Logique pour l'Intelligence Artificielle

Démontration automatique et Fondements théoriques du langage PROLOG

Bernard ESPINASSE

Professeur à l'Université d'Aix-Marseille

2008

- **Transformation en clauses**
- **Algorithme d'Unification**
- **Principe de Résolution**
- **Introduction au langage prolog**

Démonstration automatique

- **Logique des prédicats du 1°ordre (LP1) indécidable**

→ *aucune procédure de décidabilité*

- **toutefois :**
  - il existe des algorithmes de **semi-décidabilité**
  - possible de **restreindre le langage LP1** pour avoir une **décidabilité**

→ *transformation d'une formule de LP1 en **forme clausale** (ensemble de clauses)*

Transformation d'une formule de LP1

- **objet : suppression des quantificateurs par la mise sous forme clausale**
- **processus en étapes :**

1- mise sous forme prenexe (∀ et ∃ en tête)

→ toute formule f de LP1 est équivalente à une formule fp sous forme prenexe

2- skolémisation (élimination des ∃)

→ la skolémisation n'assure pas l'équivalence, mais :

***Théorème de SKOLEM** : Une formule est inconsistance SSI sa forme skolémisée l'est*

3- mise sous forme clausale

→ élimination des ∀

→ mise sous forme normale conjonctive

→ mise sous forme de clauses

Mise sous forme prenex

- **forme prenexe : f: $(Q_1x_1)...(Q_nX_n)M$**

  - **Q_i: (préfixe) un quantificateur ∀, ∃ (en tête) et x_i : variable prédicative**
  - **M: (matrice) une formule sans quantificateur**

1) éliminer ⊃ et <-> sachant :

A<->B *remplacé par* (A⊃ B)∧(B⊃ A) et A⊃B *remplacé par* ¬AvB

2) accoler les ¬ aux atomes concernés sachant :

¬¬A	*équivalent à*	A
¬(AvB)	//	(¬A)∧(¬B)
¬(A∧B)	//	(¬A)v(¬B)
¬(∃x(P(x))	//	∀x(¬P(x))
¬(∀x(P(x))	//	∃x(¬P(x))

3) renommer certaines variables liées pour ne plus avoir de variables quantifiée 2 fois, sachant :

∀xP(x) *équivalent à* ∀yP(y) et ∃xP(x) *équivalent à* ∃yP(y)

4) déplacer tous les quantificateurs à gauche

Mise sous forme de skolem (élimination des ∃)

- **mettre sous forme prenex**
- **remplacer chaque** ∃ par une nouvelle fonction distincte des variables quantifiées universellement qui précèdent ce ∃

Ex:

∀x ∃y ∀z P(x,y,z) équ. à ∀x∀z P(x,f(x),z)

∀x ∃y ∃z R(x,y,z) équ.à ∀xR(x,f(x),g(x))

∃x ∀y ∀z ∃t (Q(x,y)∧(P(y,t)) équ.à ∀y ∀z (Q(a,y)∧(P(y,f(y,z)) avec a = constante

Mise sous forme normale conjonctive (FNC)

- **littéral**: un atome ou la négation d'un atome (prédicat) P(x,y), ¬Q(z),...
- **FNC = formule de la forme:** $A_1 \wedge A_2 \wedge ... \wedge A_n$ où chaque A_i = disjonction de littéraux :

f: (¬P(x)vQ(a,x))∧P(b)∧(¬Q(y,b)vP(a))

$\{A_1, A_1, ... A_n\}$ = ens. des clauses de la formule f

- **utiliser l'associativité et la distributivité (∧/∨)**

Clauses

- **Littéral =** un atome ou la négation d'un atome (prédicat) P(x,y), ¬Q(z),...
- **Clause = disjonction (ou ∨) finie (0 ou plus) de littéraux :**

Ex : C: (¬P(x,f(x)) v R(x,f(x),g(x))

