2. Logique classique - Notion de base

Logique des propositions et logique des prédicats

Logique des propositions $\in$Logique des prédicats
(logique des propositions = logique des prédicats
- variables, terme - quantificateurs)

1. Le vocabulaire de la logique des prédicats

- variables (X, Y, Z)
- constantes individuelles (a, b, c),
  fonctionnelles (f, g, h),
  prédicatives (p, q, r);
- les connecteurs (ou constantes logiques): T, $\bot$, $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\equiv$
- les quantificateurs universel ($\forall$) et existentiel ($\exists$).

2. La syntaxe

(règles pour formuler les phrases dans le langage)

- **termes**: variable, constante individuelle, forme fonctionnelle
- **forme fonctionnelle**, constante fonctionnelle + termes: $f(t_1, t_2, t_3)$.
- **forme prédicative**, constante prédicative + termes: $p(t_1, t_2, t_3)$.
- **atome** ou formule atomique: forme prédicative, une égalité entre les termes (par exemple, $t_1=t_2$).

formule (ou des formules bien formées) F:

	exemples
1. atome;	$p(t_1, t_2, t_3)$
2. T, $\bot$;	
3. $\neg A$, $A\wedge B$, $A\vee B$, $A\Rightarrow B$, $A\equiv B$	
4. $\forall xA$, $\exists xA$	$\exists x\ p(t_1, x, t_3)$

où A et B représentent une expression logique quelconque.

3. La sémantique

Une expression logique prend un sens quand on la met dans un contexte du monde réel. La sémantique d'une logique se base sur une interprétation qui permet de faire correspondre des expressions de base à la réalité.
Une interprétation est définie comme:

$I = (D, I_c, I_v)$ où:

- D est un ensemble non vide d'éléments (domaine);
- I_c: donne une signification aux constantes.

  $I_c(a)$: un élément dans D

  $I_c(f)$: $D^n \rightarrow D$,

  $I_c(P)$: $D^m \rightarrow \{0,1\}$.
- I_v: assigne un élément de D à une variable

  I_v: $V \rightarrow D$.

I peut être étendue sur toute formule (fonction de valuation V: $F \rightarrow \{0,1\}$):

- $I(\top) = 1$;
- $I(\bot) = 0$;
- Si X est une variable libre, alors $I(X)=I_v(X)$;
- Si a est une constante individuelle, alors $I(a)=I_c(a)$;
- $I(f(t_1,... t_n)) = I_c(f)(I(t_1), ..., I(t_n))$
- $I(p(t_1,... t_m)) = I_c(p)(I(t_1), ..., I(t_m))$
- Si t_1 et t_2 sont des termes,

  $I(t_1=t_2) = 1$ si $I(t_1)=I(t_2)$;

  $I(t_1=t_2) = 0$ autrement.
- $I(\neg A) = 1$ si $I(A) = 0$

  $I(\neg A) = 0$ si $I(A) = 1$;
- Pour les autres formes de formules composées:

A	B	$A \vee B$	$A \wedge B$	$A \Rightarrow B$	$A \equiv B$
0	0	0	0	1	1
0	1	1	0	1	0
1	0	1	0	0	0
1	1	1	1	1	1

- $I(\forall XA) = 1$ si $I_{a/X}(A) = 1$ pour tout élément a du domaine D

  $I(\forall XA) = 0$, autrement.

  où $_{a/X}$ signifie affecter la valeur a à X.
- $I(\exists XA) = 1$ si $I_{a/X}(A) = 1$ pour au moins un élément a du domaine D.

  $I(\exists XA) = 0$, autrement.

La valeur de vérité d'une expression dépend de

- l'interprétation (variable)
- propriétés du langage (fixe)

valeur de vérité $\leftrightarrow$ interprétation, propriétés du langage

2 approches pour étudier une logique:

théorie de modèle (approche sémantique)
système axiomatique (approche syntaxique)

4. Théorie de modèle

Modèle = l'interprétation qui rend une formule vraie
$M = (D, I_C, I_V)$

Étant donné un modèle M, la valeur de vérité d'une formule quelconque:

$\models_M A$

(A est vraie dans le modèle M)

En pratique: une réalité (des faits observés) $\rightarrow$ modèle

Si $\models_M A$, alors A est vérifiée dans la réalité.

