Lógica Computacional

Frequência

Duração: 2 horas

Universidade da Beira Interior

Segunda-Feira 9 de Janeiro de 2017

> Prova sem consulta de material pedagógico.
> É proibido o uso de calculadora e de telemóvel.
> Qualquer fraude implica reprovação imediata.
> Só serão corrigidas as provas **legíveis**.

Exercício 1 (OCaml e Indução Estrutural)

1. *Considere a seguinte função:*

   ```
   let rec misterio a b c = if c > b then misterio a b c else a (misterio a b (a c));;
   ```

   Qual é o seu tipo (na forma em que o compilador ocaml o calcula)?

2. *Define a função* `inverte` *que aceita em parâmetro uma lista e devolve em saída a lista invertida.*

3. *Dê uma definição recursiva terminal da função* `inverte` *dada no ponto anterior.*

4. *Dê uma definição da função* `inverte` *usando o combinador $fold\_left$ sobre listas.*

5. *Dê uma definição da função length que calcula o comprimento da lista em parâmetro.*

6. *Demonstra por indução estrutural que*

$$\forall l :\ 'a\ list,\ length\ l = \ length\ (inverte\ l)$$

__Resposta__:

1. Vejamos o que a definição da função `misterio` nos diz sobre os seus parâmetros.

   Para começar: tem 3 parâmetros. Logo tem um tipo da forma seguinte:

   $'a \ \rightarrow \ 'b \ \rightarrow \ 'c \ \rightarrow \ 'd$

   Resta-nos saber se temos informação para precisar mais os valores de $'a$, $'b$, $'c$, $'d$. (Relembra-se que $'a$ é o tipo do parâmetro a, etc, e que $'d$ é o tipo do valor devolvido pela função `misterio`)

   b e c são comparados, logo são do mesmo tipo: $'b \ = \ 'c$

   O parâmetro a é usado por duas vezes como função unária. Ou seja é do tipo $'e \ \rightarrow \ 'f$.

   Como a é invocada no lugar de c na função mistério (i.e. $(a\ c)$), o tipo de c coincide com o tipo de $(a\ c)$. Ou seja $'f \ = \ 'c$. Mais, como c pode ser parâmetro de a, o tipo de entrada de a é igual ao tipo de c. Ou seja: $'e \ = \ 'c$.

Resumindo (e incluindo já o que sabemos dos tipos)

$$misterio \ : ('c \to' c) \to' c \to' c \to' d$$

Como usamos um if-then-else, o tipo do valor devolvido no caso then tem de ser igual ao tipo do valor devolvido np caso else. Mais, este tipo é o tipo do retorno da função `misterio`.

Assim o tipo de `misterio a b c` (retorno do `then`) é igual ao tipo de `a (misterio a b (a c))` (retorno do `else`). Ora a função a retorna um valor igual ao tipo do parâmetro c

ou seja $'d \ = \ 'c$

Conclusão

$$misterio \ : ('c \to' c) \to' c \to' c \to' c$$

OCaml deverá responder renomeando $'c$ para $'a$, de forma inócua e totalmente idêntica..

$$misterio \ : ('a \to' a) \to' a \to' a \to' a$$

Como nota final, auxiliar, desta alínea. Esta função não tem sentido computacional muito relevante. Não parece calcular nada de concreto. Até parece permitir uma recursividade infinita directa caso $b > c$. Na verdade, no caso do *else*... também. Mas isso é outra história...

2. Exercício de resolução trivial. A única nota explicativa relevante é: quando a lista por inverter tem um elemento *el* à cabeça, a lista invertida terá esse elemento como ultimo elemento. Daí calcularmos a lista invertida da cauda (aqui *li*), e concatenarmos este resultado à esquerda da lista constituída por *el* sozinho. É juntar à cauda.