- une clause = {ens.de littéraux}
- si {}=ø on a la **clause vide**, une **clause vide** ne peut jamais être satisfaite
  → **toujours fausse**
- un {clauses} = une conjonction de clauses gouvernées par des quantificateurs universels
- **une formule est un {clauses}**

- **Mise sous forme de clauses**
  1) mettre sous forme de skolem
  2) suppression des ∀
  3) passage en FNC
  4) séparation en clauses
  5) limitation de la portée de chaque variable à une seule clause

Ex: transformer en clause la formule suivante: (∀x P(x)⊃ ¬(∃y (Q(b,y)⊃P(y))))

Algorithme d'unification (J.Pitrat et J.A.Robinson -1966)

- **Indépendant** du **système formel** considéré **(LP0, LP1)**
- Permet d'**appliquer** un **théorème T** (T: H → C) à une **formule f** en rendant **H** et **f identiques** par une suite de **substitutions** de variables libres dans H et f **= unifier H et f**

Principe:

- mise en **coincidence systèmatique** de **H** et **f** par **substitutions en parcourant en parallèle** les **arbres** associés à **f** et **T :**

→ si l'on parvient à la fin de l'hypothèse **H** => ***unification réussie***

par modus-ponens *:*

→ la conclusion **C** de **T**, **modifiée par les substitutions** effectuées donne la **nouvelle forme de f** après l'application de **T**

- plus il y a de **variables** => plus l'algorithme est **complexe**

L'algorithme d'unification

Remarques

1 - représenter **f** et **T** par des **arbres** : nœuds = symboles de l'expression (opérateur (non substituables) ou variables substituables)

2 - les **substitutions** concernent le remplacement d'une variable **x** de **f** ou **H** par un terme **t** de **f** ou **H**

3 - seules les **variables libres peuvent être substituées** (celles non quantifiées de façon universelle ou existentielle)

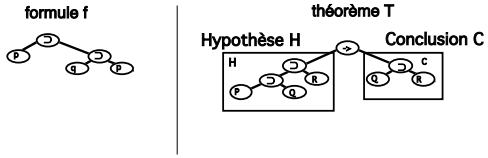
4 - il convient de **renommer** les variables de **f** et **T** avant d'appliquer l'algorithme

5 - **plusieurs ordres de parcours possibles** (ex: parcours dans un arbre) ex: profondeur d'abord

Algorithme d'unification : logique des propositions

• Soit théorème **T: ((P⊃Q)⊃R) → (Q⊃R)** et formule **f: ((p⊃(q⊃p))**

arbres associés :



• **Parcours de f et de H de T en parallèle :**

- les 2 premiers symboles rencontrés dans **f** et **H** = même opérateur ⊃
- on progresse dans les 2 arbres :
  - dans **f** on rencontre la variable **p**,
  - dans **H** on rencontre l'opérateur ⊃ qui annonce un nouveau terme **t**

-> il faut changer **p** en **t** où **t** = sous-arbre => **1° substitution p en t** :

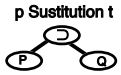
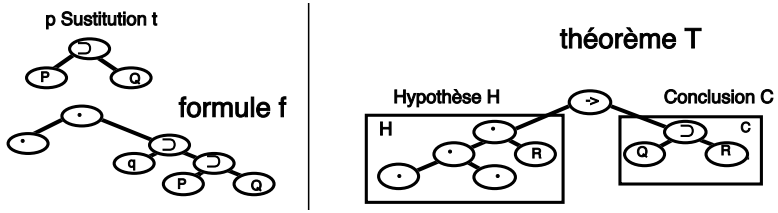


Illustration de l'algorithme d'unification



Les symbôles déjà considérés dans le parcours sont pointés dans la figure (•).

