

4. Satisfactibilitat

En general, les fórmules són certes en algunes interpretacions i falses en d'altres. A la secció 4.1 definim uns termes en aquesta relació, específicament els de la satisfactibilitat i la insatisfactibilitat. A la secció 4.2 presentem el mètode de taulers per a demostrar la insatisfactibilitat, i després de veure el tema preparatori de la unificació a la secció 4.3, en presentem un altre mètode, el de la resolució, a la secció 4.4. A la secció 4.5 presentem el tema propiament relacionat de la programació lògica.

4.1 Models

(1) **Definició** (*satisfacció, model*).

Una interpretació *satisfà* la fórmula A , o n'és un *model*, ssi A és certa en la interpretació.

(2) **Definició** (*tautologia, contradicció, contingència*).

- Una fórmula és *tautològica* o una *tautologia* ssi qualsevol interpretació la satisfà;
- és *contradictòria* o una *contradicció* ssi no és satisfeta per cap interpretació;
- és *contingent* o una *contingència* ssi és satisfeta per unes interpretacions i no per altres.

(3) **Comentari.**

Una fórmula és exclusivament contingent, tautològica o contradictòria.

(4) **Exemples.**

- $P \vee \neg P$, $\neg(P \wedge \neg P)$, i $P \rightarrow P$ són tautologies.
- $P \wedge \neg P$ i $\forall x(Px \wedge \neg Pf(x))$ són contradiccions.
- P és una contingència.

(5) **Comentari.**

- Una fórmula A és
 - tautològica ssi $A \equiv \blacksquare$,
 - contradictòria ssi $A \equiv \square$,
 - i contingent ssi $A \not\equiv \square$ i $A \not\equiv \blacksquare$.
- Igualment, $A \equiv B$
 - ssi $A \leftrightarrow B$ és una tautologia
 - ssi $\neg(A \leftrightarrow B)$ és una contradicció.

(6) **Definició** (*satisfactibilitat* d'una fórmula).

- Una fórmula A és *satisfactible* ssi té un model,
la qual cosa escrivim $\models A$;
- altrament és *insatisfactible*,
la qual cosa escrivim $\not\models A$.

(7)

tautologies	contingències	contradiccions
satisfactibles		insatisfactibles

Exercici 4.1. Com es poden definir mútuament satisfactibilitat i equivalència?

Solució. La fórmula A és satisfactible ssi A no és equivalent a $\square$, és a dir, $\models A$ ssi $A \not\equiv \square$; A és equivalent a B ssi $(A \wedge \neg B) \vee (B \wedge \neg A)$ és insatisfactible, és a dir, $A \equiv B$ ssi $\not\models (A \wedge \neg B) \vee (B \wedge \neg A)$.

(8) **Definició** (*satisfacció, model i satisfactibilitat* d'un conjunt de fórmules).

- Una interpretació *satisfà* un conjunt de fórmules, o n'és un *model*, ssi les satisfà totes.
- – Un conjunt $\{A_1, \dots, A_n\}$ de fórmules és *satisfactible* ssi té un model,
la qual cosa escrivim $\models A_1, \dots, A_n$;
– altrament, és *insatisfactible*,
la qual cosa escrivim $\not\models A_1, \dots, A_n$.

(9) **Proposició** (*compartiment dels models*).

$\Gamma \cup \{A \wedge B\}$ i $\Gamma \cup \{A, B\}$ comparteixen models, perquè una interpretació és un model de $\Gamma \cup \{A \wedge B\}$ ssi és un model de $\Gamma \cup \{A, B\}$.

(10) **Corol·lari** (*compartiment de la satisfactibilitat*).

$\models A \wedge B, \Gamma$ ssi
 $\models A, B, \Gamma$.

- (11) **Proposició** (*conservació de la satisfactibilitat sota la substitució uniforme de variables lliures per constants noves*).

Siguin $\{x_1, \dots, x_n\}$ les variables lliures en un conjunt Γ de fórmules i siguin $\{a_1, \dots, a_n\}$ constants que no apareixen a Γ .

Llavors,

$$\begin{aligned} & \models \Gamma \text{ ssi} \\ & \models \Gamma\{a_1/x_1, \dots, a_n/x_n\}. \end{aligned}$$

- (12) **Comentari.**

Els conjunts de fórmules de (11) no comparteixen models en general.

- (13) **Teorema** (*conservació de la satisfactibilitat sota l'Skolemització*).

Si no hi ha cap f a $\{\forall y_1 \dots \forall y_n \exists x A\} \cup \Gamma$ i $A\{f(y_1, \dots, y_n)/x\}$ és lliure,

$$\begin{aligned} & \models \forall y_1 \dots \forall y_n \exists x A, \Gamma \text{ ssi} \\ & \models \forall y_1 \dots \forall y_n (A\{f(y_1, \dots, y_n)/x\}), \Gamma. \end{aligned}$$

El símbol f s'anomena una *funció d'Skolem*; quan $n = 0$, per a $f()$ entenem una constant (d'Skolem).

- (14) **Comentari.**

Els conjunts de fórmules de (13) no comparteixen models en general.

Per demostrar que unes fórmules són satisfactibles n'hi ha prou de citar-ne un model, és a dir, donar una interpretació que les fa totes certes alhora.

(15) **Exemple.**

$$\models \neg \forall x \exists y Cxy, Pab, Pbc, Pcd, \neg \exists x \exists y \exists z \exists w (Pxy \wedge Pyz \wedge Pzw \wedge Cxz \wedge Cyw)$$

Perquè té, per exemple, el model amb domini $\{\alpha, \beta, \gamma, \delta\}$ i valuació $[a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma, d \mapsto \delta, P \mapsto \{\langle \alpha, \beta \rangle, \langle \beta, \gamma \rangle, \langle \gamma, \delta \rangle\}, C \mapsto \{\langle \alpha, \delta \rangle, \langle \delta, \alpha \rangle, \langle \beta, \gamma \rangle, \langle \gamma, \beta \rangle\}]$.

Però no podem demostrar la insatisfactibilitat sobre la base de comprovar totes les interpretacions, perquè n'hi ha un nombre infinit. Abordarem la insatisfactibilitat primer pel mètode de *taulers* i després pel mètode de *resolució*, que es basa en la *unificació*.

4.2 Taulers¹

Un *tauler* és un arbre d'enunciats crescut d'un conjunt d'enunciats inicials a la seva arrel de tal manera que el conjunt inicial és satisfactible si ho és alguna de les branques senceres de l'arbre. En aquesta secció suposem el llenguatge de predicats amb el vocabulari lògic $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists\}$.