   Porquê não usar a operação :: no lugar de @? precisamente porque :: é a operação que junta um elemento à cabeça de uma lista (de $'a$ e $'a\ list$ para $'a\ list$), e aqui queremos juntar no fim. Usamos @ para esse propósito (não se esqueça que @ está a espera de duas listas para realizar a concatenação, daí ser invocada com [*el*] e não somente *el*).

```
let rec inverte l =
  match l with
  | [] → []
  | el::li → (inverte li)@[el]
```

3. Resposta trivial igualmente. O truque é acumular numa lista em parâmetro os elementos que vamos "descascando" da lista l. Como vamos juntando na ordem da exploração da lista l, o último elemento de l é o o ultimo a ser colocado à cabeça do acumulador: o acumulador acaba por ser a lista na ordem inversa!

```
let rec inverte_rt l acc=
  match l with
  | [] → acc
  | el::li → (inverte_rt li (acc::acc))

let inverte l = inverte_rt l []
```

4. Nesta alínea, retomamos a ideia que fundamentou a função recursiva terminal da alínea anterior: vamos explorar a lista da esquerda para a direita (é o percurso feito pela função `fold_left`) e acumular os elementos lidos para um acumulador.

```
let rec inverte l = fold_left (fun a el → el::a) [] l
```

5. A função *length* é igualmente trivial.

```
let rec length l =
  match l with
  | [] → 0
  | _::li → 1 + length li
```

ou até mesmo:

```
let rec length l = fold_left (fun a _ -> a+1) 0 l
```

6. Queremos demonstrar por indução estrutural sobre as listas que

$$\forall l :\ 'a\ list,\ length\ l = \ length\ (inverte\ l)$$

Por indução estrutural sobre as listas, basta considerar dois casos. O caso de base em que a lista é vazia e o caso indutivo em que a lista é construída com base num elemento e de uma outra lista

(B) Caso de base. Será $length\ [] = \ length\ (inverte\ [])$

Por definição da função $inverte$, $inverte\ [] = []$, logo $length\ [] = length\ (inverte\ []) = 0$. Caso demonstrado.

(I) Caso indutivo. Seja a um elemento, e l' uma lista tal que (HI - Hipótese de indução)

$$(HI)\ \ length\ l' = \ length\ (inverte\ l')$$

Queremos demonstrar que

$$length\ (a :: l') = \ length\ (inverte\ (a :: l'))$$

Por definição da função $inverte(a :: l') = (inverte\ l')@[a]$.

Assim $(length\ ((inverte\ l')@[a])) = length(inverte\ l') + length[a] = length(inverte\ l') + 1$

Por (HI) $length(inverte\ l') + 1 = lengthl' + 1$

Igualmente por definição da função $length$, temos $length\ (a :: l') = 1 + length\ l'$

Ou seja, $length\ (a :: l') = 1 + length\ l' = length(inverte\ l') + 1 = (length\ ((inverte\ l')@[a]))$. Como queríamos demonstrar.

Para concluir, demonstrado o caso de base e o caso inductivo, podemos afirmar

$$\forall l :\ 'a\ list,\ length\ l = \ length\ (inverte\ l)$$

□

Exercício 2 (Alguém copiou nesta sala!)
O professor chamou os cincos alunos suspeitos e questionou cada um deles para saber quem deles cometeu esta ínfame afronta. O resumo das intervenções é o seguinte:

- *António: Não foi o Eduardo, foi o Bernardo*
- *Bernardo: Não foi o Carlos, nem foi o Eduardo*
- *Carlos: Foi o Eduardo, não foi o António*
- *David: Foi o Carlos, foi o Bernardo*
- *Eduardo: Foi o David, não foi o António*

Por experiência própria, o professor sabe que cada aluno suspeito mente exactamente uma vez em cada duas afirmações básicas feitas.

Quem copiou? Justifique.

Resposta: A tese é a seguinte: O Carlos copiou. Vamos demonstrar esta tese.

Ao analisar as 5 confissões, vemos que a confissão do Bernardo se destaca das outras: é a única que afirma duas negações.

Vamos tirar proveito deste facto: como o Bernardo mente uma vez, ou foi o Carlos ou foi o Eduardo.

Prossigamos com uma análise por caso (eliminação da disjunção em Dedução Natural) complementada por um raciocínio por absurdo (redução ao absurdo, no contexto da Dedução Natural):

Vamos considerar as duas hipóteses alternativamente. Vamos começar pelo Eduardo

Hipótese: Foi o Eduardo. Mas então neste caso a mentira do Carlos foi dizer que foi o Eduardo. Logo, neste caso, não foi o António. Do facto de saber que não foi o António, depreendemos que foi o David (nos dizeres do Eduardo - já que o Eduardo não mente sobre a responsabilidade do António). Mas se foi o David , então não foi o Eduardo. **Contradição**. Não pode ter sido o Eduardo.

