas le cas, alors les prémisses à vérifier deviennent de nouveaux buts, appelés **sous - buts**, et le processus est réitéré.

Les principales conditions d'arrêt :

- L'ensemble des sous - buts est vide (succès) = tous les sous - buts ont été vérifiés et le problème est résolu
- Impasse ou échec :
  - Soit un des sous - buts n'est pas vérifiable avec la règle courrante et il faut choisir une nouvelle règle pour le vérifier, et si cela n'est pas possible, alors il y a échec.

Schéma d'algorithme :

Procédure vérifier (But)

    Ok = Faux
    Cas 1 Si But ∈ BF alors Ok = Vrai

Cas 2 Si But est demandable (attribution d'une valeur externe)
alors Ok = réponse externe à la question
Cas 3 Si BR (But) = ensemble de règles qui concluent sur But
Tant que Ok = faux et Br (But) non vide
- sélectionner R dans BR (But)
- Ok = vérifier conditions (R)
- BR (But) = BR (But) - {R}

⇕ Il suffit qu'une règle soit déclanchable
⇕ **Nœud "OU", "OR"**

Procédure vérifier - conditions (Règle)
Ok = Vrai
C = conditions (Règles)
Tant que C non vide et Ok = Vrai
Ok = vérifier (Premier (C))
C = C - {Premier (C)}

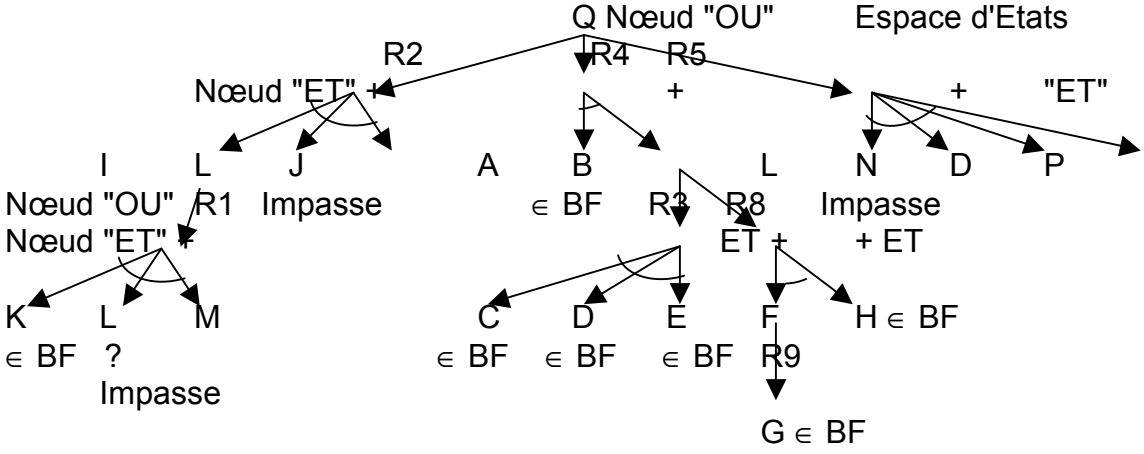
⇕ Vérifier toutes les conditions de la règle
⇕ **Nœud "ET", "AND"**
⇕ Analogie avec AND booléen

Remarque: Comme pour le chaînage avant, il est possible d'avoir une mémorisation immédaite ou différée des nouveaux faits établis.

Chaînage arrière avec mémorisation immédiate

Cycle 1 : Sélection = {R2, R4, R5} But = {Q} Choix = R2
Application = vérifier les conditions de R2 Sous - But = {I, L, J}
Cycle 2 : But = {I} Sélection = {R1} Choix = R1
Application = vérifier les conditions de R1 Sous - But = {K, L, M}
Cycle 3 : But = {K} K ∈ BF
Cycle 4 : But = {L} L ∉ BF, non demandable, BR (L) = 0 **Impasse**
R2 n'est pas déclanchable
Cycle 5 : Sélection = {R4, R5} Choix = R4 Sous - But = {A, B}
Cycle 6 : But = {A} A ∈ BF
Cycle 7 : But = {B} Sélection = {R3, R8} Choix = R3 Sous - But = {C, D, E}
Cycle 8 : But = {C} C ∈ BF
Cycle 9 : But = {D} D ∈ BF
Cycle 10 : But = {E} E ∈ BF
Cycle 11 : But = {} **Succès**

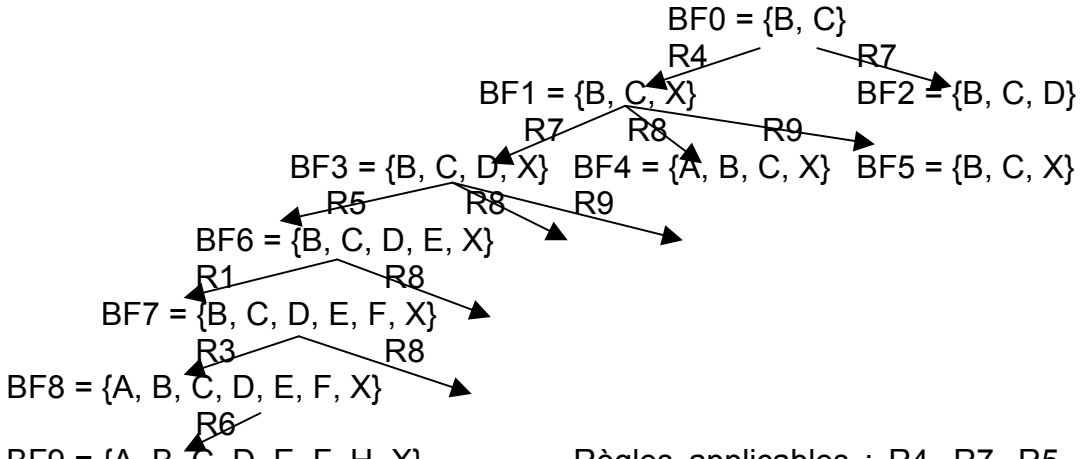
Alternance de Nœuds "ET" et de Nœuds "OU"
Arbre ET - OU



Exercice: R1 Si B et D et E Alors F
R2 Si G et D Alors A
R3 Si C et F Alors A
R4 Si B Alors X
R5 Si D Alors E
R6 Si X et A Alors H
R7 Si C Alors D
R8 Si X et C Alors A
R9 Si X et B Alors D BF = {B, C} Problème : vérifier H

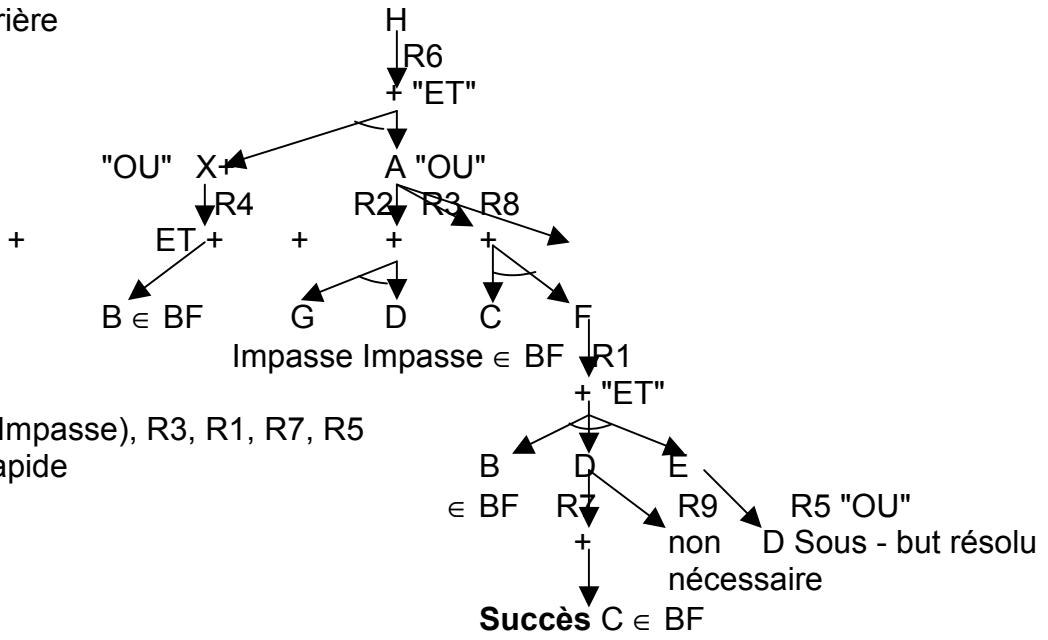
Vérifier H avec le chaînage avant (mémorisation immédiate) et le chaînage arrière (mémorisation immédiate).

Dessiner l'arbre et ou asocié au chaînage arrière



BF9 = {A, B, C, D, E, F, H, X} Règles applicables : R4, R7, R5, R1, R3, R6

Chaînage arrière



R6, R4, R2 (Impasse), R3, R1, R7, R5

Résolution rapide

R6
X R4
A R8

Remarques:

1) il est impossible d'affirmer qu'une sorte de chaîne est meilleure qu'une autre
2) Pas d'utilisation de variables (ni de connecteurs booléens comme "OU" ou "NON"

Eléments sur les graphes et les structures ordonnées

Dans ce chapitre nous essayerons de formaliser les notions d'espace d'état et de parcours d'état pour la notion de graphe.

Un graphe est une structure très générale et à partir de cette notion il est possible, des notions de structures plus régulières comme les arbres et les treillis (utilisées dans les représentations hiérarchiques et en apprentissage).

1 - Graphes et relations

Un graphe G = (E, Γ) est la donnée de 2 ensembles, E un ensemble de **sommets** (ou **points**) et Γ un ensemble **d'axes** (des couples de sommets)

Notations x Γ y = l'arc (x, y) x = origine de l'arc y = extrèmité de l'arc

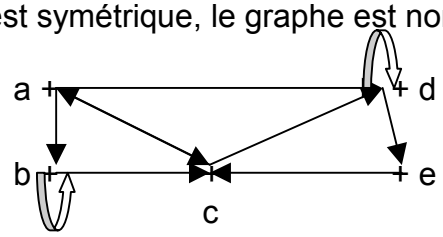
$\Gamma(x) = \{y \in E \mid x \Gamma y\}$ = ensemble des sommets en relation avec x

$\Gamma^{-1}(x) = \{y \in E \mid y \Gamma x\}$ = ensemble des sommets en relation avec x (avec une orientaion différente)

$x \Gamma y \neq y \Gamma x$

Γ = relation et donc elle peut posséder les propriétés de relations. En particulier, elle peut être symétrique $x \Gamma y = y \Gamma x$ $\Gamma(x) = \Gamma^{-1}(x)$

Si Γ est symétrique, le graphe est non orienté (les arcs s'appellent des **arêtes**)



E = {a, b, c, d, e }

Γ (b) = {b, c} Γ^{-1} (b) = {a, b}

Γ (c) = {a, d} Γ^{-1} (c) = {a, b, c}

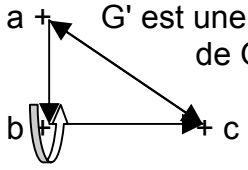
graphe non orienté = le même sans les flèches

2 - Notion de Sous - Graphe, chemins et arbres

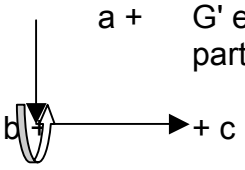
Définition: Un **sous - graphe** G' = (E', Γ') de G = (E, Γ) est tel que :

- E' $\leq$ E et Γ' $\leq \Gamma$ (inclusion ensembliste)
- Si l'ensemble des arcs qui joint les sommets de E' dans G est conservé, dans ce cas G' est une **restriction** de G à E', sinon G' est un **sous - graphe partiel** de G

E' = {a, b, c}



G' est une restriction de G à E'



G' est un sous - graphe partiel de G

Définition: Un **chemin** (Chaîne dans un graphe non orienté) est une suite d'arcs

$\gamma_0, \gamma_1 \ldots \gamma_n = (x_0, x_1), (x_1, x_2), \ldots (x_n, x_{n+1})$, telle que l'extrèmité de x_{i+1} de (x_1, x_{i+1}), correspond avec l'origine x_{i+1} de $\underbrace{(x_{i+1}, x_{i+2})}_{\gamma_{i+1}}$.

L'origine du chemin est x_0

L'extrèmité du chemin est x_n

Si $x_0 = x_n$, le chemin est un **circuit** (cycle si G non orienté)

Un chemin élémentaire est constitué **d'arcs tous différents**

Définition: Arbre

Un **arbre** est un graphe non orienté sans cycle la connexité G = (E, Γ)
$\forall$ x, y $\in$ E, il existe une chaîne qui relie x à y

Nous ne considérons que les **arbres enracinés** = il existe un sommet distingué appelé **racine** de l'arbre qui sert de référence.

Variantes de vocabulaire Sommet ➔ nœud
Arête ➔ branche

Rappels: Systèmes à base de règles
Chaînage avant
Chaînage arrière
Parcours de graphe
Parcours d'un espace d'états
Initial ➔ but
But ➔ Etat initial admissible
(Parcours d'arbre ET - OU)

Introduction aux graphes
Exploration de graphes Profondeur
Largeur
Graphe Strutures régulières
Treillis symétrique
Réflexive
Transitive
Anti - symétrique
G = (E, Γ) ➔Symétrique
Non symétrique
Non orienté orienté

3 - Parcours ou exploration de Graphe

3 - 1 Définitions

On appelle **parcours de graphe** ou **exploration de graphe** tout procédé déterministe qui permet de fixer un ordre de parcours = étant donné l'ensemble de sommets déjà exploré, quel est le prochain sommet à explorer ?

Nature des graphes étudiés (espace d'états)

Les graphes sur lesquels nous travaillons sont non orientés et ils sont supposés **enracinés** = il existe un sommet distingué appelé la racine à partir duquel commence le parcours (état initial dans un epace d'Etat).

Un graphe enraciné peut posséder des cycles.
Il faut donc s'arranger pour détecter les cycles (problème de la boucle infinie)

Notion de couche :

Etant donné un graphe G = (E, Γ) muni d'une racine α on définit la notion de **couche** de la façon suivante : $\forall$ x et y $\in$ E, d (x, y) = Inf {lg (x, y)}

Distance du plus petit chemin allant de x à y longueur d'une chaîne de x à y
(le plus petit chemin n'est pas nécessairement unique)

pour le graphe G = (E, Γ) enraciné en α :

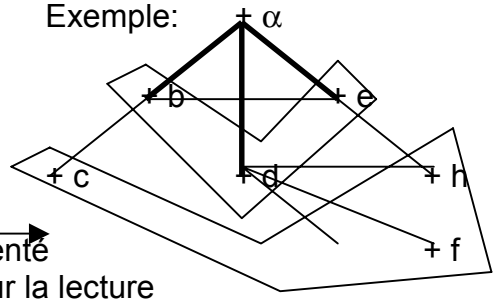
C_0 (α) = {d} couche d'ordre 0

C_1 (α) = {x $\in$ E | d (α, x) = 1}

C_2 (α) = {x $\in$ E | d (α, x) = 2}

On peut prendre n'importe quel élément du graphe et définir les couches qui lui sont associées.

Exemple:



orienté
pour la lecture
cycle α, b, e

C_0 (α) = {α}

C_1 (α) = {b, d, e}

C_2 (α) = {c, f, g, h}

3 - 2 Parcours en largeur

Le parcours en largeur d'un graphe enraciné G = (E, Γ) s'appuie sur la notion de couche et consiste à explorer depuis la racine α le graphe, couche par couche, jusqu'à explorer le sommet désiré.

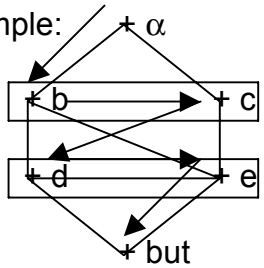
L'algorithme associé utilise une file, structure de donnée premier entré - premier sorti FIFO (insertion en queue et extraction en tête de file)

Remarque: le parcours de graphe nécessite un **marquage** des sommets de façon à ne pas parcourir un "cycle infini"

Algorithme largeur

G = (E, Γ)
α = racine de G, file {α} but - atteint = faux
but = but à atteindre
Jusqu'à file vide ou but atteint faire
    Si la tête de file = but alors but - atteint = vrai ne rien faire d'autre
    Sinon engendrer les successeurs non marqués de la tête de la file dans l'ordre "gauche droite"
    (pour un élément x, les sucesseurs de x sont donnés par Γ (x))
    Γ (x) = {y $\in$ E | x Γ y}
Si but - atteint = vrai alors succès
Sinon échec

Exemple:



File = {α} α est marqué
Γ (α) = {b, c}
File = {b, c} b et c sont marqués
Γ (b) = {d, e, α}
File = {c, d, e} d et e sont marqués
Γ (c) = {α, e}
File = {d, e}
Γ (d) = {b, but, e}
File = {e, but} but est marqué
Γ (e) = {b, d, but, c}

File = {but}

Dans ce parcours les couches sont respectées = elles sont explorées l'une après l'autre dans le sens gauche - droite

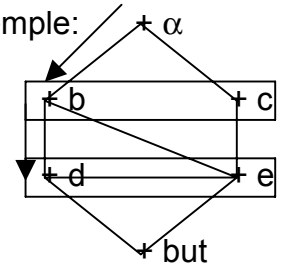
3 - 3 Parcours en profondeur

Le principe du parcours en profondeur est d'explorer le graphe à partir de la racine en allant "la plus à gauche et le plus profond possible"
Pour ce faire, à la place d'utiliser une file comme dans le parcours en largeur, on utilise une **pile** (dernier entré - premier sorti insertion et extraction au sommet) LIFO

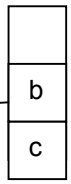
Algorithme profondeur

G = (E, Γ) de racine α
Pile = {α}, α est marqué
But - atteint = faux, but
Jusqu'à pile vide ou but - atteint faire
  Si sommet de pile = but alors but - atteint = vrai (ne rien faire d'autre)
  Sinon + engendrer les sucesseurs non marqués du sommet de pile
    + dépiler
    + empiler les sucesseurs dans l'ordre droite - gauche en les marquant
Si but - atteint = vrai alors succès
Sinon echec

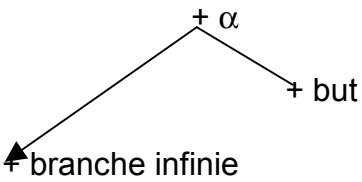
Exemple:



Pile = {α} α est marqué
Γ (α) = {b, c}
Pile = {b, c} b et c sont marqués
Γ (b) = {d, e, α}
Pile = {d, e, c} d et e sont marqués
Γ (d) = {b, but, e}
Pile = {but, e, c} but est marqué But arreint

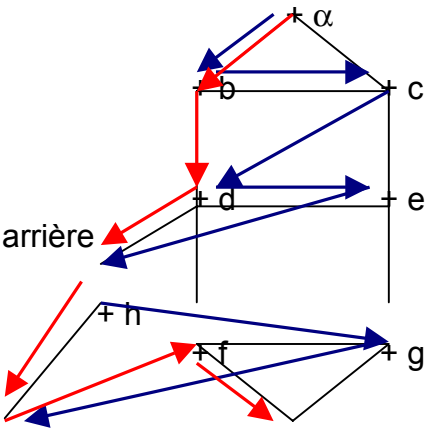


Remarque: Il ne faudrait pas conclure d'après l'exemple que le parcours en profondeur permet d'atteindre le but plus rapidement.



Par exemple: les branches infinies posent un problème et une profondeur maximale d'exploration doit quelquefois être fixée

Exemple: Parcours en largeur et en profondeur de α et de ω



Largeur	**Profondeur**	
File = {α}	Pile = {α}	
File = {b, c}	Pile = {b, c}	
File = {c, d, e}	Pile = {d, e, c}	
File = {d, e}	Pile = {h, f, e, c}	
File = {c, h, g}	Pile = {i, f, e, c}	
File = {h, f, g}	Pile = {f, e, c}	Retour
File = {f, g, i}	Pile = {ω, g, e, c}	But - atteint = ω
File = {g, i, ω}		
File = {i, ω}		
File = {ω}	But - atteint = ω	


+ i
+ ω

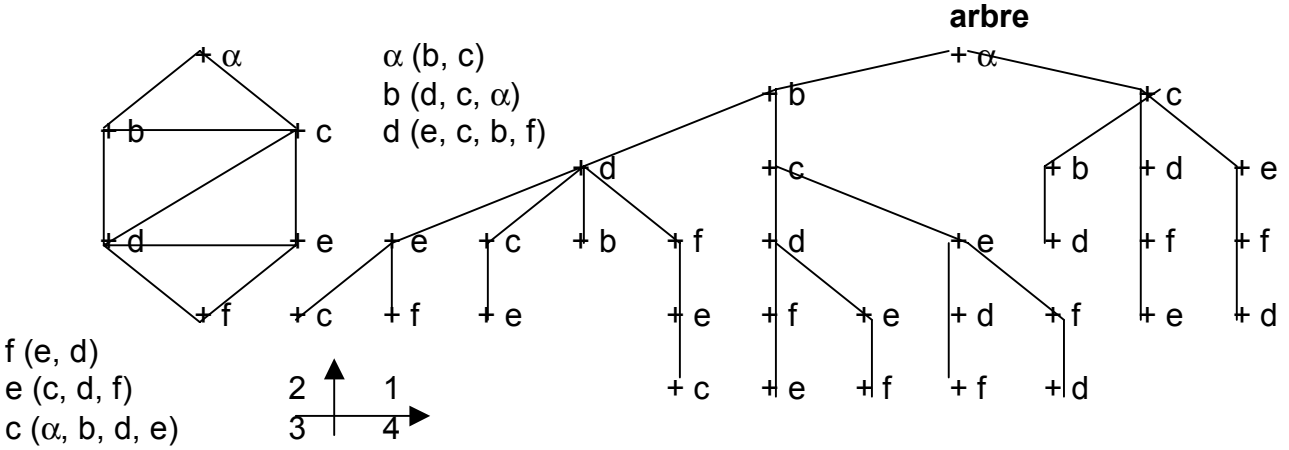
3 - 4 Une application du parcours en profondeur :

Transformation d'un graphe en arbre avec duplication des sommets

Il est possible de "transformer un graphe en un arbre" : l'arbre correspond au graphe où les cycles ont été supprimés.

Principe et règles de construction

- Parcours du graphe en profondeur
- Une branche est crée et correspond au parcours courant : la branche s'arrête dès que le sucesseur examiné est déjà sur la branche
- Pour un sommet donné il y a autant de branches que de sucesseurs qui ne sont pas déjà sur la branche ➔ duplication des sommets



4 - L'Algoritme A* ou la recherche ordonnée

4 - 1 Introduction

Dans les parcours en largeur et en profondeur on recherche un chemin dans le graphe qui mêne de la racine au but. On ne se préoccupe pas de la qualité du chemin, ces parcours sont d'ailleurs qualifiés de **parcours aveugles**

Cette fois nous allons parcourir le graphe en essayer de trouver le "meilleur" chemin entre la racine α et le but ω = le meilleur au sens d'une **fonction d'évaluation**

Une fonction d'évaluation associe un coût élémentaire à chaque arête du graphe et le coût du chemin est la somme des coûts élémentaires des arêtes qui composent le chemin.

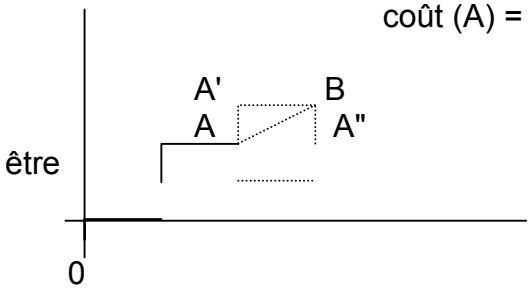
Si $\gamma = \gamma_1, \gamma_2, \ldots \gamma_n$ ou les γ_i, sont des arêtes coût $\gamma = \sum_{i=1}^{n} \text{coût}(\gamma_i)$

Une fonction d'évaluation mesure le coût d'un chemin avec 2 points de vue =

(I) **le passé** = coût du chemin déjà parcouru

(II) **le futur** = estimation du coût du chemin restant à parcourir

coût (A) = évaluation (A) = passé (A) + futur (A)



(4) estimation

une bonne estimation peut être

La distance euclidienne

Remarque : La mise au point de fonctions d'évaluation est une activité complexe en dehors des domaines métriques.

*4 - 2 L'Algorithme A**

Deux Définitions

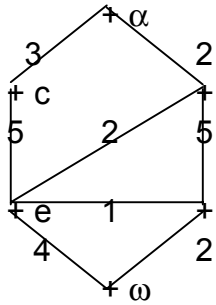
(I) Le coût d'un sommet est le coût du chemin optimal qui le joint depuis la racine
"optimal" = selon la fonction d'évaluation, soit le minimal, soit le maximal
(en règle générale on considère le coût minimal)

(II) Un chemin α à un sommet s est admissible si et seulement si son coût est inférieur au coût de s. En particulier, le coût d'un sommet évolue au cours du parcours.