Els taulers es construeixen estenent fulles amb enunciats més petits segons els enunciats que ja apareixen en una branca.

(16) **Definició** (*branca tancada d'un tauler*).

Una branca d'un tauler és *tancada* si conté un enunciat i aquest enunciat negat, la qual cosa assenyalarem amb una $\times$ a la seva fulla.

(17) **Comentari.**

Els enunciats de qualsevol branca tancada d'un tauler són insatisfactibles; qualsevol conjunt de fórmules que conté un enunciat i aquest enunciat negat és insatisfactible: l'enunciat i l'enunciat negat no es poden ser certs alhora.

¹Aquesta secció s'ha escrit consultant uns apunts de Rafel Farré.

Com a primer exemple d'una regla per a estendre branques de taulers hi ha la regla següent per a la negació:

$$(18) \quad \begin{array}{c} \neg\neg A\checkmark \\ | \\ A \end{array}$$

La regla diu que quan hi ha un enunciat $\neg\neg A$ (no marcat com a ja usat), es pot estendre *cada* fulla no tancada que domina amb A i marcar $\neg\neg A$ com a usat. Les altres regles per a estendre branques es classifiquen en regles α , β , γ i δ . Són les següents. Regles- α :

$$(19) \quad \begin{array}{ccc} A \wedge B\checkmark & \neg(A \vee B)\checkmark & \neg(A \rightarrow B)\checkmark \\ | & | & | \\ A & \neg A & A \\ B & \neg B & \neg B \end{array}$$

Regles- β :

$$(20) \quad \begin{array}{ccc} A \vee B\checkmark & \neg(A \wedge B)\checkmark & A \rightarrow B\checkmark \\ / \quad \backslash & / \quad \backslash & / \quad \backslash \\ A \quad B & \neg A \quad \neg B & \neg A \quad B \end{array}$$

$$\begin{array}{ccc} A \leftrightarrow B\checkmark & \neg(A \leftrightarrow B)\checkmark & \\ / \quad \backslash & / \quad \backslash & \\ A \quad \neg A & A \quad \neg A & \\ B \quad \neg B & \neg B \quad B & \end{array}$$

Regles- γ :

$$(21) \quad \begin{array}{ccc} \forall x A & \neg \exists x A & \\ | & | & \\ A\{t/x\} & \neg A\{t/x\} & \text{On } t \text{ és qualsevol terme tancat} \end{array}$$

Regles- δ :

$$(22) \quad \begin{array}{ccc} \exists x A\checkmark & \neg \forall x A\checkmark & \\ | & | & \\ A\{a/x\} & \neg A\{a/x\} & \text{On } a \text{ és una constant nova} \end{array}$$

Notem que en les regles- γ , i només en aquestes, no es marca l'enunciat com a usat: les regles- γ es poden reutilitzar sense límit.

(23) **Definició** (*tauler per a un conjunt d'enunciats Γ*).

El tauler inicial per a un conjunt d'enunciats Γ és un *tauler per al conjunt d'enunciats Γ* . I si es té un tauler per a un conjunt d'enunciats Γ , és també un *tauler per al conjunt d'enunciats Γ* , el resultat d'aplicar una de les regles anteriors a un enunciat no marcat com a usat, estenent *totes* les fulles no tancades dominades per l'enunciat amb les filles segons la regla.

(24) **Lema** (*conservació de la satisfactibilitat en els taulers*).

Per a qualsevol tauler, el conjunt inicial és satisfactible sii alguna branca de l'arbre és satisfactible.

Demostració. Per a la regla de negació s'ha de demostrar que per a qualsevol conjunt d'enunciats Γ , $\Vdash \Gamma, \neg\neg A$ sii $\Vdash \Gamma, \neg\neg A, A$. Per a les regles- α s'ha de demostrar que per a qualsevol conjunt d'enunciats Γ , $\Vdash \Gamma, A \wedge B$ ssi $\Vdash \Gamma, A \wedge B, A, B$, etc. Per a les regles- β s'ha de demostrar que per a qualsevol conjunt d'enunciats Γ , $\Vdash \Gamma, A \vee B$ ssi $\Vdash \Gamma, A \vee B, A$ o $\Vdash \Gamma, A \vee B, B$, etc. Per a les regles- γ s'ha de demostrar que per a qualsevol conjunt d'enunciats Γ , $\Vdash \Gamma, \forall x A$ ssi $\Vdash \Gamma, \forall x A, A\{t/x\}$, etc. Per a les regles- δ s'ha de demostrar que per a qualsevol conjunt d'enunciats Γ , $\Vdash \Gamma, \exists x A$ ssi $\Vdash \Gamma, \exists x A \wedge A\{a/x\}$, etc.

(25) **Definició** (*tauler tancat*).

Un tauler és *tancat* sii totes les seves branques són tancades.

(26) **Teorema** (*solidesa dels taulers*).

Per a qualsevol conjunt inicial d'enunciats Γ , si un tauler per a Γ és tancat, Γ és insatisfactible.

Demostració. Pel lema (24) de conservació de satisfactibilitat en taulers, el conjunt inicial Γ és satisfactible sii alguna branca és satisfactible. Però totes les branques d'un tauler tancat són insatisfactibles, per tant el conjunt inicial d'un tauler tancat és insatisfactible.

També enunciem la *completesa* de taulers:

(27) **Teorema** (*completesa de taulers*).

Si un conjunt inicial d'enunciats Γ és insatisfactible, llavors hi ha un tauler tancat per a Γ .

Demostració. La demostració d'aquest teorema es troba fora de l'abast d'apunts introductoris. Per a una versió en català consulteu, per exemple, Morrill (2001) *Lògica de primer ordre*, Edicions UPC, Barcelona.

Com a primer exemple de la construcció d'un tauler considerem el conjunt d'enunciats $\{\neg P \wedge Q, P \vee \neg Q\}$. Comencem escrivint aquests enunciats inicials un sota l'altre:

$$(28) \quad \begin{array}{c} \neg P \wedge Q \\ P \vee \neg Q \end{array}$$

Ara apliquem la regla- α per al primer enunciat:

$$(29) \quad \begin{array}{c} \neg P \wedge Q \checkmark \\ P \vee \neg Q \\ \hline \neg P \\ Q \end{array}$$

I apliquem la regla- β per al segon enunciat:

$$(30) \quad \begin{array}{c} \neg P \wedge Q \checkmark \\ P \vee \neg Q \checkmark \\ | \\ \neg P \\ Q \\ \swarrow \quad \searrow \\ P \qquad \neg Q \end{array}$$

Veiem que a la branca a l'esquerra hi ha l'enunciat i l'enunciat negat $\{P, \neg P\}$ i a la branca a la dreta hi ha l'enunciat i l'enunciat negat $\{Q, \neg Q\}$. Per tant tanquem aquestes branques:

$$(31) \quad \begin{array}{c} \neg P \wedge Q \checkmark \\ P \vee \neg Q \checkmark \\ | \\ \neg P \\ Q \\ \swarrow \quad \searrow \\ P \qquad \neg Q \\ \times \qquad \times \end{array}$$

Ambdues branques són tancades, per tant el conjunt inicial és insatisfactible.