Consistance: A est consistante s'il y a au moins un modèle

Validité: A est *valide* si elle est toujours vraie quelque soit l'interprétation. $\models A$.

Par exemple, on a: $\models (A \vee \neg A)$.

(la validité d'une formule ne dépend que des propriétés du langage)

Conséquence logique:

$E \models A$

toutes les interprétations qui rendent vraies toutes les formules de E rendent également vraie la formule A.

5. Système axiomatique

Le but est de caractériser un système logique par des propriétés syntaxiques.

Composition:

Axiomes: formules considérées comme valides
(propriétés du langage)

Règles d'inférence: $\dfrac{A_1, A_2, ..., A_n}{A}$

Exemple: MP: $\dfrac{A,\ A \Rightarrow B}{B}$

Un système axiomatique pour la logique des prédicats:

Axiomes:

A1 $(P \Rightarrow (Q \Rightarrow P))$

A2 $((P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \Rightarrow Q) \Rightarrow (P \Rightarrow R)))$

A3 $((\neg P \Rightarrow \neg Q) \Rightarrow ((\neg P \Rightarrow Q) \Rightarrow P))$

A4 $\forall X(P \Rightarrow Q) \Rightarrow (P \Rightarrow \forall XQ)$
où x ne figure pas dans P et elle est libre dans Q.

A5 $(\forall XQ \Rightarrow Q_{t/X})$
où $Q_{t/X}$ désigne la formule obtenue en remplaçant uniformément x par t dans Q et que ni x ni aucune variable de t n'est quantifiée dans Q.

Règles d'inférence

MP Si P et $P \Rightarrow Q$ sont des théorèmes, alors Q aussi.

$$\frac{P,\ P \Rightarrow Q}{Q}$$

G Si x n'est pas lié dans le théorème P, alors $\forall XP$ est un théorème.

$$\frac{P(X)}{\forall XP(X)}$$

Théorèmes: démontrables à partir des axiomes en appliquant les règles d'inférence.

démonstration: une liste ordonnée d'axiomes, de règles d'inférence et de théorèmes déjà connus qui ont permis d'obtenir ce théorème.

$\vdash_\Sigma A$: A est démontrable dans le système Σ.

$E \vdash_\Sigma A$: A est démontrable à partir de E dans le système Σ.

Système axiomatique correspond-t-il à la réalité (sémantique)
Adéquation: Si $\vdash A$ alors $\models A$
Complétude: Si $\models A$ alors $\vdash A$

Dans un système axiomatique adéquat et complet, on peut déduire toutes les formules valides et rien que cela.

6. Quelques équivalences

On montre ici quelques équivalences qu'on se servira plus tard.

$A \vee A \equiv A \wedge A) \equiv A$ (idempotence)
$(A \vee B) \vee C \equiv A \vee (B \vee C)$
$(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$ (associativité)
$(A \vee B) \wedge C \equiv (A \vee C) \wedge (B \vee C)$
$(A \wedge B) \vee C \equiv (A \wedge C) \vee (B \wedge C)$ (distributivité)
$\neg\neg A \equiv A$
$\neg(A \vee B) \equiv \neg A \wedge \neg B$
$\neg(A \wedge B) \equiv \neg A \vee \neg B$
$\neg(\exists X\ P) \equiv \forall X\ \neg P$
$\neg(\forall X\ P) \equiv \exists X\ \neg P$
$(A \equiv B) \equiv ((A \Rightarrow B) \wedge (B \Rightarrow A))$
$(A \Rightarrow B) \equiv (\neg A \vee B)$

7. Application de la logique dans l'IA

La logique a été utilisée dans l'IA à plusieurs endroits:

- utiliser des expressions logiques pour représenter des faits, des connaissances
- utiliser la logique pour inférer des conclusions

Dans une application en IA, face à un problème, le premier travail consiste à d'abord créer une représentation pour ce problème. Cette étape est appelée la *conceptualisation*. La logique a été d'abord utilisée pour représenter des faits qu'on observe dans une application. Par exemple, dans un monde de blocks, si on observe que le block *a* est posé sur le block *b*, alors on peut représenter ce fait par l'expression `sur(a, b)`. On peut également représenter des connaissances en logique. Par exemple, le fait que le block *a* est sur le block *b* implique que le block *b* n'est pas libre (c'est-à-dire qu'on ne peut pas poser un autre block sur *b*, ou qu'un bras mécanique ne peut pas prendre le block *b*). Cette connaissance peut être exprimée comme:

$\forall X \forall Y$ `sur(X,Y)` $\Rightarrow \neg$ `libre(Y).`

où `X,Y` sont des variable.

