Butlletí de la Societat Catalana de Matemàtiques
Vol. 15, núm. 1, 2000. Pàg. 55–70

Demostració automàtica*

Juan Carlos Martínez

1 Introducció

En la demostració automàtica es planteja el problema de trobar, en un cert llenguatge formal, mètodes mecànics de demostració de teoremes que puguin dur-se a la pràctica. El llenguatge formal que es faci servir ha de complir dos requisits bàsics:

(a) Ha de tenir una capacitat expressiva prou àmplia per poder representar problemes formals.
(b) Ha de tenir la propietat que, a través d'ell, sigui possible desenvolupar mecanismes d'inferència automàtica que puguin ser implementats en un ordinador.

En els anys seixanta, s'observà la utilitat dels llenguatges de predicats per al disseny de mètodes eficients de demostració automàtica. Robinson, l'any 1965, introduí, a través d'aquests llenguatges, l'anomenat *mètode de resolució*, que és un refinament del mètode que havia desenvolupat Herbrand l'any 1930. La resolució, tanmateix, va resultar més eficient que els mètodes de demostració automàtica que s'havien desenvolupat fins aquell moment. Actualment, és un mètode bàsic per demostrar teoremes, que s'utilitza en problemes d'Intel·ligència Artificial.

En el camp de la Informàtica, hi ha un bon nombre d'exemples de problemes computacionals, que poden ser tractats mitjançant el mètode de Resolució. Per exemple, els problemes d'anàlisi i síntesi de programes dins la programació convencional. En el cas de l'anàlisi de programes, podem descriure l'execució d'un programa per mitjà d'una fórmula α d'un llenguatge de predicats, i també es pot descriure la condició que el programa acabi amb una altra fórmula β. Aleshores es tracta de demostrar que la fórmula β es dedueix de la fórmula α, per poder verificar que el programa acabarà la seva execució. I, en el cas de la síntesi de programes, donat un conjunt de fórmules de predicats que descriuen un problema, es pot utilitzar la resolució per determinar si existeixen solucions al problema i, en cas que

∗ El contingut d'aquest treball correspon a la conferència inaugural del curs 98–99 de la Facultat de Matemàtiques de la Universitat de Barcelona.

existeixin, per poder dissenyar un diagrama de flux, per mitjà del qual es puguin trobar aquestes solucions.

Un altre exemple és l'anomenat problema de l'isomorfisme de grafs, que consisteix a saber si dos grafs (dirigits) són isomorfs o, més en general, si un graf és isomorf a un subgraf d'un altre graf. Aquest problema té interès en la pràctica, a causa del fet que una fórmula química que representa un compost orgànic es pot representar mitjançant un graf. Per consegüent, el problema de saber si dues fórmules químiques corresponen al mateix compost orgànic és un problema d'isomorfisme de grafs. En aquest cas, podem representar els grafs en qüestió per mitjà de fórmules d'un llenguatge de predicats i, aleshores, el problema es redueix a demostrar que la fórmula que descriu un dels grafs es dedueix de la fórmula que descriu l'altre graf.

D'altra banda, una de les aplicacions de la demostració automàtica més rellevants, actualment, s'ha produït en la programació declarativa, atès que, a través d'un refinament del mètode de resolució, s'ha aconseguit desenvolupar un llenguatge de programació, el PROLOG , amb un nivell de programació superior al dels llenguatges convencionals, que disposa d'un mecanisme d'execució de raonament automàtic.

L'objectiu d'aquest treball és presentar el mètode de resolució i mostrar com s'utilitza en el mecanisme d'execució del PROLOG. Mostrarem aleshores, a la secció 1 i 2, les nocions de lògica que calen per poder introduir el tema de la resolució. A la secció 3, mostrarem el mètode general de resolució. A la secció 4, estudiarem el model computacional del llenguatge PROLOG. I a la secció 5, veurem, com a aplicació, un programa escrit en PROLOG que serveix per colorejar un mapa amb quatre colors, de manera que no hi hagi mai dos països veïns als quals se'ls assigni el mateix color.

2 Preliminars

En aquesta secció, estudiarem alguns dels conceptes fonamentals sobre llenguatges de predicats.

1 DEFINICIÓ *Anomenarem* alfabet dels símbols lògics *el conjunt format pels tipus de símbols següents:*

(a) Una quantitat infinita i numerable de variables $v_1, \ldots, v_n, \ldots$.
(b) Els connectius lògics $\neg$, $\vee$, $\wedge$, $\rightarrow$ *i* $\leftrightarrow$.
(c) Els quantificadors $\exists$ *i* $\forall$.
(d) Els símbols auxiliars (,) *i* , .

2 DEFINICIÓ *Un* vocabulari *és un conjunt format pels següents tipus de símbols:*

(a) Símbols de constant.
(b) Símbols de funció, cada un amb una certa arietat associada.
(c) Símbols de predicat, cada un amb una certa arietat associada.

Denotarem els símbols de predicat per $P, Q, R, \ldots$; els símbols de funció per $f, g, h, \ldots$; els símbols de constant per $a, b, c, d, \ldots$; i les variables per $u, v, x, y, z, \ldots$ En el cas de símbols de funció o de predicat, denotarem per un superíndex la seva

arietat. És a dir, escriurem f^n (R^n), si f (R) és un símbol de funció (de predicat) de n arguments.

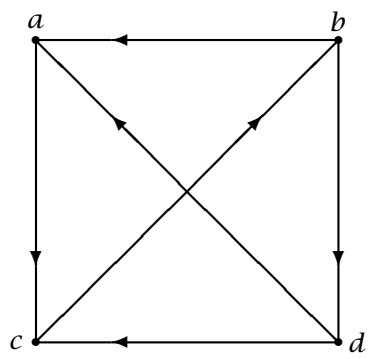
Els dos conceptes bàsics següents s'utilitzaran durant tot el treball.

