Elementos Lógicos da Programação I– Módulo III

Dedução Automática em Lógica Proposicional

1. Introdução.
2. Formas normais negativas.
3. Sistemas de "tableaux"
4. Método de Davis-Putnam

José Manuel E. Valença @ DI Universidade do Minho 1999-2000

Introdução

Os sistemas de dedução de Gentzen (dedução natural e cálculo de sequentes) constituem a forma mais flexível de exprimir a noção de dedução como um processo de transformação sintáctica. Os estados de prova e deduções são representados sintacticamente por árvores; a construção da prova pode eventualmente ser feita automaticamente mas, frequentemente, requer a intervenção do utilizador.

Estes sistemas de dedução foram apresentados para a Lógica Proposicional mas, como veremos, são facilmente extendidos a lógicas mais expressivas: 1^a ordem, temporais, etc.

A Lógica Proposicional admite uma automatização completa do processo de prova (como vimos quando apresentamos um algoritmo que determinava a validade de proposições). Por isso, no caso particular da Lógica Proposicional, faz sentido tentar encontrar algoritmos, tão eficientes quanto possível, que automatizem completamente o processo de construção de provas.

Assim ...

O objectivo deste módulo é o estudo de sistemas de dedução que sejam automatizáveis. Iremos referir

1. Métodos baseados na análise de formas normais.
2. Os sistemas de "tableaux"
3. O método de Davis-Putnam

Todos estes sistemas de dedução são justificados, essencialmente, por refutação. Genericamente os vários sistemas procuram determinar, dada uma teoria Δ, o conjunto dos modelos que refutam esta teoria e o conjunto de modelos que a validam.

Para tal, os algoritmos podem ser consideravelmente simplificados se as proposições forem previamente convertidas numa forma equivalente designada **forma normal negativa**.

Formas Normais Negativas (fnn's)

15 DEFINIÇÃO *Dado um conjunto dos símbolos proposicionais $\mathcal{P}$, uma* ***forma normal negativa*** *é uma proposição escrita com a seguinte sintaxe:*

$$\mathcal{F} \quad ::= \quad \mathcal{P} \mid \neg\mathcal{P} \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{F} \vee \mathcal{F}$$

Esta definição determina que as fnn's são símbolos proposicionais, ou negações de símbolos proposicionais, ou então conjunções ou disjunções de outras fnn's.

Se p e q forem símbolos proposicionais, então

$$\neg\mathsf{p} \vee \mathsf{q} \quad , \quad \mathsf{p} \vee (\neg\mathsf{p}) \wedge ((\neg\mathsf{p}) \vee \mathsf{q}) \quad , \quad (\neg\mathsf{p} \wedge \neg\mathsf{q}) \vee (\mathsf{p} \wedge \mathsf{q})$$

são exemplos de formas normais negativas equivalentes, respectivamente, a

$$\mathsf{p} \supset \mathsf{q} \quad , \quad ((\mathsf{p} \supset \mathsf{q}) \supset \mathsf{p}) \supset \mathsf{p} \quad , \quad (\mathsf{p} \supset \mathsf{q}) \wedge (\mathsf{q} \supset \mathsf{p})$$

Já as proposições seguintes **não** satisfazem a definição de fnn.

$$\mathsf{p} \supset \mathsf{q} \quad , \quad \neg(\mathsf{p} \vee \mathsf{q}) \quad , \quad P \wedge Q$$

A primeira, porque usa um conectivo (a implicação $\supset$) não previsto na definição de fnn. A segunda, porque a negação é aplicada a uma fórmula que não é um símbolo proposicional. Finalmente a terceira, porque os símbolos P e Q não são símbolos proposicionais mas são sim variáveis que podem ser instanciadas com qualquer proposição quer ela seja uma fnn ou não.

10 PROPOSIÇÃO *Para toda a proposição P existe uma forma normal negativa $\mathcal{N}(P)$ que lhe é semanticamente equivalente; isto é,*

$$P \models \mathcal{N}(P) \qquad e \qquad \mathcal{N}(P) \models P$$

A determinação da forma normal negativa $\mathcal{N}(P)$ é baseada nas seguintes equivalências.

$$\neg\neg P \equiv P \quad P \supset Q \equiv \neg P \vee Q \quad \neg(P \supset Q) \equiv P \wedge \neg Q$$
$$\neg(P \wedge Q) \equiv \neg P \vee \neg Q \qquad \neg(P \vee Q) \equiv \neg P \wedge \neg Q$$

Para não criar ambiguidades entre os conectivos antes da conversão e os conectivos após conversão é usual escolher, para estes últimos, um novo conjunto de símbolos. Assim usaremos, para os conectivos das formas normais negativos, a seguinte notação:

(i) A conjunção $(P \wedge Q)$ é representada por $(P\ ,\ Q)$.

(ii) A disjunção $(P \vee Q)$ é representada por $(P \mid Q)$.

Representando por $P \rightsquigarrow P'$ e $Q \rightsquigarrow Q'$ as conversões de P e Q, as equivalências acima sugerem as transformações a seguir indicadas

$\neg\neg P \rightsquigarrow P'$	
$P \wedge Q \rightsquigarrow P'\ ,\ Q'$	$\neg(P \wedge Q) \rightsquigarrow \neg P' \mid \neg Q'$
$P \vee Q \rightsquigarrow P' \mid Q'$	$\neg(P \vee Q) \rightsquigarrow \neg P'\ ,\ \neg Q'$
$(P \supset Q) \rightsquigarrow \neg P' \mid Q'$	$\neg(P \supset Q) \rightsquigarrow P'\ ,\ \neg Q'$

Os símbolos proposicionais p ou a negação de símbolos proposicionais $\neg\mathsf{p}$ (colectivamente designados por **literais**) não são alterados por esta transformação.