On continue le parcours dans les 2 arbres :

- dans **f** on rencontre un opérateur ⊃ annonçant un terme **t**
- dans **H** la variable **R**

-> on effectue une nouvelle **substitution R en t** :

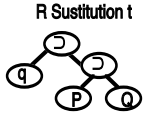
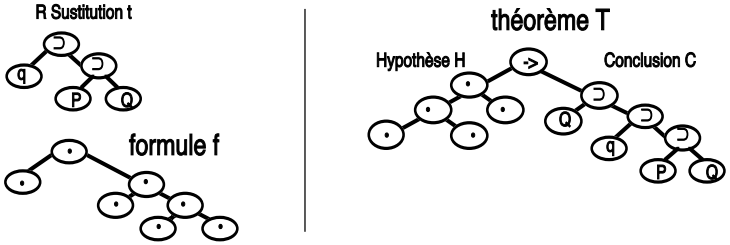


Illustration de l'algorithme d'unification



• Parcours de hypothèse **H** complet, l'**unification terminée** et **réussie**, par 2 substitutions :

- **p/ P ⊃ Q**
- **R/ q ⊃ (P ⊃ Q)**

• **f** est **changé** en (nouvelle conclusion de T) **f1: (Q ⊃ (q ⊃ (q ⊃ Q)))**

• en appliquant **T** à **f** on est parvenu au résultat (après normalisation):

(p ⊃ q) ⊃ (r ⊃ (p ⊃ q)) → (q ⊃ (r⊃ (p ⊃ q)))

• membre gauche du résultat = *théorème le plus général* que l'on peut obtenir en appliquant **T** à **f**

Principe de Résolution

• S'inspire d'une méthode de démonstration de théorème en LP1 (J.Herbrand 1930) :

- soit un théorème **T: $H_1 \wedge H_2 \wedge H_3 \wedge \ldots H_n \rightarrow C$** (où H_i = hypothèses)
- pour démontrer théorème T, il peut être plus facile de démontrer que formule :

f : $H_1 \wedge H_2 \wedge H_3 \wedge \ldots H_n \wedge (\neg C)$ est contradictoire

- montrer que f est contradictoire <=> montrer que **f = non-théorème en contenant une expression de la forme P∧¬P** (contre exemple pour f)

• pour cela Herbrand défini (**théorème d'Herbrand**) un processus de démonstration :

- consistant à faire dans f **un nombre fini de substitutions** lorsque T est initialement un théorème (sinon infini)
- s'appliquant **uniquement à des formules mises sous forme clausale**

Soit F un ensemble de formules et Fc l'ensemble des clauses obtenues à partir de F:

Fc est inconsistant <=> F est inconsistant

démonstration de théorèmes par **réfutation** :

$H_1 \wedge H_2 \wedge \ldots H_n$ –> C Théorème <=> $H_1 \wedge H_2 \wedge \ldots H_n \wedge (\neg C)$ inconsistant

Théorème de Herbrand

une **condition nécessaire** pour que E = {clauses} soit inconsistant est qu'il existe un **sous-ensemble fini contradictoire** de formules de E **complètement instanciées**

Théorème de résolution

soit les formules sous forme conjonctive : **f1:**

$C_1 \wedge C_2 \wedge \ldots (pvLi) \ldots (\neg pvLj) \ldots \wedge C_m$ (Ci = clause) et **f2 : f1∧(Li v Lj)** (Li = littéral) :

Si f2 est contradictoire => f1 est également contradictoire

- Ci=(p v Li) et Cj=(¬p v Lj) = *clauses parentes*
- (Li v Lj) = *clause résolvante*

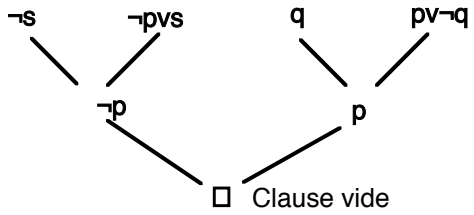
Principe de résolution

• la **clause vide** □(aucun littéral) ne peut jamais être satisfaite : elle est toujours fausse = **contradiction**	□ □
• si F comprend 2 clauses p et ¬p, F est **inconsistant** (on dérive la clause vide □)	p ¬p → □
• P,¬PvQ se résolvent en Q (**modus ponens**: P, P⊃Q I- Q)	P ¬PvQ → Q
• ¬Q,¬PvQ se résolvent en ¬P (**modus tollens**: ¬Q, P⊃Q I- ¬P)	¬Q ¬PvQ → ¬P
• ¬PvQ, ¬QvR se résolvent en ¬PvR (**règle d'enchainement**: P⊃Q, Q⊃R I- P⊃R)	¬PvQ ¬QvR → ¬PvR
• - P, ¬P se résolvent en □ (□ = clause vide)	P ¬P → □