3 DEFINICIÓ *Sigui τ un vocabulari. Definim el conjunt dels* termes $T(\tau)$ *per les regles següents:*

(1) $x \in T(\tau)$ *per a tota variable* x.
(2) $c \in T(\tau)$ *per a tot símbol de constant* $c \in \tau$.
(3) Si $f^n \in \tau$ *i* $t_1, \ldots, t_n \in T(\tau)$, *aleshores* $f(t_1, \ldots, t_n) \in T(\tau)$.

4 DEFINICIÓ *Sigui τ un vocabulari. Anomenarem* àtom *una tira de símbols* $R(t_1, \ldots, t_n)$, *on* $R^n \in \tau$ *i* $t_1, \ldots, t_n \in T(\tau)$.

Aleshores ja podem descriure un graf dirigit per una conjunció d'àtoms, representant els elements del graf per mitjà de constants i considerant un símbol de predicat P de dos arguments. Per exemple, el graf



el podem representar per mitjà de l'expressió:

$$P(a,c) \wedge P(b,a) \wedge P(b,d) \wedge P(d,c) \wedge P(d,a) \wedge P(c,b).$$

En general, necessitem no obstant fórmules més complexes per representar problemes formals. En el mètode de resolució, les fórmules que s'utilitzen són les *sentències d'un llenguatge de predicats*. Introduïm aquest concepte informalment. Per a una definició formal, remetem el lector a [3].

5 DEFINICIÓ *Una* sentència *és una tira de símbols que agrupa un conjunt finit i no buit d'àtoms, mitjançant els connectors lògics de* $\neg$, $\vee$, $\wedge$, $\rightarrow$ *i* $\leftrightarrow$, *de manera que qualsevol intervenció de qualsevol variable estigui afectada per algun dels quantificadors.*

Si ϕ és una sentència que agrupa àtoms respecte d'un vocabulari τ, direm que ϕ és una τ-sentència.

Considerem, per exemple, el vocabulari $\tau = \{ s^1, f^1, P^1, R^2 \}$. Aleshores, la tira de símbols

$$\forall x(P(x) \rightarrow \exists y(P(y) \wedge R(s(x), y) \wedge R(y, s(f(x)))))$$

és una τ-sentència.

Per poder interpretar i avaluar les sentències d'un llenguatge de predicats, introduïm el concepte següent:

6 DEFINICIÓ *Sigui τ un vocabulari. Una* τ-estructura *és un parell* (D, I) *on* D *és un conjunt no buit, que anomenarem* domini *de l'estructura, i* I *és una aplicació de domini* τ *que satisfà les condicions següents:*

(1) Si $c \in \tau$, $I(c) \in D$.

(2) Si $f^n \in \tau$, $I(f)$ *és una funció de* n *arguments sobre* D.

(3) Si $R^n \in \tau$, $I(R)$ *és una relació de* n *arguments sobre* D.

Aleshores, per avaluar el valor de veritat d'una sentència en una estructura, substituïm els símbols de constant, de funció i de predicat per les interpretacions corresponents i agafem com a domini dels quantificadors el domini de l'estructura.

Denotem per V el valor vertader i per F el valor fals. Si H és una τ-estructura i ϕ és una τ-sentència, denotem per $H(\phi)$ el resultat d'avaluar ϕ en H. Per exemple, considerem la sentència

$$\phi = \forall x(P(x) \to \exists y(P(y) \wedge R(s(x), y) \wedge R(y, s(f(x)))))$$

i l'estructura $H = (\mathbb{N}, I)$ on $I(P) = \{\, m \in \mathbb{N} \colon m \text{ és primer } \}$, $I(R) = \{\, (m, n) \in \mathbb{N}^2 \colon m \leq n \,\}$, $I(s)$ és la funció successor sobre els nombres naturals i $I(f)$ és la funció factorial. Aleshores, la interpretació de ϕ en H és la condició que expressa que «per a tot primer p existeix un primer q tal que $p < q < p! + 2$». Com és sabut que aquesta condició és certa, tenim que $H(\phi) = V$.

7 DEFINICIÓ *Sigui τ un vocabulari. Siguin $\phi_1, \ldots, \phi_n, \phi$ τ-sentències. Diem que ϕ és una* conseqüència lògica *de $\phi_1, \ldots, \phi_n$ si, per a tota τ-estructura H tal que $H(\phi_1) = V, \ldots, H(\phi_n) = V$, tenim que $H(\phi) = V$.*

Escriurem $\{\, \phi_1, \ldots, \phi_n \,\} \vDash \phi$ per expressar que ϕ és conseqüència lògica de $\phi_1, \ldots, \phi_n$.

8 DEFINICIÓ *Una τ-sentència ϕ és* insatisfactible *si per a tota τ-estructura H, $H(\phi) = F$.*

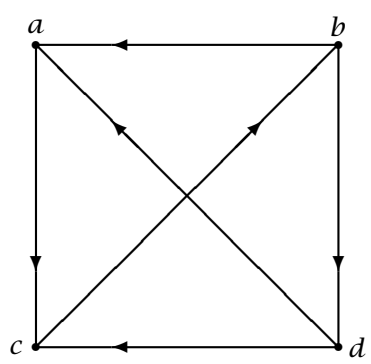
Observeu que, si $\phi_1, \ldots, \phi_n, \phi$ són sentències, les dues condicions següents són equivalents:

(1) $\{\, \phi_1, \ldots, \phi_n \,\} \vDash \phi$.

(2) $\phi_1 \wedge \ldots \wedge \phi_n \wedge \neg\phi$ és insatisfactible.

Aleshores, el que es vol aconseguir en la resolució és determinar si una sentència és conseqüència d'un conjunt finit de sentències. I per demostrar-ho es procedeix per refutació, és a dir, s'intenta demostrar que és insatisfactible la conjunció de les hipótesis del teorema amb la negació de la tesi.