Exemplos

$$(\neg\neg\mathsf{p} \vee \mathsf{p}) \supset \mathsf{p} \rightsquigarrow \neg(\neg\neg\mathsf{p} \vee \mathsf{p}) \mid \mathsf{p} \rightsquigarrow (\neg\neg(\neg\mathsf{p})\ ,\ \neg\mathsf{p}) \mid \mathsf{p}$$
$$\rightsquigarrow (\neg\mathsf{p}\ ,\ \neg\mathsf{p}) \mid \mathsf{p}$$

$$((\mathsf{p} \supset \mathsf{q}) \supset \mathsf{p}) \supset \mathsf{p} \rightsquigarrow \neg((\mathsf{p} \supset \mathsf{q}) \supset \mathsf{p}) \mid \mathsf{p} \rightsquigarrow ((\mathsf{p} \supset \mathsf{q})\ ,\ \neg\mathsf{p}) \mid \mathsf{p}$$
$$\rightsquigarrow ((\neg\mathsf{p} \mid \mathsf{q})\ ,\ \neg\mathsf{p}) \mid \mathsf{p}$$

Observando a forma das regras de conversão apresentadas no acetato 3 torna-se evidente que, com excepção da regra da dupla negação, $\neg\neg P \rightsquigarrow P'$, todas as restantes regras produzem resultados de uma de duas formas:

* uma conjunção $(\alpha_1\ ,\ \alpha_2)$
* uma disjunção $(\beta_1 \mid \beta_2)$

Torna-se possível agrupar as regras de conversão nestas duas categorias e chamar ao primeiro grupo **regras** α e ao segundo grupo **regras** β. Uma terceira categoria são as regras que lidam com os literais e com a dupla negação.

Nesta perspectiva as diferentes regras podem ser apresentados em três quadros.

η	η
p	p
$\neg$p	$\neg$p
$\neg\neg P$	P

$\alpha_1\ ,\ \alpha_2$	α_1	α_2
$P \wedge Q$	P	Q
$\neg(P \vee Q)$	$\neg P$	$\neg Q$
$\neg(P \supset Q)$	P	$\neg Q$

$\beta_1 \mid \beta_2$	β_1	β_2
$\neg(P \wedge Q)$	$\neg P$	$\neg Q$
$P \vee Q$	P	Q
$P \supset Q$	$\neg P$	Q

Estes quadros transcrevem-se directamente num programa PROLOG.

```
fnn(  P       ,  N)         :- atom(P) , N = P.
fnn(-(P)      ,  N)         :- atom(P) , N = -(P).
fnn(-(-(P))   ,  N)         :- fnn(P,N).

fnn(  P & Q   , (A1 , A2)) :- fnn(  P , A1) , fnn(  Q , A2).
fnn(-(P v Q)  , (A1 , A2)) :- fnn(-(P), A1) , fnn(-(Q), A2).
fnn(-(P => Q), (A1 , A2)) :- fnn(  P , A1) , fnn(-(Q), A2).

fnn(-(P & Q) , (B1 | B2)) :- fnn(-(P), B1) , fnn(-(Q), B2).
fnn(  P v Q  , (B1 | B2)) :- fnn(  P , B1) , fnn(  Q , B2).
fnn(  P => Q , (B1 | B2)) :- fnn(-(P), B1) , fnn(  Q , B2).
```

(*continua*)

Exemplos de execução deste programa [a] para determinar, respectivamente, as fnn's de $(\neg\neg p \vee p) \supset p$ e de $((p \supset q) \supset p) \supset p$.

```
| ?- fnn((-(-(p)) v p) => p, FN).

FN = -(p) , -(p) | p ?

yes

| ?- fnn(((p => q) => p) => p, FN).

FN = (-(p) | q) , -(p) | p ?

yes
```

Uma vez obtida a forma normal negativa de uma proposição, o que fazer?

A resposta está na **simplificação**. Uma vez simplificada uma *fnn* de forma adequada é depois simples determinar se ela é válida ou se é refutada.

Exemplo

Considere-se a seguinte *fnn* equivalente ao exemplo $((\neg p \mid q) \ , \ \neg p \mid p)$.

$$(\neg p \mid q \mid p) \ , \ (\neg p \mid p)$$

Trata-se da conjunção de duas *fnn*'s que, por seu turno, são exclusivamente formadas por uma disjunção de literais: $(\neg p \mid q \mid p)$ e $(\neg p \mid p)$.

Ambas as disjunções contêm o símbolo p e a sua negação $\neg p$; por isso, ambas as disjunções serão proposições válidas.

Logo a conjunção destas duas proposições válidas é também válida.

[a] Note-se que a conjunção " , " tem sempre prioridade superior à disjunção " | " e ambas são operações associativas; por este facto pode-se prescindir de muitos parêntesis na representação de formas normais negativa.

Simplificação de formas normais negativas

É possível simplificar uma *fnn* de duas formas possíveis:

Formas conjuntivas Uma *forma conjuntiva* é uma *fnn* formada por uma única conjunção de disjunções de literais.

Uma forma conjuntiva equivalente a $((\neg p \mid q)\ ,\ \neg p \mid p)$ será

$$(\neg p \mid q \mid p)\ ,\ (\neg p \mid p)$$

Formas disjuntivas Uma *forma disjuntiva* é formada por uma única disjunção de proposições construídas por conjunções de literais.

Uma forma disjuntiva equivalente a $((\neg p \mid q)\ ,\ \neg p \mid p)$ será

$$(\neg p\ ,\ \neg p) \mid (q\ ,\ \neg p) \mid p$$

Cada forma conjuntiva (ou disjuntiva) pode ser representada por uma lista de listas dispensando, assim, o uso de qualquer conectivo.