Algorithme A*

File = {α} (liste vide)
But - atteint = faux
Jusqu'à file vide ou but - atteint faire
Si le premier chemin atteint le but alors but - atteint = vrai (ne rien faire d'autre)
Sinon (a1) Déterminer les chemins admissibles qui peuvent être construits à partir du premier chemin de la file (un chemin ne peut être prolongé que par un sommet qu'il n'incluse pas déjà de cycle)
(a2) Si 2 chemins atteignent le même sommet, ne conserver que le chemin de coût optimal
(a3) Supprimer le premier chemin de la file
(a4) Insérer les nouveaux chemins et trier la liste selon le critère d'évaluation (chemin optimal en tête)
Si but - atteint = vrai alors succès
Sinon échec

Exemple : chemin optimal entre α et ω



File = {α(0)}
Γ (α) = {c, b } chemins = {αc(3), αb(2)}
File = {αb(2), αc(3)}
Γ (b) = {α, e, d } chemins = {αbe(4), αbd(7)}
File = {αc(3), αbe(4), αbd(7)}
Γ (c) = {α, e} αce(8) non admissible car αbe(4)}
File = {αbe(4), αbd(7)}
Γ (e) = {b, d, c, ω} αbed(5), αbeω(8), αbec(9) non admissible car αc(3)

File = {αbed(5), αbd(7), αbeω(8)} αbedω(7) αbeω(8) n'est plus admissible
Γ (d) = {b, e, ω}
File = {αbd(7), αbedω(7)}
Γ (d) = {b, e, ω} αbde(8) non admissible car αbe(4) αbdω(9) non admissible car αbedω(7)
File = {αbedω(7)} but - atteint = vrai

Remarque : On ne s'arrête pas si un chemin atteint le but et qu'il existe un chemin incomplet de coût inférieur

A méditer : jeu du taquin

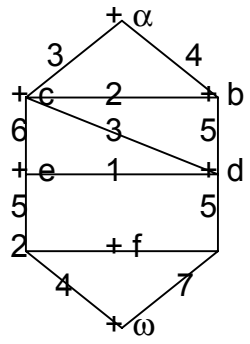
Etat initial	2	8	3	Etat final	1	2	3
	1	6	4		8	*	4

	7	*	5		7	6	5

1) Développer l'espace d'état
2) Proposer une fonction d'évaluation (avec coût du passé et estimation)
3) Utiliser A* pour trouver un chemin optimal entre l'état initial et le but

Exemple : Trouver 1 chemin optimal entre :

- α et d
- α et f
- c et f
- b et ω comparer α et ω



α et d File = {α(0)}
File = {αc(3), αb(4)}
Γ (c) = {b, α, e, d } αcb(5) non admissible
αce(9), αcd(6),
File = {αb(4), αcd(6), αce(9)}
Γ (d) = {α, c, d } αbc(6) et αbd(9) non admissibles car αcd(6)
File = {**αcd(6),** αce(9)} αcd(6) but atteint

α et f File = {α(0)}
File = {αc(3), αb(4)}
File = {αb(4), αcd(6), αce(9)}
File = {αcd(6), αce(9)}
File = {αcde(7), αcdf(11)}
File = {αcdeg(12), **αcdf(11)**} but atteint

c et f File = {αcdf(8)}

α et ω File = {**αcdegω(16)**}

b et ω File = {b(0)}
File = {bc(2), αb(4), bd(5)}
File = {αb(4), bd(5), bce(8)}
bcd(5)
File = {bd(5), bce(8)}
bcd(5)
File = {bde(6), bdf(10)}
File = {bdeg(11), bdfω(17)}
File = {**bdegω(15)**} but atteint

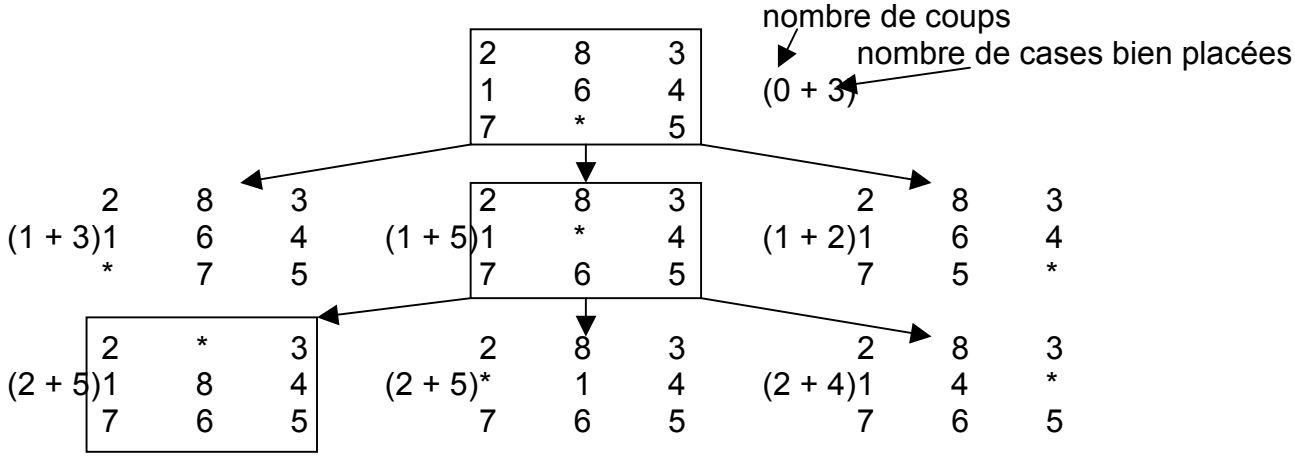
Taquin:

	2	8	3	Etat initial	1	2	3	Etat final
	1	6	4		8	*	4	
	7	*	5		7	6	5	

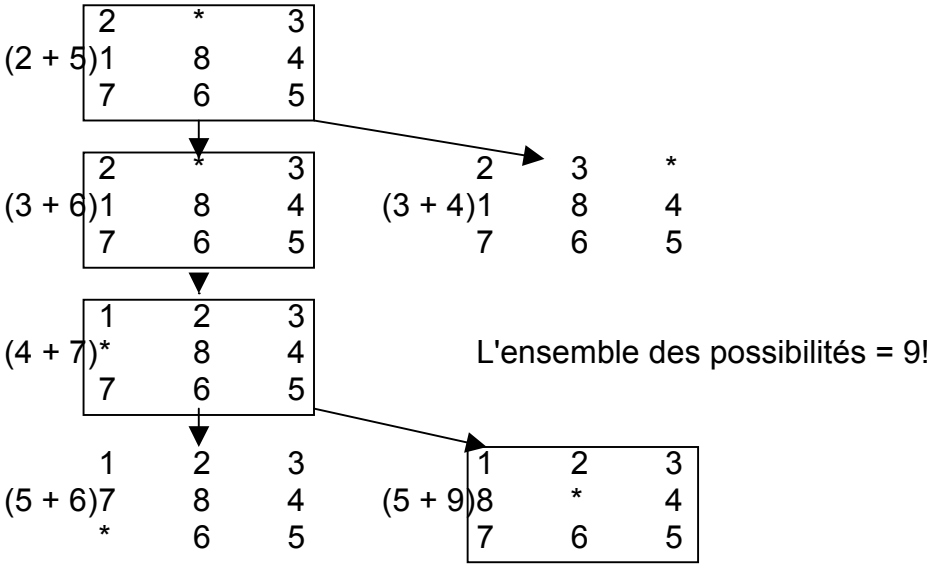
Fonction d'évaluation: eval = p + f

passé : nombre de coups déjà joués

estimation du futur : nombre de coups bien placés / nombre de cases bien placées



2 au même coup



5 - Exploration d'arbres de jeux

Les jeux auquels nous nous intéressons sont des jeux à 2 joueurs où chaque joueur joue alternativement.

L'**univers du jeu** - ou **univers du problème** - peut être représentée par 1 arbre où apparaissent les différentes situations possibles du jeu.

Il est supposé qu'il existe une **fonction d'évaluation** qui associe une valeur à chaque situation du jeu.

Remarque: Les valeurs des situations évoluent au cour du jeu.

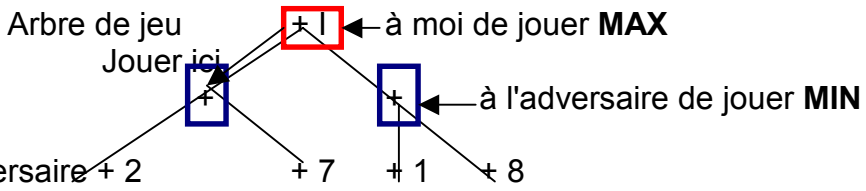
5 - 1 Arbre de jeu et parcours min - max

Dans 1 arbre de jeu, on considère les **couches** selon le départ du jeu. Une couche peut être **amie** - c'est à moi de jouer - ou **ennemie** - c'est à l'adversaire de jouer.

Lorsque la couche est amie, on va chercher à maximiser nos gains = cette couche correspond à ce qui est appelé un **niveau max**.

Lorsque la couche est ennemie, on va chercher à minimiser nos pertes = cette couche correspond à ce qaui est appelé un **niveau mini**.

Exemple:



Les situations terminales correpondent à une évaluation 2 coups à l'avance

Contexte:

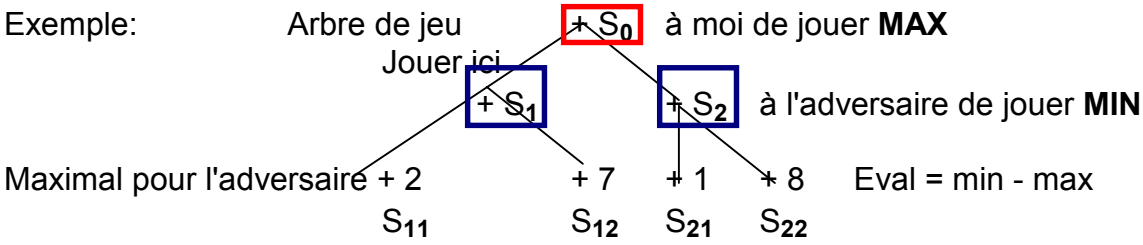
- 2 coups possibles = gauche et droite
- évaluation 2 coups à l'avance

➔ il faut choisir la branche qui correspond à un gain maximal sachant que l'adversaire va de même maximiser ses gains c'est le principe de **l'algorithme min - max**

Principe de l'algorithme min - max

On se donne une profondeur maximale de l'évaluation (nombre de coups à l'avance pour lequel on a une évaluation) notée p_{max}, une fonction d'évaluation et une situation initiale.

1) Si la profondeur p_{max} est atteinte, retourner l'évaluation associée au noued courant
2) Si le niveau du jeu est min, alors appliquer min - max aux descendants du noued courant et retourner le minimum des valeurs résultats.
3) Si le niveau du jeu est max, alors appliquer min - max aux descendants et retourner le maximum des résultats.



eval (s_0) = max (eval (s_1), eval (s_2)) = 2

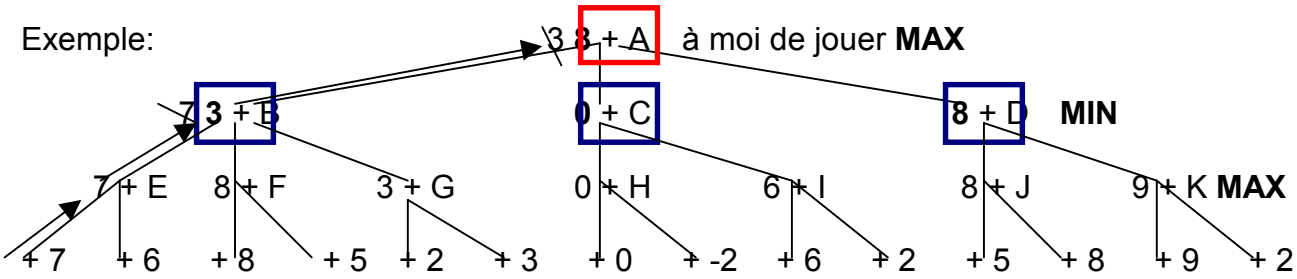
eval (s_1) = min (eval (s_{11}), eval (s_{12}) = 2

2 7

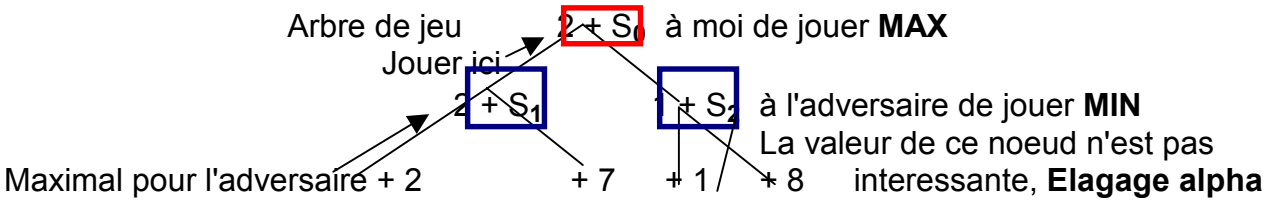
eval (s_2) = min (eval (s_{21}), eval (s_{21}) = 1

1 8

Intérprétation des résultats: Je joue s_1. Une fois en s_1,tout est à recommencer



5 - 1 Elagage alpha - beta



Les règles d'élagage:

(R1) Si tout les successeurs d'un nœud on été éxaminés, alors la valeur provisoire du nœud devient sa **valeur définitive**

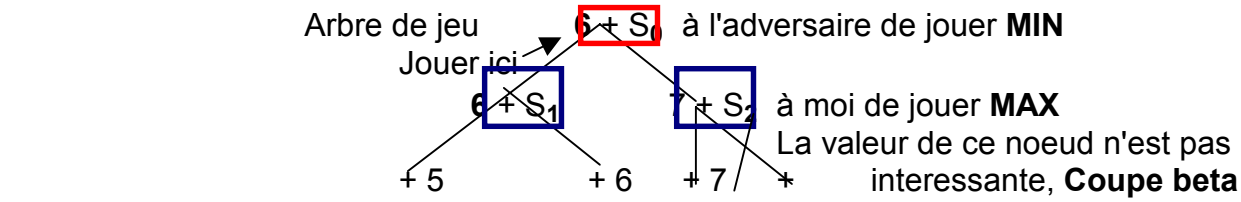
(R2) Si un nœud N de **niveau max** a une valeur provisoire v_1 et qu'un de ses descendants a une valeur définitive v_2, alors la valeur provisoire de N est **max (v_1, v_2)**
Idem pour un nœud N de niveau min

(R3) **Coupe alpha:**

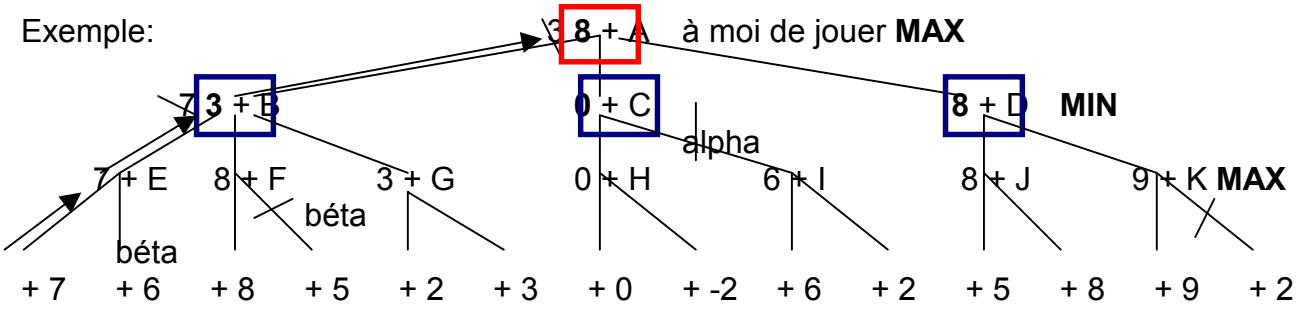
Si M_i est un nœud de **niveau min** avec un ancêtre A(M_i) tel que: **val(M_i) < val(A(M_i))**, alors il est inutile d'explorer la descendance restante de M_i.

(R4) **Coupe beta:**

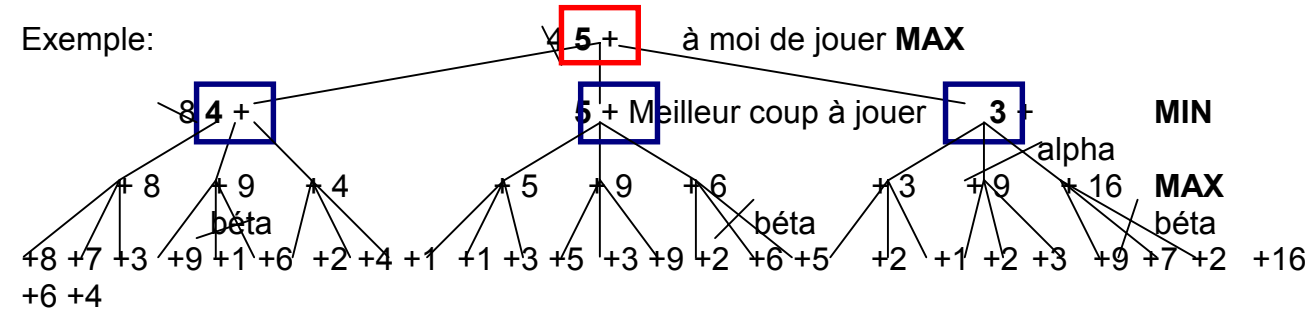
Si M_a est un nœud de **niveau max** avec un ancêtre $A(M_a)$ tel que: **$val(A(M_a)) \leq val(M_a)$**, alors il est inutile d'explorer la descendance restante de M_a.



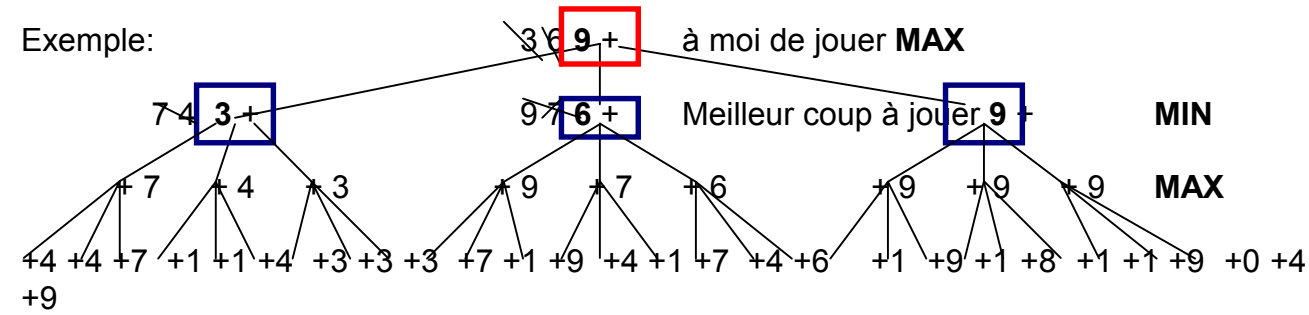
Exemple:

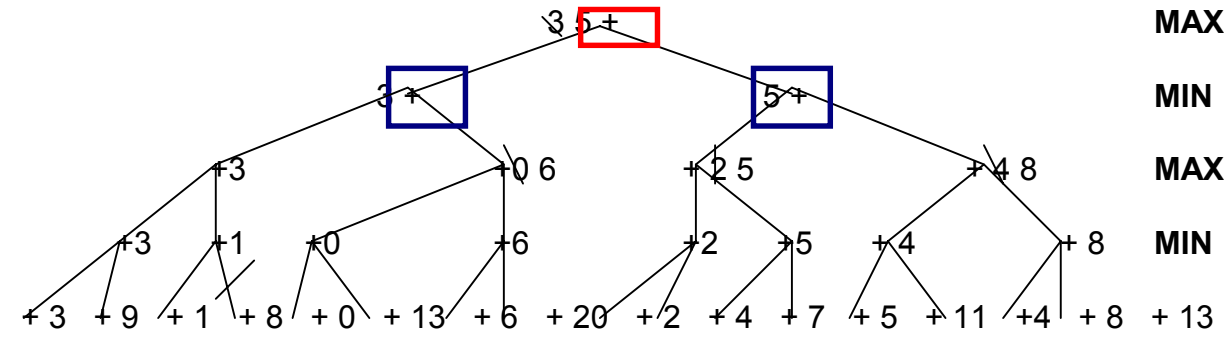


Exemple:



Exemple:





6 - Brève introduction aux structures ordonnées

6 - 1 Relation d'équivalence et partitions

Une relation d'équivalence R sur un ensemble E est:

(I) **reflexive:** $\forall x \in E, x R x$

(II) **symétrique:** $\forall x, y \in E: x R y \Rightarrow y R x$

(III) **transitive:** $\forall x, y, z \in E: x R y$ et $y R z$ alors $x R z$

Une relation d'équivalence induit une **partition en classes** de E, ou la classe de x est l'ensemble des éléments équivalents à x, un élément a une classe et une seule; les classes sont disjointes et leur réunion est égale à E.

Notation: La partition déterminée R se note E/R et s'appelle **espace quotient**

6 - 2 Relation d'ordre et ensemble ordonné

Une d'ordre ρ sur un ensemble E est:

(I) **reflexive:** $\forall x \in E, x \rho x$

(II) **transitive:** $\forall x, y, z \in E: x \rho y$ et $y \rho z$ alors $x \rho z$

(III) **antisymétrique:** $\forall x, y \in E: x R y \Rightarrow y R x$

Notation: On mote souvent une relation d'ordre avec $\leq$ ou $\subseteq$.

Un ensemble muni d'une relation d'ordre $(E, \leq)$ s'appelle **un ensemble ordonné**

L'ordre est **total** si tous les éléments sont comparables, sinon il est **partiel**.

Exemples: Un ordre total: $(N, \leq)$ Un ordre partiel: $(P(E), \subseteq)$

$E = \{1, 2, 3\}$ $P(E) = \{0, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$

$\{1\} \subseteq \{1, 2\}$ $\{1\} \not\subseteq \{2, 3\}$

$\{1, 3\}$

2 ordres ρ_1 et ρ_2 sont **comparables** si $\forall x, y \in E: x \rho_1 y \Rightarrow x \rho_2 y$ on dit que ρ_1 est inclus dans ρ_2

6 - 3 Maximaux, Minimaux, Treillis

Définition: x est un **majorant** d'une partie X de E si $\forall y \in X, y \leq x$

x est un **minorant** d'une partie X de E si $\forall y \in X, x \leq y$

x est un élément **maximal** d'une partie X si:

- x est un **majorant** de X, $x \in X$
- $\nexists z \in X: x \leq z$

x est un élément **minimal** d'une partie X si:

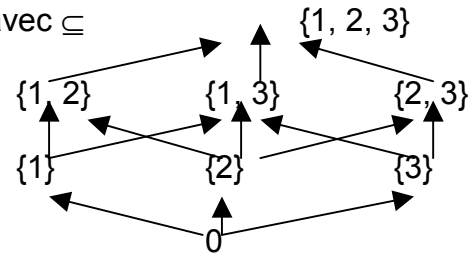
- x est un **minorant** de X, $x \in X$
- $\nexists z \in X: z \leq x$

Définition: Treillis

Un trellis est un ensemble ordonné $(E, \leq)$ tel que tout couple d'éléments (x, y) possède une **borne supérieure** (un plus petit majorant) notée $x \vee y$ ("x ou y")et une **borne inférieure** (un plus grand minorant) notée $x \wedge y$ ("x et y").

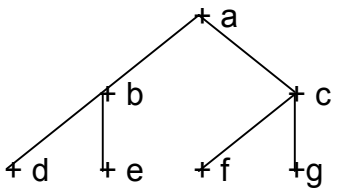
Borne supérieure: ➔ sup - demi - treillis borne inférieure: ➔ inf - demi - treillis

Exemple: L'ensemble P(E) des parties de E avec $\subseteq$



trellis ⟶ = inclus dans

- Un **arbre** est un sup - demi - treillis où tous les éléments sauf 1, la racine, possèdent un ascendant unique.



⟶ = inclus dans

$\begin{cases} a \vee b = a \\ a \wedge b = b \end{cases}$ $\forall$ x et y dans l'arbre

si $x \vee y = x$ alors $x \wedge y = y$

➔ vrai pour les éléments sur une même branche

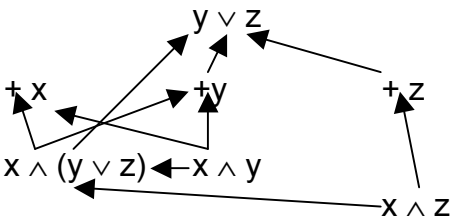
6 - 4 Treillis particuliers

1) Treillis distributif

Un trellis est distributif si les opérations sup et inf sont distributives l'une par rapport à l'autre.

