Programação Funcional
14ª Aula — Um verificador de tautologias

Pedro Vasconcelos
DCC/FCUP

2014

Proposições lógicas

Uma *proposição lógica* é construida apartir de:

constantes T, F (*verdade* e *falsidade*)

variáveis $a, b, c, \ldots$

conectivas lógicas $\wedge, \vee, \neg, \implies$

parêntesis $(,)$

Exemplos:

$$a \wedge \neg b$$

$$a \wedge ((\neg a) \implies F)$$

$$(\neg(a \vee b)) \implies ((\neg a) \wedge (\neg b))$$

Tabelas de verdade das conectivas

a	b	$a \wedge b$
F	F	F
T	F	F
F	T	F
T	T	T

a	b	$a \vee b$
F	F	F
T	F	T
F	T	T
T	T	T

a	$\neg a$
F	T
T	F

a	b	$a \Longrightarrow b$
F	F	T
T	F	F
F	T	T
T	T	T

Tautologias

Uma proposição cujo valor é *verdade* para qualquer atribuição de valores às variáveis diz-se uma tautologia.

Exemplo:

a	$\neg a$	$a \vee \neg a$
F	T	T
T	F	T

Conclusão: $a \vee \neg a$ é uma tautologia.

Representação de proposições

Vamos definir um tipo recursivo para representar proposições.

```
data Prop = Const Bool                -- constantes
          | Var Char                  -- variáveis
          | Neg Prop                  -- negação
          | Conj Prop Prop            -- conjunção
          | Disj Prop Prop            -- disjunção
          | Impl Prop Prop            -- implicação
            deriving (Eq,Show)
```

Representação de proposições (cont.)

Exemplo: a proposição

$$a \implies ((\neg a) \implies F)$$

é representada como

```
Impl (Var 'a') (Impl (Neg (Var 'a')) (Const False))
```

Associação de valores a variáveis

Para atribuir valores de verdade às variáveis vamos usar uma *lista de associações*.

Exemplo: a atribuição

$$\begin{cases} a = T \\ b = F \\ c = T \end{cases}$$

é representada pela lista

```
[('a',True),('b',False),('c',True)]
```

Associação de valores a variáveis (cont.)

Definimos:

- *listas de associações* entre *chaves* e *valores*;
- uma função para procurar o valor associado a uma chave.

```
type Assoc ch v = [(ch,v)]

find :: Eq ch => ch -> Assoc ch v -> v
find ch assocs = head [v | (ch',v)<-assocs, ch==ch']
```

É uma função parcial: dá um erro se não encontrar a chave!

Calcular o valor duma proposição

Vamos definir o *valor de verdade* de uma proposição por recursão.

O primeiro argumento é uma *atribuição de valores* às variáveis.

```
type Atrib = Assoc Char Bool

valor :: Atrib -> Prop -> Bool
valor s (Const b)  = b
valor s (Var x)    = find x s
valor s (Neg p)    = not (valor s p)
valor s (Conj p q) = valor s p && valor s q
valor s (Disj p q) = valor s p || valor s q
valor s (Impl p q) = not (valor s p) || valor s q
```

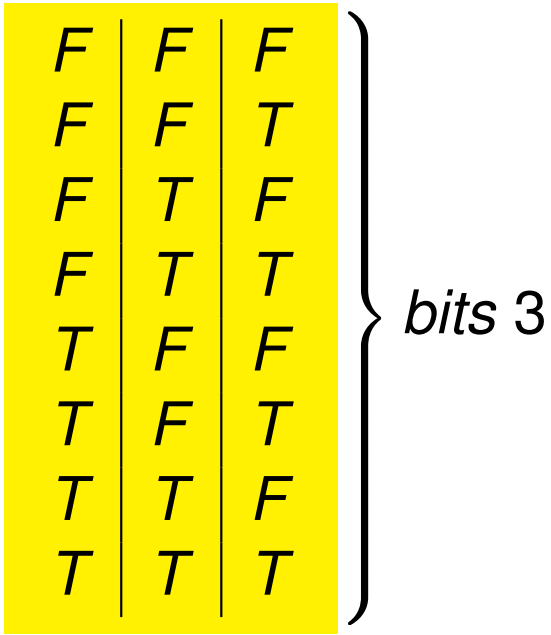
Gerar atribuições às variáveis

- Para n variáveis distintas há 2^n linhas na tabela de verdade.
- Como obter todas as atribuições de forma sistemática?
- Vamos escrever uma função para gerar *todas* as sequências de n boleanos (cf. exercício 3.10):

```
bits :: Int -> [[Bool]]
```