Os dois exemplos anteriores são representados pelas seguintes listas

$$[\,[\neg p\ ,\ q\ ,\ p]\ ,\ [\neg p\ ,\ p]\,] \qquad [\,[\neg p\ ,\ \neg p]\ ,\ [q\ ,\ \neg p]\ ,\ [p]\,]$$

Temos assim duas abordagens possíveis à representação de proposições sem usar conectivos:

(i) conjunções de disjunções de literais, no caso das formas conjuntivas, ou

(ii) disjunções de conjunções de literais, no caso das formas disjuntivas.

Os conectivos desaparecem e através de simples algoritmos de manipulação de listas será possível, como veremos, averiguar se a proposição é válida ou se pode ser refutada.

A construção de formas normais conjuntivas (ou disjuntivas) a partir de formas normais negativas é o nosso próximo objectivo.

Antes, porém, é preciso sistematizar o tipo de operações genéricas que são admissíveis nas "listas de listas de literais". Vamos lidar, essencialmente, com duas operações:

soma A soma de duas destas listas, μ e υ, representada por $\mu + \upsilon$, é a concatenação das duas listas. Por exemplo,

$$[\,[a,b]\,,\,[c,d]\,] \;+\; [\,[x,y]\,,\,[z,w]\,] \quad = \quad [\,[a,b]\,,\,[c,d]\,,\,[x,y]\,,\,[z,w]\,]$$

produto O produto $\mu \times \upsilon$ é a lista formada por todas as concatenações de elementos de μ com elementos de υ. Por exemplo

$$\begin{aligned}[\,[a,b]\,,\,[c,d]\,] \;\times\; [\,[x,y]\,,\,[z,w]\,] \quad &= \\ = \quad [\,[a,b,x,y]\,,\,[c,d,x,y]\,,\,[a,b,z,w]\,,\,[c,d,z,w]\,]&\end{aligned}$$

O programa seguinte implementa estas definições em PROLOG

```
soma(U,V,W) :-
        append(U,V,W).

produto(U,[],[]).
produto(U,[P|V], W) :-
        map(U, append(P), U1) , produto(U,V,V1), soma(U1,V1,W).
```

O predicado `map` implementa a conhecida função de ordem superior com a mesma designação[a] que aplica uma "função" (neste caso o predicado `append(P)`) a todos os elementos de uma listas.

[a] Está implementado, conjuntamente com outros predicados "de ordem superior" num texto `auxs.pl` disponível na página da disciplina.

Considere-se duas proposições em forma normal conjuntiva

$$\mu \equiv (a \wedge b) \qquad \text{e} \qquad \upsilon \equiv (c \wedge d)$$

sendo a, b, c, d literais; como listas teríamos

$$\mu \equiv [[a], [b]] \qquad \text{e} \qquad \upsilon \equiv [[c], [d]]$$

A associatividade e distributividade dos operadores $\wedge, \vee$ permite-nos concluir

$$\begin{aligned} \mu \wedge \upsilon &\equiv a \wedge b \wedge c \wedge d \\ \mu \vee \upsilon &\equiv (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d) \end{aligned}$$

Como listas teríamos

$$\begin{aligned} \mu \wedge \upsilon &\equiv [[a], [b], [c], [d]] \quad \equiv \quad \mu + \upsilon \\ \mu \vee \upsilon &\equiv [[a, c], [a, d], [b, c], [b, d]] \quad \equiv \quad \mu \times \upsilon \end{aligned}$$

Este exemplo sugere a justificação da seguinte proposição

11 PROPOSIÇÃO *Se μ e υ são duas formas conjuntivas representadas por listas de listas de literais então $\mu \wedge \upsilon$ é representado pela lista $\mu + \upsilon$ e $\mu \vee \upsilon$ é representado pela lista $\mu \times \upsilon$.*

Trocando os papeis dos símbolos $\wedge$ e $\vee$ conclui-se também

12 PROPOSIÇÃO *Se μ e υ são duas formas disjuntivas representadas por listas de listas de literais então $\mu \vee \upsilon$ é representado pela lista $\mu + \upsilon$ e $\mu \wedge \upsilon$ é representado pela lista $\mu \times \upsilon$.*

O seguinte programa PROLOG calcula formas conjuntivas e disjuntivas de acordo com estas duas proposições.

(*continua*)

```
forma_conj(P, [[P]])  :- literal(P).
forma_conj((A,B), U)  :-
        forma_conj(A,A1) , forma_conj(B,B1), soma(A1,B1,U).
forma_conj((A;B), U)  :-
        forma_conj(A,A1) , forma_conj(B,B1), produto(A1,B1,U).

forma_disj(P, [[P]])  :- literal(P).
forma_disj((A;B), U)  :-
        forma_disj(A,A1) , forma_disj(B,B1), soma(A1,B1,U).
forma_disj((A,B), U)  :-
        forma_disj(A,A1) , forma_disj(B,B1), produto(A1,B1,U).

literal(P)  :-  atom(P) ; (P = -(Q) , atom(Q)).
```

No seguinte exemplo determina-se a forma normal negativa, e a respectiva simplificação em formas normais conjuntiva e disjuntiva, de

$$((p \supset q) \supset p) \supset p \quad , \quad \neg((\neg p \supset \neg q) \supset q \supset p)$$

```
| ?- fnn( ((p=>q)=>p)=>p , FN) ,
        forma_conj(FN,FC) , forma_disj(FN,FD).

FC = [[p,q,-(p)],[p,-(p)]],
FD = [[-(p),-(p)],[-(p),q],[p]],
FN = (-(p);q),-(p);p ?
yes
| ?- fnn(-(((p=>q)=>p)=>p) , FN) ,
        forma_conj(FN,FC) , forma_disj(FN,FD).

FC = [[p,p],[p,-(q)],[-(p)]],
FD = [[-(p),-(q),p],[-(p),p]],
FN = (p,-(q);p),-(p) ?
yes
```

Formas fechadas

Designaremos por **caminhos** as listas de literais que ocorrem nas formas normais conjuntivas e disjuntivas.