À ce stade, on peut mentionner la différence entre un fait (une donnée) et une connaissance. En règle générale, un fait ou une donnée réfère à une situation particulière observée. Elle ne peut pas être généralisée. En revanche, une connaissance réfère à une famille de situations similaires. On peut utiliser les deux exemples ci-dessus pour illustrer cette différence. Par `sur(a,b)`, on veut simplement dire que dans la situation particulière qu'on a observée, le block *a* est sur le block *b*. Cela n'est vrai que dans cette situation particulière. Par contre, l'implication

$\forall X \forall Y$ `sur(X,Y)` $\Rightarrow \neg$ `libre(Y)`

est vérifiée pour toutes X et Y. Elle couvre donc une famille de situations.

Note: La distinction entre donnée et connaissance n'est souvent pas aussi nette. Une connaissance peut parfois être traitée comme une donnée, selon le problème. Mais c'est cette *généralisation* qui fait la différence conceptuelle entre une connaissance et une donnée.

Le fait de pouvoir représenter un problème est intéressant seulement si on peut utiliser cette représentation pour effectuer des tâches intelligentes, comme le raisonnement. Ainsi, un autre rôle important de la logique est de développer un mécanisme permettant à une machine d'effectuer des raisonnements. Le système de modèles et le système axiomatique qu'on a décrits plus haut vise précisément à fournir un tel mécanisme. Ainsi, en utilisant la règle d'inférence Modus Ponens (MP), on peut inférer `¬libre(b)` à partir du fait et la connaissance décrits ci-dessus.
Convention:
1. Dans la suite de description, nous utilisons la même convention que le livre de Luger et Stubblefield: Un mot commençant par une majuscule représente une variable, et un mot commençant par une minuscule ou entre '…' représente une constante.
2. Il existe un ordre de priorité parmi les connecteurs logiques. Dans l'ordre décroissant, on a:

¬
∧ ∨
⇒ ≡
∀ ∃

Ainsi, l'expression `∀X∀Y sur(X,Y) ⇒ ¬ libre(Y)` implique le parenthésage suivant:

```
∀X∀Y (sur(X,Y) ⇒ (¬ libre(Y)))
```

8. Comment représenter des faits et des connaissances véhiculés en langue naturelle

La langue naturelle (l'anglais, le français, etc.) est un véhicule qu'on utilise pour communiquer, entre autres, des faits et des connaissances. Si on arrive à représenter ce que la langue naturelle véhicule en expression logique, il serait possible de modéliser la plupart de choses dans le monde réel. Donc, la question qu'on se pose est comment traduire une phrase de langue naturelle en une expression logique.
Cette tâche de traduction est très complexe. Elle pose parfois même des problèmes pour un être humain. Ainsi, on ne prétend pas ici de traduire n'importe quel type de phrase. Les phrases qu'on traite ici sont très simples.

Un objet est de certaine catégorie

Exemple: Jean est un homme.
Avant de traduire une phrase, il faut d'abord déterminer ceux qu'on utilise comme prédicat et comme arguments. Ensuite, il faut relier les prédicats selon la relation exprimée dans la phrase.
Pour cette phrase, la catégorie `homme` peut être utilisée comme un prédicat pour désigner qu'un objet est de la catégorie homme. La personne `Jean` (un objet) peut être considérée comme un argument (et le seul argument) de ce prédicat. Ainsi, la traduction de cette phrase est:

```
homme('Jean').
```

Un groupe nominal

Si une phrase contient le segment "un homme", il signifie qu'il existe un homme (tel que…). Ainsi, la traduction de ce segment est:

$\exists X$ homme(X)

Un fait

Une phrase comme "Un homme marche" représente un fait particulier observé. Sa signification est "il existe un homme tel qu'il marche". Un certain objet doit posséder 2 qualités: être un homme et marche. Ainsi, la traduction de cette phrase est la suivante:

$\exists X$ homme(X) $\wedge$ marche(X).