Com a segon exemple de la construcció de taulers considerem el conjunt inicial $\{\exists x(Px \wedge \neg Qx), \forall x(\neg Px \vee Qx)\}$:

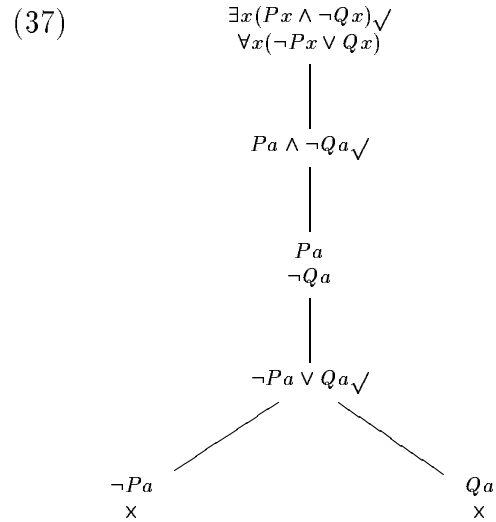
$$(32) \quad \begin{array}{c} \exists x(Px \wedge \neg Qx) \\ \forall x(\neg Px \vee Qx) \end{array}$$

$$(33) \quad \begin{array}{c} \exists x(Px \wedge \neg Qx) \checkmark \\ \forall x(\neg Px \vee Qx) \\ | \\ Pa \wedge \neg Qa \end{array}$$

$$\begin{array}{c}
 (34) \quad \begin{array}{c} \exists x(Px \wedge \neg Qx) \checkmark \\ \forall x(\neg Px \vee Qx) \end{array} \\
 \quad \quad \quad | \\
 \quad \quad \quad \begin{array}{c} Pa \wedge \neg Qa \checkmark \end{array} \\
 \quad \quad \quad | \\
 \quad \quad \quad \begin{array}{c} Pa \\ \neg Qa \end{array}
 \end{array}$$

$$\begin{array}{c}
 (35) \quad \begin{array}{c} \exists x(Px \wedge \neg Qx) \checkmark \\ \forall x(\neg Px \vee Qx) \end{array} \\
 \quad \quad \quad | \\
 \quad \quad \quad \begin{array}{c} Pa \wedge \neg Qa \checkmark \end{array} \\
 \quad \quad \quad | \\
 \quad \quad \quad \begin{array}{c} Pa \\ \neg Qa \end{array} \\
 \quad \quad \quad | \\
 \quad \quad \quad \neg Pa \vee Qa
 \end{array}$$

$$\begin{array}{c}
 (36) \quad \begin{array}{c} \exists x(Px \wedge \neg Qx) \checkmark \\ \forall x(\neg Px \vee Qx) \end{array} \\
 \quad \quad \quad | \\
 \quad \quad \quad \begin{array}{c} Pa \wedge \neg Qa \checkmark \end{array} \\
 \quad \quad \quad | \\
 \quad \quad \quad \begin{array}{c} Pa \\ \neg Qa \end{array} \\
 \quad \quad \quad | \\
 \quad \quad \quad \neg Pa \vee Qa \checkmark \\
 \quad \quad \quad \swarrow \quad \searrow \\
 \neg Pa \quad \quad \quad Qa
 \end{array}$$



(38) **Definició** (*literal*).

Un *literal* és un àtom o un àtom negat.

(39) **Exemples.**

- $\neg Qy$, Pax , R i $\neg S$ són literals.
- No són literals $\neg\neg R$ ni $R \wedge S$ ni $R \vee S$.

(40) **Definició** (*branca oberta d'un tauler*).

Una branca d'un tauler és *oberta* sii conté només enunciats marcats com a usats i literals, sense cap enunciat i enunciat negat, la qual cosa assenyalem amb una $\bigcirc$ a la seva fulla.

(41) **Lema.**

El conjunt inicial d'un tauler amb una branca oberta és satisfactible.

Demostració. La demostració d'aquest teorema es troba fora de l'abast d'apunts introductoris. Per a una versió en català consulteu, per exemple, Morrill (2001) *Lògica de primer ordre*, Edicions UPC, Barcelona.

Considereu per exemple un tauler per al conjunt inicial $\neg P \wedge Q, (P \wedge R) \vee R$:

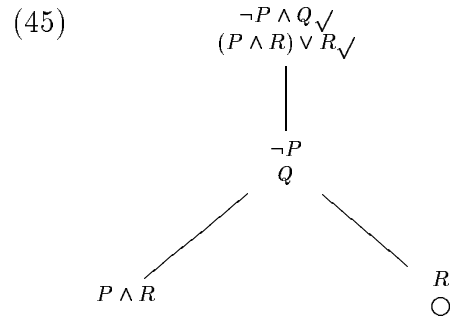
$$(42) \quad \begin{array}{c} \neg P \wedge Q \\ (P \wedge R) \vee R \end{array}$$

$$(43) \quad \begin{array}{c} \neg P \wedge Q \checkmark \\ (P \wedge R) \vee R \\ \hline \neg P \\ Q \end{array}$$

$$(44) \quad \begin{array}{c} \neg P \wedge Q \checkmark \\ (P \wedge R) \vee R \checkmark \\ \hline \neg P \\ Q \\ \swarrow \quad \searrow \\ P \wedge R \quad R \end{array}$$

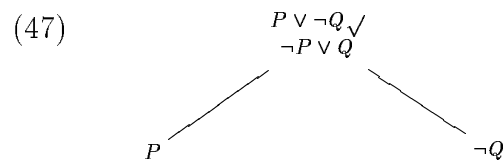
La branca a la dreta consisteix només en enunciats usats i literals sense cap

enunciat i enunciat negat, per tant es pot assenyalar com a oberta:

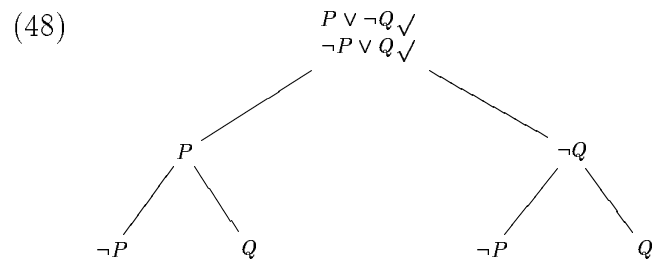


Per tant, el conjunt inicial és satisfactible.