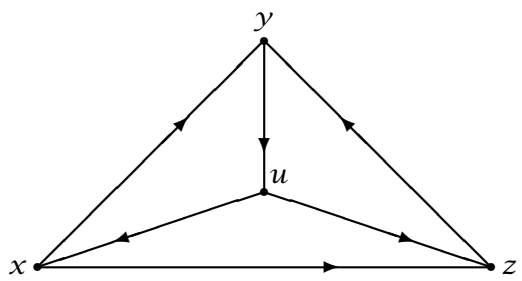
Considerem ara el graf G_1 que hem donat anteriorment:



que representem, considerant els elements com a constants, per mitjà de la sentència

$$\phi_1 = P(a,c) \wedge P(b,a) \wedge P(b,d) \wedge P(d,c) \wedge P(d,a) \wedge P(c,b).$$

Aleshores, si volem demostrar que aquest graf és isomorf al graf G_2 següent:



denotem els elements de G_2 per mitjà de variables, i descrivim G_2 mitjançant la sentència

$$\phi_2 = \exists x \exists y \exists z \exists u (P(x,y) \wedge P(y,u) \wedge P(u,x) \wedge P(u,z) \wedge P(z,y) \wedge P(x,z)).$$

Aleshores, la sentència $(\phi_1 \rightarrow \phi_2)$ expressa que G_1 i G_2 són isomorfs. Veurem a la secció 3 com és possible provar aquest fet per resolució, demostrant que la sentència $\phi_1 \wedge \neg\phi_2$ és insatisfactible.

3 Formas de Skolem

En el mètode de resolució, s'utilitza una caracterització de Skolem, que simplifica notablement el problema de determinar si una sentència és insatisfactible. Per introduir aquesta caracterització, necessitem algunes nocions prèvies.

9 DEFINICIÓ *(a) Anomenarem* literal *un àtom o la negació d'un àtom.*
(b) Una cláusula *és una disjunció (possiblement buida) de literals.*
(c) Si $\phi = \phi_1 \vee \ldots \vee \phi_n$ és una clàusula, definim el conjunt de membres de ϕ per $Mb(\phi) = \{\phi_1, \ldots, \phi_n\}$.

10 DEFINICIÓ *Una sentència ϕ està en* forma prenexa*, si ϕ és de la forma $Q_1x_1 \ldots Q_nx_n\psi$, on $Q_1, \ldots, Q_n \in \{\exists, \forall\}$ i ψ és una conjunció de clàusules. Direm aleshores que $Q_1x_1 \ldots Q_nx_n$ és el* prefix *de ϕ.*

El resultat següent expressa que qualsevol sentència té una forma prenexa.

11 TEOREMA *Per a tota τ-sentència ϕ existeix una τ-sentència ϕ' en forma prenexa, tal que, per a tota τ-estructura H, $H(\phi \leftrightarrow \phi') = V$.*

Per demostrar el teorema 11, podem seguir l'algorisme següent:

(1) Aplicant les equivalències lògiques corresponents, s'eliminen els símbols d'implicació i d'equivalència en la fórmula ϕ.
(2) Aplicant novament equivalències lògiques, es porten els símbols de negació davant dels àtoms.
(3) Canviant el nom de les variables que calgui, es porten els quantificadors a l'esquerra de la fórmula.
(4) Novament, mitjançant equivalències lògiques, es transforma la fórmula sense quantificadors que queda en una conjunció de clàusules.

El lector pot consultar [3] per a una prova detallada del teorema 11.

12 DEFINICIÓ *Una sentència ϕ està en* forma de Skolem*, si ϕ és de la forma $\forall x_1 \dots \forall x_n \psi$, on ψ és una conjunció de clàusules.*

13 TEOREMA *Per a tota τ-sentència ϕ existeix una τ'-sentència ϕ' en forma de Skolem amb $\tau' \supseteq \tau$ tal que:*

$$\phi \text{ insatisfactible} \iff \phi' \text{ insatisfactible.}$$

A continuació, posem de manifest un algorisme que permet construir per a qualsevol sentència ϕ una sentència ϕ' que satisfà la condició del teorema 13. Pel teorema 11, podem suposar que las sentències amb què treballem estan en forma prenexa. Aleshores, a cada sentència ϕ en forma prenexa, li assignem una sentència en forma de Skolem ϕ^{sk} per inducció sobre $k(\phi)$ = nombre d'intervencions del quantificador existencial en el prefix de ϕ. Si $k(\phi) = 0$, posem $\phi^{sk} = \phi$. Suposem que $k(\phi) > 0$. Tenim que ϕ és de la forma $\forall x_1 \dots \forall x_n \exists y Q_1 z_1 \dots Q_m z_m \psi$ amb ψ sense quantificadors. Ara definim un terme t de la manera següent. Si $n = 0$, definim $t = c$, on c és un símbol de constant nou. I, si $n > 0$, definim $t = f(x_1, \dots, x_n)$ on f és un símbol de funció de n arguments nou. Sigui ψ' l'expressió que resulta de substituir en ψ tota intervenció de la variable y pel terme t. Sigui $\chi = \forall x_1 \dots \forall x_n Q_1 z_1 \dots Q_m z_m \psi'$. Aleshores, per inducció, definim $\phi^{sk} = \chi^{sk}$.

Ara, és possible comprovar que ϕ és insatisfactible si i només si ϕ^{sk} és insatisfactible.

14 DEFINICIÓ *A la sentència ϕ^{sk} se la denomina* forma de Skolem *de ϕ.*

Per exemple, una forma de Skolem de la sentència

$$\forall x \exists y \exists z((\neg P(x,y) \wedge Q(x,z) \wedge S(y)) \vee R(x,y,z))$$

és la sentència

$$\forall x((\neg P(x, f(x)) \vee R(x, f(x), g(x))) \wedge (Q(x, g(x)) \vee R(x, f(x), g(x))) \wedge (S(f(x)) \vee R(x, f(x), g(x)))).$$

4 Resolució

Denotem per $\square$ la clàusula buida. En el mètode de resolució, es tracta d'inferir $\square$ a partir d'un conjunt finit de clàusules, la insatisfactibilitat del qual es vol demostrar. El concepte clau en el mètode de resolució és el de resolvent de dues clàusules. Prèviament, introduïm aquest concepte en el context de les clàusules sense variables, per fer-ho més entenedor.