Só Resta o Carlos.

Poderíamos terminar aqui... temos o nosso culpado. Mas só para confirmar, vamos agora explorar o outro caso.

Hipótese: Foi o Carlos. Então o David mente quando diz que foi o Bernardo. Não foi o Bernardo. Logo o António tem razão quando diz que não foi o Eduardo (mentiu na outra afirmação que incrimina o Bernardo). Se não foi o Eduardo, o Carlos diz a verdade quando diz que não foi o António. Neste caso o Eduardo diz a verdade quando diz que não foi o António.

Conclusão: se emitirmos a hipótese de que o culpado é o Carlos, conseguimos confirmar que por consequência não foi mais nenhuma outro.

Conclusão: Culpado confirmado, foi o Carlos.

□

Exercício 3 (Sintaxe)

1. *Dê e apresente os cálculos dos conjuntos das variáveis livres e mudas da seguinte fórmula* φ

$$R(x,y) \lor \forall x.((\forall y.R(z,x) \to Q(x)) \land (P(x,y) \to Q(z)))$$

2. *Calcule, se possível,* $\varphi\{f(y)/x\}$.

<u>**Resposta**</u>:

1. As variáveis livres são $R(x,y) \lor \forall x.((\forall y.R(z,x) \to Q(x)) \land (P(x,y) \to Q(z)))$ são as seguintes:

$$\begin{aligned}
& VL(\ R(x,y) \lor \forall x.((\forall y.R(z,x) \to Q(x)) \land (P(x,y) \to Q(z)))\ )\ = \\
& VL(\ R(x,y)\ ) \cup VL(\ \forall x.((\forall y.R(z,x) \to Q(x)) \land (P(x,y) \to Q(z)))\ )\ = \\
& \{x,y\} \cup VL(\ (\forall y.R(z,x) \to Q(x)) \land (P(x,y) \to Q(z))\ ) \setminus \{x\}\ = \\
& \{x,y\} \cup (VL(\ (\forall y.R(z,x) \to Q(x))\ ) \cup VL(\ (P(x,y) \to Q(z))\ )) \setminus \{x\}\ = \\
& \{x,y\} \cup (VL(\ (R(z,x) \to Q(x)\ ) \setminus \{y\} \cup \{x,y,z\}) \setminus \{x\}\ = \\
& \{x,y\} \cup (\{x,z\} \setminus \{y\} \cup \{x,y,z\}) \setminus \{x\}\ =\ \{x,y\} \cup \{y,z\}\ = \\
& \{x,y,z\}
\end{aligned}$$

as variáveis ligadas (mudas) são:

$$\begin{aligned}
& VM(\ R(x,y) \lor \forall x.((\forall y.R(z,x) \to Q(x)) \land (P(x,y) \to Q(z)))\ )\ = \\
& VM(\ R(x,y)\ ) \cup VM(\ \forall x.((\forall y.R(z,x) \to Q(x)) \land (P(x,y) \to Q(z)))\ )\ = \\
& \{\} \cup VM(\ \forall x.((\forall y.R(z,x) \to Q(x)) \land (P(x,y) \to Q(z)))\ )\ = \\
& VM(\ (\forall y.R(z,x) \to Q(x)) \land (P(x,y) \to Q(z))\ ) \cup \{x\}\ = \\
& (VM(\ R(z,x) \to Q(x)\ ) \cup \{y\} \cup VM(\ (P(x,y) \to Q(z))\ )) \cup \{x\}\ = \\
& \{\} \cup \{y\} \cup \{\} \cup \{x\}\ =\ \{x,y\}
\end{aligned}$$

2. A substituição $\varphi\{f(y)/x\}$ é possível. Em todos os termos em que x é livre (nomeadamente $R(x, y)$), y também. Logo não há risco de captura de variáveis.

   $R(x, y) \vee \forall x.((\forall y.R(z, x) \to Q(x)) \wedge (P(x, y) \to Q(z)))\ \{f(y)/x\} =$
   $R(f(y), y) \vee \forall x.((\forall y.R(z, x) \to Q(x)) \wedge (P(x, y) \to Q(z)))$

□

Exercício 4 (Cálculo booleano)
Tendo em conta as definições de base dos operadores da álgebra de Boole demonstre:

$$\ominus(b_1 \oplus b_2) = (\ominus b_1) \otimes (\ominus b_2)$$

Resposta: Este resultado por demonstrar é uma das duas leis de De Morgan. Como requerido, vamos demonstrá-la com base nas definições dos operadores booleanos (i.e. $\ominus, \oplus$ e $\otimes$). Obviamente, o exercício explicita que não se pode usar as proprias leis de De Morgan para demonstrar... as leis de De Morgan.