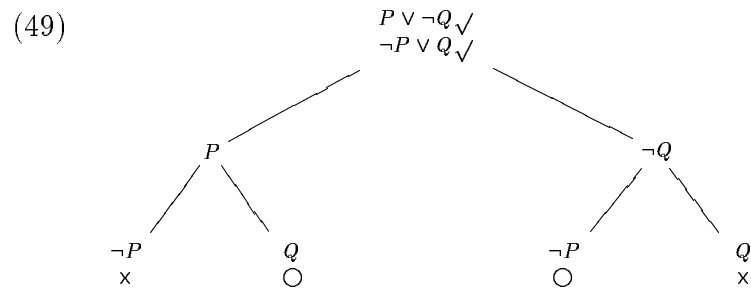
Considereu el conjunt inicial $\{P \vee \neg Q, \neg P \vee Q\}$:



Ara, l'aplicació de la regla al segon enunciat ha d'estendre *totes* dues fulles (no tancades) que domina:

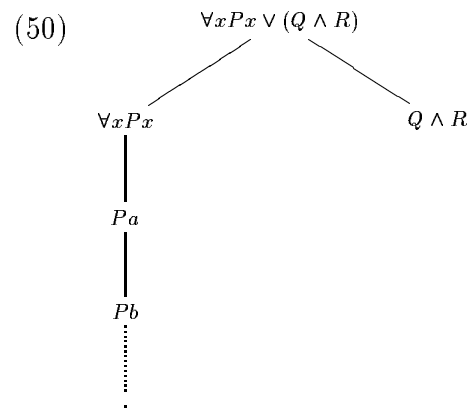


Algunes de les branques són tancades i d'altres obertes:

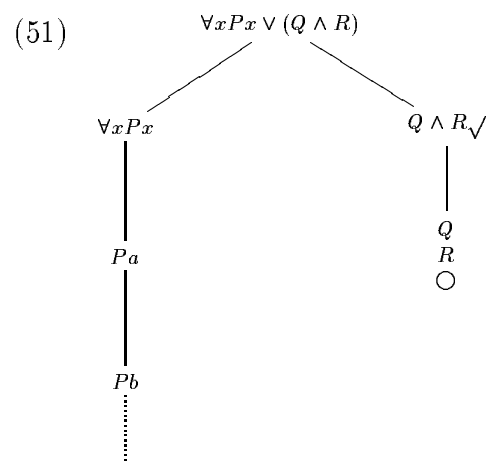


Per tant el conjunt inicial és satisfactible.

Considerem el tauler (50).



Podríem continuar per sempre aplicant la regla per a $\forall x Px$; d'altra banda, amb una sola aplicació a $Q \wedge R$ trobem una branca oberta, i per tant podem treure la conclusió que el conjunt inicial és satisfactible:



Això ilustra que una construcció intel·ligent de taulers hauria d'operar per

amplitud i no per profunditat. Considerem també el tauler següent:

$$(52) \quad \begin{array}{c} \forall x Px \\ Q \wedge \neg Q \\ \vdots \\ Pa \\ \vdots \\ Pb \\ \vdots \\ \vdots \end{array}$$

Aquí també veiem que podríem continuar per sempre aplicant la regla per a $\forall x Px$, mentre una sola aplicació a $Q \wedge \neg Q$ donaria una branca tancada, i per tant el resultat que el conjunt inicial és insatisfactible:

$$(53) \quad \begin{array}{c} \forall x Px \\ Q \wedge \neg Q \checkmark \\ \vdots \\ Pa \\ \vdots \\ Pb \\ \vdots \\ Q \\ \neg Q \\ \times \end{array}$$

Això ilustra que una construcció intel·ligent de taulers hauria d'aplicar regles als enunciats d'una branca cíclicament.

Un *procediment de decisió* per a la satisfactibilitat en el llenguatge proposicional és un algorisme que per a qualsevol conjunt inicial d'enunciats ens diria (correctament) després d'un temps finit si el conjunt és satisfactible o insatisfactible. La calculació de taules veritatives constueix un procediment de decisió per a la satisfactibilitat en el llenguatge proposicional. Els taulers en constueixen un altre. Diem doncs que el problema de la satisfactibilitat en el llenguatge proposicional és *decidable*.

Un *procediment de decisió* per a la satisfactibilitat en el llenguatge de predicats seria un algorisme que per a qualsevol conjunt inicial d'enunciats ens diria (correctament) després d'un temps finit si el conjunt és satisfactible

o insatisfactible. Els taulers ens donen un procediment de decisió per al llenguatge de predicats? Si es fa una construcció intel·ligent de taulers, està garantit que trobarem en un temps finit un tauler tancat si el conjunt inicial és insatisfactible. Però si el conjunt inicial és satisfactible és possible que la construcció d'un tauler, tot i ser intel·ligent, no acabi mai. Així els taulers no ens donen cap procediment de decisió per al llenguatge de predicats. La dificultat queda amb les regles- γ que es poden fer servir un nombre de vegades sense límit sense poder marcar l'enunciat com a usat. Aquesta dificultat no es pot eludir; el problema de la satisfactibilitat en el llenguatge de predicats és *indecidible*. S'ha demostrat que no existeix cap procediment de decisió de cap tipus per a la satisfactibilitat en el llenguatge de predicats: suposar altrament donaria lloc a una contradicció matemàtica.

Tot i la no decidabilitat, una heurística pràctica en la construcció de taulers és d'aplicar regles- α abans de regles- δ abans de regles- β abans de regles- γ . Per exemple, les constants que introduïm en regles- δ s'acostumen a utilitzar posteriorment en regles- γ .

4.3 Unificació

La unificació tracta d'una certa compatibilitat entre expressions.

(54) **Definició** (*unificador, unificació, unificable*).

Un *unificador* de dues expressions ϕ i ϕ' és una substitució θ tal que $\phi\theta = \phi'\theta$; diem que $\phi\theta$ ($=\phi'\theta$) és la *unificació* de ϕ i ϕ' sota θ .

Dues expressions són *unificables* ssi tenen un unificador.

(55) **Exemples.**

Els parells d'àtoms següents tenen els unificadors indicats.

