Logica per l'Informatica

Ricorsione ed Induzione strutturale

12/12/2023

Esercizio 1: Riscaldamento booleano.
Ricorda la definizione dei booleani come tipi induttivi[1]:

$$\texttt{Bool} ::= \texttt{true} \mid \texttt{false}$$

Ricorda la funzione ricorsiva strutturale $\texttt{not} : \texttt{Bool} \to \texttt{Bool}$ definita da:

$$\texttt{not}\,\texttt{true} := \texttt{false}$$

$$\texttt{not}\,\texttt{false} := \texttt{true}.$$

1. Definisci una funzione ricorsiva strutturale[2] $\texttt{and} : \texttt{Bool} \to \texttt{Bool} \to \texttt{Bool}$ che presi in input $b_1 : \texttt{Bool}$ e $b_2 : \texttt{Bool}$ ritorna in output il booleano corrispondente alla "e" booleana delle usuali tavole di verità classiche. Per esempio, $\texttt{and}\,\texttt{false}\,\texttt{true} = \texttt{false}$.

   Definisci una funzione ricorsiva strutturale $\texttt{or} : \texttt{Bool} \to \texttt{Bool} \to \texttt{Bool}$ che presi in input $b_1 : \texttt{Bool}$ e $b_2 : \texttt{Bool}$ ritorna in output il booleano corrispondente alla "o" booleana delle usuali tavole di verità calssiche. Per esempio, $\texttt{or}\,\texttt{false}\,\texttt{true} = \texttt{true}$.

2. Dimostra il seguente[3]:

   Theorem 1

   $$\forall b_1 : \texttt{Bool}.\, \forall b_2 : \texttt{Bool}. \qquad \texttt{not}\,(\texttt{and}\,b_1\,b_2) = \texttt{or}\,(\texttt{not}\,b_1)\,(\texttt{not}\,b_2).$$

[1] Ricorda che si intende che come assioma abbiamo sempre la formula: $\texttt{true} \neq \texttt{false}$.

[2] L'induzione strutturale come l'avete vista nel corso prevede che si vada per induzione solo sul primo argomento della funzione. Quindi dovete pensare a come definire la "e" di due booleani analizzando solo le forme del primo booleano. Analogo per la "o", che è richiesta nelle linee sotto.

[3] Questo esprime, in termini di tavole di verità (dunque di semantica classica), una delle leggi di De Morgan che abbiamo gia dimostrato in DN!

Esercizio 2: Riscaldamento aritmetico.

Ricorda la definizione dei numeri naturali come tipo induttivo:

$$\texttt{Nat} ::= \texttt{O} \mid \texttt{S}\,\texttt{Nat}$$

e come sempre, diciamo che il termine $\texttt{O}$ rappresenta il numero naturale $0 \in \mathbb{N}$, il termine $\texttt{1} := \texttt{S}\,\texttt{O}$ rappresenta $1 \in \mathbb{N}$, il termine $\texttt{2} := \texttt{S}\,(\texttt{S}\,\texttt{O})$ rappresenta $2 \in \mathbb{N}$ ecc (si tratta della rappresentazione unaria dei naturali).

Ricorda la funzione ricorsiva strutturale $+ : \texttt{Nat} \to \texttt{Nat} \to \texttt{Nat}$ che avete visto in aula, che calcola il valore di tipo $\texttt{Nat}$ corrispondente alla somma dei numeri naturali corrispondenti ai due input di tipo $\texttt{Nat}$:

$$\begin{array}{lcl} \texttt{O} + m & := & m \\ (\texttt{S}\,n) + m & := & \texttt{S}\,(n + m). \end{array}$$

1. Definisci una funzione ricorsiva strutturale $\texttt{isZero} : \texttt{Nat} \to \texttt{Bool}$ che ritorna $\texttt{true}$ sse l'input è $\texttt{O}$ e ritorna $\texttt{false}$ altrimenti[4].

   Dimostra il seguente[5].

   Theorem 2

   $$\forall n : \texttt{Nat}. \qquad (n \neq \texttt{O}) \leftrightarrow (\texttt{isZero}\,n = \texttt{false}).$$

2. Definisci una funzione ricorsiva strutturale $\cdot : \texttt{Nat} \to \texttt{Nat} \to \texttt{Nat}$ che calcola il valore $n \cdot m : \texttt{Nat}$ corrispondente al prodotto dei numeri naturali corrispondenti ad input $n : \texttt{Nat}$ e $m : \texttt{Nat}$.