Um caminho diz-se **fechado** se contém, simultaneamente, um símbolo proposicional p e a sua negação $\neg\mathsf{p}$.

Uma **forma fechada** é uma forma normal (conjuntiva ou disjuntiva) que tem <u>todos</u> os seus caminhos fechados.

Exemplos

A forma normal conjuntiva representativa de $((\mathsf{p} \supset \mathsf{q}) \supset \mathsf{p}) \supset \mathsf{p}$ tem os caminhos $[\mathsf{p}, \mathsf{q}, \neg\mathsf{p}]$ e $[\mathsf{p}, \neg\mathsf{p}]$; ambos os caminhos são fechados.

Já o mesmo não acontece com os caminhos $[\neg\mathsf{p}, \neg\mathsf{p}]$, $[\neg\mathsf{p}, \mathsf{q}]$ e $[\mathsf{p}]$ da forma disjuntiva.

Inversamente, em $\neg(((\mathsf{p} \supset \mathsf{q}) \supset \mathsf{p}) \supset \mathsf{p})$, é a forma disjuntiva que tem todos os caminhos fechados.

Para a proposição $(((\mathsf{p} \supset \mathsf{q}) \supset \mathsf{p}) \supset \mathsf{p})$, a forma conjuntiva é fechada mas a forma disjuntiva não o é.

Inversamente $\neg(((\mathsf{p} \supset \mathsf{q}) \supset \mathsf{p}) \supset \mathsf{p})$ tem uma forma conjuntiva não fechada e uma forma disjuntiva fechada.

As duas cláusulas seguintes implementam em PROLOG as definições de caminho e forma fechada. Recorrem a um predicado auxiliar `todos` que aplica a todos os elementos da lista `Caminhos` o predicado `fechado`.

```
fechado(Caminho)  :- member(P,Caminho), member(-(P),Caminho).
forma_fechada(Caminhos) :- todos(Caminhos, fechado).
```

Validade de formas normais

Nas formas normais conjuntivas os caminhos representam disjunções de literais $a_1 \vee a_2 \vee \cdots \vee a_n$.

Se for fechado um dos a_i coincide com um símbolo p e outro coincide com a sua negação. Portanto o caminho representa uma proposição da forma

$$\mathsf{p} \vee \neg\mathsf{p} \vee a$$

em que a denota todos os restantes literais. Em Lógica Proposicional Clássica uma tal proposição é uma tautologia.

Uma forma conjuntiva representa uma proposição da forma

$$(a_1 \vee \cdots \vee a_n) \wedge \cdots \wedge (b_1 \vee \cdots \vee b_m)$$

Se todos os seus caminhos forem fechados representa uma conjunção de tautologias; deste modo é também uma tautologia.

Nas formas disjuntivas os caminhos representam conjunções $a_1 \wedge \cdots \wedge a_n$. Se o caminho for fechado esta proposição contém um sub-termo $\mathsf{p} \wedge \neg\mathsf{p}$; forçosamente será não-válido em qualquer modelo.

Assim caminho fechados em formas disjuntivas representam proposições equivalentes ao absurdo $\perp$.

Uma forma disjuntiva representa uma proposição da forma

$$(a_1 \wedge \cdots \wedge a_n) \vee \cdots \vee (b_1 \wedge \cdots \wedge b_m)$$

Se for fechada representará a disjunção de proposições equivalentes ao absurdo; por isso será também equivalente ao absurdo.

Justifica-se, assim, o resultado seguinte

13 Proposição *Uma forma normal conjuntiva fechada é uma tautologia. Uma forma normal disjuntiva fechada é equivalente ao absurdo.*

A proposição 13 justifica o seguinte programa PROLOG que automatiza o teste de validade e de inconsistência (equivalência ao absurdo) de uma proposição em Lógica Proposicional Clássica.

```
valido(P)        :-
        fnn(P,FN), forma_conj(FN, FC), forma_fechada(FC).

inconsistente(P) :-
        fnn(P,FN), forma_disj(FN, FD), forma_fechada(FD).
```

Para testar a validade de $((\mathtt{p} \supset \mathtt{q}) \supset \mathtt{p}) \supset \mathtt{p}$ podem ser usados ambos os predicados desde que `inconsistente` seja aplicado à negação da proposição que se pretende testar.

```
| ?- valido(((p => q) => p) => p).
yes

| ?- inconsistente(-(((p => q) => p) => p)).
yes
```

Uma tautologia da forma $P \supset Q$ pode ser provada por inconsistência de $P \wedge \neg Q$.

Por exemplo, $(\mathtt{p} \supset \mathtt{q}) \supset (\neg\mathtt{q} \supset \neg\mathtt{p})$ pode ser provada da seguinte forma

```
| ?- inconsistente( (p => q) & -(-(q) => -(p))).
yes
```

ou da forma seguinte

```
| ?- valido((p => q) => (-(q) => -(p))).
yes
```

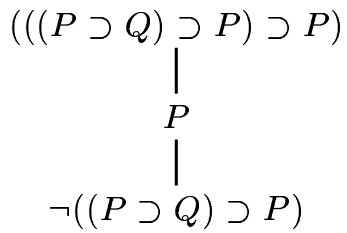
Sistemas de "Tableaux"

"*Tableaux*" são, essencialmente, uma forma diferente de apresentar as deduções do sistema LK (sem a regra de corte).

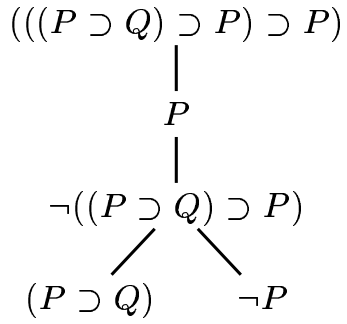
Tal como em LK a correcção de uma *tableaux* é normalmente justificada através da noção de **refutação**; neste caso são designados por **tableaux semânticos**.