Px	$Pf(y)$	$\{f(a)/x, a/y\}, \{f(y)/x\}, \dots$
Pab	Qab	cap
$P03x$	$P0yy$	$\{3/y, 3/x\}$
$Pf(a)$	$Pg(a)$	cap
$Pxf(x)$	$PAf(b)$	cap
Px	$Pf(x)$	cap
Pxx	Px	cap
$Pf(a)$	$Pf(a, a)$	cap
Pxy	Pyx	$\{x/y\}, \{y/x\}, \dots$

(56) **Definició** (*unificador més general*).

Un unificador θ de dues expressions ϕ i ϕ' n'és un *unificador més general* (UMG) ssi per a tot unificador σ de ϕ i ϕ' hi ha una substitució ρ tal que $\phi\theta\rho(= \phi'\theta\rho) = \phi\sigma(= \phi'\sigma)$.

(57) **Exemples.**

Px i $Pf(y)$ tenen un unificador més general $\{f(y)/x\}$.

Pxy i Pyx tenen unificadors més generals $\{x/y\}$ i $\{y/x\}$, que diem que són *variants alfabètiques*.

(58) **Definició** (*problema d'unificació, solució a un problema d'unificació*).

Un *problema d'unificació* és un conjunt de parelles de termes.

Una *solució al problema d'unificació* π és una substitució que és un unificador de tota parella a π .

L'algoritme següent calcula una solució a un problema d'unificació, si és que n'hi ha.

(59) **Definició** (*algoritme d'unificació per transformacions*).

```

unif( $\pi, \sigma$ )
cas  $\pi$  de
  []: sortir amb  $\sigma$ ;
  [ $t = t|\pi'$ ]: unif( $\pi', \sigma$ );
  [ $f(t_1, \dots, t_n) = f(t'_1, \dots, t'_n)|\pi'$ ]: unif( $[t_1 = t'_1, \dots, t_n = t'_n|\pi'], \sigma$ );
  [ $x = t|\pi'$ ] o [ $t = x|\pi'$ ] sense cap  $x$  a  $t$ : unif( $\pi'\{t/x\}, [t/x](\sigma\{t/x\})$ );
  [ $x = t|\pi'$ ] o [ $t = x|\pi'$ ] amb  $x$  un subterme proper de  $t$ :
    fallar amb “no unificables (ocurrència)”;
  [ $f \dots = g \dots |\pi'$ ]:
    fallar amb “no unificables (conflicte)”.

```

(60) **Comentari.**

- Hi ha dos paràmetres: un *problema d'unificació* π que representem amb una llista $[t_1 = t'_1, \dots, t_m = t'_m]$ i una *substitució* σ que representem amb una llista $[t''_1/x_1, \dots, t''_n/x_n]$.
- Cada pas transforma el problema d'unificació i/o la substitució. Si hi ha una solució al problema d'unificació inicial, l'execució l'acaba buidant i retorna la substitució final; altrament, l'execució s'atura imprimint “no unificables”.
- L'únic punt de no-determinisme es produeix respecte si una equació $x = y$ es tracta amb x/y o y/x , la qual cosa dóna lloc a variants alfabètiques.

Clarament, dos àtoms són unificables ssi tenen el mateix predicat de la mateixa arietat i el problema d'unificació dels termes corresponents té solució. Hi ha la següent:

(61) **Proposició.**

Si $Pt_1 \dots t_n$ i $Pt'_1 \dots Pt'_n$ són unificables, $\text{unif}([t_1 = t'_1, \dots, t_n = t'_n], [])$ en calcula un unificador més general.

(62) **Exemple.**

Per unificar $Pxf(x, y)$ i $Pg(y)f(g(a), z)$ hi ha l'execució:

π	σ
$[x = g(y), f(x, y) = f(g(a), z)]$	$[]$
$[f(g(y), y) = f(g(a), z)]$	$[g(y)/x]$
$[g(y) = g(a), y = z]$	$[g(y)/x]$
$[y = a, y = z]$	$[g(y)/x]$
$[a = z]$	$[a/y, g(a)/x]$
$[]$	$[a/z, a/y, g(a)/x]$

Ens dona UMG $\{a/z, a/y, g(a)/x\}$.

Exercici 4.2. Unifiqueu:

- i. Pxy i $Pyf(z)$
- ii. Pxa i Pyb
- iii. $Pxg(x)$ i Pyg
- iv. $Pxg(x)y$ i $Pzug(u)$

4.4 Resolució

(63) **Definició** (*literal complementari*).

El *literal complementari* d'un literal L és el literal que conté el mateix àtom amb polaritat complementària, la qual cosa escrivim $\overline{L}$.

(64) **Exemples.**

- $\overline{\neg Qy} = Qy$.
- $\overline{Pax} = \neg Pax$.

(65) **Definició** (*parella complementària*).

Una *parella complementària* està formada per un àtom i aquest àtom negat.

El mètode de resolució *no* treballa directament amb *fórmules*, sinó amb *clàusules*.

(66) **Definició** (*clàusula*).

Una *clàusula* és un conjunt de literals.

Una clàusula es llegeix com la clausura universal (és a dir, amb totes les variables quantificades universalment a fora) de la disjunció dels seus literals.

(67) **Exemples.**

- Una clàusula (binària)
 $\{Pax, \neg Qy\}$ es llegeix $\forall x \forall y (Pax \vee \neg Qy)$ i s'escriu $Pax \vee \neg Qy$.
- Una clàusula (unitària)
 $\{Pax\}$ es llegeix $\forall x Pax$ i s'escriu Pax .
- La clàusula (buida)
 $\{\}$ es llegeix $\square$ i s'escriu $\square$.

(68) **Definició** (*renomenament* d'una clàusula).

Siguin $\{x_1, \dots, x_n\}$ les variables en una clàusula C , i siguin les variables $\{x'_1, \dots, x'_n\}$. Llavors $C\{x'_1/x_1, \dots, x'_n/x_n\}$ és una *renomenament* de C .

(69) **Comentari.**

Les lectures de tots els renomenaments d'una clàusula són equivalents.

En el mètode de resolució hi ha dues regles: la regla de resolució clausular i la regla de factorització:

(70) **Definició** (*regla de resolució clausular*).

$$\frac{C \cup \{L\} \quad \{M\} \cup D}{C \cup D\theta} \theta, \text{ si } \theta \text{ és un UMG de } \{L, \overline{M}\}$$

Aquesta regla afirma que si θ és un UMG de $\{L, \overline{M}\}$, llavors de $C \cup \{L\}$ i $\{M\} \cup D$ es pot derivar la *resolvent* $C \cup D\theta$.