15 DEFINICIÓ *Per a tot literal ψ definim*

$$\sim \psi = \begin{cases} \neg\psi, \text{ si } \psi \text{ és un àtom.} \\ \phi, \text{ si } \psi = \neg\phi \text{ per a algun àtom } \phi. \end{cases}$$

16 DEFINICIÓ *Siguin ϕ_1, ϕ_2, ϕ clàusules sense variables. Direm que ϕ és un* resolvent bàsic *de ϕ_1, ϕ_2, si existeix un literal ψ tal que $\psi \in Mb(\phi_1)$, $\sim\psi \in Mb(\phi_2)$ i*

$$Mb(\phi) = (\, Mb(\phi_1) \setminus \{\, \psi \,\}) \cup (\, Mb(\phi_2) \setminus \{\, \sim\psi \,\}).$$

Per exemple, un resolvent de les clàusules $P(a) \vee Q(b)$ i $\neg P(a) \vee \neg R(b,c)$ és la clàusula $Q(b) \vee \neg R(b,c)$.

Ara, en considerar clàusules en el context general, el primer que necessitem, per definir la noció de resolvent, és establir un criteri que ens permeti aparellar àtoms. Per exemple, podem aparellar una variable x amb una constant a i així obtenim, com a resolvent de $P(x) \vee Q(b)$ i $\neg P(a) \vee \neg R(b,c)$, la clàusula $Q(b) \vee \neg R(b,c)$. Tanmateix, no és possible computar cap resolvent de les clàusules $P(x) \vee Q(b)$ i $\neg P(f(x)) \vee \neg R(b,c)$, perquè la variable x intervé en el terme $f(x)$.

Aleshores, el criteri que busquem ve donat per l'anomenat *algorisme d'unificació*. En primer lloc, hem de definir què s'entén per conjunt unificable.

17 DEFINICIÓ *(a) Una* substitució *és un conjunt finit de parelles $\{\, t_1/x_1, \ldots, t_n/x_n \,\}$, on $t_1, \ldots, t_n$ són termes i $x_1, \ldots, x_n$ són variables diferents dos a dos.*

(b) Una substitució $\{\, t_1/x_1, \ldots, t_n/x_n \,\}$ és bàsica*, si $t_1, \ldots, t_n$ són variables.*

(c) Si ϕ és un àtom (o, en general, una combinació booleana d'àtoms) i $\lambda = \{\, t_1/x_1, \ldots, t_n/x_n \,\}$ és una substitució, denotem per $\phi\lambda$ la tira de símbols que resulta de reemplaçar en ϕ tota intervenció de x_i per t_i ($i = 1, \ldots, n$). Aleshores direm que $\phi\lambda$ és una realització *de ϕ.*

(d) Si W és un conjunt de literals i λ és una substitució, escrivim $W\lambda = \{\, w\lambda : w \in W \,\}$. Aleshores es diu que λ és un unificador *de W, si $|W\lambda| = 1$. Es diu que W és* unificable*, si té algun unificador.*

18 DEFINICIÓ *Descrivim informalment* l'algorisme d'unificació. *Rep com a entrada un conjunt W d'àtoms, finit i amb almenys dos elements, i determina si W és unificable. A més, si W és unificable, l'algorisme dóna com a sortida un unificador de W. Per això, en cada pas de còmput, l'algorisme tracta d'aparellar una variable x d'un dels àtoms amb un terme t d'un altre àtom de manera que x, t estiguin alineats en els àtoms escollits i x no intervingui en t. Si es pot trobar aquesta variable x i aquest terme t, se substitueix en cada àtom tota intervenció de x per t.*

Per exemple, considerem el conjunt

$$\{ P(c, x, f(g(y))), P(z, f(z), f(u)) \}.$$

Tenim que, en aquest conjunt, c està alineat amb z, x està alineat amb $f(z)$ i $g(y)$ està alineat amb u. Llavors, podem aparellar c amb z, obtenint el conjunt d'àtoms

$$\{ P(c, x, f(g(y))), P(c, f(c), f(u)) \}.$$

En aquest nou conjunt, x està alineat amb $f(c)$ i $g(y)$ està alineat amb u. Aleshores, aparellem x amb $f(c)$, així obtenim el conjunt

$$\{ P(c, f(c), f(g(y))), P(c, f(c), f(u)) \}.$$

Ara, aparellant $g(y)$ amb u, arribem al conjunt unitari

$$\{ P(c, f(c), f(g(y))) \}.$$

Per tant, s'obté com a sortida l'unificador $\{ c/z, f(c)/x, g(y)/u \}$.

Per a una descripció rigurosa de l'algorisme d'unificació, remetem el lector a [2].

19 DEFINICIÓ *Siguin ϕ_1, ϕ_2, ϕ clàusules. Diem que ϕ és un* resolvent *de ϕ_1, ϕ_2, si es compleixen les tres condicions següents:*

(1) Existeixen substitucions bàsiques ξ, η, tals que $\phi_1\xi$ i $\phi_2\eta$ són clàusules sense variables en comú.

(2) Existeixen $L \subseteq Mb(\phi_1), M \subseteq Mb(\phi_2)$, amb $L, M \neq \emptyset$, tals que el conjunt N d'àtoms que intervenen en $L\xi \cup M\eta$ és unificable.

(3) Per a algun unificador σ de N obtingut mitjançant l'algorisme d'unificació, es compleix:

(a) $L\xi\sigma = \{ \chi \}$, $M\eta\sigma = \{ \sim \chi \}$ per a algun literal χ.

(b) $Mb(\phi) = (Mb(\phi_1) \setminus L)\xi\sigma \cup (Mb(\phi_2) \setminus M)\eta\sigma$.