Une connaissance (une observation généralisée)

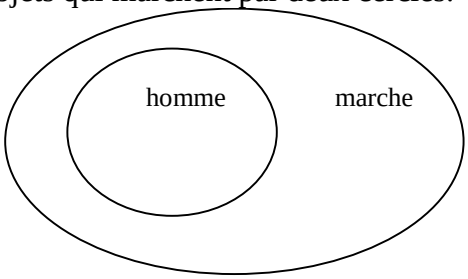
Par la phrase "Tout homme marche", on exprime une qualité générale de l'homme. Cette phrase peut être comprise comme "Si un objet est un homme, alors il marche". Les deux phrases ont exactement le même sens. Ainsi, sa traduction est:

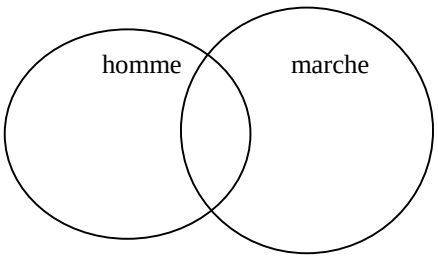
$\forall X$ homme(X) $\Rightarrow$ marche(X).

L'erreur souvent commise pour traduire ce type de phrase est de tenter de traduire "homme" et "marche" comme 2 qualités parallèles à un objet comme suit:

$\forall X$ homme(X) $\wedge$ marche(X).

Cette traduction correspond à la phrase "Tout objet est un homme et qu'il marche". C'est évidemment pas le même sens.
Pour mieux voir la différence, on peut représenter l'ensemble des hommes et l'ensemble d'objets qui marchent par deux cercles:





La première figure correspond à "tout homme marche": L'ensemble des hommes est compris dans l'ensemble des objets qui marchent. La second figure correspond à la fausse traduction: Cette fausse traduction désigne simplement l'ensemble des objets dans l'intersection des deux cercles. Elle n'exprime pas la relation d'inclusion entre les deux ensembles.

La phrase "Tout homme marche" peut être aussi exprimée comme "Tous les hommes marchent", "Chaque homme marche", etc.

Voyons un exemple un peu plus compliqué: "Tout homme qui est jeune marche". Ici, l'objet désigné par le sujet (tout homme qui marche) est qualifié par deux prédicats: homme et jeune. Pour un tel objet, on observe qu'il marche. Ainsi, sa traduction est:

$\forall X$ homme(X) $\wedge$ jeune(X) $\Rightarrow$ marche(X).

L'erreur fréquente pour cette phrase est de la traduire comme suit:

$\forall X$ (homme(X) $\Rightarrow$ jeune(X)) $\Rightarrow$ marche(X).

Cette phrase correspond à "Si tous les hommes sont jeunes, alors ils marchent". Mais qu'arrive-t-il s'il y a des hommes jeunes, et d'autres vieux?

Les phrases qu'on vient de voir peuvent aussi être exprimées en "si … alors …". Par exemple: "Tout homme est mortel" est équivalente à "si un objet est un homme, alors il est mortel". La traduction des phrases en "si … alors …" est directe.
Donnons quelques autres exemples:
Chacun a un père: $\forall X \exists Y$ père(Y,X).
Note: Ici, on choisit à définir père comme un prédicat à deux arguments. On peut aussi traduire cette même phrase par: $\forall X$ a_père(X). Mais cette seconde traduction est moins raffinée. Il sera plus difficile de raisonner en se basant sur cette seconde traduction.
Chaque père a une femme: $\forall Y(\exists X$ père(Y,X)) $\Rightarrow$ ($\exists Z$ femme(Z,Y)).
Remarquer la différence de qualification sur X et Y dans père en comparaison avec la phrase précédente. Cela est due au fait qu'ici on s'intéresse à tous les pères, tandis que précédemment, on s'intéressait à tous les enfants.

Négation

Ex: Il n'y a pas d'homme qui marche = Aucun homme ne marche.

$\neg(\exists X$ homme(X) $\wedge$ marche(X)).

$\forall X$ homme(X) $\Rightarrow \neg$marche(X).

Ces deux traductions sont équivalentes.

Ex: Tous les hommes ne marchent pas.
Cette phrase est ambigüe. Elle peut signifier "Aucun homme ne marche", ou bien "certains hommes ne marchent pas". Ainsi, il y a deux traductions correspondant à ces deux sens. La première traduction est identique que l'exemple précédent. La seconde traduction est comme suit:

$\exists X$ homme(X) $\wedge \neg$marche(X).