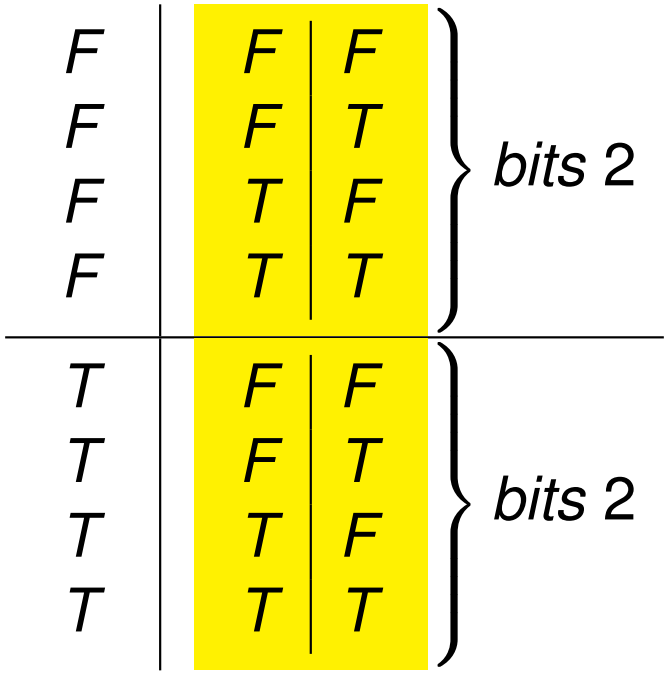
Gerar atribuições às variáveis (cont.)

Exemplo, as sequências de comprimento 3 (três variáveis):



Gerar atribuições às variáveis (cont.)

Podemos decompor em duas cópias da tabela para 2 variáveis com uma coluna extra:



Gerar atribuições às variáveis (cont.)

Em geral: vamos gerar as sequências de forma recursiva.

```
bits :: Int -> [[Bool]]
bits 0 = [[]]
bits n = [b:bs | bs<-bits (n-1), b<-[False,True]]
```

Gerar atribuições às variáveis (cont.)

Falta ainda gerar atribuições; começamos por listar todas as variáveis numa proposição.

```
vars :: Prop -> [Char]
vars (Const _)   =  []
vars (Var x)     =  [x]
vars (Neg p)     =  vars p
vars (Conj p q)  =  vars p ++ vars q
vars (Disj p q)  =  vars p ++ vars q
vars (Impl p q)  =  vars p ++ vars q
```

Gerar atribuições às variáveis (cont.)

A função seguinte gera todas as as atribuições de variáveis duma proposição:

```
atribs :: Prop -> [Atrib]
atribs p = map (zip vs) (bits (length vs))
     where vs = nub (vars p)
```

(A função *nub* da biblioteca *Data.List* remove repetidos.)

Verificar tautologias

Uma proposição é tautologia se e só se for verdade para todas as atribuições de variáveis.

```
tautologia :: Prop -> Bool
tautologia p = and [valor s p | s<-atribs p]
```

Verificar tautologias (cont.)

Alguns exemplos:

```
> tautologia (Var 'a')
False

> tautologia (Impl (Var 'p') (Var 'p'))
True

> tautologia (Disj (Var 'a') (Neg (Var 'a')))
True
```

Exercícios

1. Escrever uma função que calcula a lista das atribuições que tornam uma proposição *falsa* (i.e. uma lista de contra-exemplos).
2. Escrever um programa para imprimir a tabela de verdade duma proposição.