Exemple de résolution (logique des propositions)

Soit l'expression en forme conjonctive **f1 : ¬s∧q∧(p∨¬q) ∧(¬p∨s)** ; on a les 4 clauses :

C1: ¬s, C2: q, C3: (p ou ¬q), C4: (¬p ou s)



- 1° résolution entre clauses **C1** et **C4** -> clause résolvante **C5: ¬p**
- 2° résolution entre **C2** et **C3** -> clause résolvante **C6: p**
- clause **C6 unifiée** avec clause **C5** -> **résolvante vide**

d'où :

- formule f1 de départ est **contradictoire**
- on a **établit** le **théorème** **T : q∧(p∨¬q) ∧(¬p∨s) → s**

Principe de résolution appliqué aux clauses concrêtes (ground-résolution)

- clause **concrête** : tous les littéraux ne contiennent aucune variable (ex: ¬P(a)vQ(a,f(b));...)

Soit 2 clauses concrêtes :
C1: G1v...Gn
C2: ¬G1vH2vH3v...vHm

G1 dans C1 et **¬G1** dans C2 sont dit **littéraux complémentaires**
- en appliquant le principe de résolution à C1 et C2 (clauses parentes) on obtient la clause C3 (résolvante de C1 et C2): **C3: G2v...GnvH2v...Hm**

Ex :

C1: PvR	C1: ¬PvQvR
C2: ¬PvQ	C2: ¬QvS
C3: RvQ	C3: ¬PvRvS

- si une clause C est atteinte à partir d'un ensemble S de clauses en un nombre fini de pas, alors on a déduit C de S : **C est une conséquence logique**

- si l'on a obtenu la clause vide □, alors on a fait une **réfutation** ou donné une preuve de S : **S est un théorème**

Principe de résolution appliqué à des clauses avec variables (unification)

• soit 2 clauses avec variables : **C1: P(x)vQ(x)** et **C2:¬P(f(x))vR(x)**

→ il n'y a plus de paires complémentaires à cause des variables, pour en obtenir il faut faire des subtitutions:

f(a) *substitué à* x dans C1
a *substitué à* x dans C2

• on obtient les **instances de C1 et C2:**

C1': P(f(a))vQ(f(a))
C2':¬P(f(a))vR(a)
C3': Q(f(a))vR(a) résolvante de C1' et C2'

• de façon générale **si l'on avait substitué f(x) à x dans C1 on aurait obtenu :**

C1': P(f(x))vQ(f(x))
C2':¬P(f(x))vR(x)
C3': Q(f(x))vR(x) C3' = résolvante la plus générale de C1' et C2'

Ainsi, l'opération **d'unification** permet, **par substitution,** de **transformer** des clauses en vue de faire apparaître des littéraux complémentaires

Exemple de résolution dans LP1

Soit :

x, y, z, t, et v des variables,
a, b, c des constantes,
P, Q, R des prédicats,

• **la forme conjonctive f1: P(x,y)∧Q(t)∧R(v)∧(¬P(a,z)v¬Q(b)v ¬Q(c))**
• **l'unification fournit trois substitutions et trois résolvantes possibles:**

S: {x/a, y/z} donne f2 = ¬Q(b)v¬Q(c),
S': {t/b} donne f'2 = ¬P(a, z)v¬Q(c)
S'': {t/c} donne f''2 = ¬P(a, z)v¬Q(b)