Il y a beaucoup d'expressions en langue naturelle qu'on ne peut pas traduire adéquatement en expression logique. Cela illustre à la fois la complexité de la langue naturelle est le pouvoir expressif limité de la logique. Par exemple, la phrase "cet homme marche" désigne une personne en particulier. Cela nécessite une façon de déterminer un objet (homme) particulier dans le domaine d'interprétation. Cela s'avère souvent difficile (bien qu'il existe des méthodes pour le faire).

9. Unification

L'unification est un algorithme pour déterminer les substitutions qui permettent de rendre deux expressions identiques.
Une substitution est le remplacement d'une variable par une autre expression. On le représente comme Exp/Var. Par exemple, b/X signifie le remplacement de la variable X par la constante b.

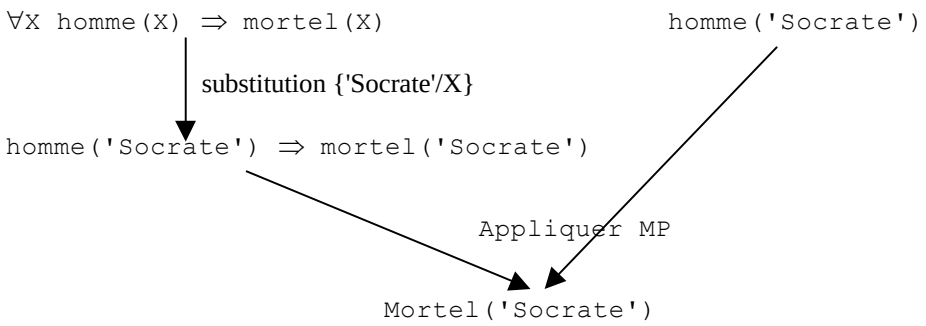
L'unification est une étape nécessaire pour faire de l'inférence. Par exemple, dans l'exemple classique de Socrate, on a les deux expressions suivantes:

∀X homme(X) ⇒ mortel(X). (Tout homme est mortel)
homme('Socrate'). (Socrate est un homme)

On ne peut pas utiliser directement la règle d'inférence Modus Ponens (MP) qui est comme suit:

$$\frac{A,\ A \Rightarrow B}{B}$$

parce que la règle demande deux expressions A qui sont IDENTIQUES. Bien que pour un être humain, il soit facile d'associer homme(X) avec homme('Socrate'), c'est n'est pas une association directe pour une machine, car elles sont syntaxiquement différentes. Pour qu'elle deviennent identiques, il faut qu'on remplace la variable X dans homme(X) par la constante 'Socrate'. C'est cette substitution que l'unification essaie de trouver. Ainsi, l'inférence de "Socrate est mortel" se fait comme dans la figure suivante:



Comment trouver les substitutions nécessaires pour unifier deux expressions? Si on les cherche directement à partir des deux expressions qui peuvent être complexes, la tâche est difficile. Pour la simplifier, on transforme les deux expressions en une forme standard, appelée *forme clausale.*

10. Forme clausale

Dans une forme clausale, il n'y a que les connecteurs ¬ et ∨. Le connecteur ∧ est implicitement mis entre deux clauses. Il n'y a plus de quantificateur. Toute variable est implicitement quantifié par ∀. Les variables indépendantes sont nommées différemment.
La transformation d'une expression quelconque en forme clausale se fait en plusieurs étapes:

1. transformer ⇒ et ≡ en utilisant les équivalences suivantes:
   $A \Rightarrow B \equiv (\neg A \vee B)$
   $(A \equiv B) \equiv ((\neg A \vee B) \wedge (\neg B \vee A))$
2. ramener la négation directement sur les prédicats, en utilisant les équivalences.
3. Si deux variables sont quantifiées par deux quantificateurs différents, donnez-leur des noms différents.

4. Suppression de $\exists$:
   Si une variable quantifiée par $\exists$ est à l'extérieur de tout quantificateur $\forall$, alors remplacer la variable par une constante. Deux variables différentes sont remplacées par des constantes différentes.
   Si une variable quantifiée par $\exists$ est à l'intérieur d'un quantificateur $\forall X$, alors remplacer la variable par f(X).
   Cette fonction f(X) est appelée une fonction Skolem. Elle permet de refléter la dépendance éventuelle entre la variable et X.
5. Enlever tout $\forall X$.
6. Transformer l'expression en Forme Normale Conjonctive, par ex. $(A \vee \neg B) \wedge (C \vee D \vee E)$
7. Séparer les éléments reliés par $\wedge$ en différentes clauses. Assurer que des variables dans des clauses différentes ont des noms différents.