3. Dimostra il seguente:

   Theorem 3

   $$\forall n : \texttt{Nat}. \qquad n \cdot \texttt{O} = \texttt{O}.$$

Esercizio 3: Il tipo delle formule.

Definiamo un nuovo tipo induttivo $\texttt{Formula}$ come segue:

$$\texttt{Formula} ::= x_1 \mid x_2 \mid x_3 \mid \texttt{Formula} \wedge \texttt{Formula} \mid \texttt{Formula} \vee \texttt{Formula} \mid \neg\texttt{Formula}$$

Per esempio, $\varphi_0 := x_2 \wedge (x_3 \vee \neg(x_1 \wedge x_2)) \wedge (\neg x_1 \vee x_1)) : \texttt{Formula}$.

[4] Si, è banale... ma ad essere pedanti bisogna fornire una dimostrazione del fatto che la funzione che hai dato verifica davvero le condizioni richieste. Questo è proprio ciò che il Teorema 3 chiede di fare, ed in questo caso sarà banale.

[5] Ricorda che $\leftrightarrow$ è sempre zucchero sintattico per la congiunzione delle due implicazioni opposte.

L'idea è che ad ogni valore del tipo `Formula` si può essere associare un booleano, che è il suo valore di verità, una volta che abbiamo specificato un valore di verità (un booleano) per ogni variabile x_1, x_2, x_3.

Questo tipo rappresenta allora le formule proposizionali (senza implicazione) con tre proposizioni atomiche non specificate (le variabili x_1, x_2, x_3), dove le forme $\wedge, \vee$ e $\neg$ avranno i loro soliti significati delle tavole di verità della logica classica.

Per esempio, se decido di associare `true` a x_1, `true` a x_2 e `false` a x_3, allora il valore di verità di φ_0 di sopra si calcola sostituendo alle variabili i loro valori di verità scelti, e poi eseguendo il calcolo booleano ottenuto: $\texttt{true} \wedge ((\texttt{false} \vee \neg(\texttt{true} \wedge \texttt{true})) \wedge (\neg\texttt{true} \vee \texttt{true}))$ e poi leggendo $\wedge, \vee$ e $\neg$ come, rispettivamente, la "e", la "o" e la negazione booleana. Facendo il conto, vediamo che il valore di verità associato a φ_0 con quella scelta di valori di verità per le variabili, è `false`.

Formalizziamo ora questa idea:

1. Definisci la funzione ricorsiva strutturale

$$\texttt{truth} : \texttt{Formula} \to \texttt{Bool} \to \texttt{Bool} \to \texttt{Bool} \to \texttt{Bool}$$

   tale che $\texttt{truth}\,\varphi\,b_1\,b_2\,b_3$ sia il valore booleano di verità associato a φ, data la assegnazione di valori di verità seguente: b_1 a x_1, b_2 a x_2 e b_3 a x_3.

2. Sia $\texttt{GG}$[6] : $\texttt{Formula} \to \texttt{Formula}$ la funzione ricorsiva strutturale definita da:

$$\begin{array}{lcl} \texttt{GG}\,x_1 & := & x_1 \\ \texttt{GG}\,x_2 & := & x_2 \\ \texttt{GG}\,x_3 & := & x_3 \\ \texttt{GG}\,(\neg\varphi) & := & \neg\texttt{GG}\,\varphi \\ \texttt{GG}\,(\varphi_1 \wedge \varphi_2) & := & \texttt{GG}\,\varphi_1 \wedge \texttt{GG}\,\varphi_2 \\ \texttt{GG}\,(\varphi_1 \vee \varphi_2) & := & \neg(\neg(\texttt{GG}\,\varphi_1) \wedge \neg(\texttt{GG}\,\varphi_2)). \end{array}$$

   2.1)

   Verifica che si ha: $\texttt{GG}\,(x_1 \vee \neg x_1) = \neg(\neg x_1 \wedge \neg\neg x_1)$.