• **la démonstration s'achève :**
  • soit lorsque Li et Lj sont trouvés tous les deux vides,
  • soit lorsqu'une contradiction interne sera exibée par une résolvante, c'est le cas pour f2 et la seconde clause de f1:

S''': {t/b} donne f3 =¬Q(c) qui avec la substitution
S'''': {t/c} donne résolvante f4= □ (clause nulle)

ainsi f1 était contradictoire

Intérêts et limites de la résolution

• **Intérêts:**
  • les **hypothèses**, la **conclusion** et les **clauses résolutoires intermédiaires** jouent le **même rôle**
  • les démonstrations sont ainsi **obtenues en avançant dans les deux directions** à la fois:

hypothèses → conclusion et **conclusion → hypothèses**

  • la **preuve** est faite **lorsque 2 trajets parcourus se rencontrent** :

Limites :

• **Limite due à l'explosion combinatoire: comment choisir convenablement :**
  • **clauses et littéraux à unifier** pour atteindre le plus rapidement la clause vide
• **Limitée à la démonstration de théorème:**
  • elle ne peut pas inventer des hypothèses et si l'hypothèse envisagée n'est pas un théorème, elle développe alors un arbre infini

PROLOG : la programmation logique

• **Développé par A.COLMERAUER en 1971** (compréhension et traduction automatique langues naturelles)

• Bati sur **LP1** et sur la **méthode d'Herbrand-Robinson,** ainsi :

  • dans un programme prolog, l'**unité élémentaire** n'est plus l'instruction **mais un énoncé logique**

  • peut être considéré comme un **interpréteur** de **formules de LP1**

  • il permet de **poser un probléme au système** : Prolog langage de programmation déclaratif

  • il permet **résoudre le problème** en fournissant une **réponse** si elle existe (par unification et résolutions)

Clauses de Horn et conventions Prolog

• **Clauses en général** :

C : $(\neg q1 \vee \neg q2 \vee ... \vee \neg qn) \vee (p1 \vee p2 \vee ... \vee pk)$ soit $(P \vee Q <=> \neg P \supset Q)$:

$C : (q1 \wedge q2 \wedge ... \wedge qn) \supset (p1 \wedge p2 \wedge ... \wedge pk)$

q_i = littéraux négatifs; p_j = littéraux positifs

• **Clauses de Horn** : permettent de limiter effort d'unification : **1 seul littéral positif** :

C: $p \vee (\neg q_1 \vee \neg q_2 \vee ... \vee \neg q_n)$ soit : **C: $(q_1 \wedge q_2 \wedge ... \wedge q_n) \supset p$**

• **Conventions Prolog :**

- les signes $\supset$ et $\neg$ sont **omis** à l'intérieur des clauses
- on distingue la **tête** et la **queue** de la règle séparées par le symbole **->**
- clause **implicative** : **C: $(q_1 \wedge q_2 \wedge ... \wedge q_n) \supset p$,** en Prolog : **p -> q_1 q_2 ... q_n ;**
- clause **but** : **C: $(q_1 \wedge q_2 \wedge ... \wedge q_n) \supset$** (sans littéral positif), en Prolog : **q_1 q_2 ... q_n;**
- clause **assertions** : **p** (1 littéral positif et sans littéral négatif), en Prolog : **p ->;**

Exemple 1: Le syllogisme de Socrate

formule (1) *tout homme est mortel,*
formule (2) *Socrate est un homme,*
formule (3) *alors Socrate est mortel*

• formule (1) = thèorème traduite en **clause de Horn implicative** :

C1: homme(x) $\supset$ mortel(x) soit :

C1: mortel(x) $\wedge$ $\neg$homme(x)

traduite en Prolog par : **mortel(x) -> homme(x);**

• formule (2) = affirmation traduite en la **clause de Horn assertionnelle** :

C2: homme(Socrate)

traduite en prolog par : **homme(Socrate) ->;**

Remarque :

- –> doit être interprétée comme un **SI** plutôt que d'un implique
- l'absence de terme à droite de –> signifie que ce qui est vérifié à gauche est un fait vérifié en toutes circonstances

