UNIVERSIDAD AUTÓNOMA DE MÉXICO

Facultad de Ciencias

Autor: Adrián Aguilera Moreno

Lógica Computacional

Semanal 3

Para cada uno de los siguientes ejercicios, **justifica ampliamente** tu respuesta.

1. **Realiza** las siguientes sustituciones **eliminando** los paréntesis superfluos en el resultado y **mostrando paso a paso** el procedimiento.

   a) $((q \vee r)[q, p := \neg p, s] \to (r \wedge \neg(r \leftrightarrow p)))[p, r, q := r \vee q, q \wedge p, s]$.

   b) $(u \vee t) \to (\neg r \leftrightarrow (u \leftrightarrow s))[r, u, t := u, t, r]$.

   $\triangledown$ **Solución:** Analizando los incisos tenemos que

   a) Sea $\alpha = [q, p := \neg p, s]$ y $\beta = [p, r, q := r \vee q, q \wedge p, s]$, entonces

$$\begin{aligned}
((q \vee r)\alpha \to (r \wedge \neg(r \leftrightarrow p)))\beta &= ((q\alpha \vee r\alpha) \to (r \wedge \neg(r \leftrightarrow p)))\beta \\
&= ((\neg p \vee r) \to (r \wedge \neg(r \leftrightarrow p)))\beta \\
&= ((\neg p \vee r)\beta \to (r \wedge \neg(r \leftrightarrow p))\beta) \\
&= \neg p\beta \vee r\beta \to r\beta \wedge \neg(r \leftrightarrow p)\beta \\
&= \neg(r \vee q) \vee (q \wedge p) \to (q \wedge p) \wedge \neg(r\beta \leftrightarrow p\beta) \\
&= \neg(r \vee q) \vee (q \wedge p) \to (q \wedge p) \wedge \neg(q \wedge p \leftrightarrow r \vee q)
\end{aligned}$$

   b) Sea $\alpha = [r, u, t := u, t, r]$, entonces

$$\begin{aligned}
(u \vee t) \to (\neg r \leftrightarrow (u \leftrightarrow s))\alpha &= (u \vee t) \to (\neg r\alpha \leftrightarrow (u \leftrightarrow s)\alpha) \\
&= (u \vee t) \to (\neg u \leftrightarrow (u\alpha \leftrightarrow s\alpha)) \\
&= (u \vee t) \to (\neg u \leftrightarrow (t \leftrightarrow s)) \\
&= u \vee t \to (\neg u \leftrightarrow (t \leftrightarrow s))
\end{aligned}$$

   $\triangleleft$

2. *a)* **Define recursivamente** la función pa que dada una fórmula φ, devuelve el número de paréntesis abiertos "(" que tiene φ.

   b) **Define recursivamente** la función pc que dada una fórmula φ, devuelve el número de paréntesis cerrados ")" que tiene φ.

   c) Sea $\varphi = (((\neg p \wedge q) \vee \neg r) \to r)$. Demuestra que

$$pa(\varphi) - pc(\varphi) = 0$$

$\triangledown$ **Solución:**

a) Definimos $pa : PROP \cup \{\wedge, \vee, \neg, \to, \leftrightarrow, (,)\} \to \mathbb{N} \cup \{0\}$. Así, para

$$\varphi \in ATOM \cup \{\wedge, \vee, \neg, \to, \leftrightarrow,)\}$$

se tiene que $pa(\varphi) = 0$. También, $pa\big(()\big) = 1$. Para $\alpha, \mu \in PROP$ se tiene que

$$\begin{aligned}
pa\big((\alpha)\big) &= pa\big(()\big) + pa(\alpha) + pa\big()\big) \\
pa(\neg\alpha) &= pa(\neg) + pa(\alpha) \\
pa(\alpha * \mu) &= pa(\alpha) + pa(*) + pa(\mu)
\end{aligned}$$

donde $* \in \{\wedge, \vee, \neg, \to, \leftrightarrow\}$.

b) Definimos $pc : PROP \cup \{\wedge, \vee, \neg, \rightarrow, \leftrightarrow, (,)\} \rightarrow \mathbb{N} \cup \{0\}$. Así, para

$$\varphi \in ATOM \cup \{\wedge, \vee, \neg, \rightarrow, \leftrightarrow, (\}$$

se tiene que $pc(\varphi) = 0$. También, $pc\big()\big) = 1$. Para $\alpha, \mu \in PROP$ se tiene que

$$\begin{array}{rcl} pc\big((\alpha)\big) & = & pc\big(()+pc(\alpha)+pc\big()\big) \\ pc(\neg\alpha) & = & pc(\neg)+pc(\alpha) \\ pc(\alpha * \mu) & = & pc(\alpha)+pc(*)+pc(\mu) \end{array}$$

donde $* \in \{\wedge, \vee, \neg, \rightarrow, \leftrightarrow\}$.