(71) **Definició** (*regla de factorització clausular*).

$$\frac{C}{C\theta} \theta, \text{ si } \theta \text{ és un UMG de } \{L, M\} \subseteq C$$

Aquesta regla afirma que si θ és un UMG de dos literals de C , llavors es pot derivar $C\theta$ (en la qual els literals unificables es col·lapsen en un).

(72) **Definició** (*arbre de resolució*).

Un *arbre de resolució* partint d'un conjunt de clàusules inicials és un arbre en el qual:

- cada arbre local és una instància de resolució clausular o de factorització;
- cada fulla és un renomament d'una clàusula inicial o bé d'una clàusula derivada anteriorment;
- l'arrel és la clàusula buida.

Tenim el següent:

(73) **Teorema** (*correcció del mètode de resolució*).

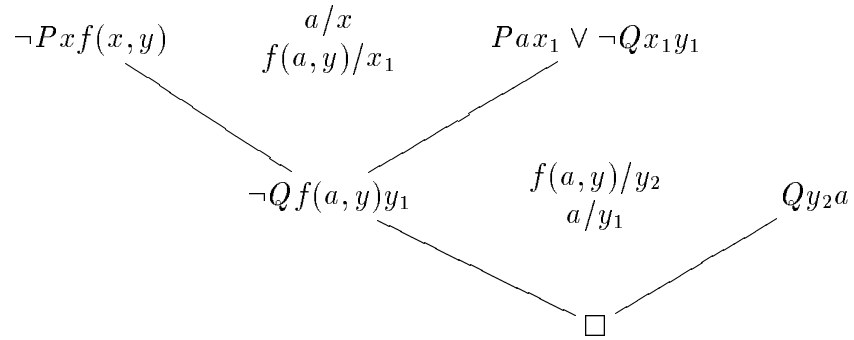
Un conjunt $\mathcal{O}$ de clàusules és insatisfactible ssi hi ha un arbre de resolució partint d' $\mathcal{O}$.

(74) **Exemple.**

Partint de les clàusules

$$\neg Pxf(x, y), Pax_1 \vee \neg Qx_1y_1, Qy_2a$$

hi ha l'arbre de resolució



l'existència del qual demostra que

$$\not\models \forall x \forall y \neg Pxf(x, y), \forall x \forall y (Pax \vee \neg Qxy), \forall x Qxa$$

(75) **Comentari.**

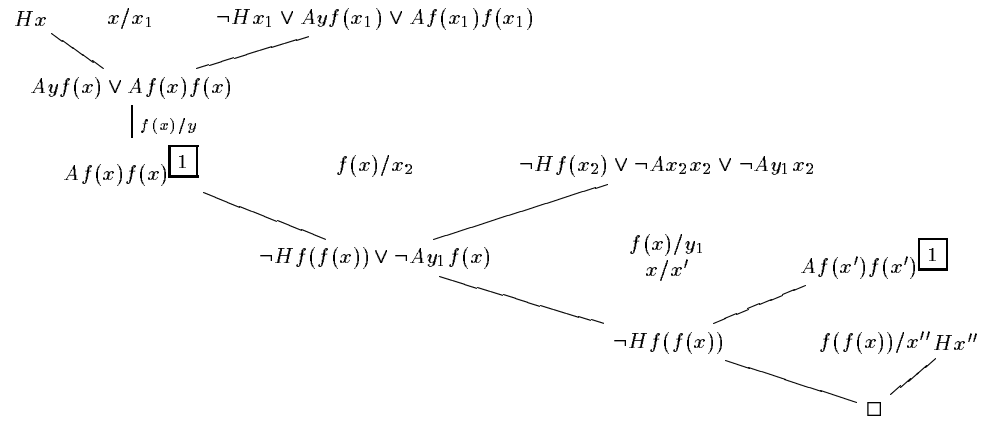
- El conjunt de clàusules $\{Px, \neg Pf(x)\}$ és insatisfactible, però si n'agafem els renomenaments Py i $\neg Pf(y)$ no hi podem aplicar la regla de resolució clausal, perquè una variable i un terme que conté aquesta variable com a subterme proper no són unificables. Un cas com aquest passa només per la casualitat que els renomenaments triats comparteixen desafortunadament una variable.
- Podem evitar d'una vegada per sempre aquest perill renomenant amb variables totalment noves cada clàusula que entra com a fulla en un arbre de resolució: a clàusules diferents, a ocurrències distintes de la mateixa clàusula i a repeticions de clàusules derivades. D'això se'n diu *standardització a part*.

(76) **Exemple.**

Que

$$\not\models \forall x Hx, \forall x \forall y (\neg Hx \vee Ayf(x) \vee Af(x)f(x)), \forall x \forall y (\neg Hf(x) \vee \neg Axx \vee \neg Ayx)$$

queda demostrat per



(77) **Definició (clausularització).**

Una *clausularització* d'un conjunt de fórmules és un conjunt de clàusules que resulta de:

- convertir cada fórmula en FNPC per normalització (conservant la satisfactibilitat; de fet, conservant els models);
- convertir cada FNPC en una FNPC tancada a base de substituir uniformement les variables lliures per constants noves (conservant la satisfactibilitat, tot i no conservar els models);
- eliminar els existencials per Skolemització, traient successivament els existencials dels prefixos (conservant la satisfactibilitat, tot i no conservar els models);
- treure una clàusula de cada conjuntand de cada matriu de cada FNPC sense existencials.

és una conclusió anterior. Notem que tots els arbres de resolució d'aquesta secció han sigut lineals.

(80) **Teorema** (*completesa de la resolució lineal*).

La resolució lineal (incloent els passos secundaris) és refutacionalment complet; és a dir, si un conjunt de clàusules és insatisfactible, hi ha un arbre de resolució lineal partint d'aquestes amb arrel $\square$.

4.5 Programació lògica²

La *programació lògica* és un paradigma computacional per a treure conclusions de conjunts de fórmules, anomenats programes lògics. Aquestes fórmules són clàusules d'una forma especial anomenades clàusules de Horn, i les conclusions s'obtenen per una versió de resolució lineal anomenada resolució SLD.

La resolució lineal només amb passos primaris es diu *resolució lineal d'entrada*. La resolució lineal d'entrada no és completa per a les clàusules en general, però sí que és completa (i sense factorització clausular) per a les *clàusules de Horn*:

(81) **Definició** (*literal positiu i literal negatiu*).

Un *literal positiu* és qualsevol àtom. Un *literal negatiu* és qualsevol àtom negat.