$\begin{cases} x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) \\ x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) \end{cases}$

et ou et ou et



2) Treillis complémenté

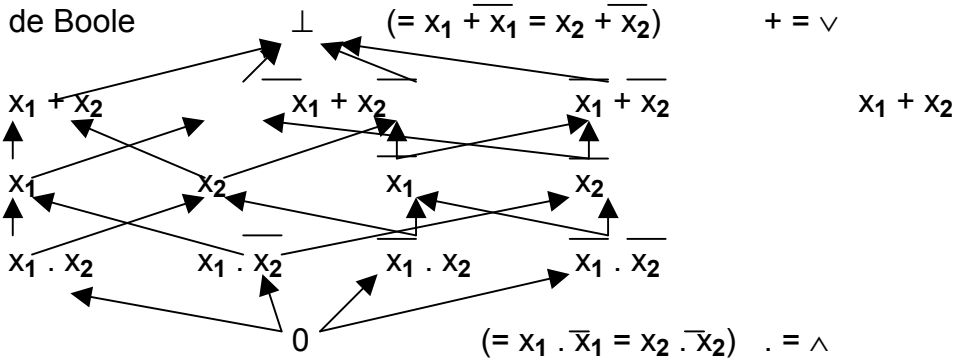
Un trellis est complémenté s'il possède un élément maximalnoté T, un élément minimal noté $\perp$, et si tout élément $\wedge$ possède un complémentaire x qui vérifie:

$\begin{cases} x \vee \overline{x} = T \\ x \wedge \overline{x} = \perp \end{cases}$

Un trellis complémenté s'appelle un trellis de Boole

Théorème: Dans un trellis de Boole, tout élément a un complémentaire unique

Trellis de Boole



Définition: Algèbre de Boole

Une algèbre de Boole est un trellis de Boole qui vérifie:

(I) $\vee$ et $\wedge$ sont commutatives et associatives

(II) $\vee$ et $\wedge$ sont idempotentes { $x \vee x = x$; $x \wedge x = x$

(III) $\vee$ et $\wedge$ sont absorbantes { $x \vee (y \wedge z) = x \; \forall \; y$; $x \wedge (y \vee z) = x \; \forall \; y$

Introduction au calcul des propositions

0 - Rappels des treillis à l'algèbre de Boole

(E, $\geq$) treillis: $x \vee y$ borne supérieure

$x \wedge y$ borne inférieure

complémenté + { treillis T(1), $\perp$(0), complémenté : x a un complémentaire $\overline{x}$

$x \vee \overline{x} = T$

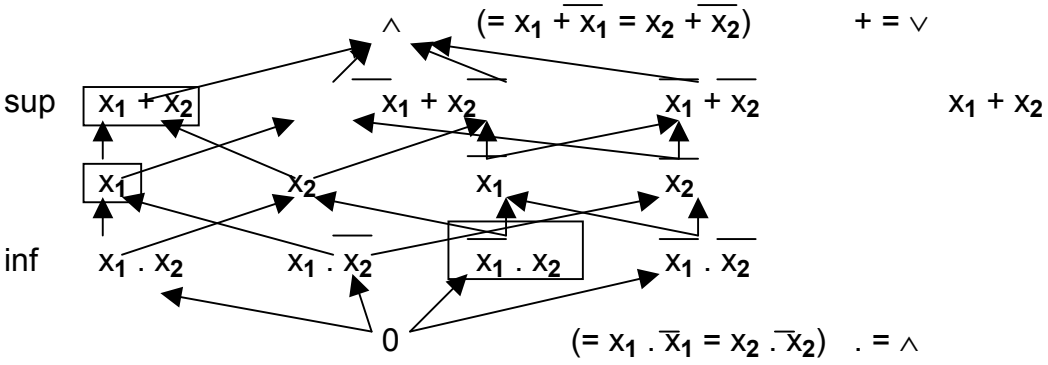
$x \wedge \overline{x} = \perp$

distributif = treillis de Boole + { treillis distributif : $\vee$ et $\wedge$ sont distributives

→ Algèbre de Boole; (B, 0, 1, +, ., -)

- treillis de Boole
- + et . sont associatives et commutatives
- + et . sont idem potentes { $x + x = x$; $x \, . \, x = x$
- + et . sont absorbantes { $x + x \, . \, y = x$; $x \, . \, (x + y) = x$

Exemple du treillis des connecteurs booléens (2 variables)



Remarque sur le treillis: sens $\longrightarrow$ dans le treillis

Ordre partiel

➔ soit l'inclusion des supports

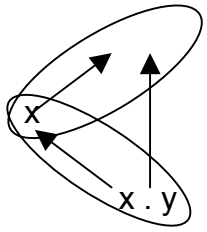
➔ soit l'implication booléenne

Support d'une fonction booléenne

Définition: soit $f \in F_n$ (Fonction booléenne à n variables)

Le support de f noté Sf, est l'ensemble des points des $B_n = \{0, 1\}^n$ ou f vaut 1

Propriétés sur le support:



$Sx = x + x \cdot y$

1

$x_1 \vee x_1 \vee x_2$

$x_1 \rightarrow x_1 \vee x_2$

$\mathbf{[x_1 \rightarrow x_1 + x_2]}$

$Sf + g = Sf \cup Sg$

$Sf \cdot g = Sf \cap Sg$

$\overline{Sf} = L_{Bn} \setminus Sf$

pour le treillis

$B_2 = \{0, 1\}^2 = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$

$x_1 \rightarrow x_1 + x_2$ $\quad x_1 \rightarrow x_1 \vee x_2$

$S_{x1} = \{(1, 0), (1, 1)\}$

$\subseteq$

$S_{x1 + x2} = \{(1, 0), (1, 1), (0, 1)\}$

$\overline{x_1} \cdot x_2 \leq x_1 \leq x_1 + \overline{x_1} \cdot x_2 \leq 1 \leq x_1 + x_2$

$x_1 + \overline{x_1} \cdot x_2 \geq \overline{x_1} \cdot x_2$

$x_1 + \overline{x_1} \cdot x_2 \geq x_1 \cdot x_2$

$x_1 \geq x_1 \cdot x_2$ $\quad x_1 + \overline{x_1 \cdot x_2} = x_1 + x_2$

$x_1 + \overline{x_1} \cdot x_2 \geq x_2 = \overline{x_1} \cdot x_2 + x_1 \cdot x_2$

Propriétés de treillis:

$a \geq b$ et $a \geq c$ alors $a \geq b \vee c$

$a \leq b$ et $a \leq c$ alors $a \leq b \wedge c$

Définitions formes normales de fonctions booléennes:

(I) Forme normale conjonctive

Toute fonction booléenne à n variables peut se mettre sous la forme d'une somme de **monômes canoniques conjonctifs**.

Un monôme est **canonique** s'il contient toutes les fonctions projections (tous les x_i)

Il est conjonctif si c'est un produit.

Ex: $x_1 \cdot x_2$ $\quad x_1 \cdot \overline{x_2} \overline{x_1} \cdot x_2 \overline{x_1} \cdot \overline{x_2}$

$f(x_1, \ldots, x_n) = \sum f(\varepsilon_i, \ldots \varepsilon_n)\, x_1^{\varepsilon 1} \cdot x_2^{\varepsilon 2} \cdots x_n^{\varepsilon n}$ $\quad (\varepsilon_i, \ldots \varepsilon_n) \in B_n$

$x_i^{\varepsilon i} = x_i^{\varepsilon i} = 1$ alors x_i

sinon $\overline{x_i}$ $\quad$ monôme canonique conjonctif

$f(x_1, x_2, x_3) = x_1 \cdot x_2 + \overline{x_1} \cdot x_3$

B3 $\quad Sf = \{(1, 1, 0), (1, 1, 1), (0, 1, 1), (0, 0, 1)\}$

8 points

$f(x_1, x_2, x_3) = x_1^1 \cdot x_2^1 \cdot x_3^0 + x_1^1 \cdot x_2^1 \cdot x_3^1 + x_1^0 \cdot x_2^1 \cdot x_3^1 + x_1^0 \cdot x_2^0 \cdot x_3^1$

$f(x_1, x_2, x_3) = x_1 \cdot x_2 \cdot \overline{x_3} + x_1 \cdot x_2 \cdot x_3 + x_1 \cdot x_2 \cdot \overline{x_3} + \overline{x_1} \cdot x_2 \cdot x_3$

(II) Forme normale disjonctive

Toute fonction booléenne à n variables peut se mettre sous la forme d'un produit de monômes **canoniques disjonctifs**

Un monône est disjonctif si c'est une somme de fonctions projection $\overline{x_i}$ ou x_i

$f(x_1, \ldots, x_n) = \prod f(\varepsilon_i, \ldots \varepsilon_n) + x_1^{\overline{\varepsilon 1}} + x_2^{\overline{\varepsilon 2}} + \cdots + x_n^{\overline{\varepsilon n}}$ $\quad (\varepsilon_i, \ldots \varepsilon_n) \in B_n$

$= \prod (x_1^{\varepsilon 1} + x_2^{\varepsilon 2} + \cdots + x_n^{\varepsilon n})$ $\quad (\varepsilon_i, \ldots \varepsilon_n) \notin Sf$

$f(x_1, x_2, x_3) = x_1 \cdot x_2 + \overline{x_1} \cdot x_3$

$Sf = \{(0, 0, 0), (1, 0, 0), (1, 0, 1), (0, 1, 0)\}$

$f(x_1, x_2, x_3) = (\overline{x_1}^0 + \overline{x_2}^0 + \overline{x_3}^0) . (\overline{x_1}^1 + \overline{x_2}^0 + \overline{x_3}^0) . (\overline{x_1}^1 + \overline{x_2}^0 + \overline{x_3}^1) . (\overline{x_1}^0 + \overline{x_2}^1 + \overline{x_3}^0)$

$f(x_1, x_2, x_3) = (x_1 + x_2 + x_3) . (\overline{x_1} + x_2 + x_3) . (x_1 + \overline{x_2} + x_3) . (\overline{x_1} + x_2 + x_3)$

Remarque: Toute fonction booléenne peut s'écrire comme une somme ou un produit de fonctions projections positives ou négatives.

Composition de fonctions booléennes et famille génératrice

$f \in F_n$, g_1, …. $g_n \in F_k$

$h(x_1, \ldots x_k) = f\ (g_1\ (x_1 \ldots. x_k), \ldots\ldots , (g_n\ (x_1 \ldots. x_k))$

h est une **fonction composée** de f et des g_i, avec $h \in F_k$, soit F une famille de fonctions booléennes.

Comp (F): ensemble de toutes les fonctions bolléennes qui peuvent être construites en composant les fonctions de F et les projections (positives ou négatives).

Une famille F est génératrice si: Comp (F) = $\cup F_r$ $\quad 1 \geq 0$

Exemple: $\{+, ., \overline{\ }\}$ (formes normales conjonctives ou disjonctives)

$\{+, \overline{\ }\}$ $\}$ familles génératrices minimales
$\{. , \overline{\ }\}$ $\}$

$\overline{x_1 + x_2} = \overline{x_1} . \overline{x_2}$ $\quad x_1$ XOR x_2

$\{+, \overline{\ }\} \rightarrow .$

$\overline{x_1 . x_2} = \overline{x_1} + \overline{x_2}$

$\{. , \overline{\ }\} \rightarrow +$

$\{0, \rightarrow\} x_1 = x_1 \rightarrow 0$ même support

$x_1 + x_2 = \overline{x_1} \rightarrow x_2$

$x_1 . x_2 = [(x_1 \rightarrow 0) \rightarrow (x_2 \rightarrow 0)] \rightarrow 0$

$x_1 \rightarrow x_2$

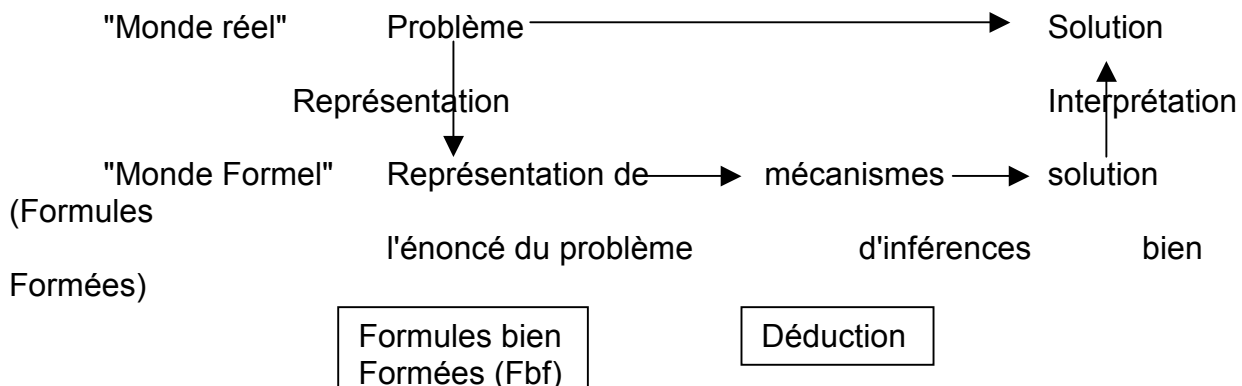
x_1	x_2	$x_1 \rightarrow x_2$
0	0	1
0	1	0
1	0	1
1	1	1

$x_1 \rightarrow 0 = 1$ (0, *) $\quad 0 \rightarrow 0$ } vrai
$x_1 \rightarrow x_2 = 1$ (0, 0), (0, 1), (1, 1) $\quad 0 \rightarrow 1$ }
$x_1 \rightarrow x_2 = 0$ (1, 0)

$\overline{0} \rightarrow 1$ }
$x \rightarrow y = \overline{x} + y$ $\quad \overline{0} \rightarrow 1$ }

1 - Introduction à la problématique du calcul des propositions

(I) un point de vue logique sur la résolution de problèmes



de Fbf vraies on construit des Fbf vraies

(II) Introduction intuitive

Le CP0 (calcul des propositions 0) correspond à l'étude de la logique sans variable ou les constituants élémentaires de la logique sont des **propositions** de forme: [sujet verbe complément]

Une proposition est **insécable** et ne peut prendre que 2 valeurs vrai ou faux

Proposition + connecteurs booléens
- insécable non séparables

Exemple de propositions

Il - fait - noir
Le - garçon - énerve - la - fille
Le - cours - n'est - pas - facile ou les - élèves - sont - fatigués
Bleu et vert

(III) Problèmes qui se posent en CP0

+ représenter un énoncé à l'aide des Formules du CP
+ Savoir si une formule est toujours vraie
+ Savoir si une formule se déduit d'un ensemble de formules

2 - Le langage du calcul des propositions

$P = \underbrace{\{a, b, c, ...\}}_{\text{Symboles de propositions}} \cup \{T, \bot\}$

P = ensemble de variables propositionnelles qui ont la particularité de ne pouvoir prendre que les valeurs vrai (1) ou faux (0).

B = {$\vee$, $\wedge$, ➔, ←➔, ¬} ou = $\vee$ et = $\wedge$ non = ¬ ( ) = 2 parenthèses

Définition: L'ensemble des **propositions** noté Prop est le plus petit ensemble tel que:

+ $P \subseteq$ Prop
+ si φ et $\psi \in$ Prop, alors ¬φ et ¬ψ φ {$\vee$, $\wedge$, ➔, ←➔, ¬} ψ sont dans Prop

Exemple: a $\vee$ b beau $\wedge$ violet p ➔ q jaune $\wedge$ rouge ➔ orange

Remarque:

(1) Il est possible de prendre 1 ensemble minimal de connecteurs booléens. Par exemple {$\vee$, ¬} (famille génératrice).

Une formule du CP0 est donnée par la grammaire: F ➔ a $\in$ P | ¬F | F $\vee$ F

Les autres connecteurs booléens apparaissent comme des abréviations:

$\varphi \wedge \psi$ = ¬(¬$\varphi \vee$ ¬ψ)
φ ➔ ψ = ¬$\varphi \vee \psi$
φ ←➔ ψ = (φ ➔ ψ) $\wedge$ (ψ ➔ φ)

Une formule construite à partir de $\vee$ et ¬ s'appelle une **clause**.

(2) Ecriture et sens des Formules

Une formule du CP0 correspond à l'écriture d'une fonction booléenne.

Il faut aussi se donner les moyens d'associer un sens aux Formules c'est le rôle de **l'interprétation**

Par exemple: $\underbrace{1 + 1 = I + I}_{\text{Vraie dans certains cas I interprété comme 1}} = 2 = II$ écriture = syntaxique

1 + 1 = 2 N et addition des entiers sémantique de l'écriture

3. - Aspects sémantiques du CP0

Les aspects sémantiques consernent la façon d'associer une valeur de vérité à une Formule du CP0.

Pour associer cette valeur de vérité, on utilise les notions de **valuation** - **interprétation** - et de **domaine d'interprétation**

Pour une formule contenant n variables proposionnelles, la sémantique de la formule va être donnée par un tableau à 2^n lignes et p colonnes ou p désigne le nombre de constituants de la Formule.

Le tableau s'appelle une table de vérité et est gérée avec les règles que nous allons voir.

3 - 1 Valuations

Définition: Une valuation sert à associer une valeur de vérité à une formule du CP0 une fois que la valeur de vérité des constituants a été fixée.
La valeur de la formule s'obtient avec les règles avec les règles suivantes:

$\vee$ = Prop ➔ {0, 1}

+ $\vee$ (T) = 1 et $\vee$ ($\perp$) = 0
+ $\vee$ ($\varphi \vee \psi$) = max ($\vee$ (φ), $\vee$ (ψ))
+ $\vee$ ($\varphi \wedge \psi$) = min ($\vee$ (φ), $\vee$ (ψ))
+ $\vee$ ($\neg\varphi$) = 1 - $\vee$ (φ)
+ $\vee$ (φ ➔ ψ) = 0 si $\vee$ (φ) = 1 et $\vee$ (ψ) = 0 sinon 1
+ $\vee$ (φ ←➔ ψ) = 1 si $\vee$ (φ) = $\vee$ (ψ)

Exemples : Table de vérité de ($\neg$p ➔ $\neg$q) ←➔ (p ➔ q) = φ
($\neg$q ➔ $\neg$p) ←➔ (p ➔ q) = ψ

p	q	$\neg$p	$\neg$q	$\neg$p ➔ $\neg$q	p ➔q	φ	$\neg$q ➔ $\neg$p	ψ
0	0	1	1	1	1	1	1	1
0	1	1	0	0	1	0	1	1
1	0	0	1	1	0	0	0	1
1	1	0	0	1	1	1	1	1

ψ est une tautologie

3 - 2 Propriétés sémantiques des Formules

(I) Une formule φ est dite satisfiable s'il existe une valuation qui lui donne la valeur 1
Une telle valuation s'appelle un **modèle de φ**.
Par exemple, la valuation p, q (0, 0) est un **modèle de φ**.

Une formule qui n'est pas satisfiable est dite aussi contradictoire.
Par exemple: p $\wedge$ $\neg$p n'est jamais satisfiable

(II) Une formule est une tautologie si elle est satisfiable pour toute valeur. Si φ est un tautologie, on note |= φ
Par exemple: p $\vee$ $\neg$p est une tautologie

(III) Soient {φ_1 …. φ_k} un ensemble de Formules et φ une formule.

La Formule φ est une conséquence sémantique de l'ensemble $\{\varphi_1 \ldots. \varphi_k\}$, ce qui se note $\{\varphi_1 \ldots. \varphi_k\} \models \varphi$ si tous les modèles de $\{\varphi_1 \ldots. \varphi_k\}$ sont des modèles de φ

Remarque: un modèle de $\{\varphi_1 \ldots. \varphi_k\}$ n'est autre qu'un modèle de $\varphi_1 \wedge \varphi_2 \wedge \ldots. \varphi_k$ = les φ_i doivent être vraies toutes simultanément.

(IV) Equivalence et incompatibilité

2 formules φ et ψ sont **équivalentes sémantiquement si $\varphi \models \psi$ et $\psi \models \varphi$**

2 formules φ et ψ sont **incompatibles** si elles n'ont aucun modèle commun

Exemple: $p \rightarrow q$ et $\neg p \vee q$ sont sémantiquement équivalentes

p	q	$p \rightarrow q$	$\neg p$	$\neg p \vee q$
0	0	1	1	1
0	1	0	1	1
1	0	1	0	0
1	1	1	0	1

$\neg p$ et p sont incompatibles

(V) $\{\varphi_1 \ldots. \varphi_k\} \models \varphi$ Modèles $\{\varphi_1 \ldots. \varphi_k\} \subseteq$ Modèles (φ)

Modèles $\{\varphi_1 \ldots. \varphi_k\} \cap$ Modèles $(\neg\varphi) = 0$

incompatible

$\{\varphi_1 \ldots. \varphi_k\} \models \varphi$ ssi $\{\varphi_1 \ldots. \varphi_k, \neg\varphi\}$ est contradictoire, il n'existe aucun modèle pour $\varphi_1 \wedge \varphi_2 \ldots. \wedge \varphi_k \wedge \neg\varphi$

- Procédé construction de Formules
- Valuation ➔ modèle et tautologie (toujours vrai) $\models \varphi$

Aspects Sémantiques Valuation: $\{\varphi_1 \ldots. \varphi_k\} \models \varphi$ ssi $\{\varphi_1 \ldots. \varphi_k, \neg\varphi\}$ est contradictoire

$p \vee \overline{p}$

4. - Aspects syntaxiques du CP0

- Savoir si une Formule est toujours vraie (sans passer par la sémantique)?
- Savoir si une Formule se déduit d'un ensemble de Formules

+ Système Formels] cadre syntaxique

+ La Résolution] 2 méthodes de preuve dans 1 cadre syntaxique

+ La méthode des tableaux syntaxiques

4 - 1 Notion de système Formel

Définition:

(I) Un système formel est constitué de:

- 1 alphabet fini de symboles, (P)
- processus de construction de Formules bien formées F ➔ $a \in P \quad | \neg F | F \vee F$
- 1 ensemble d'axiomes
- 1 ensemble de règles d'inférence de la forme $\frac{p,\ p \rightarrow q}{q}$

  $\alpha_1, \alpha_2, \ldots. \alpha_n \vdash \beta_1, \beta_2, \ldots. \beta_n$

  où les α_i et β_j sont des FbF

  $\vdash$ dérivation syntaxique ($\models$)

Un système Formel pour le CP0

+ P: alphabet de symboles propositionnels

+ F → a ∈ P | ¬F | F ∨ F
+ axiomes
  (A1) p → (q → p)
  (A2) (p → (q → r)) → ((p → q) → (p → r))
  (A3) ((¬p → ¬q) → ((¬p → q) → q))
+ règles d'inférence $\frac{p,\ p \rightarrow q}{q}$ **modus ponens**
+ règles de démonstration:
  Toutes les occurrences d'un symbole dans une Formule peuvent être remplacées par la même Formule
+ règles de simplification
  $p \vee \neg q \vee \neg p$ $p \vee p \vee \neg q$ $p \vee \neg q$ $p \vee \neg q$ Idempotence

Remarque sur les axiomes

p	q	p → q	p → (q → p)
0	0	1	1
0	1	0	1
1	0	1	1
1	1	1	1

→ tautologie

(II) Notion de démonstration

Une démontration dans 1 système Formel est 1 suite finie de formules φ_1, …. φ_n où chaque φ_i est soit 1 axiome soit le résultat de l'appliccation d'une règle d'inférence à φ_j, avec j < i

La dernière Formule, φ_n, s'appelle un **théorème** (le théorème est le correspondant syntaxique de la notion de tautologie)

Exemple de démonstration dans le système Formel

Demontrer p |— p est théorème

(φ_1) (A_1) p → (q → p)

(φ_2) substitution de q par p → q dans (A_1) = (p → ((p → q) → p)

(φ_3) (A_2) p → (q → r) → ((p → q) → (p → r)

(φ_4) substitution de q par p → q et de r par p dans (A_2): (p → ((p → q) → p)) → (p → (p → q)) → (p →p))

(φ_5) modus ponens à (φ_2) et (φ_4) (p → (p → q) → (p →p)

(φ_6) substitution de q par p dans (A_1): (p → (p → p))

(φ_7) modus ponens à (φ_5) et (φ_6) (p → p): théorème

4 - 2 Correction, complétude et décidabilité

Définition:

(I) **Correction:** Un système Formel pour le CP0 possède la propriété de correction (est correct) si les théorèmes correspondent aux tautologies (pour la notion de valuation associée au CP0). |— φ alors |= φ
théorème tautologie

(II) **Complétude:** Un système Formel pour les CP0 possède la propriété de **complétude** (ou est **complet**) si les tautologies correspondent aux théorèmes $\models \varphi$ alors $\vdash \varphi$

Si une Formule est une tautologie, alors il est possible de la prouver syntaxiquement.

Remarque: Correction et complétude sont des propriétés qui se généralisent à tout système Formel.

Théorème: Le système Formel associé au CP0 est correct et complet

(III) **Décidabilité:** Une logique est décidable s'il existe une procédure permettant de séparer les théorèmes des non théorèmes.

Théorème: CP0 est décidable

Remarque: Il existe plusieurs algorithmes pour prouver qu'une Formule est théorème, mais ces algorithmes sont de complexité exponentielles: le problème de la satisfiabilité d'une formule boléenne est NP - complet

Soit φ une formule de CP0

Pour savoir si φ est une tautologie, il suffit de calculer la valuation de φ sur chacun des points de B_n, où n est le nombre de variables propositionnelles constituant φ. Or le nombre de calculs à faire est 2^n.

4 - 3 Formes normales et notion de clause

Définition:

(I) Un **littéral** désigne une variable propositionnelle sous forme positive ou négative.

Une clause est une disjonction de littéraux: $C = a_1 \vee a_1 \vee \ldots \vee a_n \vee \neg b_1 \vee \neg b_2 \vee \ldots \vee \neg b_m$

Théorème: Toute formule du CP0 peut se mettre sous le Forme d'une conjonction de clauses.

$\varphi = c_1 \wedge c_2 \wedge \ldots \wedge c_k$ où les c_i sont des clauses.

Cette conjonction de clauses s'appelle **Forme normale** ou **Forme clausale** de φ

Ce théorème repose sur le fait que $\{\neg, \vee, \wedge\}$ est **génératrice**.

(II) Clauses particulières:

+ Une formule dont la Forme normale est vide est une tautologie.

Pourquoi: $p \vee \neg p$ se réécrit "1"

$c \wedge 1$ se réécrit c ➔ 1 est "simplifié"

➔ une formule dont la forme normale est vide est une disjonction de 1, donc une tautologie.

