Lógica Computacional
Conjuntos completos de conectivas

DCC/FCUP

2020/21

Funções de verdade (ou Booleanas)

p	q	$p \wedge q$
$\perp$	$\perp$	$\perp$
$\perp$	$\top$	$\perp$
$\top$	$\perp$	$\perp$
$\top$	$\top$	$\top$

x	y	$F(x,y)$
$\perp$	$\perp$	$\perp$
$\perp$	$\top$	$\perp$
$\top$	$\perp$	$\perp$
$\top$	$\top$	$\top$

A tabela de verdade duma fórmula ϕ, com $n > 0$ variáveis proposicionais $p_1, \ldots p_n$, define uma função de verdade

$$F_\phi : \{\top, \perp\}^n \longrightarrow \{\top, \perp\}$$

tal que $F_\phi(x_1, \ldots, x_n) = v_X(\phi)$,onde $v_X(p_i) = x_i$ para $i \in \{1 \ldots n\}$ e $x_i \in \{\top, \perp\}$.

Funções de verdade

Qualquer função de $f : \{\top, \bot\}^n \longrightarrow \{\top, \bot\}$, com $n > 0$ diz-se uma função de verdade.
Para $n = 1$ temos 4 funções:

x_1	$f_1(x_1)$
$\top$	$\bot$
$\bot$	$\bot$

x_1	$f_2(x_1)$
$\top$	$\bot$
$\bot$	$\top$

x_1	$f_3(x_1)$
$\top$	$\top$
$\bot$	$\bot$

x_1	$f_4(x_1)$
$\top$	$\top$
$\bot$	$\top$

Existem 16 funções de verdade de aridade 2. Prove!
E quantas funções existem de aridade n, para $n > 0$?

Conjunto de conectivas completo

Um conjunto de conectivas C diz-se completo se para qualquer função de verdade f existe uma fórmula ϕ com n variáveis proposicionais e contendo só conectivas de C, tal que $F_\phi = f$.

Proposição: O conjunto de conectivas $\{\wedge, \vee, \neg\}$ é completo.

Demonstração.

Mostramos por indução sobre n. Para $n = 1$ existem 4 funções de verdade:

x_1	f_1
$\top$	$\bot$
$\bot$	$\bot$

x_1	f_2
$\top$	$\bot$
$\bot$	$\top$

x_1	f_3
$\top$	$\top$
$\bot$	$\bot$

x_1	f_4
$\top$	$\top$
$\bot$	$\top$

Sendo $\phi_1 = p \wedge \neg p$, $\phi_2 = \neg p$, $\phi_3 = p$, $\phi_4 = p \vee \neg p$, tem-se que $F_{\phi_i} = f_i$ para $1 \leq i \leq 4$.

$\square$

Conjunto de conectivas completo

Demonstração.

Supondo que a hipótese é válida para $n \geq 0$, seja $f : \{\top, \bot\}^{n+1} \longrightarrow \{\top, \bot\}$. Consideremos duas funções de verdade n-árias f_1 e f_2 tal que:

$f_1(x_1, \ldots, x_n) = f(x_1, \ldots, x_n, \top)$ e

$f_2(x_1, \ldots, x_n) = f(x_1, \ldots, x_n, \bot)$.

Por hipótese de indução existem ϕ_i, com variáveis $p_1, \ldots, p_n$, tal que $F_{\phi_i} = f_i$ para $i = 1, 2$. Tome-se

$\phi = (p_{n+1} \wedge \phi_1) \vee (\neg p_{n+1} \wedge \phi_2)$, então $F_\phi = f$.

$\square$

Proposição: O conjunto de conectivas $\{\neg, \rightarrow\}$ é completo.

Demonstração.

Basta ver que $\phi \wedge \psi \ \equiv \neg(\phi \rightarrow \neg\psi)$ e $\phi \vee \psi \ \equiv (\neg\phi \rightarrow \psi)$. $\square$

Fórmulas de Horn

Uma *fórmula de Horn* é uma fórmula em forma normal conjuntiva em que em cada disjunção (cláusula) existe no máximo um literal positivo.

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

Numa fórmula de Horn, as disjunções $\neg p_1 \vee \ldots \vee \neg p_n \vee p$ também se podem escrever como:

$$(p_1 \wedge \ldots \wedge p_n) \rightarrow p$$

ou se p não existe (ou é $\perp$):

$$(p_1 \wedge \ldots \wedge p_n) \rightarrow \perp$$

ou se os p_i não existem:

$$\top \rightarrow p$$

Fórmulas de Horn

Nem todas as fórmulas têm uma fórmula de Horn equivalente!