Exemple 2: Le repas léger

• on considère qu'un repas léger est composé d'**un plat suivi d'un désert**. On a ainsi la **clause de Horn implicative** :

C1: plat(x) $\wedge$ desert(y) $\supset$ repas(x,y) soit :

C1: $\neg$plat(x) $\vee$ $\neg$desert(y) $\vee$ repas(x,y)

traduite en Prolog par :

repas(x,y) -> plat(x) desert(y);

• on dispose aussi de **4 faits** : **clauses de Horn assertionnelles** :

C2:plat(poisson),
C3:plat(viande),
C4:dessert(glace),
C5:dessert(fruit),

traduites en prolog ainsi :

plat(poisson) -> ;
plat(viande) -> ;
dessert(glace) -> ;
dessert(fruit) -> ;

Interprétation des connaissances dans Prolog

• Prolog = démonstration de théorème suivant le principe de résolution

• un programme en Prolog = suite de clauses ordonnées

• Prolog **choisit 2 clauses et essaie de les unifier** (par substitutions) :

- les clauses sont balayées **selon leur ordre d'arrivée** et
- dans chaque clause, **les littéraux sont balayés de droite à gauche**.

• le but est de démontrer qu'**une conjonction de littéraux p_i est le conséquent d'un ensemble de clauses de Horn C**, soit :

$$C \text{ I- } \exists x_1, x_2 ... x_n \ (p_1 \wedge p_2 \wedge ... \wedge p_k)$$

ceci revient à démontrer que la formule :

$C \wedge \forall x_1, x_2 ... x_n \ (\neg p_1 \vee \neg p_2 \vee ... \vee \neg p_k)$ est contradictoire.

• si l'inconsistance de cet ensemble de clauses est démontrée, le but $p_1 \wedge p_2 \wedge ... \wedge p_k$ se déduit de C pour certaines substitutions des variables $x_1, x_2 ... x_n$ reportées par Prolog

Interprétation des connaissances dans Prolog

Démonstrateur Prolog (chaînage arrière) :

Etape 1: il part d'une **pile de buts égale aux buts initiaux** $p_1 \wedge p_2 \wedge ... \wedge p_k$

Etape 2: il cherche à **résoudre le littéral du sommet de la pile de buts :**

- il prend la 1°clause de C non encore utilisée : $q_1 \vee \neg q_2 \vee ... \vee \neg q_m$ (notée $q_1 \rightarrow q_2 ... q_m$) telle que p_1 et q_1, soient **unifiables**
- il tente **d'effacer** la nouvelle pile de buts : $\neg q_2 \vee ... \vee \neg q_m \vee \neg p_2 \vee ... \vee \neg p_k$ où l'on effectue la **substitution** qui a permis l'unification des 2 littéraux
- si plus de clause de C ne convient pour cet effacement, il **revient en arrière** (backtracking) **jusqu'à trouver une résolution** pour laquelle il existe une clause de C qui convienne et qui n'a pas été tentée

Etape 3: si plus de littéral à résoudre => on a obtenu la **clause vide et :**

- **il reporte la substitution** des variables du but ayant permis cette résolution
- il **revient en arrière** jusqu'à trouver une résolution avec une clause de C qui n'a pas été tentée

Etape 4: il retourne à l'étape 2

Etape 5: il s'arrête quand il n'y a plus de clauses à sélectionner pour la résolution

Exemple 1: Le syllogisme de Socrate

	clauses de Horn	traduction Prolog
implicative	C_1: ¬homme(x) ∧ mortel(x) (1)	**mortel(x) -> homme(x);**
faits	C_2: homme (Socrate) (2)	**homme(Socrate) ->;**
but	C_3: mortel (Socrate) (3)	**mortel(Socrate);**

- but à démontrer : **Socrate est mortel** (clause but):