+ Formes simples pour une clause

a
$\neg b$
$a \vee \neg b$
} une clause est toujours satisfiable

Il existe une seule clause qui n'est jamais satisfiable et qui s'appelle la **clause vide**. se note

4 - 4 Normalisation d'une Formule du CP0

Algorithme de normalisation

(1) Traitement des abréviations

Exemple: (¬p ➔ q) ∨ (q ➔ ¬p)
(1) (¬(¬p) ∨ q) ∨ (¬q ∨ ¬p)
(2) (p ∨ q ∨ ¬q ∨ ¬p)
(3) p ∨ q ∨ ¬q ∨ ¬p ➔ 1 la clause se simplifie ➔ théorème En particulier cette formule est une tautologie

Abréviations:

p ➔➔ p se réécrit (p ➔ q) ∧ (q ➔ p)
p ➔ q se réécrit ¬p ∨ q

(2) Glissement des négations à l'intérieur des parenthèses et traitement des négations

$\neg(p \vee q)$ se réécrit $\neg p \wedge \neg q$

$\neg(p \wedge q)$ se réécrit $\neg p \vee \neg q$

$\neg\neg p$ se réécrit p

(3) Distributivité de $\wedge$ et $\vee$

$p \vee (q \wedge r)$ se réécrit $(p \vee q) \wedge (p \vee r)$

$p \wedge (q \vee r)$ ← est déjà une forme clausale ne doit pas être modifiée

(4) Simplification: $p \vee 1$ se réécrit 1 et se simplifie

$p \vee p$ se réécrit p

$\neg p \vee \neg p$ se réécrit $\neg p$

(5) **Notation:** $\varphi = c_1 \wedge c_2 \wedge \ldots \wedge c_n$

φ: c_1

c_2

c_n ou on écrit une clause par ligne

Remarque sur l'exemple:

p	q	$\neg p$	$\neg p \rightarrow q$	$q \rightarrow \neg p$	φ
0	0	1	0	1	1
0	1	1	1	1	1
1	0	0	1	1	1
1	1	0	1	0	1

→ tautologie

Un autre exemple: $\neg(\neg(q \vee r \rightarrow p) \rightarrow \neg(q \wedge t \rightarrow p)$

$\neg(\neg\neg(\neg(q \vee r) \vee p) \vee \neg(\neg q \wedge t) \vee p)$

$\neg((\neg q \wedge \neg r) \vee p) \vee ((q \wedge t) \wedge \neg p)$

$(q \vee r) \wedge \neg p \vee (\neg q \vee \neg t \vee p)$

$\begin{cases} q \vee r \\ \neg p \\ \neg q \vee \neg t \vee p \end{cases}$

4 - 5 Le principe de résolution

Présentation intuitive

(C_1) $a \vee b$ **Attention:** la résolution porte sur **une** variable propositionnelle

(C_2) $\neg a \vee c$ $a \neq b, \neg b, c \neg c$

(C_3) $a \vee c$

Le principe de résolution consiste à dériver la clause (C_3) des clauses (C_1) et (C_2). La clause (C_3) s'appelle la **résolvante** de (C_1) et (C_2).

Schémas d'inférence associés à la résolution: $\neg a \vee b \vee c,\ a \vee d$ résolution en général

$$\frac{}{b \vee c \vee d}$$

$\frac{\neg a \vee b,\ a}{b}$ **modus ponens** $\frac{\neg a \vee b,\ \neg b}{\neg a}$ **modus tollens** (preuve par l'absurde)

Notion de démonstration en utilisant le principe de résolution

(I) **Preuve de réfutation:** Soit S un ensemble de clauses
Une **réfutation** de S consiste à dériver la clause vide à partir de S

(II) Résolution dans le CP0
Théorème: Soit S un ensemble de clauses **S est non satisfiable ssi S |—**
Conséquence: $\{\varphi_1 \ldots. \varphi_n\} \models \varphi$ ssi $\{\varphi_1 \ldots. \varphi_k, \neg\varphi\}$ non satisfiable.
$\{\varphi_1 \ldots. \varphi_k, \neg\varphi\}$ non satisfiable ssi $\{\varphi_1 \ldots. \varphi_k, \neg\varphi\} \vdash$
$\{\varphi_1 \ldots. \varphi_k, \neg\varphi\} \models \varphi$ ssi $\{\varphi_1 \ldots. \varphi_k, \neg\varphi\} \vdash$

Procédé pour démontrer qu'une formule se déduit d'un ensemble de formules (on supposera que les formules sont sous forme normale)
φ = tautologie **$\models \varphi$ ssi $\neg\varphi \vdash$** ➔ φ est une tautologie ssi on peut dériver à partir de $\neg\varphi$

(III) Exemple et modalité opérationnelles

…
p
…
$\neg$p
…

$\neg(\neg(q \vee r \rightarrow p) \rightarrow \neg(q \wedge t \rightarrow p)$

1. $q \vee r$
2. $\neg p$
3. $\neg q \vee \neg t \vee p$

Résolution sur 1 et 3 pour q 4. $r \vee \neg t \vee p$
Résolution sur 2 et 3 pour p 5. $\neg q \vee \neg t$

Cette formule n'est pas une tautologie parce qu'on n'arrivera jamais à dériver

Exemple: Montrer que $\{a \vee b \vee \neg d,\ \neg a \vee c \vee \neg d,\ \neg b,\ d\} \models c$
ssi $\{a \vee b \vee \neg d,\ \neg a \vee c \vee \neg d,\ \neg b,\ d,\ \neg c\} \vdash$

1	$a \vee b \vee \neg d$	Résolution sur 1 et 3 pour b:	6	$a \vee \neg d$	
2	$\neg a \vee c \vee \neg d$				$\neg b$
3	$\neg b$	Résolution sur 4 et 6 pour d:	7	a	
4	d	Résolution sur 2 et 7 pour a:	8	$c \vee \neg d$	
5	$\neg c$	Résolution sur 4 et 8 pour d:	9	c	
		Résolution sur 5 et 9 produit:			

Exemple: $\varphi = (q \vee (r \rightarrow (s \rightarrow (p \vee (s \rightarrow q))))) \vee (r \rightarrow (\neg p \rightarrow (\neg s \vee q)))$
Est - ce que φ est une tautologie
- Forme clausale de $\neg\varphi$
- $\neg\varphi \vdash$

$\neg\varphi = \neg[(q \vee (r \rightarrow (s \rightarrow (p \vee \underbrace{(s \rightarrow q)}_{\neg s \vee q}))))) \vee (r \rightarrow (\underbrace{\neg p \rightarrow (\neg s}_{s \vee \neg q} \vee q)))]$

$\underbrace{p \vee \neg s \vee q}_{\neg(\neg p) \vee \neg s \vee q}$

$\underbrace{\neg s \vee p \vee \neg s \vee q}_{\neg r \vee p \vee \neg s \vee q}$

$\underbrace{\neg r \vee \neg s \vee p \vee \neg s \vee q}_{\vee\ \neg r \vee p \vee \neg s \vee q}$

$\neg(q \vee \neg r \vee \neg s \vee p \vee \neg s \wedge q \quad \vee \quad \neg r \vee p \vee \neg s \wedge q) \quad = \neg q \wedge r \wedge s \wedge \neg p$

¬q
r
s
¬p ⊢ φ n'est pas une tautologie

p	q	r	s	¬r	¬s	φ
0	0	0	0	1	1	1
0	0	1	1	0	0	0

$= q \vee \neg r \vee \neg s \vee p$

2^4 lignes

➔ φ n'est pas une tautologie

$\varphi = (r \rightarrow (s \rightarrow (p \wedge \underbrace{(s \rightarrow q)}_{\neg s \vee q})))) \vee (r \rightarrow \underbrace{(p \rightarrow (s \vee q)}_{\neg p \vee s \vee q}))$

$\underbrace{\neg p \wedge (\neg s \vee q)}$ $\quad \neg r \vee \underbrace{\neg p \vee s \vee q}$

$\underbrace{\neg s \vee p \wedge (\neg s \vee q)}$ $\quad \vee \neg r \vee \neg p \vee s \vee q$

$\underbrace{\neg r \vee \neg s \vee p \wedge (\neg s \vee q)}$ $\quad \vee \quad \neg r \vee \neg p \vee \neg s \vee q$

$\neg r \vee \neg s \vee p \wedge (\neg s \vee q) \quad \vee \quad \neg r \vee \neg p \vee \neg s \vee q$

$\neg r \vee \neg s \vee (p \wedge \neg s) \vee (p \wedge q) \vee \neg p \vee q$

$\Psi = \neg (\neg r \vee \neg s \vee (p \wedge \neg s) \vee (p \wedge q) \vee \neg p \vee q) \quad = r \wedge s \wedge (\neg p \vee s) \wedge (\neg p \vee \neg q) \wedge p \wedge \neg q$

r
s
¬p ∨ s
¬p ∨ ¬q
p
¬q Ψ n'est pas une tautologie

Systèmes Formels - Méthode de résolution pour le CP0

5. - Rappel sur les systèmes Formels "Aspects syntaxiques des Formules"

Un système Formel est composé de:

(I) Un alphabet fini de symboles (P = {a, b, c ….. } variables propositionnelles)

(II) Un procédé de construction fbf ($X \rightarrow a \in P \mid \neg X \mid X \vee X$)

(III) Ensenble d'axiomes (éventuellement vide)

(IV) Règles d'inférence

+ règles de simplification: $a \vee a \vee b \quad a \vee b$

$\neg a \vee \neg a \vee b \quad \neg a \vee b$

Remarque: les 2 règles se retrouvent dans la construction d'une forme clausale

+ $\dfrac{a,\ a \rightarrow b}{a}$ modus ponens $\qquad \dfrac{a \vee b,\ \neg a \vee c}{b \vee c}$

Résolution:

Exemple de système Formel

+ P = {a, b, c}

+ Processus de construction: φ est une fbf ssi $\varphi \in \{a, b, c\}^* \lambda \in \{a, b, c\}^*$

$\overline{\text{mot vide}}$ λ a = λ a = a

+ Axiome: a b a c a a ➔ forme donnée qui intervient dans les démonstrations
+ Rèles d'inférences

(R1) Si A c B est un thèorème où A, B $\in$ {a, b, c} * alors a A c B a est un théorème
(R1) Si A c B est un thèorème où A, B $\in$ {a, b, c} * alors A a c a B est un théorème

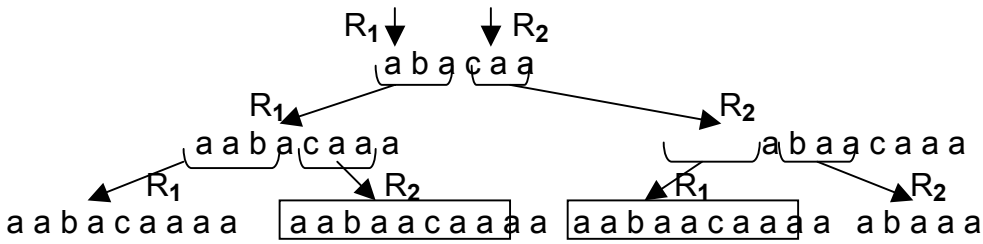
Question:

(I) Est- ce que a a b a a c a a a a est un théorème ?
(II) a b a a c a a a a est un théorème ?
(III) a b a b a c a a a est un théorème ?

?

(I) a b a c a a |— a a b a a c a a a a

R_1, R_2 A B



a a b a a c a a a a est un théorème Formel:

a b a c a a a a b a c a a a, a b a a c a a a, a a b a a c a a a a, a a b a a c a a a a

φ_1 φ_2 (R_1) φ_3 (R_2) φ_4 (R_2)
φ_5 (R_1)

(II) a b a c a a ➔ a b a a c a a a a ➔ a b a a a c a a a a
(R_2) (R_1)

Il n'existe aucune séquence d'application des règles permettant de prouver a a b a a c a a a a est un théorème

Remarque: l'axiome contient un nombre pair de a et les 2 règles d'inférences ajoutent chacune 2 "a", donc le nombre de a est toujours pair.

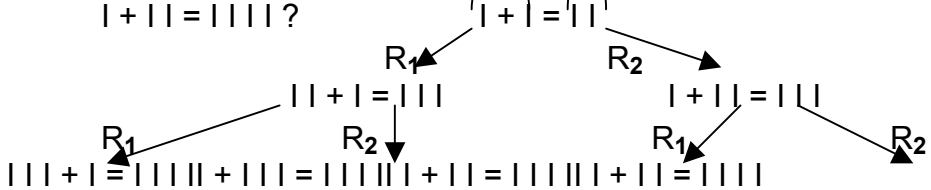
(III) a b a b a c a a a (R_1) ➔a a b a c a a a
(R_2) ➔a b a a c a a a

Il n'est pas possible de prouver que a b a b c a a a est un théorème, car il n'y a aucun moyen d'insérer un "b"

Un autre exemple: + P = {I, +, =}
+ Construction de fbf: concaténétion sur P
+ Axiome: I + I = I I
+ Règles d'inférences: si A = B alors I A = I B
si A = B alors A I = B I

Montrer que: I I + I I = I I I I
I + I I I = I I I I
I I I + I = I I I I
I + I I = I I I I ?

A B



Théorème Théorème Théorème Théorème

I + I I = I I I I non théorème. Dans une formule bien formée, on a 1 nombre pair de I

Résolution: Intuitivement: c_1 $a \vee b$

c_2 $\neg a \vee c$

c_3 $b \vee c$ **résolvante** de c_1 et c_2 et elle est le résultat de l'application de la **résolution** de c_1 et c_2

Si on veut démontrer que:

$\{\varphi_1 \ldots. \varphi_n\} \models \varphi$ ssi $\{\varphi_1 \ldots. \varphi_k, \neg\varphi\}$ est contradictoire

dérivation sémantique (valuation)

$\{\varphi_1 \ldots. \varphi_k, \neg\varphi\} \vdash$ ← clause vide

dérivation résolution

Clause Forme générale: $a_1 \vee a_2 \vee \ldots. \vee a_n \vee \neg b_1 \vee \neg b_2 \vee \ldots. \vee \neg b_n$

"Clauses de base" a, $a \vee \neg b$, $\neg b$ } **une clause est toujours satisfiable**

$a \vee \neg a$ ➔ 1 s'efface

$a \wedge \neg a$ n'est pas 1 clause, mais est une **forme clausale!**

= la seule clause non satisfiable

Exemple 8: $a \rightarrow b \models (a \rightarrow \neg b) \rightarrow \neg a$ $\{\underbrace{a \rightarrow b}_{f_1} \models \underbrace{\neg [(a \rightarrow \neg b) \rightarrow \neg a]}_{f_2}\} \vdash$

Mise sous forme clausale de f_1 et f_2

f_1 $\neg a \vee b$

f_2 $\neg [\underbrace{(a \rightarrow \neg b)}_{\neg a \vee \neg b} \rightarrow \neg a]$

$\neg [\neg (\neg a \vee \neg b) \vee \neg a]$ $(\neg a \vee \neg b) \wedge a$ **forme clausale !!**

1 $\neg a \vee b$

2 $\neg a \vee \neg b$

3 a

Résolution 1 et 2 4 $\neg a$ } 2 étapes

Résolution 3 et 4 5

Autre cheminrésolution de 1 et 3 6 b

Résolution 6 et 2 7 $\neg a$ } 3 étapes

Résolution 7 et 3 8

Résolution 2 et 3 9 $\neg b$

Résolution 9 et 1 10 $\neg a$ } 3 étapes

Résolution 10 et 3 11

Exemple 2 $(a \rightarrow b) \rightarrow c \models a \rightarrow (b \rightarrow c)$ $\{(a \wedge b) \rightarrow c \models \neg [a \rightarrow (b \rightarrow c)]\} \vdash$

Formes clausales: φ_1 $\neg (a \wedge b) \vee c = \neg a \vee \neg b \vee c$

φ_2 $\neg [a \rightarrow \underbrace{(b \rightarrow c)}_{\neg b \vee c}]$

$\neg (\neg a \vee \neg b \vee c)$ $= a \wedge b \wedge \neg c$

1 $\neg a \vee \neg b \vee c$

2 a

3 b

4 $\neg c$

Exemple 2: |= (a ➔ b) ➔ ((a ➔ (b ➔ c)) ➔ (a ➔ c)) ➔ Ψ
¬Ψ |— ⬌ Ψ est une tautologie
¬[(a ➔ b) ➔ ((a ➔ (b ➔ c)) ➔ (a ➔ c))]

¬a ∨ b ¬b ∨ c ¬a ∨ c

¬a ∨ ¬b ∨ c

¬[¬(¬a ∨ b) ∨ ¬(¬a ∨ ¬b ∨ c) ∨ ¬a ∨ c] = (¬a ∨ b) ∧ (¬a ∨ ¬b ∨ c) ∧ a ∧ ¬c

forme clausale:
1 ¬a ∨ b
2 ¬a ∨ ¬b ∨ c
3 a
4 ¬c

Résolution 1 et 2: 5 ¬a ∨ c
5 et 3 6 c
6 et 4 7

Exemple 9: φ = ¬[¬(a ∨ b ➔ c) ➔ ¬(a ∧ b ➔ c)] ➔ ¬φ |—
¬φ = ¬(¬[¬(a ∨ b ➔ c) ➔ ¬(a ∧ b ➔ c)])

(a ∨ b ➔ c) ∨ ¬(¬(a ∧ b) ∨ c))
¬(¬[¬(a ∨ b) ∨ c) ∨ (a ∧ b ∧ ¬c]
[(¬a ∧ ¬b) ∨ c)] ∨ (a ∧ b ∧ ¬c)
[(¬a ∨ c) ∧ (¬b ∨ c)] ∨ (a ∧ b ∧ ¬c) (¬a ∨ c ∨ a) ➔ 1 (¬a ∨ c ∨ b) ∨ (a ∧ b ∧ ¬c)
(¬a ∨ c ∨ ¬c) ➔1

1 ¬a ∨ c ∨ b
2 a ∧ b ∧ ¬c résolution 1 et 2 ➔ 1

6. - La méthode des tableaux sémantiques (TS)

La méthode des tableaux sémantiques est une autre méthode de preuve (système déductif) qui est une alternative à la résolution.

6 - 1 Définition en terme de système Formel

+ P = {a, b, c, ….} variables propositionnelles
+ Procédé de construction des fbf du CP0
+ un ensemble d'axiomes vides
+ un ensemble de règles d'inférence (2 pour chaque connecteur booléen)

Règles d'inférences

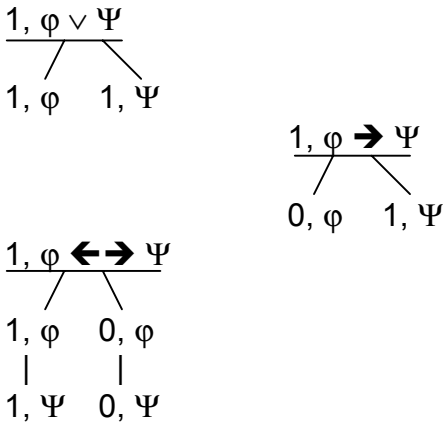
Règles positives **Règles négatives**

Si la valuation de ¬φ est 1 alors la Valuation de φ est 0 $\frac{}{1, \varphi}$ 0, ¬φ

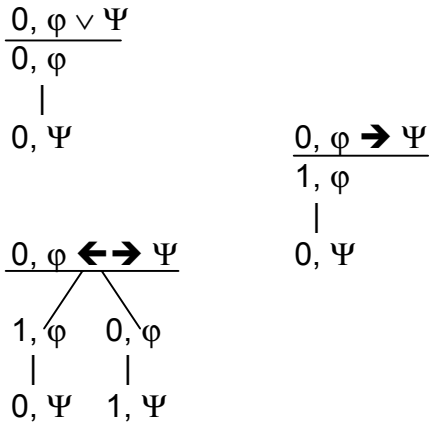
$\frac{1, \neg\varphi}{0, \varphi}$ $\frac{1, \varphi \wedge \Psi}{\begin{array}{l}1, \varphi \\ 1, \Psi\end{array}}$ $\frac{0, \varphi \wedge \Psi}{0, \varphi \mid 0, \Psi}$

symbolise et Début de "branche" et naissance de 2 branches "ou"

Règles positives



Règles négatives

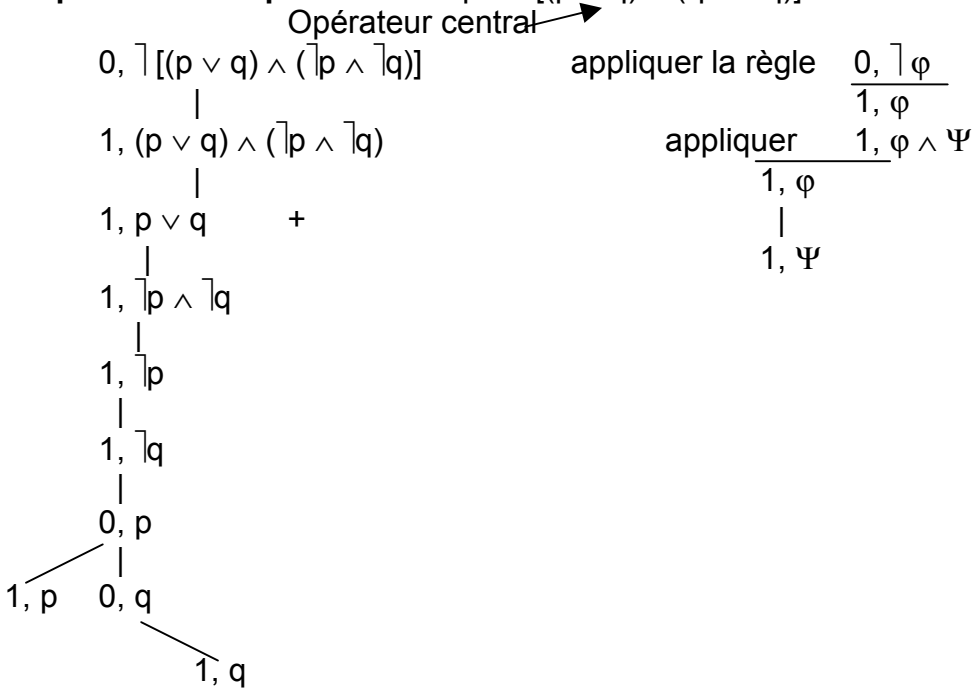


6 - 2 Les principes intuitifs de la méthode des tableaux sémantiques TS

(1) La méthode de Ts s'appuie sur la **réfutation** (comme la résolution):
Pour prouver que φ est un théorème, on cherche à montrer que ⌝ φ est une absurdité ou conduit à une absurdité

(2) Un **tableau sémantique** est un **arbre fini étiqueté** qui se construit de la façon suivante:
- Si φ est la formule à prouver, la racine de l'arbre est 0, φ
- On choisit une branche de l'arbre et un nœud (ε, Ψ) ou ε = 0 ou 1, et on applique la règle d'inférence liéeau symbole "central" associé à Ψ
- Une branche ne peut plus être développée à partir du moment où elle contient 2 nœuds de la forme (1, Ψ) et (0, Ψ)

Un premier exemple: Prouver que ⌝ [(p ∨ q) ∧ (⌝p ∧ ⌝q)] est un théorème



6 - 3 Définitions Formelles

(1) Un nœud d'une branche d'un tableau sémantique est dit réduit si une règle d'inférence lui à été appliquée (on n'applique pas 2 fois une règle d'inférence à un nœud).
S'il n'est pas réduit, un nœud est dit **ouvert** ou en **attente**.

(2) Une branche est **contradictoire**, si elle contient un nœud de la forme (0, Ψ) et de la forme (1, Ψ). La branche est alors **fermée** et n'est plus développée.

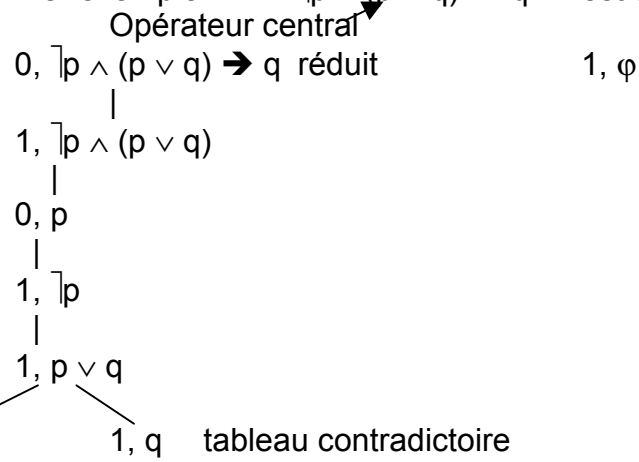
(3) Une branche est **terminée** si elle est fermée (contradictoire) où tous les nœuds sont réduits.

(4) Un tableau est terminé si toutes les branches sont terminées. Un tableau est contradictoire si toutes les branches le sont.