c) Veamos que

$$\begin{array}{rcl} pa\big((((\neg p \wedge q) \vee \neg r) \rightarrow r)\big) & = & pa\big(()+pa\big(((\neg p \wedge q) \vee \neg r) \rightarrow r\big)+pa\big()\big) \\ & = & 1+pa\big(((\neg p \wedge q) \vee \neg r)\big)+pa(\rightarrow)+pa(r) \\ & = & 1+pa\big(()+pa\big((\neg p \wedge q)\big)+pa(\vee)+pa(\neg r)+pa\big()\big) \\ & = & 2+pa\big(()+pa\big(\neg p \wedge q\big)+pa\big()\big)+pa(\neg)+pa(r) \\ & = & 3+pa(\neg p)+pa(\wedge)+pa(q) \\ & = & 3+pa(\neg)+pa(p) \\ & = & 3 \end{array}$$

además, se tiene que

$$\begin{array}{rcl} pc((((\neg p \wedge q) \vee \neg r) \rightarrow r)) & = & pc\big(()+pc\big(((\neg p \wedge q) \vee \neg r) \rightarrow r\big)+pc\big()\big) \\ & = & pc\big(((\neg p \wedge q) \vee \neg r)\big)+pc(\rightarrow)+pc(r)+1 \\ & = & pc\big(()+pc\big((\neg p \wedge q)\big)+pc(\vee)+pc(\neg r)+pc\big()\big)+1 \\ & = & pc\big(()+pc\big(\neg p \wedge q\big)+pc\big()\big)+pc(\neg)+pc(r)+2 \\ & = & pc(\neg p)+pc(\wedge)+pc(q)+3 \\ & = & pc(\neg)+pc(p)+3 \\ & = & 3 \end{array}$$

Luego, es claro que

$$\begin{array}{rcl} pa((((\neg p \wedge q) \vee \neg r) \rightarrow r)) - pc((((\neg p \wedge q) \vee \neg r) \rightarrow r)) & = & 3-3 \\ & = & 0 \end{array}$$

$\triangleleft$

Desafío extra...

- **Define recursivamente** una función compress que elimina los elementos consecutivos repetidos de una lista.

  Ejemplo:

  `compress([a,a,a,a,i,i,i,u,d,d,d,d,d,a,a,a,a,a,a,a]) = [a,i,u,d,a]`

  $\triangledown$ **Solución:** Para este inciso definimos una función auxiliar que nos indique si un elemento arbitrario esta contenido en una lista, esto es

$$\texttt{contains(x:xs, a)}^1 = \begin{cases} \texttt{true} & \text{si x} = \text{a} \\ \texttt{false} & \text{si x} = \phi \\ \texttt{contains(s, a)} & \text{si x} \neq \text{a} \end{cases}$$

[1] `contains:[_] → [_]`.

Ahora definamos la función `compress:[_] → [_]`, esto es

$$\texttt{compress(x:xs)} = \begin{cases} \texttt{compress(s)} & \text{si } \texttt{contains(s, x)} = \texttt{true} \\ \texttt{x ++ compress(s)} & \text{si } \texttt{contains(s, x)} = \texttt{false} \end{cases}$$

◁

– **Demuestra**, usando tu definición, que:

`compress([1, 2, 2, 3, 3, 3]) = [1, 2, 3]`

Demostración: Veamos la ejecución de `compress` en las siguientes líneas:

$$\begin{aligned} \texttt{compress}\big(\texttt{[1, 2, 2, 3, 3, 3]}\big) &= \texttt{1 ++ compress}\big(\texttt{[2, 2, 3, 3, 3]}\big) \\ &= \texttt{1 ++ compress}\big(\texttt{[2, 3, 3, 3]}\big) \\ &= \texttt{1 ++ 2 ++ compress}\big(\texttt{[3, 3, 3]}\big) \\ &= \texttt{1 ++ 2 ++ compress}\big(\texttt{[3, 3]}\big) \\ &= \texttt{1 ++ 2 ++ compress}\big(\texttt{[3]}\big) \\ &= \texttt{1 ++ 2 ++ 3} \\ &= \texttt{[1, 2, 3]} \end{aligned}$$

Cabe destacar que por cada llamada a `compress(x)` se llama a lo más $\frac{|x|\cdot(|x|+1)}{2}$ veces a la función `contains` o al menos 1 vez. □