Per exemple, considerem les clàusules $\phi_1 = P(x) \vee Q(f(x)) \vee Q(z)$ i $\phi_2 = R(x, y) \vee \neg Q(x)$. Prenem $\xi = \{ u/x \}$, $\eta = \emptyset$. D'aquesta manera, donem un nom nou a la variable x de ϕ_1, per tal que no se sobreposi amb la variable x de ϕ_2. Ara, prenem $L = \{ Q(f(x)), Q(z) \}$ i $M = \{ \neg Q(x) \}$. Tenim $N = \{ Q(f(u)), Q(z), Q(x) \}$, que és unificable per $\{ f(u)/z, f(u)/x \}$. Per tant, $P(u) \vee R(f(u), y)$ és un resolvent de ϕ_1, ϕ_2.

20 DEFINICIÓ *Sigui Φ un conjunt de clàusules.*

(a) Definim $R(\Phi)$ com el conjunt
$\{ \phi :$ *existeixen $\phi_1, \phi_2 \in \Phi$ tals que ϕ és un resolvent de ϕ_1, ϕ_2 $\}$.*

(b) Definim $R^0(\Phi) = \Phi$, $R^{n+1}(\Phi) = R(R^n(\Phi))$.

El resultat següent és el teorema central de resolució .

21 TEOREMA *Siguin* $\chi_1, \ldots, \chi_n, \chi$ *sentències. Sigui*

$$\forall x_1 \ldots \forall x_m (\phi_1 \wedge \ldots \wedge \phi_k)$$

una forma de Skolem de $\chi_1 \wedge \ldots \wedge \chi_n \wedge \neg\chi$. *Sigui* $\Phi = \{\, \phi_1, \ldots, \phi_k \,\}$. *Aleshores, són equivalents:*

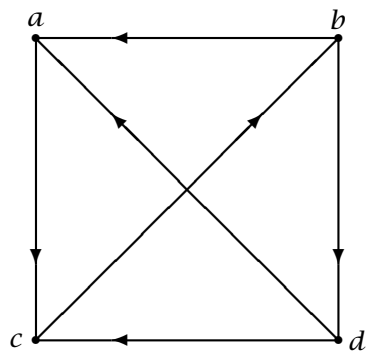
(1) $\{\, \chi_1, \ldots, \chi_n \,\} \models \chi$.
(2) $\square \in R^n(\Phi)$ *per a algun* $n \geq 0$.

El teorema 21 dóna lloc a un procediment automàtic de demostració per saber si una sentència és conseqüència d'un conjunt finit de sentències, en cas que ho sigui. El procediment consisteix a anar computant els conjunts $R^n(\Phi)$, i comprovar si en algun pas s'obté la clàusula buida. Observeu que si el teorema fos fals, és a dir, si no es cumplís la condició (1), el procediment de demostració no tindria final. D'altra banda, com a conseqüència del teorema d'indecidibilitat de Church, sabem que no és possible desenvolupar un mètode de demostració automàtica que s'aturi sempre i determini si el teorema en qüestió és vertader o fals.

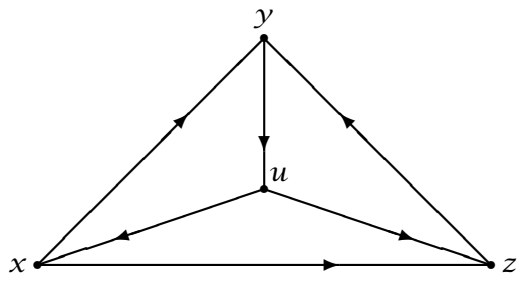
El lector interessat podrà trobar una demostració del teorema 21 a [2].

Quan el mètode de resolució es duu a la pràctica, se sol incorporar-hi diverses restriccions a fi de fer-lo eficient. En el cas del problema de l'isomorfisme de grafs, quan computem un resolvent de dues clàusules, s'ha de complir que una d'elles sigui un àtom i s'ha d'elegir en l'altra clàusula el primer literal, és a dir, el que estigui més a l'esquerra. Considerem els dos grafs que hem posat de manifest anteriorment:

G_1 :



G_2 :



Considerem les sentències:

$$\phi_1 = P(a,c) \wedge P(b,a) \wedge P(b,d) \wedge P(d,c) \wedge P(d,a) \wedge P(c,b),$$

$$\phi_2 = \exists x \exists y \exists z \exists u (P(x,y) \wedge P(y,u) \wedge P(u,x) \wedge P(u,z) \wedge P(z,y) \wedge P(x,z)).$$

Llavors, per provar que G_1 i G_2 són isomorfs, hem de demostrar que ϕ_2 és conseqüència lògica de ϕ_1. I per això, demostrem per resolució que $\phi_1 \wedge \neg\phi_2$ és insatisfactible. Obtenim aleshores les premisses següents:

1. $P(a,c)$.
2. $P(b,a)$.
3. $P(b,d)$.
4. $P(d,c)$.
5. $P(d,a)$.
6. $P(c,b)$.
7. $\neg P(x,y) \vee \neg P(y,u) \vee \neg P(u,x) \vee \neg P(u,z) \vee \neg P(z,y) \vee \neg P(x,z)$.

Ara, resolem 4 amb 7 (elegint el primer literal de 7), i així obtenim la substitució $\{d/x, c/y\}$, i obtenim com a resolvent:

8. $\neg P(c,u) \vee \neg P(u,d) \vee \neg P(u,z) \vee \neg P(z,c) \vee \neg P(d,z)$.

A continuació, resolem 8 amb 6, i així traiem l'aparellament b/u, i obtenim com a resolvent:

9. $\neg P(b,d) \vee \neg P(b,z) \vee \neg P(z,c) \vee \neg P(d,z)$.

Ara, resolem 9 amb 3, i obtenim:

10. $\neg P(b,z) \vee \neg P(z,c) \vee \neg P(d,z)$.

Aleshores, elegim 2 i 10, obtenint l'aparellament a/z i el resolvent:

11. $\neg P(a,c) \vee \neg P(d,a)$.

A continuació, resolem 1 amb 11, derivant:

12. $\neg P(d,a)$.