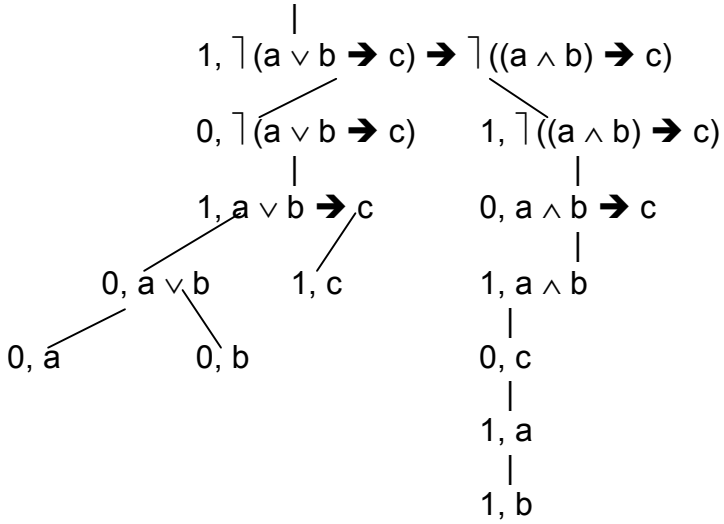
(5) Une preuve de φ par la méthode de TS est un tableau contradictoire de racine 0, φ

Un dernier exemple: $\neg p \wedge (p \vee q) \rightarrow q$ est un théorème

Opérateur central



Exemple: $0, \neg(\neg(a \vee b \rightarrow c) \rightarrow \neg((a \wedge b) \rightarrow c))$

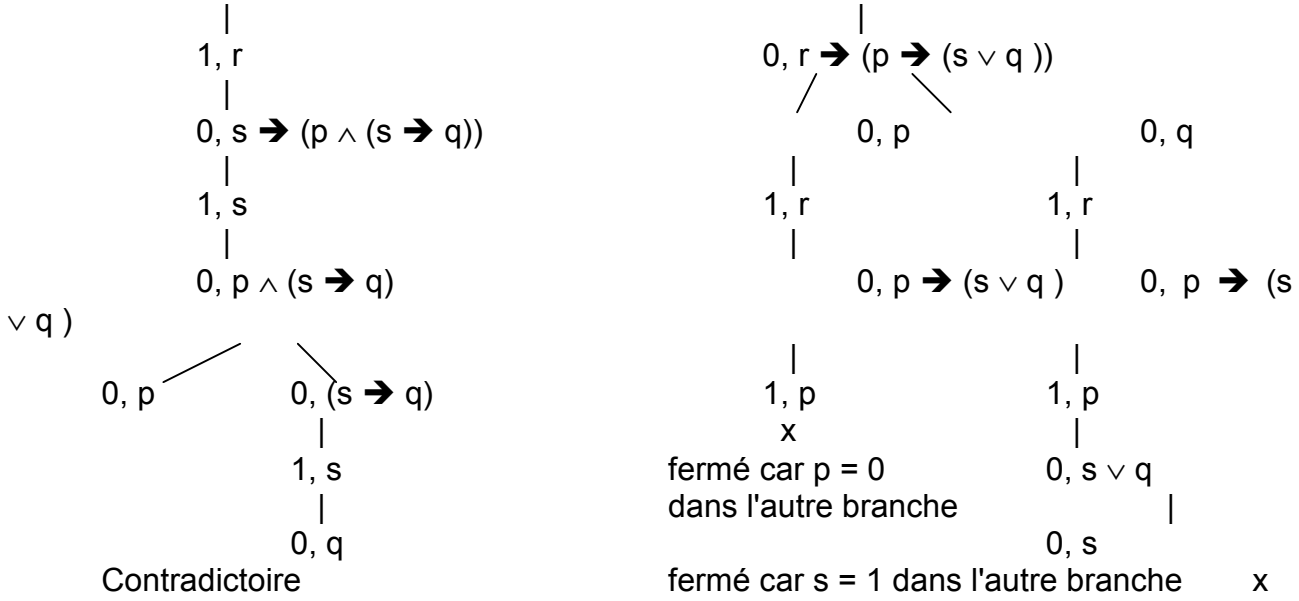


Exemple: $(r \rightarrow (s \rightarrow (p \wedge (s \rightarrow q)))) \vee (r \rightarrow (p \rightarrow (s \vee q))))$

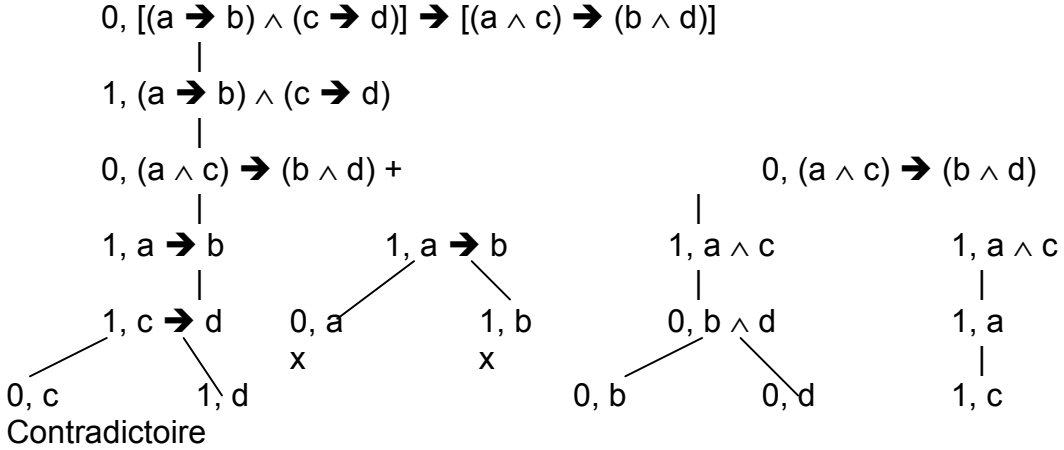
$0, (r \rightarrow (s \rightarrow (p \wedge (s \rightarrow q)))) \vee (r \rightarrow (p \rightarrow (s \vee q))))$

$0, (r \rightarrow (s \rightarrow (p \wedge (s \rightarrow q))))$

$0, (r \rightarrow (p \rightarrow (s \vee q))))$ + (attente) $0, r \rightarrow (p \rightarrow (s \vee q))$



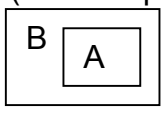
Exemple: $[(a \rightarrow b) \wedge (c \rightarrow d)] \rightarrow [(a \wedge c) \rightarrow (b \wedge d)]$



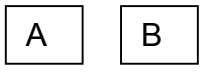
7. - Représentation d'énoncés avec des Formules du CP0

Nous allons dresser un catalogue des phrases types qui sont facilement représentables par les Formules du CP0.

(I) "Quantification universelle" (la vraie quantification sera vue par le CP1)
Tous les A sont des B
$A \rightarrow B$
$A \subseteq B$ $\quad \neg A \vee B$



(II) Les A ne sont pas des B (A et B disjoints) (universelle)
$A \rightarrow \neg B$
$A \subseteq \_ B$
Aucun A n'est un B



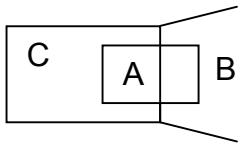
(III) Remarque: "Aucun A n'est un B" n'est pas la négation de "Tous les A sont des B"
$A \rightarrow \neg B$ $\quad \neg (A \rightarrow B) = \neg (\neg A \vee B) = A \wedge \neg B$

Quantification existentielle: certain A ne sont pas des B

Quelques A ne sont pas des B

(IV) Tous les A sauf les B sont des C

$A \wedge \neg B \rightarrow C$



Variantes: A moins qu'ils ne soient des B, les A sont des C

A moins que ou sauf expriment une exception

Exemple: 3 suspects B, J, et S et on cherche le coupable en fonction des témoignages suivants:

1) B déclare: J est coupable et S est innocent
2) J déclare: Si B est coupable alors S est coupable
3) S déclare: Je suis innocent mais au moins 1 des 2 autres est coupable.

Représentation des témoignages, 3 variables propositionnelles: B est coupable: B
J est coupable: J
S est coupable: S

1) $J \wedge \neg S$
2) $B \rightarrow S$
3) $\neg S \wedge B \vee J$ Système de clause S: 1. 1 J
1. 2 $\neg S$
2 $\neg B \vee S$
(3. 1 $\neg S$)
3. 2 $B \vee J$

Est - ce que le système S est cohérent ?

Le système est cohérent ssi S |—

1 J
2 $\neg S$
3 $\neg B \vee S$
4 $B \vee J$ Résolution 2 et 3: 5 $\neg B$
Résolution 3 et 4 6 $S \vee J$
Résolution 4 et 5 7 J

Résultat: + J ← coupable et on n'obtient jamais: $\neg J$
+ $\neg S$ S
+ $\neg B$ B

Exemple:

(1) Si l'unité système est ratée alors je suis bon pour septembre à moins d'avoir la moyenne en anlyse et une bonne note en IA
(2) L'unité système est ratée
(3) La note en analyse est sous la moyenne

Est - ce que je suis bon pour septembre découle des phrases 1, 2, 3 ?

Propositions l'unité système ratée: RATEE
Je suis bon pour septembre: SEPTEMBRE
Avoir la moyenne en analyse: MOYENNE_A
Bonne note en IA: IA

Première traduction

(1) $RATEE \wedge (MOYENNE\_A \wedge IA) \rightarrow SEPTEMBRE$
(2) RATEE
(3) $\neg MOYENNE\_A$

Question: SEPTEMBRE

(1) $\neg RATEE \vee (MOYENNE\_A \wedge IA) \vee SEPTEMBRE$

Distributivité

(C1) ⌝RATEE ∨ MOYENNE_A ∨ SEPTEMBRE
(C2) ⌝RATEE ∨ IA ∨ SEPTEMBRE
(C3) RATEE
(C4) ⌝MOYENNE_A
(⌝Q) ⌝SEPTEMBRE {C1, C2 C3, C4} |= SEPTEMBRE ?
{C1, C2 C3, C4, ⌝SEPTEMBRE } |—
Résolution C1 et C3 (C5) MOYENNE_A ∨ SEPTEMBRE
Résolution C4 et C5 (C6) SEPTEMBRE
Résolution C6 et ⌝Q(C7)

Exercice 5: 3 indéfini ➔ universelle

1	A ➔ ⌝B
2	A ➔ B
3	A ➔ B
4	A ∧ ⌝B ➔ C
5	A ➔ B
6	A ➔ B

Requin	requin
Equipé	poisson bien équipé
⌝Equipé	poisson mal équipé = ⌝(poisson bien équipé)
3_rangées	poisson qui possède 3 rangées de dents
menuet	poisson sachant danser le menuet
mepris	poisson digne de mépris
⌝mepris	⌝poisson digne de mépris = poisson qui ne mérite pas le mépris
gentil	poisson qui sont gentils avec les enfants
corpulent	poisson corpulent

1)	requin ➔ ⌝(⌝Equipé)	(C1)	⌝requin ∨ Equipé
2)	⌝menuet ➔ mépris	(C2)	menuet ∨ mépris
3)	⌝3_rangées ➔ ⌝Equipé	(C3)	3_rangées ∨ ⌝Equipé
4)	poisson ∧ ⌝requin ➔ gentil	(C4)	requin ∨ gentil
	assertion = "1"		
5)	corpulent ➔ ⌝menuet	(C5)	⌝corpulent ∨ ⌝menuet
6)	3_rangées ➔ ⌝mépris	(C6)	⌝3_rangées ∨ ⌝mépris
7)	corpulent ➔ gentil	(C7)	⌝corpulent ∨ gentil

Question: {1, 2, 3, 4, 5, 6} |= 7 {1, 2, 3, 4, 5, 6, ⌝7} |—
⌝(C7):corpulent ∧ ⌝gentil

Exercice 3:

membre	écossais		écossais
	Porte des chausettes oranges		orange
	Porte des jupes		jupe
	Marié		marié
Articles	Sort le dimanche		dimanche

1	⌝écossais ➔ orange	(C1)	écossais ∨ orange
2	jupe ∨ orange	(C2)	jupe ∨ orange
3	marié ➔ ⌝dimanche	(C3)	⌝marié ∨ ⌝dimanche
4	dimanche ➔➔ écossais	(C4)	⌝dimanche ∨ écossais
			⌝écosais ∨ dimanche
5	jupe ➔ écossais ∧ marié	(C5)	⌝jupe ∨ (écosais ∧ marié) ⌝jupe ∨ écosais
			⌝jupe ∨ marié

6 écossais ➔ jupe (C6) ⌉écossais ∨ jupe

Question: Existe - il des membres ?

Il existe des membres ssi le système est cohérent. Le système est cohérent ssi il n'est pas possible d'avoir {1, 2, 3, 4, 5, 6} |—

articles

Remarque:

1) il est inutile d'appliquer la résolution sur C4.1 et C4.2 qui produit à chaque fois 1
2) la résolution sur C4.1 et C4.2 ne produit en aucun cas la clause vide, car la résolution porte que sur un atome.
3) Même remarque que 1 sur C5.1 et C6

Résolution: C1 et C2: C7 écossais ∨ jupe
Résolution: C5.1 etC7: C8 écossais
Résolution: C6 et C8: C9 jupe
Résolution: C5.2 et C9: C10 marié
Résolution: C3 et C10: C11 ⌉dimanche
Résolution: C4.2 et C11: C12 ⌉écossais
Résolution: C8 et C12: C11

Exemple: {1, 2, 3, 4, 5, 6} |— 6 les 5 clauses donnent la 6ème
{1, 2, 3, 4, 5, ⌉6} |—
incohérent

⌉(A ←➔ B) = ⌉(A ➔ B ∧ B ➔ A) = ⌉(A ➔ B) ∨ ⌉(B ➔ A) = (A ∧ ⌉B) ∨ (B ∧ ⌉A)
(A ∨ B) ∧ (⌉B ∨ ⌉A) = (⌉A ➔ B) ∧ (B ➔ ⌉A)

Exercice 7: proposition: chaton aime le poisson poisson
Est réfractaire à l'étude réfractaire
Sans queue sans queue
Prêt à jouer avec un gorille gorille
Moustachus moustachu
Amoureux de l'étude ⌉
réfractaire

⌉(chaton réfractaire à l'étude) a les yeux verts verts

1 poisson ➔ ⌉réfractaire (C1) ⌉poisson ∨ ⌉réfractaire
2 sans queue ∨ ⌉gorille (C2) ⌉sans queue ∨ ⌉gorille
3 moustachu ➔ poisson (C3) ⌉moustachu ∨ poisson
4 ⌉réfractaire ➔ ⌉vert (C4) réfractaire ∨ ⌉vert
5 ⌉moustachu ➔ sans queue (C5) moustachu ∨ sans queue !
6 vert ➔ ⌉gorille (C6) ⌉vert ∨ ⌉gorille
{1, 2, 3, 4, 5, ⌉6} |—

Exercice 1: {a1, a2, a3, ⌉ci} |—
c1
c2
c3
c4

Propositions: 1) Ulysse prend le bus prend-le-bus
Le bus en retard bus -retard
Ulysse rate son rendez-vous rater-rdv

2) Ulysse doit rentrer chez lui — rentrer
Ulysse est déprimé — déprimé
3) Ulysse décroche job — décroche-job

a1) prend-le-bus ∧ bus-retard ➔ rater-rdv — ¬ prend-le-bus ∨ ¬bus-retard ∨ rater-rdv
a2) rater-rdv ∧ déprimé ➔ ¬ rentrer — ¬ rater-rdv ∨ ¬ déprimé ∨ ¬ rentrer
a3) ¬ décrocher-job ➔ déprimé ∧ rentrer — décrocher-job ∨ (déprimé ∧ rentrer)
a31) décrocher-job ∨ déprimé
a32) décrocher-job ∨ rentrer
a4) résolution de a2 et a31 puis a2 + a31 et a32 — ¬ rater-rdv ∨ décrocher-job
c1) prend-le-bus ∧ bus-retard ➔ décrocher-job — ¬ prend-le-bus ∨ ¬ bus-retard ∨ décrocher-job

{a1, a2, a3, ¬c1} |—
¬ c1) prend-le-bus ∧ bus-retard ∧ ¬décrocher-job — ¬ c11) prend-le-bus
¬ c12) bus-retard
¬ c13) ¬ décrocher-job

résolution de a1 et ¬ c11 (f1) ¬ bus-retard ∨ rater-rdv
résolution de f1 et ¬ c12 (f2) rater-rdv
résolution de f2 et a2 (f3) ¬ déprimé ∨ ¬ rentrer
résolution de f3 et a3 (f4) décrocher-job ∨ ¬ rentrer
résolution de f4 et ¬ c13 (f5) ¬ rentrer
résolution de f5 et a32 (f6) décrocher-job
résolution de f6 et ¬ c13

c2) bus-retard ➔ ¬ prend-le-bus ∨ ¬ rater-rdv — ¬ bus-retard ∨ ¬ prend-le-bus ∨ ¬ rater-rdv

{a1, a2, a3, ¬c2} |—
¬ c2) prend-le-bus ∧ bus-retard ∧ rater-rdv — ¬ c21) prend-le-bus
¬ c22) bus-retard
¬ c23) rater-rdv

résolution de a1 et ¬ c21 (f7) ¬ bus-retard ∨ rater-rdv
résolution de f7 et ¬ c22 (f8) rater-rdv
résolution de f8 et a4 (f9) décrocher-job ⟍ |—

c3) bus-retard ∨ rater-rdv ➔ déprimé — ¬ bus-retard ∧ ¬ rater-rdv ∨ déprimé

{a1, a2, a3, ¬c3} |—
¬ c3) bus-retard ∨ rater-rdv ∧ déprimé — ¬ c31) bus-retard ∨ rater-rdv
¬ c32) déprimé
résolution de a1 et ¬ c31 (f10) ¬ prend-le-bus ∨ rater-rdv
imposible d'obtenir la clause ¬ prend-le-bus ne peut jamais être effacé par la résolution.

c4) rentrer ∧ prend-le-bus ➔ (bus-retard ➔ ¬ déprimé)
rentrer ∧ prend-le-bus ∧ bus-retard ➔ ¬ déprimé
¬ rentrer ∨ ¬ prend-le-bus ∨ ¬ bus-retard ∨ ¬ déprimé

{a1, a2, a3, ¬c4} |—
¬ c4) rentrer ∧ prend-le-bus ∧ bus-retard ∧ déprimé — ¬ c41) rentrer
¬ c42) prend-le-bus
¬ c43) bus-retard
¬ c44) déprimé
résolution de a1 et a2 (f11) ¬ prend-le-bus ∨ ¬ bus-retard ∨ ¬ déprimé ∨ ¬ rentrer
après 4 résolutions on obtient

Introduction au calcul des prédicats du premier ordre

Le calcul du prédicat du 1er ordre CP1 est une logique avec variable qui étend le CP0 avec les notions de variables, de quantificateurs et de prédicats.

1. - Introduction intuitive

Si on considère la phrase: Pierre est un garçon

En CP0, on peut considérer cette phrase comme unique proposition.

Tant que la phrase est isolée, il n'y a pas de problème à priori.

Peut - on considérer les constituants de cette phrase?

Peut - on considérer Pierre en tant que tel?

Pierre ne peut - être une proposition et Pierre désigne un individu

Il serait faux d'écrire Pierre ➔ garçon

faux

Si on considère en plus: Paul est un garçon

Comment faire pour factoriser les informations et faire émerger les liens entre les 2 phrases?

➔ On introduit les notions de **variables** et de **prédicats**

Une variable peut être considérée au sens mathématique du terme comme un élément auquel on peut associer une valeur (différentede vrai, faux).

Un prédicat peut être considéré comme une fonction qui possède des arguments et qui est à valeurs {vrai, faux} ou {0, 1}

Par exemple: garçon (x) = si x désigne un garçon alors 1 sinon 0

Prédicat unaire

Garçon (Pierre) Pierre et Paul doivent être considérés comme des constantes qui

Garçon (Paul) intancient la variable x dans graçon(x)

Pierre aime Marie

Aime peut être considéré comme un prédicat binaire qui établit une relation entre 2 individus.

Aime (Pierre, Marie) Les contraintes Pierre et Marie instancient les variables x et y du prédicat binaire aime (x, y)

1. Relation entre 1 individu et 1 classe d'individu: garçon (Pierre) "Instanciation de prédicat unaire"
2. Relation entre 2 individus: aime (Pierre, Marie) "Instanciation de prédicat binaire"
3. Il est possible de définir des relations entre 2 classes d'individus: Les garçon aimene les filles en CP0 garçon ➔ aime fille
   fille ➔ aime garçon

en CP1: $\forall x \forall y$ garçon (x) $\wedge$ fille (y) ➔ aime (x, y)

on reviendra sur ce type de phrase au prochain cours

Il n'est pas licite d'écrire:

Aime (garçon, fille) si garçon et fille sont des symboles de prédicat composition: prédicat (prédicat) passage au 2ème ordre

A côté de la notion de prédicat, on trouve la notion de **fonction**. Une fonction est à considérer au sens mathématique du terme:

Une fonction f: E ➔ F est définie sur un domaine E et à valeur dans le co-domaine F et à x fait correspondre f(x)

Exemple taille: Personne ➔ Echelle = {petite, moyenne, grande}
Pierre ➔ taille (Pierre) = moyenne
Nombre réels = 1.75 E et F sont des ensembles **différents de {0, 1}**

Composition d'une fonction et d'un prédicat

P(x) à valeur dans {0, 1}
Prédicat variable ou élément fonctionnel

Personne ➔ Maison ➔ Nombre
Pierre ➔ Habitation (Pierre) ➔ Prix (habitation (Pierre))
Prédicat chère (habitation (Pierre)) ∈ {0, 1}

Un prédicat peut avoir pour argument une fonction, mais pas l'inverse f(P(x)) n'est pas licite

Ce qui est bon marché est rare: rare (x) prédicat bon-marché (z) prédicat
Il n'est pas licite d'écrire: rare (bon-marché (x))

2. - La construction des Formules du CP1

2 - 1 Langage des Formules

+ un ensemble V de **symboles de variables** V = {x, y, z …. }
+ un ensemble de **symboles de fonctions** F = {F_0, F_1, F_2, ….. F_n} où F_0 est l'ensemble des constantes (fonctions à 0 variables)F_i est l'ensemble des fonctions à i variables.
+ un ensemble de **symboles de prédicats** R = {R_0, R_1, R_2, ….. R_n} où R_0 correspond aux prédicats à 0 variables propositionnelles, R_k correspond aux prédicats à k variables.
+ un ensemble de quantificateurs {∀, ∃}
universel, **tous** existanciel, **certains** (quelques)
+ un ensemble de connecteurs booléens {¬, ∨, ∧, ➔, ➔➔}

La construction des Formules proprement dite:

(I) **Les termes:** + une constante (∈ F_0) où une variable sont des termes
+ si f ∈ F_n, t_1, ….. t_n sont des termes alors f(t_1, ….. t_n) est un terme

(II) **atomes ou formules atomiques:**
+ Si , t_1, ….. t_n sont des termes et P ∈ R_n (symbolede prédicat n-aire) alors P(t_1, ... t_n) est un **atome** ou **Formule atomique**

(III) **Littéral:** Un littéral est un atome ou une mégation d'atome

(IV) **Formules du 1er ordre:**
- un littéral est une formule
- Si φ est une formule, x une variable alors ∀ x φ, ¬ φ sont des formules

$\exists$ x φ

- Si φ et Ψ sont des Formules alors $\varphi \wedge \Psi$, $\varphi \vee \Psi$, $\varphi \rightarrow \Psi$, $\varphi \leftrightarrow \Psi$ sont des formules

Exemple: $\forall x \forall y P(x, y) \wedge \exists z \forall x Q(z, y, x)$
La quantification $\forall u \forall v P(f(u), g(v)) \wedge Q(u, v)$
Porte sur les variables fonctions Formules bien formées

2 - 2 Les caractéristique des variables et des Formules du CP1

Une sous - formule est une formule qui a servi à la construction d'une formule qui l'englobe.

Une sous - formule débute par un certain nombre de quantificateurs qui définissent la portée des variables.

$\forall x \varphi(x) \rightarrow \forall z \exists y \varphi(z) \wedge \delta(y)$ $\varphi(x) \rightarrow \forall z \Psi(x, z)$ Formule
sous formules sous formule sans quantificateur
$\forall z \Psi(z) \wedge \exists y \delta(y)$

Variables libres et liées

Une occurrence de la variable x dans une formule φ est dite si x est introduite dans une sous - formule où figure un quantificateur $\forall$ x ou $\exists$ x

On dit que x est dans le champ du quantificateur

Exemple: $\forall x \varphi(x) \rightarrow \forall y \Psi(y) \wedge \forall z \delta(z)$ x, y, z sont liées
$\forall z \exists y \varphi(x) \wedge \Psi(z) \wedge \delta(y)$ z et y sont liées
x n'est dans le champ d'aucun quantificateur, et est dite libre

Attention à la position des parenthèses et des variables:

(φ_1) $\forall x P(x, f(x)) \rightarrow Q(x)$ x est libre
x est liée
(φ_2) $\forall x P(x, f(x)) \rightarrow Q(x))$

Formules closes, prénexes et polies (ou propres)

(I) Une formule est dite **close** si elle ne contient aucune variable libre
(φ_1) n'est pas close, mais (φ_1) l'est.

(II) Une formule est dite **prénexe** si tous les quantificateurs de la Formule sont en tête de formule.
$\forall x \varphi(x) \rightarrow \exists y \Psi(y) \vee \delta(z)$ non close, non prénexe
$\forall x \exists y \varphi(x) \rightarrow \Psi(y) \vee \delta(z)$ non close, prénexe
$\forall x \exists y \forall y (\varphi(x) \rightarrow \Psi(y) \vee \delta(z))$ close et conexe

(III) Formule polie ou propre:
Une formule φ est polielorsque l'ensemble des variables libres de φ est distinct de celui des variables liées de φ, et lorsque toutes les occurrences d'une même variable sont dans le champ d'un seul quantificateur.
$(\forall x P(x, f(x))) \rightarrow Q(x)$ Formule non polie
$\forall x \exists y \varphi(x) \rightarrow \varphi(y) \wedge \delta(z)$ Formule est polie
pour rendre une formule polie, il suffit de renommer les variables

Exemple: pas close, pas prénexe, pas polie libre

$\forall x ((\exists y P(x, y))) \rightarrow \forall z Q(x, y, z) \wedge \forall y \exists x \delta(f(x), y)$

renommage $\forall x ((\exists u P(x, u) \rightarrow \forall z Q(x, y, z) \wedge \forall v \exists t \delta(f(t), v)$

Polie et conexe $\forall x \exists u \forall z \forall v \exists t (P(x, u) \rightarrow Q(x, y, z) \wedge \delta(f(t), v)$

2 - 3 Notion de Substitution

Soit φ une formule polie, x une variable et t un terme. La substitution de x par t dans φ consiste à remplacer toutes les occurrences de x par t dans φ à condition que x ne figure pas dans t.

La substitution se note $x \rightarrow t$ ou x/t $\varphi(x/t)$ dénote la formule φ ou x se substitue par t

Exemple: $\forall x (\exists u P(x, u) \rightarrow \forall z Q(x, y, z) \wedge \forall v \exists t \delta(f(t), v))$

Substitution de y par f(n(z), x)

Renommage de x par w et z par s dans φ: $\forall w (\exists u P(w, u) \rightarrow \forall s Q(w, y, s) \wedge \forall v \exists t \delta(f(t), v))$

Ne pas substituer x par f(x) !!