Façamos uma análise por caso.

Caso $b_1 = 0$. Neste caso, $\ominus(0 \oplus b_2) = \ominus b_2$ e $(\ominus 0) \otimes (\ominus b_2) = \ominus b_2$

ou seja $\ominus(0 \oplus b_2) = (\ominus 0) \otimes (\ominus b_2)$, CQD.

Caso $b_1 = 1$. Neste caso $\ominus(1 \oplus b_2) = \ominus 1 = 0$ e $(\ominus 1) \otimes (\ominus b_2) = 0$. Assim $\ominus(1 \oplus b_2) = (\ominus 1) \otimes (\ominus b_2)$. CQD.

Conclusão: $\ominus(b_1 \oplus b_2) = (\ominus b_1) \otimes (\ominus b_2)$

□

Exercício 5 (Consequência Semântica)
Considere a afirmação seguinte:

$$\{\varphi \to (\delta \wedge \psi), (\gamma \vee \psi) \to \delta\} \models \delta \to \varphi$$

Verifique se esta é verdadeira ou falsa, usando o método clássico (via projecção para a álgebra de Boole).

Resposta: Usar o método semântico clássico implica perceber se todas as valorações que satisfaçam $\varphi \to (\delta \wedge \psi)$ e $(\gamma \vee \psi) \to \delta$ satisfaçam igualmente $\delta \to \varphi$. Vejamos.

Resposta curta: Consideremos a valoração V tal que $V(\varphi) = 0$, $V(\delta) = 1$, $V(\gamma) = 1$, $V(\psi) = 1$.

$V(\varphi \to (\delta \wedge \psi)) = 1$

$V((\gamma \vee \psi) \to \delta) = 1$

$V(\delta \to \varphi) = 0$

Esta valoração é um contra-exemplo à validade da consequência semântica enunciada. A afirmação é falsa.

Resposta alternativa, mais em detalhe:

$V(\varphi \to (\delta \wedge \psi)) = \ominus V(\varphi) \oplus (V(\delta) \otimes V(\psi)) = (\ominus V(\varphi) \oplus V(\delta)) \otimes (\ominus V(\varphi) \oplus V(\psi))$

$V((\gamma \vee \psi) \to \delta) = \ominus(V(\gamma) \oplus V(\psi)) \oplus V(\delta) = (\ominus V(\gamma) \otimes \ominus V(\psi)) \oplus V(\delta) = (\ominus V(\gamma) \oplus V(\delta)) \otimes (\ominus V(\psi) \oplus V(\delta))$

$V(\delta \to \varphi) = \ominus V(\delta) \oplus V(\varphi)$

Desta imediata projecção para o domínio dos booleanos, depreende-se que a valoração da primeira fórmula é positiva nos casos:

- ou $V(\varphi) = 1$, neste caso temos necessariamente $V(\delta) = 1$ e $V(\psi) = 1$ para que a formlula seja satisfeita;
- ou então basta $V(\varphi) = 0$ (δ e ψ não influenciam neste caso aqui o estatuto da primeira fórmula)

No caso da segunda fórmula, as valorações que a validam tem a seguinte forma:

- ou $V(\delta) = 0$ e neste caso temos necessariamente $V(\varphi) = 0$ e $V(\gamma) = 0$ para que a formlula seja satisfeita;
- ou então basta que $V(\delta) = 1$

para que as duas fórmulas sejam simultaneamente satisfeita a valoração deve respeitar as seguintes condições (resultante da junção dos casos anteriores, olhando só para as variáveis envolvidas δ e φ na fórmula $\delta \to \varphi$):

- $V(\delta) = 1$ então $V(\varphi) = 1$ necessariamente para podermos satisfazer a primeira fórmula. Esta valoração satisfaz igualmente a segunda fórmula.