Exemple:

$\exists Z \forall X\ (\forall Y\ p(X,Y)) \Rightarrow \neg(\forall Y\ q(X,Y) \Rightarrow r(X,Y,Z))$

1. $\exists Z \forall X\ \neg(\forall Y\ p(X,Y)) \vee \neg(\forall Y\ \neg q(X,Y) \vee r(X,Y,Z))$
2. $\exists Z \forall X\ (\exists Y\ \neg p(X,Y)) \vee (\exists Y\ q(X,Y) \wedge \neg r(X,Y,Z))$
3. $\exists Z \forall X\ (\exists Y1\ \neg p(X,Y1)) \vee (\exists Y2\ q(X,Y2) \wedge \neg r(X,Y2,Z))$
4. $\forall X\ \neg p(X,f1(X)) \vee (q(X,f2(X)) \wedge \neg r(X,f2(X),a))$
5. $\neg p(X,f1(X)) \vee (q(X,f2(X)) \wedge \neg r(X,f2(X),a))$
6. $(\neg p(X,f1(X)) \vee q(X,f2(X))) \wedge (\neg p(X,f1(X)) \vee \neg r(X,f2(X),a))$
7. $\{\neg p(X1,f1(X1)) \vee q(X1,f2(X1)),$
   $\neg p(X2,f1(X2)) \vee \neg r(X2,f2(X2),a)\}$

Dans le résultat, on a deux clauses.

11. Trouver la substitution la plus générale

Une substitution est un ensemble d'association de forme Exp/Var. Les deux contraintes suivantes doivent être vérifiées:

1. Une variable ne peut pas être associée à deux expressions;
2. Une variable substituée ne peut pas être incluse dans une expression substituante.

Ainsi, {a/X, f(b)/Y, W/Z} est une substitution légale. Tandis que {g(Y)/X, f(X)/Y} et {a/X, f(b)/X, W/Z} ne le sont pas.

L'algorithme pour trouver la substitution la plus générale est décrite dans Luger & Stubblefield, pp. 71. Ci-dessus, cet algorithme a été complété et légèrement modifié pour qu'il soit plus facile à implanter en Prolog.

```
function unify(E1, E2);
begin
        case
                both E1 and E2 are constant or empty list:
                        if E1 = E2 then return {}
```

```
            else return FAIL;
        E1 is a variable:
            if E1 occurs in E2 then return FAIL
            else return {E2/E1};
        E2 is a variable:
            if E2 occurs in E1 then return FAIL
            else return {E1/E2};
        E1 or E2 is a constant or empty list:
            return FAIL;
        otherwise:
            begin
            if E1 and E2 are lists then LE1 := E1, LE2 := E2
            else LE1 := list form of E1, LE2 := list form of E2
            HE1 := first element of LE1;
            HE2 := first element of LE2;
            SUBS1 := unify(HE1, HE2);
            if SUBS1 = FAIL then return FAIL;
            TE1 := apply(SUBS1, rest of LE1);
            TE2 := apply(SUBS1, rest of LE2);
            SUBS2 := unify(TE1, TE2);
            if SUBS2 = FAIL then return FAIL;
            else return composition(SUBS1, SUBS2);
            end;
        end
end
```

Dans cet algorithme, on utilise deux autres fonctions: apply et composition. La première fonction applique une substitution sur une expression. L'effet de cette application est de remplacer toutes les occurrences d'une variable par l'expression qui la substitue. Par exemple:

apply({a/X, f(b)/Y, W/Z}, p(X, X, Y, V)) donne p(a, a, f(b), V).

On peut aussi noter cette application par p(X, X, Y, V) {a/X, f(b)/Y, W/Z}.

La second fonction - composition - permet de regrouper deux substitutions en une seule. Soit S1 et S2 deux substitutions, S1 • S2 (ou composition(S1, S2) de l'algorithme) est obtenu comme suit:

1. La substitution S2 est appliquée à S1;
2. S2 est ajoutée dans le résultat.

L'application de la première étape consiste à remplacer les variables des parties substituantes de S1. Par exemple, l'application de {a/X, b/Y} sur {f(X)/W, g(Y,X)/V} donne {f(a)/W, g(b,a)/V}.