(82) **Definició** (*clàusula de Horn*).

Una *clàusula de Horn* és una clàusula que conté com a màxim un literal positiu.

(83) **Teorema** (*completesa de la resolució lineal d'entrada sense factorització per a clàusules de Horn*).

Resolució lineal d'entrada sense factorització és refutacionalment completa per a clàusules de Horn.

²Aquesta secció s'ha escrit tot consultant uns apunts d'Edelmira Pasarella.

Les clàusules de Horn es divideixen en els *objectius* i les *clàusules de programa*:

(84) **Definició** (*objectiu*).

Un *objectiu* és una clàusula que no conté cap literal positiu, la qual cosa escrivim $\leftarrow A_1, \dots, A_k$.

(85) **Definició** (*clàusula de programa*).

Una *clàusula de programa* és una clàusula que conté exactament un literal positiu.

Les clàusules de programa es divideixen en les *regles* i els *fets*:

(86) **Definició** (*regla*).

Una *regla* és una clàusula de programa que conté almenys un literal negatiu, la qual cosa escrivim $A \leftarrow B_1, \dots, B_m$; A es diu el *cap* de la regla i $B_1, \dots, B_m$ es diu el *cos* de la regla.

(87) **Definició** (*fet*).

Un *fet* és una clàusula de programa que no conté cap literal negatiu, llavors és un sol literal positiu, la qual cosa escrivim $A \leftarrow$.

(88) **Definició** (*clàusula definida*).

Una *clàusula definida* és una clàusula amb els seus literals ordenats en una seqüència.

(89) **Definició** (*programa lògic definit*)

Un *programa lògic definit* és un conjunt finit de clàusules definides de programa.

Considerem la resolució lineal d'entrada que parteix d'un programa lògic definit i un objectiu definit $\leftarrow A_1, \dots, A_k$. Començarà:

$$(90) \quad \leftarrow A_1, \dots, A_k \quad \quad A \leftarrow B_1, \dots, B_m$$

Cada clàusula troncal serà un objectiu i cada segona premissa serà una clàusula definida de programa del programa lògic definit. Cada pas de resolució es farà respecte a una parella complementària que consisteix en un literal negatiu de l'objectiu a l'esquerra, i el literal positiu que és el cap de la clàusula de programa a la dreta. Una *funció de selecció* és una funció que, donat un objectiu definit, retorna un dels seus literals negatius com a *literal seleccionat*. Una resolució lineal d'entrada que parteix d'un programa lògic definit i un objectiu definit és una *resolució lineal amb funció de selecció per a clàusules definides* si i només si a cada pas de resolució, el membre de la parella complementària que ve de l'objectiu és el triat per una funció de selecció. D'això se'n diu *resolució SLD* (de l'anglès Linear resolution with Selection function for Definite clauses).

Aquí considerem la resolució SLD en la qual el literal seleccionat és sempre el primer (el de més a l'esquerra).

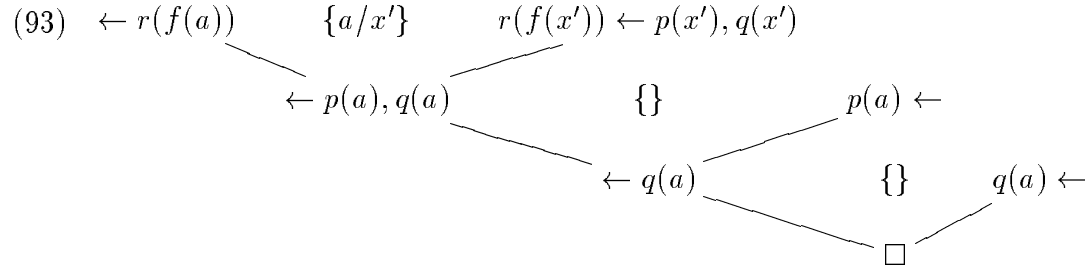
(91) **Definició** (*SLD-derivat*).

Sigui P un programa definit, $\leftarrow A_1, \dots, A_n$ un objectiu definit, i $A \leftarrow B_1, \dots, B_k$ una clàusula de programa definida de P amb les variables renomenades. Si A_1 i A són unificables amb UMG θ , llavors l'objectiu definit $\leftarrow (B_1, \dots, B_k, A_2, \dots, A_n)\theta$ és *SLD-derivat* de $A_1, \dots, A_n$ y $A \leftarrow B_1, \dots, B_k$ usant θ .

Per exemple, sigui el programa lògic definit següent:

$$(92) \quad \begin{aligned} p(a) &\leftarrow \\ p(b) &\leftarrow \\ q(a) &\leftarrow \\ r(f(x)) &\leftarrow p(x), q(x) \end{aligned}$$

Hi ha la resolució SLD següent per a l'objectiu $\leftarrow r(f(a))$:



En *derivacions SLD* ometem les segones premisses:

(94) **Definició** (*derivació SLD*).

Una *derivació SLD* d'un objectiu G respecte d'un programa definit P és una seqüència possiblement infinita $\langle G_0, \theta_0 \rangle, \langle G_1, \theta_1 \rangle, \dots$ on $G_0 = G$, $\theta_0 = \{ \}$ i cada G_{i+1} és SLD-derivat de G_i i alguna clàusula de P usant θ_{i+1} .

Una derivació SLD pot tenir èxit o fallar:

(95) **Definició** (*derivació SLD amb èxit o refutació SLD*).

Una derivació SLD és una *derivació SLD amb èxit* o una *refutació SLD* si i només si és finita i finalitza amb la clàusula buida.

(96) **Definició** (*derivació SLD amb falla*).

Una derivació SLD és una *derivació SLD amb falla* si i només si és finita i finalitza amb un objectiu diferent de la clàusula buida l'àtom del primer literal del qual no unifica amb el cap de cap de les clàusules de programa.

(97) **Definició** (*SLD-resposta calculada d'un objectiu respecte d'un programa lògic*).

Sigui $\langle G_0, \theta_0 \rangle, \dots, \langle G_n, \theta_n \rangle$ una refutació SLD de l'objectiu G respecte del programa lògic P . La *SLD-resposta calculada de G respecte de P* és la composició $\theta_1 \cdots \theta_n$ restringida a les variables que ocorren lliures a G .

(98) **Exemple.**

La següent és una refutació SLD de l'objectiu $\leftarrow r(f(a))$ respecte del programa lògic (92):

$$\langle \leftarrow r(f(a)), \{\} \rangle, \langle \leftarrow p(a), q(a), \{a/x'\} \rangle, \langle \leftarrow q(a), \{\} \rangle, \langle \square, \{\} \rangle$$

Correspon a la resolució lineal (93). La resposta calculada és $\{\}$ ja que x no és lliure a $r(f(a))$.