Exemplo:
Consideremos a refutação da proposição $((P \supset Q) \supset P) \supset P$. Note-se que, P e Q denotam proposições arbitrárias e, neste aspecto, esta abordagem distingue-se da abordagem baseada em formas normais negativas.

A refutação de $((P \supset Q) \supset P) \supset P$ implica a refutação de P **e** a refutação de $\neg((P \supset Q) \supset P)$.
Na notação de *tableaux*, esta "regra de expansão" escreve-se sob a forma de uma árvore em que as duas proposições resultantes surgem no mesmo caminho.



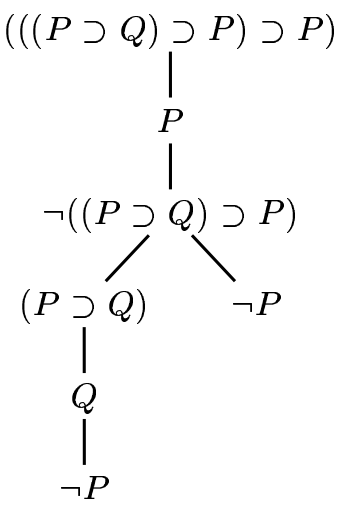
A refutação de $\neg((P \supset Q) \supset P)$ implica a refutação de $(P \supset Q)$ **ou** a refutação de $\neg P$.
O *tableau* respectivo cria uma ramificação de caminhos, um para cada uma das alternativas.



(*continua*)

Finalmente, refutar $P \supset Q$ implica refutar Q **e** refutar $\neg P$.

O *tableau* resultante é uma árvore formada por dois **caminhos** definidos pelas duas sequências de proposições. Cada caminho denota proposições que são refutadas conjuntivamente: isto é, refutar a proposição na raiz do caminho implica refutar **todas** as proposições nesse caminho.



□

Os vários caminhos iniciados na raiz de uma *tableaux* esgotam as possibilidades de refutação dessa raiz; se a raiz é refutada **algum** dos caminhos tem de ser refutado.

No exemplo anterior os dois caminhos eram

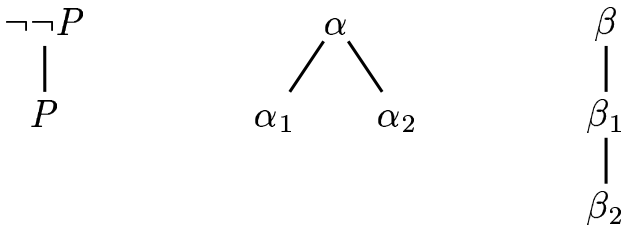
$$[\ ((P \supset Q) \supset P) \supset P\ ,\ P\ ,\ \neg((P \supset Q) \supset P)\ ,\ (P \supset Q)\ ,\ Q\ ,\ \neg P\ ]$$
$$[\ ((P \supset Q) \supset P) \supset P\ ,\ P\ ,\ \neg((P \supset Q) \supset P)\ ,\ \neg P\ ]$$

e ambos contêm o par de proposições $[P, \neg P]$. Caminhos nestas circunstâncias (contendo uma proposição e a sua negação) dizem-se **fechados**.

Caminhos fechados não podem ser refutados porque tal implicaria refutar simultaneamente P e $\neg P$. Por isso, uma *tableaux* onde todos os caminhos são fechados, diz-nos que a raiz não é refutável.

Um *tableau* onde todos os caminhos são fechados chama-se ***tableau* fechado** e afirma a validade da sua raiz.

Uma **expansão** de uma proposição P, representada por P^*, é uma das seguintes árvores



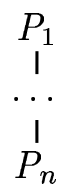
em que as proposições α e β são determinadas pelos seguintes quadros

regras conjuntivas		
α	α_1	α_1
$(P \wedge Q)$	P	Q
$\neg(P \vee Q)$	$\neg P$	$\neg Q$
$\neg(P \supset Q)$	P	$\neg Q$

regras disjuntivas		
β	β_1	β_1
$(P \vee Q)$	P	Q
$\neg(P \wedge Q)$	$\neg P$	$\neg Q$
$(P \supset Q)$	$\neg P$	Q

Um **tableau** para uma sequência de proposições $\{ P_1, \cdots, P_n \}$ é uma árvore construída de uma das formas seguintes

(1) A árvore não ramificada determinada pela sequência $[P_1, \cdots, P_n]$ é um *tableau* para esta sequência de proposições.



(2) Se $\mathbf{T}$ é um *tableau* para a sequência $\{ P_1, \cdots, P_n \}$, se P é um nodo em $\mathbf{T}$ (não necessariamente uma folha) e se P^* é a **expansão** de P então a árvore que resulta da substituição de P por P^* em $\mathbf{T}$ é ainda um *tableau* para a mesma sequência.

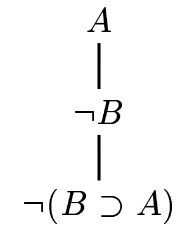
Nota:
A substituição, numa árvore de um nodo n por uma árvore $\mathcal{T}$ implica a afixação das subárvores de n a cada uma das folhas de $\mathcal{T}$.

16 DEFINIÇÃO *Um "tableau" para a teoria* $\{\neg\Gamma, A\}$, *fechado, determina a relação de dedução* $\Gamma \vdash A$.

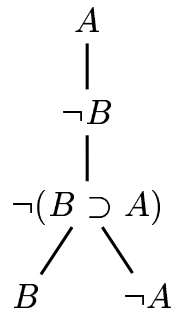
Exemplo

Construir um *tableau* que determine a relação de dedução $\{B \supset A, B\} \vdash A$.

Para impor a relação $B \supset A, B \vdash A$ é necessário construir um *tableau* fechado para a teoria $\{\neg(B \supset A), \neg B, A\}$.
O *tableau* inicia-se construindo uma árvore não ramificada com estas três proposições (a ordem entre elas é irrelevante).