I ara, resolent 5 amb 12, arribem a $\square$.

També, és interessant observar com la substitució $\{d/x, c/y, b/u, a/z\}$ que hem obtingut en aplicar l'algorisme d'unificació, és un isomorfisme entre els dos grafs. El refinament del mètode de resolució, que es fa servir en el problema de l'isomorfisme de grafs, és l'anomenat *mètode de la V-resolució*. En general, quan aquest mètode s'utilitza en el cas del problema de l'isomorfisme de grafs, l'ordinador no tan sols ens indicaria que els dos grafs són isomorfs sinó que, a més, ens donaria com a sortida un isomorfisme entre ambdós grafs. Un tractament detallat d'aquest mètode es troba a [2].

Per a més exemples d'aplicació de la resolució, remetem el lector a [2], [5], [6] i [7].

5 El llenguatge PROLOG

El nostre propòsit en aquesta secció és descriure, a través d'un refinament del mètode general de resolució, presentat a la secció anterior, el mecanisme d'execució del llenguatge PROLOG. En primer lloc, definim les clàusules que s'utilitzen en aquest llenguatge de programació.

22 DEFINICIÓ *Una* regla *és una clàusula de la forma* $\phi \vee \neg\phi_1 \vee \ldots \vee \neg\phi_n$*, on* $n \geq 0$ *i* $\phi, \phi_1, \ldots, \phi_n$ *són àtoms.*

A una regla $\phi \vee \neg\phi_1 \vee \ldots \vee \neg\phi_n$ *se la denota per l'expressió equivalent* $\phi \leftarrow \phi_1 \wedge \ldots \wedge \phi_n$*. Aleshores hom diu que* $\phi_1 \wedge \ldots \wedge \phi_n$ *és l'*antecedent *de la regla, i* ϕ *el* conseqüent.

23 DEFINICIÓ *Un* programa lògic *és una seqüència finita de regles.*

Intuïtivament, parlant en termes de programació convencional, si $R = \phi \leftarrow \phi_1 \wedge \ldots \wedge \phi_n$ és una regla d'un programa lógic, podem pensar R com una funció, l'encapçalament de la qual és ϕ i el cos d'instruccions $\phi_1, \ldots, \phi_n$.

24 DEFINICIÓ *Un* objectiu *és una expressió de la forma* $\phi_1 \wedge \ldots \wedge \phi_n$*, on* $n \geq 1$ *i* $\phi_1, \ldots, \phi_n$ *són àtoms.*

Quan es té un programa lògic, es vol recaptar informació del programa, escrivint per a aconseguir-ho un objectiu.

Si $\phi = \phi_1 \wedge \ldots \wedge \phi_n$ és un objectiu, escriurem $\sim \phi = \neg\phi_1 \vee \ldots \vee \neg\phi_n$. Observeu que $\sim \phi$ és una expressió equivalent a $\neg\phi$.

Per establir el significat d'un programa i d'un objectiu, se suposa que les variables, que intervenen en una clàusula d'un programa, estan quantificades universalment (és a dir, per $\forall$), i les variables que intervenen en un objectiu, ho estan existencialment (és a dir, per $\exists$). Escriurem $\phi(x_1, \ldots, x_n)$ per expressar que $x_1, \ldots, x_n$ són les variables que intervenen en ϕ. Aleshores, si $P = \langle \phi_1(\overline{x}_1), \ldots, \phi_n(\overline{x}_n) \rangle$ és un programa lògic i $\psi(\overline{y})$ és un objectiu, es tracta de demostrar que

$$\{ \forall\overline{x}_1\phi_1, \ldots, \forall\overline{x}_n\phi_n \} \models \exists\overline{y}\psi.$$

I per aconseguir-ho, es demostra que és insatisfactible la sentència que expressa la negació del teorema, és a dir:

$$\forall\overline{x}_1\phi_1 \wedge \ldots \wedge \forall\overline{x}_n\phi_n \wedge \forall\overline{y}\neg\psi.$$

25 DEFINICIÓ *Descrivim l'*interpretador per a programes lògics. *Rep com a entrades un programa lògic* P *i un objectiu* χ*, i dóna com a sortida una realització de* χ *si el còmput té èxit, i dóna «fallida» en cas contrari. Utilitzarem una variable* RS *on guardarem el resolvent que obtinguem en cada pas de còmput, i una variable* VS *(variable de sortida) on tindrem, al final, el resultat que busquem en el cas en què el còmput tingui èxit. L'algorisme consta aleshores de les etapes següents:*

Etapa 1. *Assignar* $\sim\chi$ *a la variable* RS *i* χ *a la variable* VS*.*

Etapa 2. *Elegir un literal* $\neg\psi$ *de* RS *i una clàusula* $\phi = \psi' \leftarrow \psi'_1 \wedge \ldots \wedge \psi'_n$ *del programa* P*, amb les variables canviades de nom, de manera que el conjunt* $\{\psi, \psi'\}$ *sigui unificable. Si no existeixen tal literal i tal clàusula, passar a l'Etapa 5.*

Etapa 3. *Computar un resolvent de ϕ i RS agafant $L = \{\neg\psi\}$, $M = \{\psi'\}$ i $N = \{\psi, \psi'\}$, i assignar-lo aleshores a RS.*

Etapa 4. *Assignar a VS l'expressió que resulta d'aplicar σ a VS, on σ és l'unificador de $\{\psi, \psi'\}$, utilitzat en l'Etapa 3, i tornar aleshores a l'Etapa 2.*

Etapa 5. *Si RS = $\square$, donar sortida = VS. Si no, donar sortida «fallida».*