Ainsi, {f(X)/W, g(Y,X)/V}•{a/X, b/Y} = {f(a)/W, g(b,a)/V, a/X, b/Y}.

12. Inférence - Méthode de résolution

Une fois les expressions unifiées, on peut appliquer des règles d'inférence pour obtenir de nouvelles conclusions. En générale, l'inférence se fait de la façon suivante:

Étant donné un ensemble d'énoncés E, si en appliquant une règle d'inférence, on peut obtenir A, alors on ajoute A dans E pour obtenir E1. On continue l'inférence à partir de E1.
Ce processus d'inférence continue jusqu'à l'obtention de la conclusion dans un ensemble d'énoncés. Dans ce cas, on peut dire que la conclusion est une conséquence des énoncés E. Si après l'application de toutes les règles d'inférence, on arrive à un point En où aucune nouvelle expression ne peut s'ajouter dans En, et que la conclusion voulue n'est pas incluse dans E, on dira que la conclusion n'est pas une conséquence de E. Cependant, pour pouvoir dire cela, il faut que tous les chemins d'inférence soit exploré.

Une méthode d'inférence souvent utilisée (par exemple, dans Prolog) est la méthode de résolution. Le principe est le suivant:
Principe de résolution: Pour prouver la clause Q à partir d'un ensemble de clauses E, il suffit de prouver que E et $\neg Q$ sont incompatibles, c'est-à-dire qu'on peut inférer $\perp$ à partir de E et $\neg Q$.

Règles d'inférence: Les règles d'inférence utilisées sont les suivantes:

$$\frac{A, \neg A}{\perp} \qquad \frac{A \vee B, \neg A \vee C}{B \vee C}$$

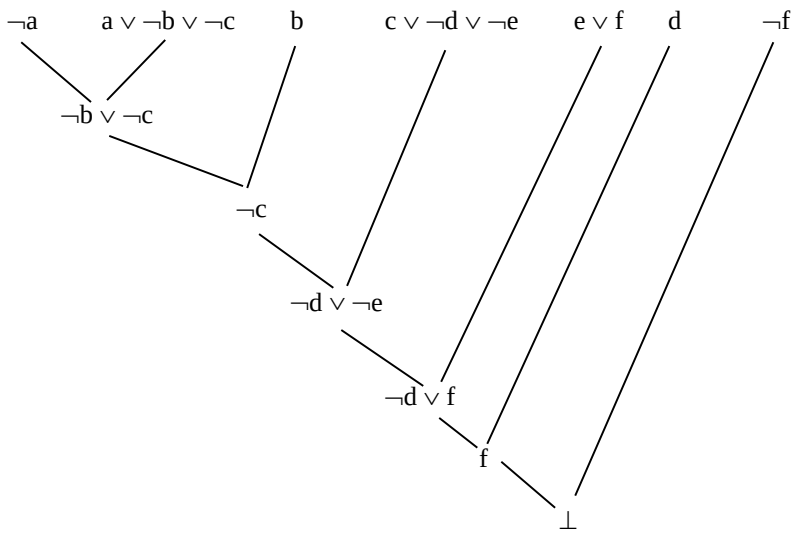
Par exemple, soit les expressions suivantes:

$b \wedge c \Rightarrow a$
b
$d \wedge e \Rightarrow c$
$e \vee f$
$d \wedge \neg f$

On voudrait prouver a. Pour cela, on transforme d'abord les expressions en forme clausales:

$a \vee \neg b \vee \neg c$
b
$c \vee \neg d \vee \neg e$
$e \vee f$
d
$\neg f$

Pour prouver a, il suffit de montrer qu'en ajoutant $\neg a$ à ces clauses, on peut arriver à $\perp$.



Évidemment, il peut exister plusieurs façons de conduire l'inférence. Par exemple, on pourrait très bien choisir à conclure d'abord en a ∨ ¬c à partir de a ∨ ¬b ∨ ¬c et b. Le choix du chemin ne change que l'efficacité de l'inférence, mais pas le résultat.

Dans cet exemple, aucune variable n'est impliquée. Cela est pour le but de simplifier l'exemple pour mieux montrer le principe de résolution. S'il y a des variables, il faudrait faire de l'unification avant d'appliquer la règle de résolution.