(99) **Exemple.**

La següent és una refutació SLD de l'objectiu $\leftarrow r(f(x))$ respecte del programa lògic (92):

$$\langle \leftarrow r(f(x)), \{\} \rangle, \langle \leftarrow p(x'), q(x'), \{x'/x\} \rangle, \langle \leftarrow q(a), \{a/x'\} \rangle, \langle \square, \{\} \rangle$$

La resposta calculada és $\{x'/x\}\{a/x'\}|_{\{x\}} = \{a/x\}$.

(100) **Exemple.**

La següent és una refutació SLD de l'objectiu $\leftarrow r(x)$ respecte del programa lògic (92):

$$\langle \leftarrow r(x), \{\} \rangle, \langle p(x'), q(x'), \{f(x')/x\} \rangle, \langle \leftarrow q(a), \{a/y\} \rangle, \langle \square, \{\} \rangle$$

La resposta calculada és $\{f(x')/x\}\{a/x'\}|_{\{x\}} = \{f(a)/x\}$.

L'espai de cerca per a refutacions SLD ve donat per un *arbre SLD*:

(101) **Definició** (*arbre SLD*).

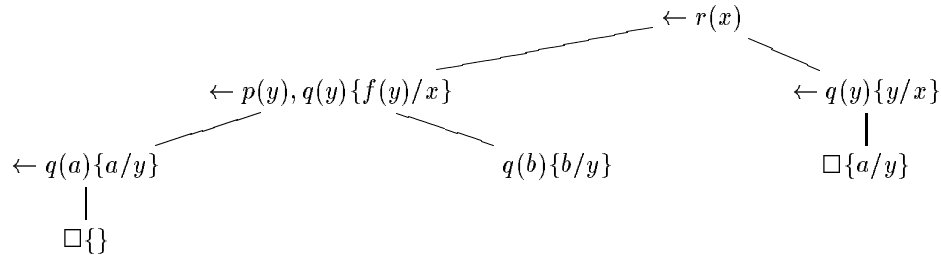
Un *arbre SLD* d'un objectiu G respecte d'un programa lògic P és un arbre d'objectius amb arrel G tal que cada node $\square$ no té filles i tal que per a cada node $\leftarrow A_0, \dots, A_n$, per a cada clàusula de programa $A \leftarrow B_1, \dots, B_k$ tal que A_0 i A són unificables amb UMG θ , el node té una filla $(B_1, \dots, B_k, A_1, \dots, A_n)\theta$.

Sigui la modificació següent del programa lògic (92):

(102) $p(a) \leftarrow$
 $p(b) \leftarrow$
 $q(a) \leftarrow$
 $r(f(x)) \leftarrow p(x), q(x)$
 $r(x) \leftarrow q(x)$

(103) **Exemple.**

L'arbre SLD de l'objectiu $\leftarrow r(x)$ respecte del programa lògic (102) és el següent:



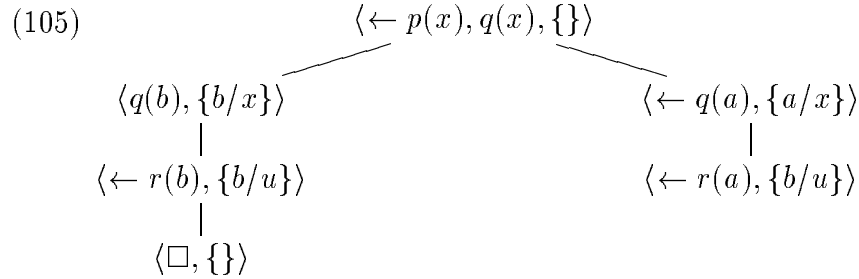
Notem com cada branca correspon a una derivació SLD. La derivació a l'esquerra és una refutació amb resposta calculada $\{f(a)/x\}$, la derivació al mig és amb falla, i la derivació a la dreta és una refutació amb resposta calculada $\{a/x\}$.

En quin ordre hem de provar de resoldre les clàusules de programa amb el literal seleccionat? En Prolog es trien en l'ordre en el qual apareixen llistades en el programa. Sigui el programa lògic següent amb les clàusules llistades en l'ordre donat:

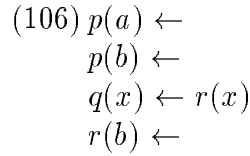
(104) $p(b) \leftarrow$
 $p(a) \leftarrow$
 $q(x) \leftarrow r(x)$
 $r(b) \leftarrow$

L'arbre SLD de l'objectiu $\leftarrow p(x), q(x)$ respecte de (104) que reflecteix l'ordre

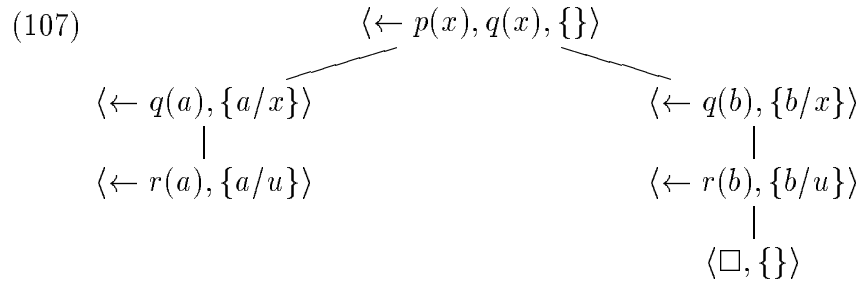
de les clàusules és el següent:



Prolog fa un recorregut per profunditat d'esquerra a dreta d'aquest arbre SLD: troba la resposta calculada $\{b/x\}$ a l'esquerra; si es demanen més solucions fa 'backtracking', tornant en el camí considerat desfent substitutions fins a trobar un node amb una branca a la dreta, i reintentant per aquesta branca, triant sempre el camí més a l'esquerra. Així segueix la branca a la dreta, que és una derivació SLD amb falla, i llavors no es donen més solucions. Considerem ara que passa quan reordenem les clàusules de (104):



L'arbre SLD per a l'objectiu $\leftarrow p(x), q(x)$ respecte del programa lògic (106) per a aquest ordre de clàusules és el següent:



Aquí Prolog fallarà a la branca a l'esquerra, farà backtracking i aleshores trobarà la refutació SLD a la branca a la dreta, retornant la resposta calculada $\{b/x\}$.