Observeu que en cada pas de còmput de l'interpretador es genera un resolvent de la forma $\neg\psi_1 \vee \ldots \vee \neg\psi_n$, on $\psi_1, \ldots, \psi_n$ són àtoms. En la pràctica, a fi de no haver d'arrossegar els símbols de negació, els interpretadors treballen amb la negació del resolvent, és a dir, amb l'expressió $\psi_1 \wedge \ldots \wedge \psi_n$, a la qual s'anomena objectiu actual*. En cada pas de còmput, l'objectiu actual és la fórmula que es vol demostrar. Aleshores, el que es fa en les Etapes 2 i 3 de l'interpretador equival a substituir en l'objectiu actual l'àtom ψ per l'antecedent de la clàusula ϕ i a aplicar aleshores a l'expressió resultant l'unificador de $\{\psi, \psi'\}$, que s'ha obtingut mitjançant l'algorisme d'unificació.*

En les diferents versions de PROLOG, el criteri per elegir un àtom de l'objectiu actual acostuma a ser el d'elegir el primer, és a dir, el que hi ha més a l'esquerra. I la simulació en PROLOG de l'elecció no determinista d'una clàusula en l'Etapa 2 de l'algorisme, es realitza mitjançant una recerca seqüencial i un retorn enrere, de la manera següent. L'interpretador elegeix la primera clàusula del programa (segons l'ordre d'aparició), el conseqüent de la qual s'unifiqui amb l'àtom elegit en l'objectiu actual. Si aquesta clàusula no existeix, l'interpretador consideraria l'objectiu de l'etapa anterior (és a dir, dóna una volta cap enrere en el còmput), elegiria de bell nou el primer àtom i a continuació escolliria, en el cas que existís, la següent clàusula del programa, el conseqüent de la qual s'unifiqués amb aquest àtom. El retorn enrere en el còmput el realitza també l'interpretador de PROLOG, quan hagi arribat a la clàusula buida. D'aquesta manera, intenta trobar totes les solucions possibles per a un objectiu que li hàgim subministrat.

Observeu que el mètode de resolució per a programes lògics que hem mostrat en l'interpretador, és un refinament del mètode general de resolució que hem mostrat a la secció 3. Per a aquest refinament, es pot demostrar un teorema de completesa semblant al teorema 21 (vegeu [4]).

En el context de la programació lògica, el símbol de conjunció «$\wedge$» es denota per una coma «,». Una substitució $\{t_1/x_1, \ldots, t_n/x_n\}$ es denota per mitjà del conjunt d'equacions $\{x_1 = t_1, \ldots, x_n = t_n\}$. En un programa en PROLOG, els identificadors que denoten constants han de començar per una lletra minúscula, i els identificadors que denoten variables per una de majúscula. Al final de cada clàusula d'un programa s'ha de posar un punt.

En el PROLOG, el tipus d'estructura de dades que s'utilitza preferentment són les llistes, a l'igual que en altres llenguatges d'Intel·ligència Artificial, com ara el LISP. Una llista és una seqüència ordenada d'elements del mateix tipus sense una longitud predeterminada. En PROLOG, els elements d'una llista es representen entre claudàtors i separats per comes. La llista buida es representa per [ ]. I una llista no buida es representa per un terme de dos arguments, separats per una barra vertical; el primer representa el primer element de la llista, i el segon la resta de la llista.

El programa següent determina si un element pertany a una llista:

$$\begin{array}{l}\mathsf{membre}(X,[X|L]).\\ \mathsf{membre}(X,[Y|L]) \leftarrow \mathsf{membre}(X,L).\end{array}$$

El programa es pot utilitzar no només per saber si un element es troba en una llista, sinó també per obtenir tots els elements d'una llista donada. Suposem, per exemple, que prenem com a objectiu:

$$\mathsf{membre}(X,[a,b,c]).$$

Aleshores el que desitgem és obtenir, en la variable de sortida, X els elements de la llista d'entrada $[a,b,c]$. Vegem com actua en aquest cas l'interpretador de PROLOG. En primer lloc, elegeix la primera clàusula del programa. Atès que aquesta clàusula no té antecedent, s'arriba a la clàusula buida $\square$, obtenint com a solució $X = a$. A continuació, l'interpretador fa una volta enrere en el còmput, elegint ara la segona clàusula del programa. S'obté com a objectiu actual l'àtom

$$\mathsf{membre}(X,[b,c]).$$

Novament, s'elegeix la primera clàusula, obtenint la solució $X = b$. Ara, l'interpretador fa una volta enrere, tornant a elegir l'objectiu

$$\mathsf{membre}(X,[b,c]).$$

Aleshores s'aparella aquest objectiu amb el conseqüent de la segona clàusula, obtenint com a objectiu actual

$$\mathsf{membre}(X,[c]).$$

Ara, s'aparella aquest àtom amb la primera clàusula, i s'arriba a la solució $X = c$.

6 El problema dels quatre colors

En aquesta secció, mostrem un programa en PROLOG per a colorejar un mapa amb quatre colors, de manera que no hi hagi dos països veïns amb el mateix color. Com és ben sabut, l'any 1976 es va resoldre la famosa conjectura segons la qual quatre colors són suficients per colorejar un mapa d'aquesta manera. El programa que veurem és el que hi ha en [8].

Prèviament, estudiarem el següent programa auxiliar:

$$\begin{array}{l}\mathsf{seleccionar}(X,[X|L],L).\\ \mathsf{seleccionar}(X,[Y|L],[Y|M]) \leftarrow \mathsf{seleccionar}(X,L,M).\end{array}$$

En aquest cas, l'interpretador va elegint els elements d'una llista que li donem com a entrada en el segon argument de l'objectiu, de manera que, quan selecciona un element de la llista, registra en el tercer argument de l'objectiu la llista resultant d'eliminar l'esmentat element. La idea és que quan l'interpretador elegeixi la primera clàusula, seleccionarà el primer element de la llista que, en aquell moment, tinguem en l'objectiu actual, i quan elegeixi la segona clàusula, aquest primer element es registrarà en el tercer argument del predicat. Aleshores, pel mètode del retorn enrere,

l'interpretador anirà elegint tots els elements de la llista que li subministrem, de la manera indicada.