Mas...
Para determinar se uma fórmula de Horn da lógica proposicional é **satisfazível** podemos usar um algoritmo eficiente!

Satisfazibilidade duma fórmula de Horn

Considere a fórmula $p \wedge \neg q \wedge (q \vee \neg p)$:

- começar por colocar numa linha as variáveis proposicionais que ocorrem na fórmula e colocar a fórmula. Ex:

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

- se alguma das variáveis proposicionais é um dos elementos da conjunção atribuir o valor $\top$ a essa variável (porquê?). Ex:

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

Satisfazibilidade duma fórmula de Horn

- Com essa informação preencher a tabela como se tivesse a construir a tabela de verdade (para essa linha), analisando cada disjunção para determinar se, para ela ser verdadeira, se pode determinar mais valores para as variáveis proposicionais:

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

- Neste caso, q tem de ser $\top$ e então isso pode ser acrescentado:

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

Satisfazibilidade duma fórmula de Horn

- E voltando a repetir este passo, obtém-se:

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

  Continuar até mais nada poder ser acrescentado.

- se no passo anterior se atribuir $\bot$ a um dos elementos da conjunção, a fórmula também fica com o valor $\bot$ e não é satisfazível. Caso contrário podemos atribuir à fórmula o valor $\top$ se atribuirmos $\bot$ às restantes variáveis proposicionais.

No exemplo que estamos a considerar, a fórmula tem o valor $\bot$ e portanto não é satisfazível.

Satisfazibilidade duma fórmula de Horn

Aplique o algoritmo anterior a

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

Satisfazibilidade

Saber se uma fórmula da lógica proposicional é satisfazível é um problema central da Ciência de Computadores pois muitos problemas de outras áreas se podem exprimir em termos dele:

- optimização: planeamento, escalonamento,etc.
- combinatória: coloração de grafos, etc.
- teorias de lógica de primeira ordem: programação linear, aritmética de reais, strings de bits, apontadores, etc. (resolutores SMT).

Mas se uma fórmula não estiver em forma normal disjuntiva ou não for duma classe para a qual é fácil determinar a satisfazibilidade, será que o único recurso é fazer a tabela de verdade?
No pior caso, SIM... isto é, não é conhecido nenhum algoritmo para a satisfazibilidade que não seja exponencial no número de variáveis e conectivas da fórmula! $P{=}NP$?

Satisfazibilidade

No caso geral, podemos encontrar algoritmos que determinam a satisfazibilidade de uma fórmula mais eficientemente que a construção de **força bruta** da tabela de verdade. A ideia geral ir construindo parcialmente uma valorização que satisfaça a fórmula ...em vez de gerar uma valorização completa (linha da tabela de verdade) e verificar se a fórmula a satisfaz!.
Este processo é especialmente simplificado se as as fórmulas estiverem em forma normal conjuntiva (FNC). Pois pode ser representado de forma compacta usando a noção de cláusula.

Cláusulas

Uma cláusula é uma disjunção de literais $l_1 \vee l_2 \vee \ldots \vee l_n$, com $n \geq 0$. Uma cláusula pode representar-se por um conjunto de literais. Se $n = 0$ dizemos que a cláusula é vazia e corresponde a $\perp$. Se $n = 1$ dizemos que a cláusula é unitária.

Exemplo:
$p \vee \neg q \vee \neg p \vee s$ pode-se representar por $\{p, \neg q, \neg p, s\}$.

Cláusulas

Conjunto de cláusulas

Qualquer fórmula da lógica proposicional em FNC pode-se representar por um conjunto de cláusulas.

$$\neg p \land (q \lor r \lor q) \land (\neg r \lor \neg s) \land (p \lor s) \land (\neg q \lor \neg s)$$

corresponde ao conjunto de cláusulas:

$$\{\{\neg p\}, \{q, r\}, \{\neg r, \neg s\}, \{p, s\}, \{\neg q, \neg s\}\}$$

Cláusulas

Literais complementares

Dado um literal l designamos por *literal complementar* o literal $\tilde{l}$ definido por:

$$\tilde{l} = \begin{cases} \neg l, & \text{se } l \text{ é uma variável (positivo)} \\ p, & \text{se } l \text{ é da forma } \neg p \text{ (negativo)} \end{cases}$$

Por exemplo $\tilde{\neg p} = p$ e $\tilde{p} = \neg p$.

Como vimos, uma cláusula é uma tautologia se contém um par de literais complementares p e $\neg p$.
Estas cláusulas podem ser retiradas do conjunto sem alterar a satisfazibilidade.