  Assim, com $V(\delta) = 1$ e $V(\varphi) = 1$, $V(\delta \to \varphi) = 1$. Ok, neste caso a consequência semântica se verifica localmente. Mas tem de se verificar para todos os casos para poder se afirmar globalmente que há consequência semântica.
- Se $V(\varphi) = 0$, $V(\delta)$ pode ter qualquer valor para satisfazer a primeira fórmula. Neste caso se $V(\delta) = 1$ a validade da segunda fórmula é garantida. Se $V(\delta) = 0$, temos de ter $V(\psi) = V(\gamma) = 0$ para garantir a validade da segunda fórmula.

  no caso $V(\varphi) = 0$ e $V(\delta) = 0$ (e as valorações para γ e ψ escolhidas de forma a validar as duas primeiras fórmulas), $V(\delta \to \varphi) = 1$.

  no caso $V(\varphi) = 0$ e $V(\delta) = 1$, $V(\delta \to \varphi) = 0$. **CQD: a consequência semântica não se confirma. A afirmação é falsa.**

$\square$

Exercício 6 (Algoritmo de Horn)
Considere a fórmula φ seguinte: $(\neg p \lor q) \land (p \lor \neg q)$

Usando o algoritmo de Horn sobre uma eventual transformação de φ, indique a natureza da fórmula φ.

Resposta: Este exercício tem uma resolução imediata.

Observemos primeiro que a fórmula está numa forma adequada para o algoritmo de Horn: é uma FNC em que há no máximo um literal positivo em cada grupo disjuntivo.

De facto, pode ser transformado em $(p \to q) \land (q \to p)$.

$\mathcal{A}(\{(p \to q), (q \to p)\}, \{\top\}) \ = \ \{\top\}$

Ora $\bot \notin \{\top\}$, logo $\mathcal{H}(\varphi) = 1$ e a fórmula em questão é possível.

$\square$

Exercício 7 (Dedução Natural)
Considere a afirmação seguinte: $\vdash (\neg q \to \neg p) \to (p \to q)$

Prove, com recurso à dedução natural, que é um teorema.

Resposta: A estratégia da demonstração é a seguinte:

Demonstrar que $(\neg q \to \neg p) \to (p \to q)$ é um teorema pode-se reduzir a demonstração de que se consegue demonstrar q tendo as provas de $\neg q \to \neg p$, e de p (é o que fazem as regras de introdução do implica da árvore de prova apresentada).

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg q \to \neg p, p, \neg q \vdash \neg q \to \neg p}\,\text{Ax} \quad \dfrac{}{\neg q \to \neg p, p, \neg q \vdash \neg q}\,\text{Ax}}{\neg q \to \neg p, p, \neg q \vdash \neg p}\,e{\to} \quad \dfrac{}{\neg q \to \neg p, p, \neg q \vdash p}\,\text{Ax}}{\neg q \to \neg p, p, \neg q \vdash \bot}}{\neg q \to \neg p, p \vdash q}\,\text{RAA (sobre } \neg q)}{\neg q \to \neg p \vdash p \to q}\,i{\to}}{\vdash (\neg q \to \neg p) \to (p \to q)}\,i{\to}
$$

Como se pode provar q, tendo $\neg q \to \neg p$ e p? Basta observar que tendo p demonstrado, não se pode ter $\neg q$. Isto porque neste caso a relação de causa consequência da qual se tem a demonstração $(\neg q \to \neg p)$ obrigaria a que tenhamos também uma demonstração de $\neg p$. Ou seja... existe uma demonstração de q. Só temos de a encontrar.

É este facto que vamos aproveitar. Provemos que não podemos ter $\neg q$ sem que isso cause uma contradição. Desta contradição temos logo q.

Este raciocínio é dado pelo uso da regra RAA. Admitindo que se tem uma prova de $\neg q$, demonstremos o absurdo.

Observemos agora o contexto do que sabemos: claramente sabemos que podemos deduzir que p é demonstrado (directo), mas que $\neg p$ também, por consequência de $\neg q$ - tal como queríamos.

Assim façamos. O absurdo pode ser demostrado porque conseguimos demonstrar p e $\neg p$. Usemos a regar de eliminação da negação para este propósito.

Demonstrar p é trivial neste ponto (é conhecimento adquirido). Usamos a regra do Axioma.

Demonstrar $\neg p$ necessita que se exiba que é a consequência de $\neg q$. É o que a regra de eliminação do implica (*Modus Ponens*) faz. A aplicação da regra do Axioma termina a construção da árvore de prova.

□