Per exemple, posem com a objectiu:

$$\mathsf{seleccionar}(X, [a, b, c], L).$$

En primer lloc, s'elegeix per unificar la primera clàusula, amb la qual cosa aconseguim la clàusula buida, obtenint la solució $X = a, L = [b, c]$. A continuació, s'efectua una volta enrere, amb la qual cosa es torna a l'objectiu

$$\mathsf{seleccionar}(X, [a, b, c], L),$$

elegint-se ara per unificar la segona clàusula del programa amb les variables amb el nom canviat, per exemple els noms $X1, Y1, L1, M1$. S'obtenen aleshores els aparellaments $X = X1$, $Y1 = a$, $L1 = [b, c]$, $L = [a|M1]$. L'objectiu actual és ara l'àtom

$$\mathsf{seleccionar}(X1, [b, c], M1).$$

Es tria aleshores la primera clàusula, i s'obtenen els aparellaments $X1 = b$, $M1 = [c]$, amb la qual cosa aconseguim la solució $X = b$, $L = [a, c]$. A continuació té lloc un retorn enrere, i així es torna a considerar l'objectiu

$$\mathsf{seleccionar}(X1, [b, c], M1).$$

Ara, s'elegeix la segona clàusula del programa, amb les variables canviades de nom, posant, per exemple, $X2, Y2, L2, M2$. Així s'obtenen els aparellaments $X1 = X2$, $Y2 = b$, $L2 = [c]$, $M1 = [b|M2]$. Aleshores, s'aconsegueix, com a objectiu actual, l'àtom

$$\mathsf{seleccionar}(X2, [c], M2).$$

Novament, es tria per unificar la primera clàusula, obtenint així els aparellaments $X2 = c$, $M2 = [\,]$, i arribant per tant a la solució $X = c$, $L = [a, b]$. A continuació, es fa una altra vegada un retorn enrere en el còmput, i s'elegeix per unificar la segona clàusula del programa, obtenint d'aquesta manera l'àtom

$$\mathsf{seleccionar}(X3, [\,], M3)$$

com a objectiu actual. Atès que el segon argument d'aquest àtom és la llista buida, l'esmentat àtom no pot unificar ni amb la primera clàusula ni amb el conseqüent de la segona i, amb això, s'acaba aquí l'execució.

Tornant al problema dels quatre colors, representem un país com un terme de tres arguments de la forma:

país (⟨nom del país⟩, ⟨color del país⟩, ⟨llista de colors dels països veïns⟩).

Per exemple, si volguéssim colorejar el mapa d'Europa, escriuríem una llista de la forma:

[país(holanda, H,[B,A]), país(bèlgica,B,[F,H,L,A]),. . . . .]

on els noms dels països són constants i els identificadors en majúscules són variables que representen els països, i als quals se'ls anirà assignant colors, d'acord amb allò que s'especifiqui en les clàusules del programa.

Vegem aleshores les clàusules de què consta el programa:

```
colorejar_mapa ([País|Països],Colors) ← colorejar_país (País,Colors),
                                        colorejar_mapa (Països,Colors).
colorejar_mapa ([ ],Colors).
colorejar_país(país(Nom,Color,Veïns), Colors) ←
                                        seleccionar(Color,Colors,Reste),
                                        membres (Veïns,Reste).
seleccionar(X, [X|L], L).
seleccionar(X, [Y|L], [Y|M]) ← seleccionar(X, L, M).
membres([X|L], M) ← membre(X, M), membres(L, M).
membres([ ], M).
membre(X, [X|L]).
membre(X, [Y|L]) ← membre(X, L).
```

Amb aquestes clàusules, l'interpretador de PROLOG seria aleshores capaç de colorejar el mapa amb els quatre colors que li subministrem. Mitjançant les dues primeres clàusules, es coloreja una llista de països amb una llista de colors, colorejant cada país de la llista, fins que la llista de països quedi buida. Per mitjà de la tercera clàusula, s'elegeix un color per al país que s'estigui considerant, utilitzant el programa «seleccionar» i, a continuació, es comprova mitjançant el programa «membres» que no hi ha cap país veí que tingui el color seleccionat. Si hi hagués algun país veí amb aquest color, l'interpretador retornaria enrere en el còmput, i elegiria aleshores un altre color per al país considerat.

Ara, si volem que l'interpretador coloregi el mapa d'Europa, posaríem les clàusules:

```
mapa (europa, [país( holanda, H,[B,A]), país(bèlgica,B,[F,H,L,A]),. . . . . ]).

colors (europa, [vermell,verd,blanc,blau]).
```

Aleshores, per poder extreure la solució que ens doni l'interpretador, posaríem la clàusula:

```
colorejar (Nom,Mapa) ← mapa (Nom, Mapa),
                       colors (Nom, Colors),
                       colorejar_mapa (Mapa, Colors).
```

Referències

[1] BRATKO, I. PROLOG*: Programming for Artificial Intelligence* , Addison-Wesley, Massachusetts, 1990.

[2] CHANG, C. L., LEE, R. C. T. *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, Nova York, 1973.

[3] EBBINGHAUS, H. D., FLUM, J., THOMAS, W. *Mathematical Logic*, Springer-Verlag, Berlin, 1994.

[4] LLOYD, J. W. *Foundations of Logic Programming*, Springer-Verlag, Berlin, 1987.

[5] LOVELAND, D. W. *Automated Theorem Proving*, North-Holland, Amsterdam, 1978.

[6] MARTÍNEZ, J. C. *Fundamentos de Demostración Automática de Teoremas*, Mathematics Preprint Series No. 219, Universitat de Barcelona, 1996.

[7] SCHÖNING, U. *Logic for Computer Scientists*, Birkhäuser, Berlin, 1989.

[8] STERLING, L., SHAPIRO, E. *The Art of* PROLOG, MIT Press, Cambridge, 1986.

DEPARTAMENT DE LÒGICA
FACULTAT DE MATEMÀTIQUES
UNIVERSITAT DE BARCELONA
GRAN VIA DE LES CORTS CATALANES, 585
08007 BARCELONA
martinez@mat.ub.es