Algoritmo de Davis-Putnam

Versão inicial de 1960 que ainda é a base de muitos dos algoritmos mais eficientes actualmente:
`http://www.satlive.org/index.jsp` ou Concurso
`http://www.satcompetition.org/`.
A ideia base do algoritmo é considerar para cada variável os possíveis valores de verdade e simplificar a fórmula de acordo com essas atribuições até se poder concluir que ela é ou não satisfazível.

Algoritmo de Davis-Putnam

Propagação Unitária

Seja S um conjunto de cláusulas. Um conjunto S' é obtido de S por propagação unitária se S' se obtém de S por repetição da seguinte transformação: se S contém uma cláusula unitária l então:

1. remover de S todas as cláusulas da forma $l \vee C'$
2. substituir em S cada cláusula da forma $\tilde{l} \vee C'$ pela cláusula C'.

Algoritmo de Davis-Putnam

Considere o seguinte conjunto de cláusulas:

$$\{p_1, \neg p_1 \vee \neg p_2, p_3 \vee p_2, \neg p_7 \vee p_2, \neg p_3 \vee p_4, \neg p_3 \vee p_5, \\ \neg p_4 \vee \neg p \vee q, \neg p_5 \vee \neg p_6 \vee r, \neg p \vee \neg q \vee p_6, p \vee p_7, \neg r \vee p_7\} \tag{1}$$

Aplicando a p_1

$$\{\neg p_2, p_3 \vee p_2, \neg p_7 \vee p_2, \neg p_3 \vee p_4, \neg p_3 \vee p_5, \neg p_4 \vee \neg p \vee q, \\ \neg p_5 \vee \neg p_6 \vee r, \neg p \vee \neg q \vee p_6, p \vee p_7, \neg r \vee p_7\} \tag{2}$$

Algoritmo de Davis-Putnam

Aplicando a propagação a $\neg p_2$

$$\begin{aligned}\{p_3, \neg p_7, \neg p_3 \vee p_4, \neg p_3 \vee p_5, \neg p_4 \vee \neg p \vee q,\\ \neg p_5 \vee \neg p_6 \vee r, \neg p \vee \neg q \vee p_6, p \vee p_7, \neg r \vee p_7\}\end{aligned} \tag{3}$$

Depois de aplicar a propagação a p_3 e $\neg p_7$, temos:

$$\{p_4, p_5, \neg p_4 \vee \neg p \vee q, \neg p_5 \vee \neg p_6 \vee r, \neg p \vee \neg q \vee p_6, p, \neg r\} \tag{4}$$

Algoritmo de Davis-Putnam

Propagando para p_4, p_5,$\neg r$ e p vem:

$$\{q, \neg p_6, \neg q \vee p_6\} \tag{5}$$

E finalmente propagando para q e $\neg p_6$:

$$\{\perp\} \tag{6}$$

Conclusão? mostra-se que o conjunto inicial de cláusulas é não satisfazível (usando neste caso só propagação unitária).

Algoritmo de Davis-Putnam

```
DLL(S) {
   input: conjunto de clausulas S
   output: satisfazivel ou nao satisfazivel
   S := propaga(S)
   if S vazio then return satisfazivel
   if S contem ⊥ then return nao satisfazivel
   l := selecciona_literal(S)
   if DLL(S ∪ {l}) = satisfazivel
      then return satisfazivel
      else return DLL(S ∪ {l̃})
}
```

Algoritmo de Davis-Putnam

A função `seleciona_literal` retorna um literal da cláusula.
Esta função poderá ser considerada um parâmetro do algoritmo, pois uma escolha adequada do literal pode tornar o algoritmo mais eficiente.

Possíveis critérios são:

- escolher uma variável que ocorre mais vezes;
- tal que o produto das ocorrências de l e $\tilde{l}$ é máximo;
- que ocorre mais vezes em cláusulas de tamanho minimal, etc.

Aplicação do Algoritmo

$$S = \{\neg p \vee \neg q, \neg p \vee q, p \vee \neg q, p \vee q\}$$

Como não tem cláusulas unitárias, temos de selecionar um literal, por exemplo, $\neg p$.
Por propagação:

$$\begin{array}{rcl} \{\neg p \vee \neg q, \neg p \vee q, p \vee \neg q, p \vee q, \neg p\} & \Rightarrow & \{\neg q, q\} \\ & \Rightarrow & \{\bot\} \end{array}$$

Aplicação do Algoritmo

Temos ainda de considerar $S \cup \{p\}$. Neste caso a propagação é:

$$\begin{array}{rcl} \{\neg p \vee \neg q, \neg p \vee q, p \vee \neg q, p \vee q, p\} & \Rightarrow & \{\neg q, q\} \\ & \Rightarrow & \{\perp\} \end{array}$$

E o algoritmo retorna nao satisfazivel .