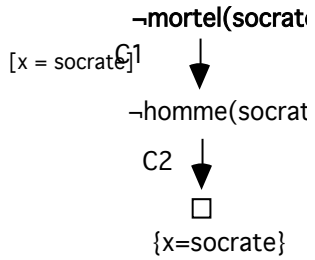
  C_3: mortel (Socrate)

- négation de l'affirmation à démonter C_3 est :

  $\neg C_3$: ¬mortel (Socrate) (3)

- Prolog **unifie** (1) et (3) en posant **[x = Socrate]** et l'on a alors :

  C_4: ¬homme (Socrate) (4)

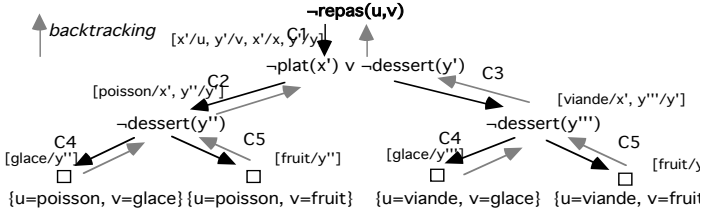


=> (4) est en **contradiction** avec (2), donc l'**affirmation (2) est prouvée**.

Exemple 2: Le repas léger

	clauses de Horn	traduction Prolog
implicative	C1: plat(x) ∧ dessert(y) ⊃ repas(x,y) C1: ¬plat(x) ∨ ¬dessert(y) ∨ repas(x,y)	**repas(x, y) -> plat(x) dessert(y);**
faits	C2: plat(poisson), C3: plat(viande), C4: dessert(glace), C5: dessert(fruit),	**plat(poisson) -> ;** **plat(viande) -> ;** **dessert(glace) -> ;** **dessert(fruit) -> ;**
but	C_6 = repas(u, v) ¬C_6 = ¬repas(u, v)	**repas(u, v);**

- pour montrer que l'ensemble de clauses est **contradictoire**, on construit l'arbre de résolution parcouru par le programme Prolog :

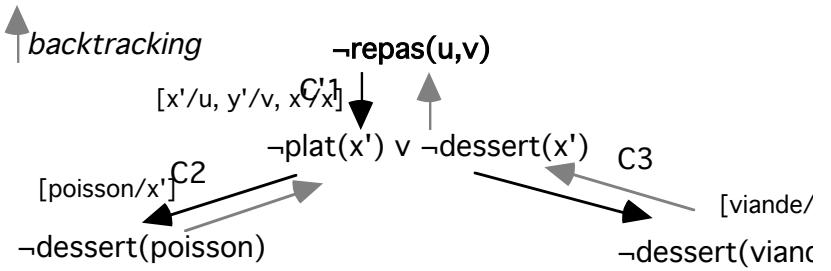


Exemple 2: Le repas léger

- dans le cas où la clause C_1 est remplacée par la clause C'_1:

  C'_1 = repas(x, y) → plat(x) dessert(x);

- le programme Prolog échoue et ne reporte aucune solution : arbre de résolution correspondant dont les feuilles sont les échecs :



- l'échec étant muet en Prolog, il est utile de disposer d'une trace qui reporte les tentatives d'unification et l'état des variables correspondant.

Intérêts et limites de la programmation logique

Intérêts :

- **En représentation des connaissances :**
  - les clauses traduisent les assertions ou théorèmes **directement sur l'univers de travail pour interprétation**
- **En interprétation des connaissances :**
  - Prolog fournit un **algorithme d'unification tout programmé** : l'arborescence de recherche (essais combinatoires d'unification des clauses 2 à 2) géré automatiquement (backtrack LIFO)
  - version Marseille : recherche en profondeur d'abord avec possibilité de retout arrière programmé (la coupure "!"): **incomplet mais plus efficace**

Limites :

- **En représentation des connaissances :**
  - limitée aux **clauses de HORN** (1 seul terme dans le membre droit),
- **En interprétation des connaissances :**
  - reste **très combinatoire des buts vers les données**
  - repose sur une systématisation du raisonnement par l'absurde : **pas naturel pour l'homme**.