A regra α para $\neg(B \supset A)$ pode ser usada.
Verifica-se que ambos os ramos deste *tableau* são fechados e, portanto, o *tableau* é fechado.



□

Genericamente um *tableau* para uma teoria $\{P_1, \cdots, P_n\}$ é uma forma conjuntiva equivalente à forma conjuntiva $[\,[\,P_1, \cdots, P_n\,]\,]$.

Cada caminho no *tableau* é visto como uma disjunção de proposições; globalmente o *tableau* interpreta-se como a conjunção dos seus diversos caminhos.

Cada transformação do *tableau* mantém a equivalência dessa forma conjuntiva à forma conjuntiva inicial.

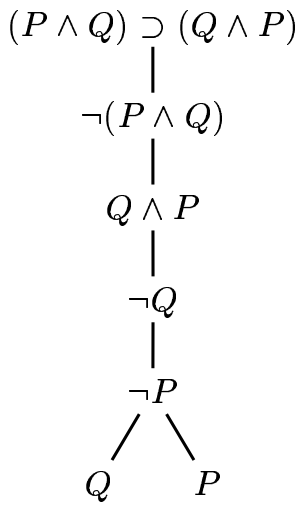
14 PROPOSIÇÃO *A relação de dedução* $\Gamma \vdash A$ *definida pela definição 16 é correcta e completa na Lógica Proposicional Clássica.*

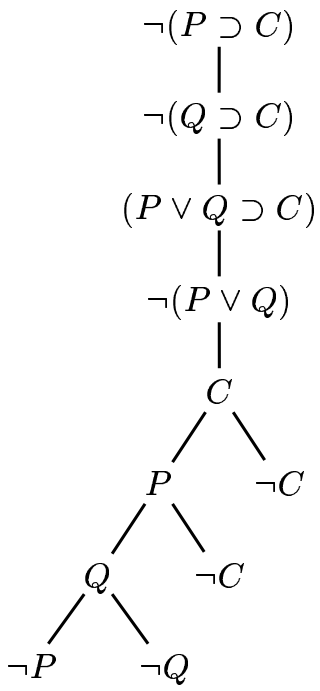
A prova deste resultado pode ser encontrada em qualquer livro de texto sobre *tableaux* [a] e segue os princípios que acabámos de sugerir.

□

Exemplos de provas, usando *tableaux*, de

$$\vdash (P \wedge Q) \supset (Q \wedge P) \qquad \text{e} \qquad \{(P \supset C), (Q \supset C)\} \vdash (P \vee Q \supset C)$$





Note-se que os *tableaux* são ambos fechados e, pela definição, determinam as relações de dedução pretendidas.

[a] Por exemplo, o texto *First Order Logic and Automated Theorem Proving*, de M. Fitting.

Método de Davis-Putman

Vimos anteriormente (acetato 11) como verificar a validade de formas normais conjuntivas e a inconsistência de formas normais disjuntivas: em ambos os casos bastava verificar se todos os caminhos eram fechados.

Pretendemos, agora, verificar a "condição oposta"; isto é,

(i) Verificar a **inconsistência** de formas normais **conjuntivas**, e

(ii) Verificar a **validade** de formas normais **disjuntivas**.

Exemplo

Pode-se provar que a proposição $\Phi \equiv (p \supset q) \wedge p \supset q$ é uma tautologia, provando que $\neg\Phi$ é inconsistente.

Note-se que $\neg\Phi \equiv (p \supset q) \wedge p \wedge \neg q$ se pode escrever

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

que é uma forma normal conjuntiva.

Para que esta proposição tenha um valor falso pelo menos um dos três factores, $(p \supset q)$, p e $\neg q$, tem de ter o valor falso; no entanto esse factor pode ser qualquer um e, para cada valoração dos símbolos proposicionais, o factor que tem o valor falso é diferente.

É assim,bastante mais complexo provar que uma forma conjuntiva é inconsistente do que provar que a forma disjuntiva equivalente é inconsistente. A forma disjuntiva equivalente, neste caso, seria

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

cuja inconsistência é verificada, simplesmente, notando que ambos os seus caminhos são fechados.

Porquê, então, usar a forma conjuntiva em vez de usar a disjuntiva?

Basicamente porque muitos problemas de inconsistência são facilmente descritos em formas conjuntivas e facilita a análise usar directamente essas formas no mecanismo de prova.

O método de Davis-Putman determina a validade (ou a inconsistência) de uma proposição Φ directamente pela definição de tautologia:

- Φ é uma *tautologia* de se verifica $\mathcal{M} \models \Phi$ para <u>todos</u> os modelos $\mathcal{M}$,
- Φ é *inconsistente* se se verifica $\mathcal{M} \not\models \Phi$ para <u>todos</u> os modelos $\mathcal{M}$.

Suponhamos que se escolhe um qualquer símbolo proposicional p que ocorra em Φ e se divide o conjunto dos modelos em duas classes: a classe dos modelos onde p tem o valor verdadeiro, representados genericamente por $\mathcal{M}_{\mathsf{p}+}$, e a classe dos modelos onde p tem o valor falso, representados genericamente por $\mathcal{M}_{\mathsf{p}-}$.

Para Φ ser uma tautologia (respectivamente, uma inconsistência) tem de ser válido (respectivamente, não-válido) em todos os modelos de cada uma das classes.

Na classe dos modelos $\mathcal{M}_{\mathsf{p}+}$ pode-se, no entanto, usar o facto de p ter o valor verdadeiro para simplificar a proposição Φ.

Seja, por exemplo,

$$\Phi \equiv (\neg \mathsf{p} \vee \mathsf{q}) \wedge \mathsf{p} \wedge \neg \mathsf{q}$$

Em todos os modelos onde p tem o valor verdadeiro, Φ equivale a

$$\Phi_{\mathsf{p}+} \equiv \mathsf{q} \wedge \neg \mathsf{q}$$

Análogamente, na classe dos modelos $\mathcal{M}_{\mathsf{p}-}$ pode-se simplificar Φ entrando em consideração com o facto de p ter o valor falso.