$x \rightarrow f(x)$

$f(f(x))$

$f(f(f(x) \ldots .)$

2 - 4 Notion d'unification

Soient L_1 et L_2 2 littéraux.

Unifier L_1 et L_2 consiste à chercher une substitution σ tel que $\sigma(L_1) = \sigma(L_1)$

Intuitivement, unifier revient à chercher une substitution qui s'applique à L_1 aussi bien qu' à L_2 et qui permet de rendre "les littéraux égaux"

Algorithme d'unification

Etant donné L_1 et L_2, former la 1ère **paire de discordances**, cad le 1er couple de termes qui sont différents

Exemple: P(f(x, g(z)), x, f (y, g(b)))

P(u, g(f(a, b)), u)

Paires de discordance

(I) f(x, g(z)) et u

(II) x et g(f (a, b))

(III) f(y, g(b)) et u

+ Si les membres de cette paire sont 2 constantes différentes, alors L_1 et L_2 ne sont pas unifiables.

P(x, a) et P(z, b) où $a \neq b$ ne sont pas unifiables

+ Si l'un des membres de la paire est un terme t et l'autre une varaible x et si t contient x alors L_1 et L_2 ne sont pas unifiables.

+ dans tous les autres cas, substituter x par t et reitérer le processus

Une remarque: Si t est une varaible, alors on peut:

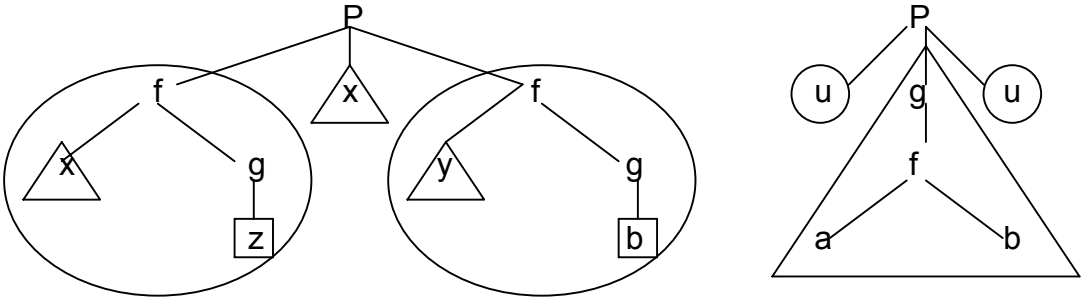
- soit substituer x par t

- soit substituer t par x

➔ processus indéterministe

Stratégie substituer en priorité les varaibles les plus contraintes: dans l'exemple: x et g(f(a, b))
(Variables qui vont être par des termes contenant le plus de contraintes possibles)

substituer x par g(f(a, b)) la substitution se représente aux 2 littéraux
(I) f(g(f (a, b))), g(z) et u y est substitué par g(f(a, b)) z est substitué par b
(III) f(y), g(b) et u x est substitué par g(f(a, b))
P(f (g (f (a, b)), g(b)), g(f (a, b)), f(g(f (a, b)), g(b)))



3. - Aspects sémantiques des Formules du CP1

3 - 1 Interprétation d'une formule du CP1

La notion d'interpréattion bien que similaire à celle employée pour le CP0 est plus compliquée car il faut tenir compte des fonctions et des variables.

Dans ce qui suit, on donne une définition intuitive de la notion d'interprétation (suffisante poue ce que nous avons à faire)

Définition: Une interprétation (Δ, I) est la donnée d'un domaine Δ et d'une fonction d'interprétation I ($.^I$) telle que:
(I) à une constante $c \in F_0$ on fait correspondre un élément noté $c^I \in \Delta$ $I: F_0 \rightarrow \Delta$
(II) à un ensemble de fonctions $f \in F_n$ ($n > 0$)on fait correspondre une fonction notée f^I: $\Delta^n \rightarrow \Delta$

$\Delta * \Delta * .... * \Delta$

(III) à un symbole de prédicat $P \in R_n$ on fait correspondre une **valeur de vérité** vrai ou faux (ou 1 ou 0) $P^I : \Delta^n \rightarrow \{0, 1\}$ avec $P \in R_n$
Interprétation d'une fonction close: (en tenant compte des varaibles)

(IV) Si la valeur de vérité de 2 formules φ et ψ est connue, alors:
$(\varphi \{\wedge, \vee, \rightarrow, \leftrightarrow\} \psi)^I = \varphi^I \{\wedge, \vee, \rightarrow, \leftrightarrow\} \psi^I$ comme en calcul des propositions
$(\neg\varphi)^I = \neg\varphi^I$

(V) **Variables et quantificateurs**
- à une varaible x on fait correspondre un élément de Δ noté x^I
- Pour une formule $\forall x \varphi$: cette formule est interprétée à vrai si pour tout $v \in \Delta$, la substitution $(\varphi (x/v))^I$ donne la valeur **vraie**.
- Dans le cas contraire, $\forall x \varphi$: cette formule est interprétée à faux

- Pour une formule $\exists x \varphi$: cette formules est interprétée à vrai s'il existe $v \in \Delta$ telle que l'interprétation $(\varphi (x/v))^{I}$ donne la valeur **vraie**.
- Dans le cas contraire, $\exists x \varphi$: cette formule est interprétée à faux

$P(t_1, \ldots T_n) \in \{0, 1\}$

Exemple 1: $\varphi = \forall x P(x)$ $\psi = \exists x \neg P(x)$ $\Delta = \{1, 2\}$

On pose $P^{I1}(1)$: vrai, $P^{I2}(2)$: faux } pour l'interprétation I_1 c'est une donnée

pour $I_1 = \varphi^{I1}$: faux

$= \psi^{I1}$: vrai valeur de vérité de ψ pour l'interprétation I_1

On pose $P^{I2}(1)$: vrai, $P^{I2}(2)$: vrai } pour l'interprétation I_1 c'est une donnée

pour $I_1 = \varphi^{I2}$: vrai valeur de vérité de φ pour l'interprétation I_1

$= \psi^{I2}$: faux

Un autre exemple: $\varphi = \forall x (P(x) \rightarrow Q(f(x), a))$ (a : constante)

On se donne: Δ et I_3 $\Delta = \{1, 2\}$

I_3: $a \rightarrow 1 = a^{I3}$):

$f \rightarrow f^{I3}: \Delta \rightarrow \Delta$ { $f^{I3}(1) = 2$, $f^{I3}(2) = 1$

$Q \rightarrow Q^{I3}: \Delta * \Delta * \Delta \rightarrow \{0, 1\}$

$Q^{I3}(1, 1) = 1$
$Q^{I3}(1, 2) = 1$
$Q^{I3}(2, 1) = 0$
$Q^{I3}(2, 2) = 1$

$x = 1$ $P^{I3}(1) \rightarrow Q^{I3}(f^{I3}(1), 1)$ la valeur de vérité de φ pour I_3 et $x = 1$ est **vrai**

$x = 2$ $P^{I3}(2) \rightarrow$ vrai = | vrai / 1

Donc: φ^{I3} la valeur de vérité de φ pour I_3 et $x = 2$ est **vraie** 1 | Vrai $\rightarrow$ | vrai = 1 | 1 vrai

3 - 2 Propriété sémantiques d'une formule du CP1

Comme en CP0, une formule se caractérise par l'étendue de sa valeur de vérité.

On s'intérese plus précisement aux formules closes (ou 1 formule où la valeur des variables libres est fixée)

Définitions:

(I) **Satisfiablilité:** Une formule φ du CP1 est satisfiable ssi il existe une interprétation I (et un domaine Δ) telle que: φ^I = 1 | vrai Dans ce cas, I est appelé modèle de φ

(II) Dans le cas contraire, φ est dite **non satisfiable** ou **inconsistante**.

(III) Une formule φ est **valide** ssi elle est satisfiable pour toute interprétation (équivalent de la notion de tautologie en CP0.

(IV) Un ensemble de formules est contradictoire ou non satisfiable s'il n'existe pas d'interprétation qui donne la valeur de vérité vrai à chacune des formules simultanément.

$\{\varphi_1, \varphi_2, ... \varphi_n\}^I \leftrightarrow \{\varphi_1 \wedge \varphi_2 \wedge ..... \wedge \varphi_n\}^I \leftrightarrow \varphi_1^I \wedge \varphi_2^I \wedge ..... \wedge \varphi_n^I$

CP0

- construction de formules
- Aspects sémantiques (tables de vérité)
- Aspects syntaxiques formes clausales

4. - Aspects syntaxiques des Formules du CP1

Comme en CP0, étant donné uns formule φ, nous cherchons à transformer φ en une **forme clausale:** φ: $c_1 \wedge c_2 \wedge \ldots\ldots \wedge c_n$ sur laquelle on va appliquer la résolution.

L'algorithme de transformation d'une formule en forme clausale est analogue à celui du CP0, mais il faut tenir compte des variables.

4 - 1 Algorithme de normanlisation

(I) **Clôture universelle:** S'il existe des variables libres dans la formule, alors il faut les **quantifier universellement** (avec renommage éventuel).

(II) **Traitement des abréviations:** $\varphi \rightarrow \psi$ se réécrit $\neg\varphi \vee \psi$
$\varphi \leftrightarrow \psi$ se réécrit $(\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi)$

(III) **Traitement des négations:**

Faire entrer les négations à l'intérieur des parenthèses en respectant les règles suivantes:

- $\neg[\neg P]$ se réécrit P prédicat quelconque d'arité quelconque
- $\neg[\forall x\ P]$ se réécrit $\exists x \neg P$
- $\neg[\exists x\ P]$ se réécrit $\forall x \neg P$

(IV) Standardisation ou étape de **politesse:** rendre la formule polie
Renommer les variables de même nom qui sont dans le champ de quantificateurs différents

(V) Mise sous forme prénexe: Tous les quantificateurs sont mis en tête de formule

(VI) **Skolem**isation: cette étape consiste à supprimer tous les quantificateurs existenciels

Deux cas: $\exists x\ \forall y\ P(x, y)$ se réécrit: $\forall y\ P(a, y)$
Aucun quantificateur avant $\exists x$ **x est tranformé en constante**

$\forall x\ \exists y\ P(x, y)$ se réécrit: $\forall x\ P(x, f(x))$

Il y a dépendance de y par rapport à x, et cette dépendance est matérialisée par une fonction dont le nom dépend du contexte du problème.
Pour les problèmes formels,la fonction est notée généralement f ou g ou h

Attention: $\forall x\ \forall z\ \exists y\ P(x, y)$ **y dépend de x et de z** donc $y \rightarrow f(x, z)$
$\forall x\ \forall z\ P(x, f(x, z))$

(VII) **Distributivité:** $(\varphi_1 \vee \psi_1) \wedge (\varphi_2 \vee \psi_2)$ **ne pas transformer !** c'est une forme clausale
$(\varphi_1 \wedge\vee \psi_1) \vee (\varphi_2 \wedge \psi_2)$ se réécrit $(\varphi_1 \vee \varphi_1) \wedge (\varphi_1 \vee \psi_2) \wedge (\psi_1 \vee \varphi_2) \wedge (\psi_1 \vee \psi_2)$

(VIII) Mise sous forme clausale:

- les quantificateurs universels sont supprimés
- les clauses sont écrites l'une au dessus de l'autre, une par ligne

Exemple: φ: $\neg[\forall x\ \exists y\ \exists z\ ((R(x, x) \rightarrow R(z, y)) \wedge (\neg R(x, x) \vee R(y, z))]$
(I) Clôture universelle: non nécessaire
(II) Abréviations: $\neg[\forall x\ \exists y\ \exists z\ ((\neg R(x, x) \vee R(z, y)) \wedge (\neg R(x, x) \vee R(y, z))]$
(III) Négations: $\exists x\ \forall y\ \forall z\ ((R(x, x) \wedge \neg R(z, y)) \vee (R(x, x) \wedge \neg R(y, z)))$

(IV) La formule est polie
(V) La formule est prénexe
(VI) Skolem: x ➔ a constante ∃x (-----) choisir un x particulier
$\forall y \forall z ((R(a, a) \wedge \neg R(z, y)) \vee (R(a, a) \wedge \neg R(y, z)))$
$(A \wedge B) \vee (A \wedge C)$ $A \wedge (B \vee C)$
$\forall y \forall z\ R(a, a) \wedge (\neg R(z, y) \vee \neg R(y, z))$

(VII) Forme clausale: 1 $R(a, a)$
2 $\neg R(z, y) \vee \neg R(y, z)$

Un autre exemple: φ: $\forall x \forall y \exists z (P(x, y) \wedge (P(y, z) \rightarrow \neg P(z, z)))$
(I) Clôture universelle: pas de variables libres
(II) Abréviations: $\forall x \forall y \exists z (P(x, y) \wedge \neg P(y, z) \vee \neg P(z, z))$
(III) Négations: néant
(IV) La formule est polie
(V) La formule est prénexe
(VI) Skolem: z dépend de x et de y $z = f(x, y)$
$\forall x \forall y (P(x, y) \wedge \neg P(y, f(x, y)) \vee \neg P(f(x, y), f(x, y)))$
(VII) Distributivité: néant
(VIII) Forme clausale: 1 $P(x, y)$
2 $\neg P(y, f(x, y)) \vee \neg P(f(x, y), f(x, y))$
renommer les variables pour rendre les clauses indépendantes
2 $\neg P(v, f(u, v)) \vee \neg P(f(u, v), f(u, v))$

Remarque: Un problème peut surgir à l'unification s'il devient nécessaire d'unifier
$P(x, y)$ et $P(f(x, y), f(x, y))$ la substitution $x \rightarrow f(x, y)$ n'est pas licite !!
D'où le renommage final

4 - 2 Principe de Résolution

Théorème fondamental (analogue au CP0)
Une formule φ est une conséquence logique d'un ensemble de formules $\{\varphi_1, \varphi_2, ... \varphi_n\}$, ce qui se note $\{\varphi_1, \varphi_2, ... \varphi_n\} \models \varphi$ si et seulement si $\{\varphi_1, \varphi_2, ... \varphi_n, \neg\varphi\} \vdash$
Comme en CP0, la seule clause contradictoire est la clause vide

Le mécanisme de résolution en CP1 $p \vee q$, $\neg q \vee q'$ Mais il faut tenir compte des variables

$$\frac{\sigma(p) \vee \sigma(q),\ \neg\sigma(q) \vee \sigma(q')}{\sigma(q) \vee \sigma(q')}$$

Il faut trouver une substitution σ telle que: $q \vee q'$

La règle de résolution s'applique via **l'unification des termes**
Etant donné 2 termes t_1 et t_2:
Les 2 termes sont unifiables s'il existe une substitution σ: $\sigma(t_1) = \sigma(t_2)$
La substitution peut agir sur (t_1 ou t_2) ou sur (t_1 et t_2)
La substitution pour un terme se propage à toutes les occurrences de ce terme dans la formule

Exemple 1: $\neg[\forall x \exists y \exists z ((R(x, x) \rightarrow R(z, y)) \wedge (\neg R(x, x) \vee R(y, z))]$
Forme clausale: 1 $R(a, a)$

2 $\neg R(z, y) \vee \neg R(y, z)$

Résolution entre 1 et 2 avec la substitution $\begin{cases} y \rightarrow a \\ z \rightarrow a \end{cases}$

Attention: une constante ne peut jamais être substituée par autre chose

La substitution sur 2 donne: $\neg R(a, a) \vee \neg R(a, a)$

la substitution des termes se propage à l'ensemble de la clause

$R(a, a)$ ➔

On en déduit que la formule $\varphi = \neg [\forall x \exists y \exists z ((R (x, x) \rightarrow R(z, y)) \wedge (\neg R(x, x) \vee R(y, z))]$ est un théorème puique $\neg \varphi \vdash$—

Exemple 2: φ: $\forall x \forall y \exists z (P (x, y) \wedge (P(y, z) \rightarrow \neg P(z, z)))$

Forme clausale: 1 $P (x, y)$

2 $\neg P(v, f(u, v)) \vee \neg P(f (u, v), f(u, v))$

Résolution de 1 et 2 avec la substitution $\{x \rightarrow v, y \rightarrow f(u, v)\}$

3 $\neg P(f (u, v), f(u, v))$

Résolution de 1 et 3 avec la substitution $\{x \rightarrow f(u, v), y \rightarrow f(u, v)\}$

On en déduit que la forule φ: $\forall x \forall y \exists z (P (x, y) \wedge (P(y, z) \rightarrow \neg P(z, z)))$ est **non satisfiable**

$\neg\varphi$ théorème:? $\neg\varphi$: $\neg [\forall x \forall y \exists z (P (x, y) \wedge (P(y, z) \rightarrow \neg P(z, z)))]$

constantes

Exemple: φ: $\exists z (Q (x, x) \vee (Q(x, y) \rightarrow Q(x, z)))$

2 variables libres x et y. Est - ce que φ est un théorème ou pas ?

Forme clausale de φ

(I) Clôture universelle: $\forall x \forall y \exists z (Q (x, x) \vee (Q(x, y) \rightarrow Q(x, z)))$

(II) Abréviations: $\forall x \forall y \exists z\ Q (x, x) \vee \neg Q(x, y) \vee Q(x, z)$

(III) Négations: néant

(IV) La formule est polie

(V) La formule est prénexe

(VI) Skolem: z dépend de x et de y $z = f (x, y)$

$\forall x \forall y \exists z\ Q (x, x) \vee \neg Q(x, y) \vee Q(x, f(x, y))$

(VII) Distributivité: néant

(VIII) Forme clausale: $Q (x, x) \vee \neg Q(x, y) \vee Q(x, f(x, y))$

On ne peut rein dire si ce n'est que $\neg\varphi$ n'est pas un théorème

Même travail sur $\neg\varphi$: $\neg [\forall x \forall y \exists z (Q (x, x) \vee (Q(x, y) \rightarrow Q(x, z)))]$

(I) Clôture universelle néant

(II) Abréviations: $\neg [\forall x \forall y \exists z (Q (x, x) \vee (Q(x, y) \rightarrow Q(x, z)))]$

$\neg [\forall x \forall y \exists z\ Q (x, x) \vee \neg Q(x, y) \vee Q(x, z)]$

(III) Négations: $\exists x \exists y \forall z \neg Q (x, x) \wedge Q(x, y) \wedge \neg Q(x, z)]$

(IV) La formule est polie

(V) La formule est prénexe

(VI) Skolem: $\{x \rightarrow a$ et $y \rightarrow b\}$

$\forall z \neg Q (a, a) \wedge Q(a, b) \wedge \neg Q(a, z)$

(VII) Distributivité: néant

(VIII) Forme clausale: 1 $\neg Q (a, a)$

2 $Q(a, b)$

3 $\neg Q(a, z)$

Résolution de 2 et 3 avec la substitution $\{z \rightarrow b\}$ et on en déduit que φ est un théorème.

Autre exemple: Même question pour φ: $\exists x \, \forall y \, \neg ((S(x) \leftrightarrow (S(y)) \wedge \forall x \, S(x))$

(I) Clôture universelle néant

(II) Abréviations: $\exists x \, \forall y \, \neg [(\neg S(x) \vee (S(y)) \wedge (\neg S(y) \vee (S(x))] \wedge \forall x \, S(x)$

(III) Négations: $\exists x \, \forall y \, [(S(x) \wedge \neg (S(y)) \vee (S(y) \wedge (\neg S(x))] \wedge \forall x \, S(x)$

(IV) Politesse: $\exists x \, \forall y \, [(S(x) \wedge \neg (S(y)) \vee (S(y) \wedge (\neg S(x))] \wedge \forall u \, S(u)$

(V) Prénexe: $\exists x \, \forall y \, \forall u \, [(S(x) \wedge \neg (S(y)) \vee (S(y) \wedge (\neg S(x))] \wedge S(u)$

(VI) Skolem: $\{x \rightarrow a\}$

$\forall y \, \forall u \, (S(a) \wedge \neg S(y)) \vee (S(y) \wedge \neg S(a)) \wedge S(u)$

(VII) Distributivité: $[((S(a) \vee S(y)) \wedge ((S(a) \vee \neg (S(a)) \wedge ((\neg S(y) \vee S(y)) \wedge ((\neg S(y)) \vee (\neg S(a))]$

$\forall y \, \forall u \, (S(a) \vee S(y)) \wedge (\neg S(y) \vee \neg S(a)) \wedge S(u)$

(VIII) Forme clausale:

1 $S(a) \vee S(y)$

2 $\neg S(y) \vee \neg S(a)$

3 $S(u)$

Résolution de 2 et 3 avec la substitution $\{u \rightarrow a\}$ 4 $\neg S(y)$

Résolution de 3 et 4 avec la substitution $\{u \rightarrow y\}$ et on en déduit que $\neg\varphi$ est un théorème.

Exercice 9 - 4): $\{(\forall x \, \forall y \, \forall z \, P(x, y) \wedge P(y, z) \rightarrow Q(z, z)), \forall x \, \exists y \, P(x, y)\} \models \forall x \, \exists y \, Q(y, x)$

$\{\varphi_1, \varphi_2\} \models \varphi_3$ ssi $\{\varphi_1, \varphi_2, \neg \varphi_3\} \vdash$

formes clausales de φ_1, φ_2, et $\neg \varphi_3\}$

(I) **Forme clausale de φ_1**

- pas de variables libres
- abréviations: $\forall x \, \forall y \, \forall z \, \neg P(x, y) \vee \neg P(y, z) \vee Q(z, z)$
- la formule est polie et prénexe
- skolemisation non nécessaire
- forme clausale: $\neg P(x, y) \vee \neg P(y, z) \vee Q(z, z)$

(II) **Forme clausale de φ_2**

- $\forall x \, \exists y \, P(x, y)$
- seul problème: skolemisation y = f(x)
- la clause: P(f (x), x)

(III) **Forme clausale de $\neg \varphi_3$**

- $\neg (\forall x \, \exists y \, Q(y, x))$ $\exists x \, \forall y \, \neg Q(y, x)$ polie et prénexe
- x = a (constante) $\forall y \, \neg Q(y, a)$
- forme clausale: $\neg Q(y, a)$

Rappel Skolem: $\exists x \, \forall y \, \ldots \rightarrow x = a$ a est une constante

$\forall x \, \exists y \, \ldots \rightarrow y = f(x)$ dépendance entre x et y

1 $\neg P(x, y) \vee \neg P(y, z) \vee Q(z, z)$

2 P(f (x), x) se réécrit: P(f (u), u) renommage de la variable x

3 $\neg Q(y, a)$ se réécrit $\neg Q(v, a)$ renommage de la variable y

résolution 3 et 1 avec les substitutions: $\{v \rightarrow x, z \rightarrow a\}$

4 $\neg P(x, y) \vee \neg P(y, a)$

résolution 2 et 4 avec les substitutions: $\{u \rightarrow a, f(u) \rightarrow f(a), y \rightarrow f(a)\}$

5 $\neg P(x, f(a))$ conséquence de $u \rightarrow a$

résolution 2 et 5 avec les substitutions: $\{u \rightarrow f(a), f(u) \rightarrow f(f(u))\}$

contrainte la plus forte conséquence de la contrainte

la substitution se propage dans le littéral lui - même et donc x $\rightarrow$ f(f (u)) substitution résultante

Remarque: 2 choses importantes

(I) Il faut faire la preuve et donc arriver à (si c'est possible)

(II) Les substitutions ont leur importance: il est possible de s'interesser à ces substitutions ➔ PROLOG

5. - Représentation de phrases du langage naturel en CP1

4 formes fondamentales

(I) **Tous les A sont des B** (universelle positive) CP0: $A \rightarrow B$

$\forall x\ A(x) \rightarrow B(x)$ $\forall x$ Homme $(x) \rightarrow$ Mortel (x)

(II) **Aucun A n'est un B** (universelle négative) CP0: $A \rightarrow \neg B$

$\forall x\ A(x) \rightarrow \neg B(x)$ $\forall x$ Homme $(x) \rightarrow \neg$ Mortel (x)

(III) **Traitement d'exceptions** (universelle avec exceptions)

Tous les A sauf les B sont des C = a moins d'être un B, les A sont des C

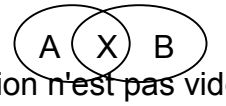
CP0: $A \wedge \neg B \rightarrow C$

$\forall x\ A(x) \wedge \neg B(x) \rightarrow C(x)$

(IV) Existancielles

Il existe des A qui sont des B en CP0: $A \wedge B$ $\exists x\ A(x) \wedge B(x)$

Il existe des A qui ne sont pas des B $\exists x\ A(x) \wedge \neg B(x)$



L'intersection n'est pas vide

Remarque: La négation d'une universelle $\neg$(Tous les A sont des B)

$\neg(\forall x\ A(x) \rightarrow B(x)) = \neg(\forall x \neg A(x) \vee B(x))$ **$= \exists x\ A(x) \wedge \neg B(x)$**

Il existe des A qui ne sont pas des B

(V) **Relations:** Tous les A sont en relation avec tous les B par R

$\forall x\ \forall y\ A(x) \wedge B(y) \rightarrow R(x, y)$

les femmes aiment les hommes: $\forall x\ \forall y$ femme $(x) \wedge$ hommes $(y) \rightarrow$ aime (x, y)

(VI) **Certains A sont en relation avec certains B par R**

$\exists x\ \exists y\ A(x) \wedge B(y) \wedge R(x, y)$

Certains hommes aiment certaines femmes $\exists x\ \exists y$ Homme $(x) \wedge$ Femme$(y) \wedge$ aime (x, y)

Forme clausale: $x \rightarrow a$, $y \rightarrow b$ Skolem

Homme $(a) \wedge$ Femme$(b) \wedge$ aime (a, b)