Neste exemplo, Φ simplificará em

$$\Phi_{\mathsf{p}-} \equiv \bot$$

Verifica-se agora que ambas as simplificações, $\Phi_{\mathsf{p}+}$ e $\Phi_{\mathsf{p}-}$, são inconsistentes; isto permite concluir que Φ é também inconsistente.

Algoritmo de Davis-Putman

Algoritmo para verificar a validade (respectivamente, inconsistência) de Φ

(1) Verifica-se se Φ pertence a uma classe de proposições particularmente simples nas quais o teste de validade (ou de inconsistência) pode ser decidido de forma trivial.

(2) Escolhe-se um qualquer símbolo proposicional p e calcula-se

   (a) Φ_{p+} , uma proposição equivalente a Φ em todos os modelos onde p tem o valor verdadeiro.

   (b) Φ_{p-} , uma proposição equivalente a Φ em todos os modelos onde p tem o valor falso.

(3) Usa-se indutivamente este mesmo algoritmo para verificar a validade (respectivamente, inconsistência) das proposições Φ_{p+} e Φ_{p-} . A proposição Φ é válida (respectivamente, inconsistente) se e só se ambos Φ_{p+} e Φ_{p-} forem válidos (respectivamente, inconsistentes).

A essência deste algoritmo está, portanto, na técnica de **separação** de Φ no par Φ_{p+} e Φ_{p-} e nas técnicas de **simplificação** de proposições.

É também necessário identificar as **proposições triviais** onde o teste de validade ou de inconsistência pode ser decidido de forma particularmente simples. inconsistentes.

Em qualquer destas técnicas a forma de representação das proposições é essencial. Por isso o método se restringe a formas normais disjuntivas, para o teste de validade, e a formas normais conjuntivas, para o teste de inconsistência.

No que se segue vamos apenas considerar o teste de inconsistência de formas conjuntivas; no entanto tudo o que vamos apresentar se converte, por dualidade, num teste de validade de formas disjuntivas.

Formas Triviais

Formas conjuntivas $(a_1 \vee \cdots \vee a_n) \wedge \cdots \wedge (b_1 \vee \cdots \vee b_m)$ são representadas por listas

$$[\,[\,a_1, \cdots, a_n\,]\,, \cdots, [\,b_1\,, \cdots\,, b_m\,]\,]$$

que podem ser vazias.

Se uma "lista interior" (um caminho) for vazia, representará a disjunção de zero proposições; ou seja é equivalente a $\bot$ já que este é elemento neutro do operador binário $\vee$.

Qualquer forma conjuntiva que contenha um caminho vazio é uma conjunção onde um dos factores é o absurdo $\bot$. O resultado da conjunção só pode ser $\bot$.

Pelo contrário se a "lista exterior" for vazia, a proposição representada por tal forma conjuntiva é uma conjunção de zero factores; será portanto uma tautologia porque os elementos neutros do operador $\wedge$ são precisamente as tautologias.

Representaremos uma tautologia genérica pelo símbolo $\top$.

Pode-se afirmar:

(i) Uma forma disjuntiva Φ que contenha um caminho vazio representa $\bot$ e, assim, é sempre inconsistente.

(ii) Uma forma disjuntiva vazia $[\,]$ representa $\top$ e, portanto, nunca é inconsistente.

Deste modo o algoritmo de Davis-Putman acusa sucesso se encontrar uma forma que contenha um caminho vazio e falha se encontrar uma forma vazia.

Separação

Uma forma conjuntiva que contenha um caminho equivalente a $\top$ representa uma proposição da forma

$$\top \wedge \Phi$$

Como essa proposição é equivalente a Φ, concluímos que:

qualquer caminho equivalente a uma tautologia pode ser eliminado de uma forma conjuntiva sem alterar o valor lógica dessa forma.

Uma outra possibilidade ocorre quando um literais num dos caminhos de Φ é equivalente a $\bot$. Neste caso a forma representa uma proposição do tipo

$$(\bot \vee P) \wedge \Phi'$$

Como $(\bot \vee P) \equiv P$, esta forma pode-se escrever simplesmente $P \wedge \Phi'$. Donde

qualquer literal equivalente a $\bot$ pode ser eliminado de qualquer caminho de uma forma conjuntiva.

Com estas duas observações podemos enunciar as seguintes regras para determinar as formas Φ_{p^+} e Φ_{p^-} que resultam de Φ, quando se assume que p é equivalente a, respectivamente, $\top$ e $\bot$.

(1) Para se obter a forma Φ_{p^+} de Φ
   (a) Retira-se de Φ todos os caminhos que contenham o literal p.
   (b) Elimina-se $\neg\mathsf{p}$ de todos os caminhos de Φ que contêm este literal.

(2) Para se obter a forma Φ_{p^-} de Φ
   (a) Retira-se de Φ todos os caminhos que contenham o literal $\neg\mathsf{p}$.
   (b) Elimina-se p de todos os caminhos de Φ que contêm este literal.

Exemplo Dado $\Phi \equiv [[\mathsf{q}, \neg\mathsf{p}], [\mathsf{p}], [\neg\mathsf{q}]]$ teremos, usando estas regras,

$$\Phi_{\mathsf{p}^+} \equiv [[\mathsf{q}], [\neg\mathsf{q}]] \qquad \Phi_{\mathsf{p}^-} \equiv [[\,], [\neg\mathsf{q}]]$$

Simplificação

A simplificação de formas conjuntivas ocorre a três situações específicas:

(I) *Quando um caminho contém uma tautologia*
Isto acontece quando o caminho é fechado (isto é, contém p e $\neg\mathsf{p}$). Neste caso o caminho contém $\mathsf{p} \vee \neg\mathsf{p} \equiv \top$ e pode, como vimos, ser eliminado da forma conjuntiva.