Exemple et exercices:

Exercice 3:

(1) Certains A sont en relation par R avec tous les B

Existancielle universelle $\exists x\ A(x) \wedge \forall y\ B(y) \rightarrow R(x, y) \leftarrow$ 2 clauses

Certains hommes aiment toutes les femmes jolies

Existancielle A relation universelle B

$\exists x$ Homme $(x) \wedge [\forall y$ Femme_jolie $(y) \rightarrow$ aime $(x, y)]$

forme clausale: 1. Homme (a)

2. $\neg$Femme_jolie $(y) \vee$ aime (a, y)

Phrase duale

(1') Tous les A sont en relation avec certains B par R

Universelle existancielle $\forall x\ \exists y\ A(x) \wedge B(y) \rightarrow R(x, y) \leftarrow$ 1 clause

$\forall x\ \exists y \neg A(x) \vee \neg B(f(x)) \vee R(x, f(x))$

$\forall x \xrightarrow{R} \forall y$	$A \rightarrow B$
$\forall x \xrightarrow{R} \exists y$	(1')
$\exists x \xrightarrow{R} \forall y$	(1)
$\exists x \xrightarrow{R} \exists y$	$A \wedge B$

(2) aucun homme n'aime une femme dépensière
∀x ∀y homme (x) ∧ femme_dépensière (y) ➔ ⌝ aime (x, y)
⌝ homme (x) ∨ ⌝ femme_dépensière (y) ∨ ⌝ aime (x, y)

(3) une femme dépensière n'est pas jolie
article indéfini ➔ universel
∀x femme_dépensière (x) ➔ ⌝ femme-jolie (x)
Forme clausale: ⌝ femme_dépensière (x) ∨ ⌝ femme-jolie (x)
Les clauses après renommage:
1. Homme (a)
2. ⌝ Femme_jolie (y) ∨ aime (a, y)
3. ⌝ homme (x) ∨ ⌝ femme_dépensière (z) ∨ ⌝ aime (x, z) renommage y en z
4. ⌝ femme_dépensière (u) ∨ ⌝ femme-jolie (u) renommage x en u
{1, 2} |= 3 ? vrai ssi {1, 2, ⌝ 3} |—

Il faut expliciter ⌝ 3: ⌝ (∀x femme_dépensière (x) ➔ ⌝ femme-jolie (x))
ou prendre la négation de 4 négation de 4
forme clausale de (⌝ φ) = ⌝ (forme clausale φ) ? Skolemisation
⌝ 4.1 femme_dépensière (u)
⌝ 4.2 femme_jolie (u)

Résolution de 2 avec ⌝ 4.2 {u ➔ y} 5. Aime (a, y)
Résolution de 2 avec ⌝ 4.1 {u ➔ z} 6. ⌝ homme (x) ∨ ⌝ Aime (x, z)
Résolution de 5 avec 6 {x ➔ a, y ➔ z} 7. ⌝ homme (a)
Résolution de 1 avec 7

Exercice 1:
1. ∀y aliment (y) ➔ aime (Jean, y) ⌝ aliment (y) ∨ aime (Jean, y)
2. universelle ∀x pomme (x) ➔ aliment (x) ⌝ pomme (x) ∨ aliment (x)
aliment (pomme)
3. ∀x poulet (x) ➔ aliment (x) ⌝ poulet (x) ∨ aliment (x) aliment (poulet)
4. ∀x ∀y personne (x) ∧ manger (x, y) ∧ vivant (x) ➔ aliment (y)
Personne mange aliment "connaissance implicite"
∀x ∀y manger (x, y) ∧ vivant (x) ➔ aliment (y)
⌝ manger (x, y) ∨ ⌝ vivant (x) ∨ aliment (x)
5. manger (Jacques, cacahouètes) ∧ vivant (Jacques)
relation complètement instanciée
5.1 manger (Jacques, cacahouètes)
5.2 vivant (Jacques)
6. ∀z manger (Jacques, z) ➔ manger (Hélène, z)
⌝ manger (Jacques, z) ∨ manger (Hélène, z)
7. Jean aime les cacahouètes aime (Jean, cacahouètes)
{1, 2, 3, 4, 5, 6} |= 7 ssi {1, 2, 3, 4, 5, 6, ⌝ 7} |—
⌝ 7 ⌝ aime (Jean, cacahouètes)
Résolution de 1 avec ⌝ 7 {y ➔ cacahouètes} 8. ⌝ aliment (cacahouètes)
Résolution de 4 avec 8 {y ➔ cacahouètes} 9. ⌝ manger (x, cacahouètes) ∨ ⌝ vivant (x)
Résolution de 5.1 avec 9 {x ➔ Jacques} 10. ⌝ vivant (Jacques)
Résolution de 10 avec 5.2

Que mange Hélène ? Quelle sont les substitutions possibles pour z dans manger (Hélène, z)
Résolution de 5.1 avec 6 {x ➔ cacahouètes} 11. manger (Hélène, cacahouètes)

On aurait pu poser comme but à résoudre: ⌉ manger (Hélène, cacahouètes) et montrer que l'on obtient
Résolution de 4 avec 5.2 {x ➔ Jacques} 12. ⌉ manger (Jacques, y) ∨ aliment (y)
Résolution de 1 avec 2 {y ➔ pomme} 15. aime (Jean, pomme)
Résolution de 1 avec 3 {y ➔ poulet} 16. aime (Jean, poulet)

Comlpément: Jacques menge toutes sortes d'aliments
∀u aliment (u) ➔ manger (Jacques, u) 17 ⌉ aliment (u) ∨ manger (Jacques, u)
Résolution de 2 avec 17 {u ➔ pomme} 18. manger (Jacques, pomme)
Résolution de 3 avec 17 {u ➔ poulet} 19. manger (Jacques, poulet)
Résolution de 6 avec 18 {z ➔ pomme} 20. manger (Hélène, pomme)
Résolution de 3 avec 17 {z ➔ poulet} 21. manger (Hélène, poulet)
3 instanciations pour z: cacahouètes
pomme
poulet
Pouvez que les cacahouètes sont des aliments? Aliment (cacahouètes) ?
22. ⌉ aliment (cacahouètes)
Résolution de 4 avec 22 {y ➔ cachouètes} 23. ⌉ manger (x, cacahouètes) ∨ ⌉ vivant (x)
Résolution de 5.1 avec 23 {x ➔ Jacques} 24. ⌉ manger (Jacques, cacahouètes)
Résolution de 5.1 avec 24 donne

Exercice 5:
1 x a un compte bancaire approvisionné être_approvisionné (x) il n'est pas utile de distinguer le compte bancaire, donc 1 relation binair'e ne s'impose pas
être_approvisionné (Marcel)
2. relation disponible entre l'individu les contes et l'individu quai des prunes (qdp)
1 relation entre 2 individus est une relation totalement instanciée
disponible (les_contes, qdp)
3. 1 **librairie** céde un **livre** à 1 **personne** relation ternaire
∀x ∀y∀z librairie (x) ∧ livre (y) ∧ personne (z) ∧ céder (x, y, z) ➔ posséder (z, y)
⌉ librairie (x) ∨ ⌉ livre (y) ∨ ⌉ personne (z) ∨ ⌉ céder (x, y, z) ∨ posséder (z, y)
4. veut_lire (Marcel, les_contes) idem à 2
5. ∀x ∀y∀z librairie (x) ∧ livre (y) ∧ personne (z) ∧ disponible (y, z) ∧ acheté (z, y) ➔ céder (x, y, z)
⌉ librairie (x) ∨ ⌉ livre (y) ∨ ⌉ personne (z) ∨ ⌉ disponible (y, x) ∨ ⌉ acheté (z, y) ∨ céder (x, y, z)
6. ∀y∀z personne (y) ∧ être_approvisionné (z) ∧ veut_lire (z, y) ➔ acheté (z, y)
⌉ personne (y) ∨ ⌉ être_approvisionné (z) ∨ ⌉ veut_lire (z, y) ∨ acheté (z, y)
7. posséder (Marcel, les_contes)
⌉7. ⌉ posséder (Marcel, les_contes) {1, 2, 3, 4, 5, 6, ⌉7} |—

Remarque: Personne fait partie de l'univers du problème, soit il est supprimé, soit il faut préciser que Marcel est 1 personne

Résolution de 3 avec ⌉7 {y ➔ les_contes, z ➔Marcel}

8. ⌉ librairie (x) ∨ ⌉ livre (les_contes) ∨ ⌉ personne (Marcel) ∨ ⌉ céder (x, les_contes, Marcel)
Résolution de 5 avec 8 {y ➔ les_contes, z ➔ Marcel}
9. ⌉ librairie (x) ∨ ⌉ livre (les_contes) ∨ ⌉ personne (Marcel) ∨ ⌉ disponible (les_contes, x) ∨ ⌉ acheté (Marcel, les_contes)
Résolution de 1 avec 6 {z ➔ Marcel}
10. ⌉ personne (Marcel) ∨ ⌉ veut_lire (Marcel, y) ∨ acheté (Marcel, y)
Résolution de 4 avec 10 {y ➔ les_contes}
11. ⌉ personne (Marcel) ∨ acheté (Marcel, les_contes)
Résolution de 9 avec 11
12. ⌉ librairie (x) ∨ ⌉ livre (les_contes) ∨ ⌉ personne (Marcel) ∨ ⌉ disponible (les_contes, x)
Résolution de 2 avec 12 13. ⌉ librairie (qdp) ∨ ⌉ livre (les_contes) ∨ ⌉ personne (Marcel)

Univers du problème: connaissances implicites { personne (Marcel); Librairie (qdp); Livre (les_contes) ➔

Exercice 6:
1. crime (y) y est 1 crime commettre (x, y) x commet y (la personne x commet l'acte (y) ∀x ∃y crime (y) ∧ personne (x) ➔ commettre (x, y) x = f(x) ➔ auteur (y)
⌉ crime (y) ∨ ⌉ personne ((auteur (y)) ∨ commettre (auteur (y), y) Skolem
2. Tous les gens malhonnêtes commettent des crimes malhonnête (x) x est 1 personne malhonnête quelqu'un d'honnête peut très bien commettre 1 crime contrairement à ce que dit l'énoncé ∀x ∀y malhonnête (x) ∧ crime (y) ➔ commettre (x, y)
Si quelqu'un commet un crime alors il est malhonnête
∀x ∀y crime (y) ∧ commettre (x, y) ➔ malhonnête (x)
⌉ crime (y) ∨ ⌉ commettre (x, y) ∨ malhonnête (x)
3. Ne sont arrêtés que les gens malhonnêtes Seul sont arêtés les gens malhonnêtes
∀u arrêté (u) ➔ malhonnête (u) arrêté (u) = u est arrêté (u est 1 personne arrêtée)
4. ∀z ∀y malhonnête (z) ∧ crime (y) ∧ arrêté (z) ➔ commettre (x, y)
⌉ malhonnête (z) ∨ ⌉ crime (y) ∨ ⌉ arrêté (z) ∨ commettre (z, y)
5. ∃y crime (y) Skolem: y ➔ a crime (a)
6. ∃z malhonnête (z) ∧ ⌉ arrêté (z) Skolem: z ➔ b malhonnête (b)
⌉ arrêté (b)
1. ⌉ crime (y) ∨ commettre (auteur (y), y)
2. ⌉ crime (y) ∨ ⌉ commettre (x, y) ∨ malhonnête (x)
3. ⌉ arrêté (u) ∨ malhonnête (u)
4. ⌉ malhonnête (z) ∨ ⌉ crime (y) ∨ ⌉ arrêté (z) ∨ commettre (z, y)
5. crime (a)
6. malhonnête (b)
⌉ arrêté (b)
Pour montrer que {1, 2, 3, 4, 5} |= 6, il faut montrer que {1, 2, 3, 4, 5, ⌉ 6} |—
Forme clausale de ⌉ 6 ⌉ (∃z malhonnête (z) ∧ ⌉ arrêté (z))
Renommage ⌉ malhonnête (t) ∨ arrêté (t)

Résolution de 4 avec ⌉ 6 avec la substitution {t ➔ z}
7. ⌉ malhonnête (z) ∨ ⌉ crime (y) ∨ commettre (z, y)
Résolution de 6 avec 7 avec la substitution {y ➔ a}

8. ⌉malhonnête (z) ∨ commettre (z, y)
Résolution de 2 avec 8 avec les substitutions {y ➔ a, x ➔ z}
9. ⌉crime (a) ∨ commettre (z, a)
Résolution de 1 avec 9 avec les substitutions {y ➔ a, z ➔ auteur(a)} 10. ⌉crime (a)
Résolution de 5 avec 10

Une brève introduction à PROLOG

Où comment nouys allons exploiter tout ce qui a été vu en CP1 avec un autre point de vue

1. - Les formules du langage Prolog

Les ingrédients du langage sont essentiellement: + des variables
+ des constantes
+ des fonctions
+ des prédicats

Remarque: Les connecteurs booléens et les quantificateurs sont implicites (quantificateurs "universelle" seulement)

Il existe 3 formes génériques:

(I) les **faits** qui sont toujours vrais

(II) les **règles** qui correspondent à des implications de type A ➔ B et dans ce cas un fait apparaît comme la conclusion d'une règle sans conditions

(III) les **questions** qui correspondent aux buts à résoudre.
Pour résoudre un tel but en CP1, on prend sa négation et on cherche à dériver la clause vide .
En Prolog, on s'intéresse plutôt à **effacer** une question comme un prédicat s'effacera dans 1 démonstration, et on mémorise les **instanciations de variables réalisées**.

2. - La notation des formules

Formes générales d'une clause: $\underbrace{A_1 \vee A_2 \vee \ldots}_{\text{Symboles positifs}} \vee \underbrace{A_n \vee \neg A_{n+1} \vee \ldots \vee \neg A_{n+m}}_{\text{Symboles négatifs}}$

En Prolog on manipule des **clauses de HORN** qui n'ont qu'un seul littéral positif.

Une clause de Horn est notée: $A_0 \vee \neg A_1 \vee \ldots \vee \neg A_n$

En terme d'implication: $A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow A_0$

A_0 = tête de clause $\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n$ = le corps de la clause
$A_1 \wedge A_2 \wedge \ldots \wedge A_n$

Un **fait** correspond à la donnée A_0 ← tête de la clause

Une **règle** correspond à une clause

Une **question** correspond à la donnée d'un littéral négatif qu'il faut effacer

Notation Prolog: $A_0 \vee \neg A_1 \vee \ldots \vee \neg A_n$ s'écrit:

Règle A_0 ← $A_1, A_2, \ldots A_n$**.** (←: :—) se lit A_0 si A_1 et A_2 et … et A_n**.**

Un **fait** A_n**.**

Une **Question**: Q?

Un exemple de programme Prolog

Une règle: repas (e, p, d) ← entrée (e), plat (p), dessert (d)**.**

Des faits: entrée (carotte)**.**
Entrée (tomate)**.**
Plat (choucroute)**.**
Plat (pate)**.**
Dessert (pomme)**.**
Dessert (mousse)**.**

Une question: Il existe différentes façons de poser les questions: plat (pate)? Répondre à cette question consiste à vérifier que cette instanciation est vraie. Plat (carotte)?
Une autre façon de poser la question: **plat (x)?**
Dans ce cas, on cherche toutes les instanciations de x qui font que le but est satisfait

3. - Le principe d'effacement

Etant donné une question Q? comment la traiter?

Etant donné Q? l'interprète Prolog recherche dans la base de clauses, les clauses dont la tête est le prédicat Q

Si l'unification de la question Q? avec la tête de la clause Q est possible, alors la tête de clause **s'efface** pour être remplacée dans le cas d'une règle par autant de nouvelles questions que de prédicats dans le corps de la clause et la question est remplacée par rien si la tête de clause correspond à un fait.

Les 3 cas principaux d'effacement: La question est: Q(z)?

(I) S'il existe un fait de la forme Q(a)**.** alors l'interprète unifie Q(z)et Q(a) et le résultat de l'unification - les substitutions réalisées - est retournée {z = a}
Exemple: plat (x)? {x = pate, x = choucroute}

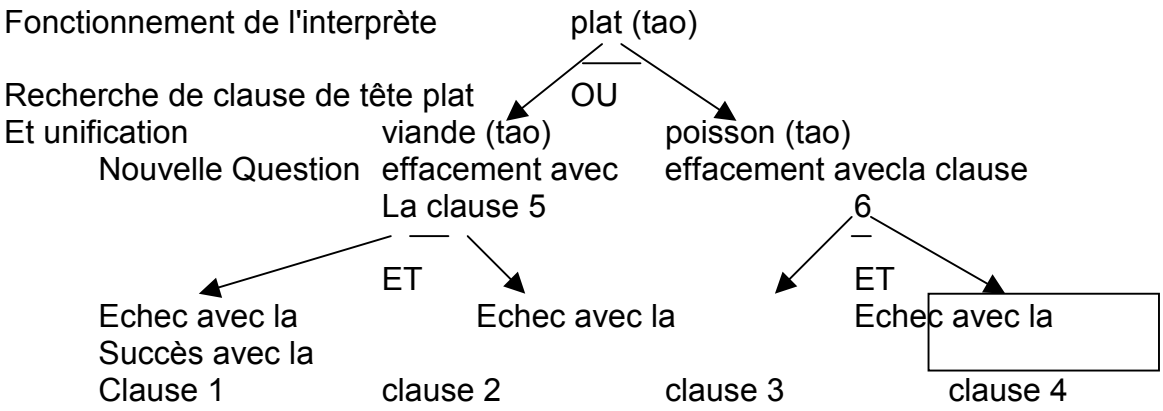
(II) S'il existe une règle de la forme Q(x) ← P(x), …., R(x) alors l'interprète unifie la question Q(z)? et la tête de clause Q(x), et la question Q(z) s'efface au profit de nouvelles questions:
P(x)? R(x)? …. Qu'il faut à leur tour effacer
Exemple: repas (carotte, choucroute, d)?
Unification avec la tête de la règle repas (e, p, d)
repas (e, p, d) ← entrée (e), plat (p), dessert (d) avec **substitution**, **propagation des substitutions** dans le corps de la clause et **génération de nouveaux buts:**
substitutions: e ➔ carotte
p ➔ choucroute
propagation des substitutions: entrée (carotte)
plat (cxhoucroute)
génération des nouveaux buts: entrée (carotte)? Vérifier que les 2 questions
plat (choucroute)? Correspondent à 2 faits
dessert (d)? ➔ trouver les instanciations possibles pour d
repas (carotte, choucroute, d)?
{e = carotte, p = choucroute, d = pomme} ∪ {e = carotte, p = choucroute, d = mousse}

(III) Etant donné Q(x)? il n'existe aucun fait et aucune règle de tête Q(a) ou encore l'unification entre Q(x) et les données de la base de clauses échoue.
Dans ce cas là, l'effacement est mis en échec.
Exemple: plat (œuf)?
L'unification avec plat (choucroute) ou plat (pate) est impossible d'où echec.
Pousse-café (y)? Il n'existe pas de clause dont la tête est pousse-café, d'où echec.

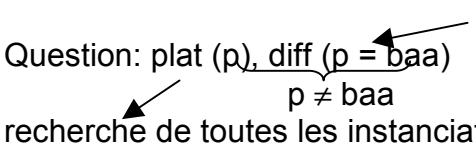
Le programme Prolog: 1. Viande (gdb)**.**

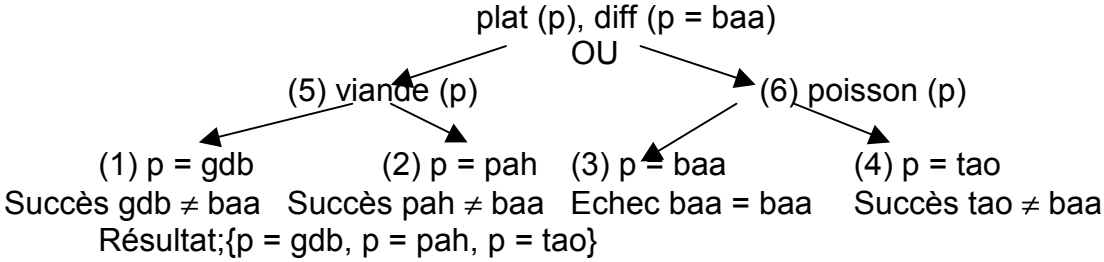
2. viande (pah)**.**
3. Poisson (baa)**.**
4. Poisson (tao)**.**
5. Plat (p) ← viande (p) Interprétation: un plat est soit
6. Plat (p) ← poisson (p) une viande soit un poisson

Question: plat (tao)?



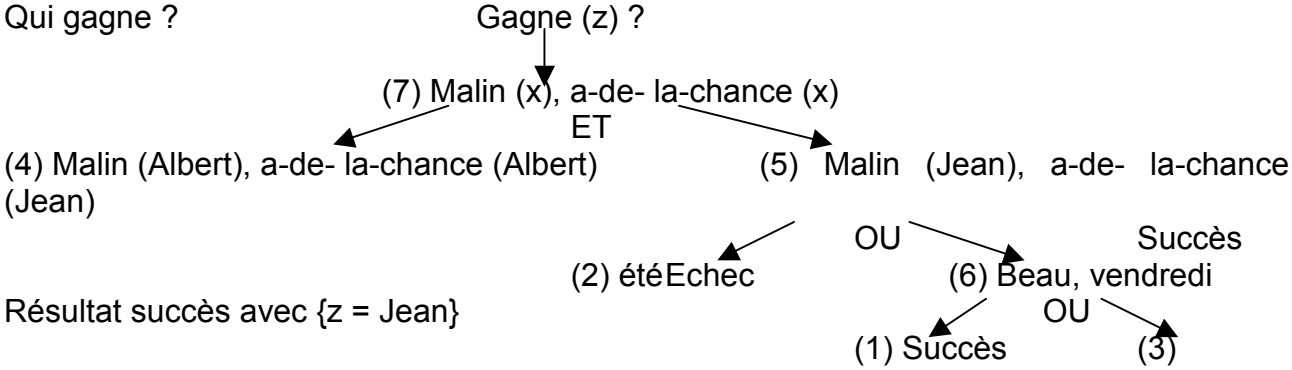
Remarque: l'interprète parcours la base séquentiellement et construit l'arbre ET - OU de résolution en profondeur





Exercice 7 (devoir) en Prolog

1. Beau**.** (Il fait beau)
2. Eté ➔ a-de- la-chance (Jean) a-de- la-chance (Jean) ← été**.**
3. Vendredi**.**
4. Malin (Albert)**.**
5. Malin (Jean)**.**
6. Beau ∧ vendredi ➔ a-de- la-chance (Jean)
   a-de- la-chance (Jean) ← Beau, vendredi**.**
7. ∀x Malin (x) ∧ a-de- la-chance (x) ➔ gagne (x)
   gagne (x) ← Malin (x), a-de- la-chance (x)**.**



Introduction aux logiques de descriptions

Les logiques de description forment une famille de langages de **représentation de connaissances** qui s'appuient à la fois sur la logique et les "représentations structurées" (au sens ou on y manipule des **structures**)

1. - Les grands principes

(I) Les éléments de base du langage sont des **concepts**, des **rôles** et des **individus**. Les concepts représentent des **classes** d'individus et les rôles des relations binaires entre les classes

(II) Chaque concept (et chaque rôle) possèdent une **description structurée** construite à partir de rôles et de **constructeurs** (ou **opérateurs**). Cette description est l'équivalent d'une fbf en CP1

(III) A chaque decription on associe par l'intermédiaire d'une fonction d'interprétation
Une interprétation (parallèle avec l'interprétation d'une formule en CP0 ou Cp1)

(IV) Les descriptions - concepts ou rôles - sont organisées en hérarchies par l'intermédiaire d'une relation de subsomption.
Intuitivement, le concept C subsume le concept D si C est plus "général" que D (le concept "couleur" subsume le concept "noir")

(V) Les opérations de base du **raisonnement** dit **terminologie** (c a d le **raisonnement** dans une logique de description) sont la **classification** d'individus:
- Classifier un concept c'est retrouver sa place dans l'ordre de la hiérarchie des concepts.
- Classifier un individu c'est retrouver les concepts dont l'individu est instance

2. - Le langage ALN et le Famille AL:

2 - 1 Le langage AL:

Le langage AL est donné par la grammaire suivante:

Concept ➔ A (Concept primitif ≈ atome en logique)

| (and C D) (conjonction des concepts C et D C ⊓ D

| (all r C) (quantification universelle où r est 1 rôle et C 1 concept s'écrit aussi: ∀r, C)

Intuitivement x r y (all r C)

Rôle ➔ ∀y instance de C

| (not A) (négation qui intervient que sur les concepts primitifs, ⌉A)

| (some r) (quantification existancielle non typée, s'écrit aussi ∃r)

x r y (some r) ∃ nécessairement au moins 1 y en relation avec x

| Top (concept le plus général ⊤)

|Bottom (concept le plus spécifique ⊥)

Exemples:

a: Une personne dont tous les enfants sont étudiants

(and Personne ← concepts primitifs

(all enfant Etudiant))

rôle

en technologie objet on avait

classs PEE

Superclass Personne, enfant type Etudiant

b: Une personne qui a 1 enfant
(and Personne ← concepts primitifs
(some enfant)) ← rôle
c: Une personne qui a des diplômes
(and Personne
(some Diplômes))
Remarque: Un rôle définit une relation entre 2 concepts; il serait plus correct d'écrire avoir - enfant ou avoir - diplômes que enfant ou diplômes.

d: Une personne dont tous les enfants sont grands et qui a des enfants
(and Personne ← concepts primitifs
(all ami Grand)
rôle (some enfant))

Autre notation: Personne ⊓ ∀ ami, Grand ⊓ ∃ enfant

2 - 3 Le langage ALN:

ALN est obtenu à partir de AL en autorisant une cardinalité sur les rôles: avoir au plus ou avoir au moins

(atleast n r) se note aussi (≥ n r)
(atmost n r) se note aussi(≤ n r)
Intuitivement: **atleast** indique r doit avoir **au moins** n valeurs
atmost indique r doit avoir **au plus** n valeurs

Que signifie avoir une valeur pour un rôle?
Rôle "voiture" (avoir une voiture)
Personne voiture → véhicule
x → y = voiture(x) y = valeur élémentaire pour le rôle
domaine du rôle co - domaine du rôle

(atleast 2 voiture) ➔ avoir au moins 2 Voitures x voiture y_1 il existe au moins y_1 et y_2
x voiture y_2

Exemples:
e: Une personne qui a au moins 2 voitures et dont toutes les voitures sont grandes
(and Personne ← concepts primitifs
(all voiture Grand)
(atleast 2 voiture)) ← rôle

f: Une personne qui a exactement 1 enfant
(and Personne
(atleast 1 enfant)
(atmost 1 enfant)
Remarque: "avoir des enfants" = (some enfant)
= (tleast 1 enfant) ← il existe **au moins** 1 enfant

Exemple: Représentation du concept "Grand parent", de Personne dont tous les enfants sont parents
(and Personne concept
concept → (all enfant Parent)) → (atleast 1 enfant) ou (atmost 1 enfant)
concept ou expression conceptuelle
(and Personne bonne façon d'écrire
(all enfant (some enfant)))

2 - 3 Les extensions de ALN:

(I) **la disjonction de concepts** ALU = Al ∪ {U} (or C D) qui s'écrit: C ⊔ D
Exemple: Un groupe dont tous les membres sont des garçons ou des filles
(and Groupe concepts primitifs
(all membre (or Garçon Fille))

(II) **quantification existancielle typée:** (c-some r C) qui s'écrit aussi ∃r**.** C
extension du (some r) où un co - domaine est fixé pour le rôle r ALE = AL ∪ {c-some}
Exemple: Une personne qui a des enfants docteurs
(and Personne
(c-some enfant Docteur))
Remarque: (and Personne ≡ (and Personne
(some enfant)) (c-some enfant Top))
(some r) ≡ (c-some r Top)

(III) La négation de concepts quelconques:
(not C) ou ⌉C ALC = AL ∪ {négation complète}
Exemple: Une personne qui n'a pas d'enfant Dfescription avec une négation
(and Personne ≡ (and Personne
(not (some enfant))) (all enfant Bottom))
D'où cela sort - il? Des règles qui suivent.
Règles: Pour tout concept C:

C ⊔ ⌉C = ⊤ (Top) ⌉⊤ = ⊥
C ⊓ ⌉C = ⊥ (Bottom) ⌉⊥ = ⊤
⌉(∀r**.** C) = ∃r**.** ⌉C
⌉(∃r**.** C) = ∀r**.** ⌉C

Exemple: ne pas avoir d'enfant ⌉(∃ enfant) = ⌉(∃ enfant**.** ⊤) = ∀enfant**.** ⊥
"ne pas avoir d'enfants" (all enfant ⊥)Autre écriture: (atmost 0enfant)
Dans ALC: ⌉(∀r**.** C) = ∃r**.** ⌉C all c-some ALE
⌉(C ⊓ D) = ⌉C ⊔ ⌉D
and or ALU
Dans AlC est aussi dans ALUE et réciproquement. ON a l'habitude de noter: ALC et non ALUE ou ALCUE on a toujours U et E

(IV) 2 constructeurs sur les rôles

(and r_1 r_2) se note $r_1 \sqcap r_2$ ALR = AL ∪ {conjonction de rôle}
Exemple: Une personne dont la taille et la force sont grandes
(and Personne ou bien (and Personne
(all (and taille force) Grand)) (all taille Grand)
(all force Grand))

La construction **restrict** ou **range**
Cette construction s'appuie à un rôle et permet de restreindre le co-domaine du rôle.
Domaine ➔ range

(restrict r C) se note aussi (r | C)
range
Exemple: Une personne dont tous les enfants docteurs sont musiciens
(and Personne
(all enfant-docteur Musicien)) (restrict enfant Docteur)
Le concept de Grand - mère (and Personne parent
(all sexe feminin)
(some (restrict enfant (some enfant))))
Exemple: avoir au moins 1 fils docteur et au moins 2 filles musiciennes
(and Personne
(atleast 1 (restrict fils Docteur))
(atleast 2 (restrict fille Musicienne)))
(c-atleast n r C) ≡ (atleast n (restrict r C))
(c-atmost n r C) ≡ (atmost n (restrict r C))
Remarque: (c-some r C) ≡ (some (restrict r C)) AL+ = AL ∪ {restrict}

3. - Concepts primitifs et concepts définis

Notion de terminilogie

Un concept primitif est comparable à un atome de la logique classique.
Un concept primitif est utilisé comme élément de base pour construire des **concepts définis**.

Un concept défini possède une **définition** et partout où il apparaît il peut être remplacé par cette définition
Qu'exprime une telle définition?

Explication sur un exemple: Description du concept défini Parent:

Parent := (and Personne
Nom de concept symbole (some enfant))
Défini de définition

Sens de cette définition:

Si x est une instance de Parent, alors nécessairement il existe y tel que x enfant de y soit vrai
Mais la réciproque est vraie aussi: **condition nécesaire**

S'il existe x instance de Personne et y tel que x enfant y alors x est instance de Parent

Les expressions conceptuelles qui composent la description d'un concept défini jouent le rôle de conditions nécessaires et suffisantes pour qu'un individu soit instance du concept défini (**moteur** de la classification)

Exemple:

1. Une équipe est définie comme un ensemble dont tous les membres sont des personnes et qui comporte au moins 2 membres
2. Une petite équipe est définie comme 1 équipe qui comprte au plus 5 membres
3. Une équipe moderne est définie comme une équipe qui comporte au plus 4 membres, au moins 1 chef qui est un membre et dont tous les chefs sont des femmes.

Représentation des éléments du problème
:< introduction au concept primitif
:= introduction au concept défini

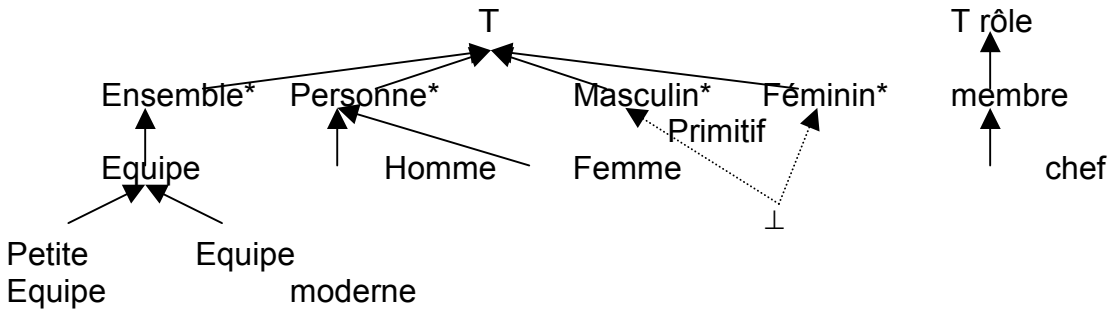
Ensemble :< ⊤ Ensemble est un concept primitif , sous concept direst de ⊤

Personne :< ⊤ Terminologie = hiérarchie des concepts

Masculin :< ⊤ **Concepts définis:**

Feminin :< ⊤ Homme := (and Personne (all sexe Masculin)) Femme := (and Personne (all sexe Feminin)

Remarque: Homme et Femme doivent être déclarés comme des concepts incompatibles



Deux Rôles: membre :< T rôle rôle primitif, sous - rôle de T rôle (rôle le plus général)
Chef :< membre chef est un sous - rôle primitif de membre

Equipe := (and Ensemble
(all membre Personne)
(atleast 2 membre))

Petite-équipe := (and Equipe
(atmost 5 membre))

Equipe-moderne := (and Equipe
(atmost 4 membre)
(atleast 1 chef)
(all chef Femme))

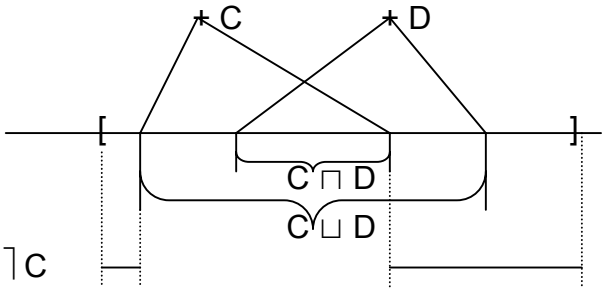
4. - La notion d'interprétation

Définition: Une **interprétation** I = (Δ, $.^I$) est la donnée de Δ **domaine** de l'interprétation et d'une **fonction d'interprétation** $.^I$ qui à un concept C fait correspondre C^I un sous - ensemble de Δ et à un rôle r un sous - ensemble de Δ * Δ noté r^I en respectent les règles données ci - dessous.

Noté: C^I = l'extension de C et coorespond à l'ensemble des instances de C
r^I = l'extension de r et correspond à un ensemble de couples d'instances instanciant r

Règles d'interprétation:

$\top^I = \Delta$	$(C \sqcap D)^I = C^I \cap D^I$	**all**
$\bot^I = \varnothing$	$(C \sqcup D)^I = C^I \cup D^I$	∃
	$(\neg C)^I = \Delta - C^I = L\Delta^{CI}$	**atleast**
		atmost
		restrict



Exercice 2

A Groupe membre (être-membre-de)Personne
A = le concept dont tous les x vérifient la propriété p co-domaine du rôle
rôle
A = groupe de personnes dont les membres de sexe féminin sont adultes
Subsumant direct rôle restriction
Co-domaine du rôle

Groupe de personnes = relation de composition le groupe est "composé de" personne

A := (and Groupe groupe dont tous les membres sont des personnes

(all membre Personne) rôle

(all membre adulte)

(all (restrict membre (all sexe feminin) adulte)))

all rôle concept concept

concepts primitifs rôles

Personne membre

Feminin sexe

Adulte posséder

(subsumant de Personne)

Groupe

Vélo

Voiture

Vehicule à moteur (subsumant de Voiture)

$P_{>20}$ (> 18) (< 20)

B: rôle possède ≈ possède (x, y)

x ⟶ y

personne élément

B := (and Personne

(atmost 1 posséder (posséder-voiture)))

posséder-vélo

posséder-voiture := (restrict posséder voiture)

rôle concept

posséder-vélo := (restrict posséder vélo)

B := (and Personne

(atmost 1 (restrict posséder

(and Voiture Vélo))))

C := (and Personne

(atmost 2 (restrict posséder Véhicule-à-moteur)))

D := (and Groupe $P_{>20}$:= (and Personne

(all membre (all age (> 20)))

(and Personne (all age (>20)))))

D := (and Groupe atleast compte les valeurs du rôle

(all membre $P_{>20}$))

Adulte := (and Personne 2 concepts: (> 18) at (> 20) **(> 20) ⊑ (> 18)**

(all age (> 18)

"toutes les personnes qui ont plus de 20ans ont nécessairementplus de 18 ans"

4. - Suite Interprétation

∀r. C **$(\text{all r C})^I = \{ x \in \Delta \mid \forall y \in \Delta : (x, y) \in r^I \rightarrow y \in C^I \}$**

∃r. C **$(\text{c-some r C})^I = \{ x \in \Delta \mid \exists y \in \Delta : (x, y) \in r^I \rightarrow y \in C^I \}$**

$(\text{some r})^I = \{ x \in \Delta \mid \exists y \in \Delta : (x, y) \in r^I \rightarrow y \in C^I \}$

x (all membre Personne) ➔ ∀x

x (c-some membre Personne) ➔ ∃x

$(\text{atleast n r})^I = \{ x \in \Delta \mid \text{card} \{ y \in \Delta \mid (x, y) \in r^I \} \geq n \}$

$(\text{atmost m r})^I = \{ x \in \Delta \mid \text{card} \{ y \in \Delta \mid (x, y) \in r^I \} \leq m \}$

5. - La relation se subsomption et la satisfiabilité

Définition: Le concept C subsume le concept D ssi:

(I) pour toute interprétation I, on a: $\mathbf{D^I \subseteq C^I}$ on note $\mathbf{D \sqsubseteq C}$

C^I = extension de C
= ensemble des individus qui sont des instances de C

(II) Un concept C est satisfiable ssi il existe une interprétation I telle que: $\mathbf{C^I = \varnothing}$

Le concept est non satisfiable sinon

Exemples standards de concepts non satifiables:

$A \sqcap \neg A$ non satisfiable

(and (atleast n r) (atmost m r)) non satisfiable où n > m

En termes de subsomption:

Un concept C n'est **pas satisfiable si** $\mathbf{C \sqsubseteq \bot}$ $\mathbf{C^I \subseteq \bot^I = \varnothing}$

(III) **Equivalence:** Les concepts C et D sont **équivalents** ssi pour toute interprétation I $\mathbf{D^I = C^I}$ on note $\mathbf{C \equiv D}$

En termes de subsomption: $\mathbf{C \sqsubseteq D}$ et $\mathbf{D \sqsubseteq C}$

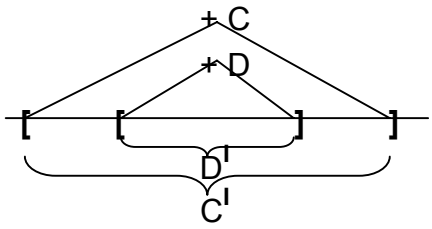
$\mathbf{C^I \subseteq D^I}$ et $\mathbf{D^I \subseteq C^I}$ $\rightarrow$ $\mathbf{C^I = D^I}$

Exemple: si $\mathbf{D \sqsubseteq C}$ alors $\mathbf{D \equiv D \sqcap C}$

$\mathbf{D^I = D^I \cap C^I = D^I \subseteq C^I}$

$\mathbf{D \sqsubseteq D \sqcap C}$ et $\mathbf{D \sqcap C \sqsubseteq D}$

$\mathbf{D^I \subseteq D^I \cap C^I}$ et $\mathbf{D^I \cap C^I \subseteq D^I}$



(IV) **Incompatibilité:** 2 concepts sont **incompatibles** s'ils ne peuvent avoir aucun descendant commun.

Autrement dit, pour toute interprétation I: $\mathbf{C^I \cap D^I = \varnothing}$

Pour la subsomption: $\mathbf{C \sqcap D \sqsubseteq \bot}$

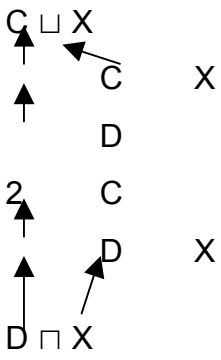
(V) Propriété dites de treillis:

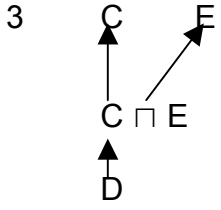
1 - Si $\mathbf{D \sqsubseteq C}$ alors $\forall$X concept: $\mathbf{D \sqsubseteq C \sqcup X}$

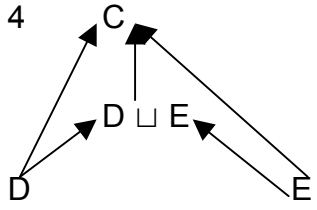
2 - Si $\mathbf{D \sqsubseteq C}$ alors $\forall$X concept: $\mathbf{D \sqcap X \sqsubseteq C}$

3 - Si $\mathbf{D \sqsubseteq E}$ et $\mathbf{D \sqsubseteq C}$ alors: $\mathbf{D \sqsubseteq C \sqcap E}$

4 - Si $\mathbf{D \sqsubseteq C}$ et $\mathbf{E \sqsubseteq C}$ alors: $\mathbf{D \sqcup E \sqsubseteq C}$







(VI) **Autres propriétés:** (all r T) ≡ T ¬(all r T) ≡ ¬T

(and Personne (all enfant T)) (c-some r ⊥) ≡ ¬⊥

(and Personne (all enfant ⊥)

6. - Test de subsomption pour ALN

Il existe 2 façons standard de tester la subsomption: par algorithme de normalisation - comparaison ou par la méthode des tableaux sémantiques.

Algorithme NC pour ALN

Normalisation

(N1) Factorisation des ⊓

$C_1 \sqcap (C_2 \sqcap C_3) \sqcap C_4 \rightarrow C_1 \sqcap C_2 \sqcap C_3 \sqcap C_4$

(N2) Factorisation des co-domaines

(all r C) ⊓ (all r D) ➔ (all r C ⊓ D)

(N3) Traitement des incohérences

A ⊓ ¬A

(atleast n r) ⊓ (atmost m r) ➔ ⊥ et n > m

Remarque: (atleast 1 r) ⊓ (atmost 0 r) ➔ ⊥

(c-some r ⊥) ➔ ⊥ (all r ⊥) ➔ ⊥

A l'issue de la normlisation, on a: $C = C_1 \sqcap C_2 \sqcap \ldots \sqcap C_n$

$D = D_1 \sqcap D_2 \sqcap \ldots \sqcap D_n$

Le concept C subsume D ssi pour tout i ∈ [1, n], il existe j ∈ [1, n] tel que: $D_j \sqsubseteq C_i$ en respectant les règles qui suivent:

[PRIM] Si C_i et D_j sont primitifs: Soit $C_i := D_j$

Soit $C_i :< D_j$ introduction au concepts primitifs

[NOT] Si $C_i = \neg C'_i$ et $D_j = \neg D'_j$ alors: $D'_j \sqsubseteq C'_i$ $\neg C'_i \sqsubseteq \neg D'_j$

[ALL] Si C_i = (all r_C C') et D_j = (all r_D D') alors $C_i \sqsubseteq D_j$ si C' ⊑ D'et $r_D \sqsubseteq r_C$

Exemples: Docteur ⊑ Bachelier ¬Bachelier ⊑ ¬Docteur

(all enfant Docteur) ⊑ (all fils Bachelier)

[ATLEAST] C_i = (atleast n r_C) D_j = (atleast m r_D) $C_i \sqsubseteq D_j$ si n ≥ m et $r_C \sqsubseteq r_D$

(atleast 2 filles) ⊑ (atleast 1 enfant) au moins

[ATMOST] C_i = (atmost n r_C) D_j = (atmost m r_D) $C_i \sqsubseteq D_j$si n ≤ m et $r_D \sqsubseteq r_C$

(atmost 2 enfants) ⊑ (atmost 3 fils) au plus

Une règle supplémentaire

[RESTRICT] (restrict r_C C) ⊑ (restrict r_D D) si C ⊑ D et $r_C \sqsubseteq r_D$

(restict fille Docteur) ⊑ (restrict enfant Bachelier)

Remarque: $(\text{restrict r C})^I = \{(x, y) \in \Delta * \Delta \mid (x, y) \in r^I \wedge y \in C^I$

$(\text{restrict r D})^I \subseteq (\text{restrict r C})^I \qquad (x, y) \in r_D{}^I \subseteq r_C{}^I \wedge y \in D^I \subseteq C^I$

$(x, y) \in r_C{}^I \wedge y \in C^I$

Exercice 6: indéfini atleast ou some

Une personne ayant des enfants artistes

C1 := (and Personne
(atleast 1(restrict enfant Artiste)))
(c-atleast 1 enfant Artiste)

avoir des fils peintres

C2 := (atleast 1(restrict fils Peintre)) **C2 ⊑ C1**

Tous les enfants sont musiciens

C3 := (all enfant musiciens)

Avoir 2 enfants fonctionnaire

C4 := (and (atleast 2(restrict enfant fonctionnaire))
(atmost 2(restrict enfant fonctionnaire)))

r_{C4}

avoir au plus 3 filles institutrices

C5 := (atmost 3(restrict filles Institutrices))

r_{C5}

(and X(atmost ….)) ⊑ (atmost ….) Institutrice ⊑ Fonstionnaire

$r_{C4} \sqsubseteq r_{C5}$? 2 < 3 (atmost 2 r_{C4}) ⊑ (atmost 3 r_{C5})

avoir au moins 3 fils policier Policier ⊑ Fonstionnaire

C5 := (atmost 3(restrict fils Policier))

r_{C6}

(atleast 2 r_{C4}) ⋢ (atleast 3 r_{C6}) ?

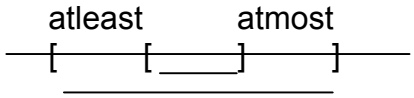
⊒ ➔ C6 ⊑ C4 dans C4 il manque la 2ème partie avec atmost

avoir au moins 2 filles cyclistes

C7 := (atleast 2(restrict fille Cycliste))

Avoir au moins 1 enfant sportif

C8 := (atleast 1(restrict enfant Sportif))



$C_7 \sqsubseteq C_8$

Exercice 2: (all r_D D) ⊑ (all r_C C) $r_C \sqsubseteq r_D$ D ⊑ C

(all membre (>20)) (all (restrict membre (all sexe feminin)) (>18)

(restrict r C) ⊑ r ➔ (restrict r T)

(atmost 1(restrict posséder (and Voiture Vélo)) ⋢ (atmost 2(restrict poséder Véhicule-à-Moteur))

Exercice 1:

Remarque sur la présentation:

(I) avoir quelque chose : x possède y ➔ se représente comme un **rôle** "possède (x, y)"
"avoir une fille" se représente par le **rôle** "fille"

(II) Ensemble de personnes: ensemble dont tous les membres sont des personnes

Concepts primitifs: Ensemble de Personnes **Rôle primitif**: membre

A := (and Ensemble
(all membre Personne
(atleast 3 fille)
(atleast 2 (restrict diplôme Imformatique) "x a un diplôme y = diplôme (x, y)"
diplôme en informatique = restriction sur le co-domaine du rôle
Personne ⟶ Diplôme ← co-domaine
x ⟶ diplôme ⟶ y = diplôme (x)

Concepts primitifs	**Rôles**	
Ensemble	membre	intrument-à-corde
Personne	fille	instrument-à-vent
Informatique	diplôme	instrument-à-percussion
Musicien-de-rock :< Musicien	guitare	tambour

A := (and Ensemble
(all membre Personne)
(atleast 3 (restrict fille (atleast 2 (restrict diplôme Informatique)))))
B := (and Ensemble
(all membre Personne)
(atleast 1 (restrict enfant (atleast 1 diplôme))))
(some r) = (atleast 1 r)
Subsomption entre A et B?

C_i = (atleast n r_C) D_j = (atleast m r_D) $C_i \sqsubseteq D_j$ si $n \geq m$ et $r_C \sqsubseteq r_D$

C_i = (restrict r_C C) $\sqsubseteq$ D_j = (restrict r_D D) $C_i \sqsubseteq D_j$ si $C \sqsubseteq D$ et $r_C \sqsubseteq r_D$

$A \sqsubseteq B$? (atleast 3 R_A) $\sqsubseteq$ (atleast 1 R_B) $R_A \sqsubseteq R_B$?

R_A = (restrict fille C_A) R_B = (restrict enfant C_B) fille $\sqsubseteq$ enfant

$C_A \sqsubseteq C_B$? C_A = (atleast 2 R'_A) C_B = (atleast 1 R'_B)

$R'_A \sqsubseteq R'_B$? R'_A = (restrict diplôme Informatique)
R'_B = diplôme = (restrict diplôme T)

C := (and Ensemble C_1
(all membre Musicien) C_2
(atmost 2 guitare) C_3
(atmost 3 tambour) C_4 avoir une guitare se représente par le rôle "guitare"

D := (and Ensemble D_1
(all membre Musicien-de-rock) D_2
(atmost 1 instrument-à-corde) D_3
(atmost 1 instrument-à-vent) D_4
(atmost 1 instrument-à-percussion))) D_5

$C \sqsubseteq D$?

C_i = (all r_C C') et D_j = (all r_D D') alors $C_i \sqsubseteq D_j$ si $C' \sqsubseteq D'$et $r_D \sqsubseteq r_C$

C_i = (atmost n r_C) D_j = (atmost m r_D) $C_i \sqsubseteq D_j$ si $n \leq m$ et $r_D \sqsubseteq r_C$

$D = D_1 \sqcap D_2 \sqcap D_3 \sqcap D_4 \sqcap D_5$

$C = C_1 \sqcap C_2 \sqcap C_3 \sqcap C_4$

$D_2 \sqsubseteq C_2$ (all membre Musicien-de-rock) $\sqsubseteq$ (all membre Musicien

$D_3 \sqsubseteq C_3$ (atmost 1 instrument-à-corde) $\sqsubseteq$ (atmost 2 guitare)
guitare $\sqsubseteq$ instrument-à-corde et 1 < 2
$D_5 \sqsubseteq C_4$ (atmost 1 instrument-à-percussion) $\sqsubseteq$ (atmost 3 tambour)
tambour $\sqsubseteq$ instrument-à-percussion et 1 < 3

⊏ ⊐ ⊑ ⊒ ⊔ ⊥ ⊤ ⊓ ⊢ ⊨ ⊭ ⊈ ⊉ ⊯

⊏ ⊐ ⊑ ⊒ ⊔ ⊥ ⊤ ⊓ ⊢ ⊨ ⊭ ⊈ ⊉ ⊯

¬ poulet (x) ∨ aliment (x) aliment (poulet) = (Δ, $\cdot^I$)
(IX) ¬ P(v, f(u, v)) ∨ ¬ P(f (u, v), f(u, v))
(x_1 …. x_k),
¬ (a ∧ b) ∨ c = ¬a ∨ ¬b∨ ¬a ∨ c] ¬ [(¬∨ (S(x)), ∧
∃x ∀y ∀z ((R (x, x) ∧ ¬ R(z, y)) ∨ (R(x, x) ∧ ¬ R(y, z)))