(II) *Quando um caminho é formado por um único literal*
Se o "caminho unitário" for da forma $[\mathsf{p}]$ então uma eventual separação de Φ usando p conduz a uma forma Φ_{p^-} que contém um caminho vazio (o que resulta de se eliminar p do caminho unitário).

Um tal Φ_{p^-} é inconsistente (por ter um caminho vazio) e não precisa de ser considerado em desenvolvimentos seguintes; resta apenas Φ_{p^+}.

Em conclusão:

se Φ *contém o caminho* $[\mathsf{p}]$ *então pode ser simplificado em* Φ_{p^+}.

Dualmente,

se Φ *contém o caminho* $[\neg\mathsf{p}]$ *então pode ser simplificado em* Φ_{p^-}.

(III) *Quando existe um "literal puro" em* Φ
Um literal p é puro em Φ se esta forma contém p mas não contém $\neg\mathsf{p}$. Identicamente $\neg\mathsf{p}$ é puro em Φ se esta forma contém $\neg\mathsf{p}$ mas não contém p.

Se p é puro em Φ, a forma Φ_{p^+} obtém-se eliminado todas os caminhos de Φ que contêm p. Ou seja pode-se escrever

$$\Phi \equiv \Phi_{\mathsf{p}^+} \wedge \cdots$$

o que significa que, Φ é inconsistente se e só se Φ_{p^+} for inconsistente.

Portanto, neste caso, Φ pode ser simplificado em Φ_{p^+}. Dualmente, se o literal puro for $\neg\mathsf{p}$, Φ pode ser simplificado em Φ_{p^-}. Em qualquer dos casos:

Se Φ *contém um literal puro podem-se eliminar todos os caminhos que contêm esse literal.*

Exemplo

Uma dedução $\Gamma \vdash P$ pode-se sempre provar (em LPC) mostrando a inconsistência da teoria $\Gamma, \neg P$. Usando esta abordagem pretende-se provar

$$\{ \mathsf{p} \wedge \mathsf{s} \supset \mathsf{r}\,,\ \mathsf{q} \supset \mathsf{s} \} \vdash \mathsf{p} \wedge \mathsf{q} \supset \mathsf{r}$$

recorrendo à inconsistência da seguinte forma normal conjuntiva

$$[\,[\neg \mathsf{p}, \neg \mathsf{s}, \mathsf{r}\,]\,,\ [\,\neg \mathsf{q}, \mathsf{s}\,]\,,\ [\,\mathsf{p}\,]\,,\ [\,\mathsf{q}\,]\,,\ [\,\neg \mathsf{r}\,]\,]$$

Começamos por simplificar esta forma usando os três caminhos formados por literais únicos. Usando o literal p, a forma simplifica-se em

$$[\,[\neg \mathsf{s}, \mathsf{r}\,]\,,\ [\,\neg \mathsf{q}, \mathsf{s}\,]\,,\ [\,\mathsf{q}\,]\,,\ [\,\neg \mathsf{r}\,]\,]$$

Usando o literal q a forma simplifica-se em

$$[\,[\neg \mathsf{s}, \mathsf{r}\,]\,,\ [\,\mathsf{s}\,]\,,\ [\,\neg \mathsf{r}\,]\,]$$

Usando o literal $\neg \mathsf{r}$ simplificamos em

$$[\,[\neg \mathsf{s}\,]\,,\ [\,\mathsf{s}\,]\,]$$

Usando o caminho singular $[\mathsf{s}]$ podemos ainda simplificar e obter

$$[\,[\,]\,]$$

Ficou-se assim reduzido à forma trivial constituída por um caminho vazio. Esta forma é inconsistente e, portanto, a forma conjuntiva inicial também é inconsistente.

Neste exemplo particularmente simples não chega a ser necessário fazer qualquer separação; limitamos-nos a usar várias vezes uma única regra (a 2^{a}) das três regras de simplificação disponíveis.

□

O seguinte programa **Prolog** implementa uma forma directa do algoritmo de Davis-Putnam.

```
verifica([])  :- false.
verifica(S)   :- member([],S).
verifica(S)   :- reduz(S,S1), !, verifica(S1).
verifica(S)   :- divide(S,U,V), !, verifica(U), verifica(V).

% Divisão
fi_mais(P,C,U,V) :-
        member(-(P),C) -> delete(C,-(P),C1), V = [C1|U] ;
        member(P,C)    ->  V = U ; V = [C|U].
fi_menos(P,C,U,V) :-
        member(P,C)    -> delete(C,P,C1), V = [C1|U] ;
        member(-(P),C) ->  V = U ; V = [C|U].
sub_mais(P,S,S1)  :- atom(P), foldr([],S,fi_mais(P),S1).
sub_menos(P,S,S1) :- atom(P), foldr([],S,fi_menos(P),S1).
divide(S,U,V)     :- em(P,S), sub_mais(P,S,U), sub_menos(P,S,V).

% Simplificação
reduz(S,S1) :- unario_mais(P,S),  sub_mais(P,S,S1)
reduz(S,S1) :- unario_menos(P,S), sub_menos(P,S,S1).
reduz(S,S1) :- removivel(S,C),   delete(S,C,S1).

% Testes
fechado(C)           :-  member(P,C), member(-(P),C).
literal_puro(P,S)    :-  atom(P), em(P,S),  \+ em(-(P),S).
literal_puro(-(P),S) :-  em(-(P),S), \+ em(P,S).
em(P,S)              :-  member(C,S), member(P,C).
removivel(S,C)       :-
        member(C,S), (fechado(C) ; member(P,C), literal_puro(P,S)).
unario_mais(P,S)     :-  member([P],S), atom(P).
unario_menos(P,S)    :-  member([-(P)],S).
```