   2.2)

   Dimostra il seguente[7]:

   Theorem 4

   $\forall\varphi : \textit{Formula}.\, \forall b_1 : \textit{Bool}.\, \forall b_2 : \textit{Bool}.\, \forall b_3 : \textit{Bool}.\quad \textit{truth}\,(\textit{GG}\,\varphi)\,b_1\,b_2\,b_3 = \textit{truth}\,\varphi\,b_1\,b_2\,b_3.$

[6] "`GG`" sta per "Gödel-Gentzen-translation", una delle cosiddette "double-negation translations", che sono delle famose trasformazioni di formule (e più generali di quella che scrivo qui). La proprietà fondamentale di queste traduzioni è che se una formula φ è dimostrabile in DN *classica*, allora $\texttt{GG}\,\varphi$ è dimostrabile in DN *intuizionista*! Per esempio, come sapete bene, il terzo escluso $x_1 \vee \neg x_1$ è dimostrabile classicamente ma *non* intuizionisticamente, eppure $\texttt{GG}\,(x_1 \vee \neg x_1)$ è dimostrabile intuizionisticamente, come potete immediatemente verificare tramite un albero di DN intuizionista.

[7] Ricorda che hai a disposizione il Teorem 1, ed anche il teorema: $\forall b : \texttt{Bool}.\, \texttt{not}\,(\texttt{not}\,b) = b$.

Esercizio 4: Il tipo delle esepressioni aritmetiche.

Definiamo un nuovo tipo induttivo `Expr` come segue:

$$\texttt{Expr} ::= \texttt{num Nat} \mid \texttt{add Expr Expr} \mid \texttt{mult Expr Expr}$$

Per esempio, i seguenti sono valori di questo tipo:

$\texttt{num 3} : \texttt{Expr}$.

$e_0 := \texttt{add}\,(\texttt{mult}\,(\texttt{num 6})\,(\texttt{num 1}))\,(\texttt{add}\,(\texttt{num 2})\,(\texttt{num 0})) : \texttt{Expr}$.

L'idea è che ad ogni valore del tipo `Expr` si può associare un valore di tipo `Nat`, che ne rappresenta il suo "valore numerico", interpretando in maniera ovvia ogni costruttore del tipo `Expr` come la rispettiva operazione aritmetica (e `num` indica semplicemente che abbiamo gia a che fare con un valore numerico), e poi facendo il calcolo aritmetico corrispondente.

Questo tipo rappresenta allora le espressioni aritmetiche (formate da somma e prodotto).

Per esempio, il valore numerico dell'espressione $e_0 : \texttt{Expr}$ di sopra, vale $\texttt{8} : \texttt{Nat}$.

1. Formalizziamo l'idea di sopra: ricordando le funzioni dell'esercizio 2, definisci una funzione ricorsiva strutturale $\texttt{eval} : \texttt{Expr} \to \texttt{Nat}$ che ritorna il valore di tipo `Nat` corrispondente al calcolo di una espressione aritmetica di tipo `Expr` in input, come descritto sopra.

   Definisci anche una funzione $\texttt{if} : \texttt{Bool} \to \texttt{Expr} \to \texttt{Expr} \to \texttt{Expr}$ ricorsiva strutturale tale tale che[8] $\texttt{if}\,b\,e_1\,e_2 = e_1$ se $b = \texttt{true}$ e $\texttt{if}\,b\,e_1\,e_2 = e_2$ se $b = \texttt{false}$.

2. Ricordando le funzioni dell'esercizio 1 e 2, definisci una funzione ricorsiva strutturale $\texttt{occZ} : \texttt{Expr} \to \texttt{Bool}$ che ritorna `true` sse $\texttt{num 0} : \texttt{Expr}$ occorre nell'espressione in input, e `false` altrimenti.

   Per esempio, $\texttt{occZ}\,(\texttt{add}\,(\texttt{mult}\,(\texttt{num 6})\,(\texttt{num 1}))\,(\texttt{add}\,(\texttt{num 2})\,(\texttt{num 0}))) = \texttt{true}$ e $\texttt{occZ}\,(\texttt{add}\,(\texttt{mult}\,(\texttt{num 6})\,(\texttt{num 1}))\,(\texttt{add}\,(\texttt{num 2})\,(\texttt{num 1}))) = \texttt{false}$.

[8] Si, è banale. Anche qui, per essere pedanti, bisognerebbe fornire la dimostrazione del fatto che la funzione che hai scritto verifica davvero le equazioni richieste... ma in questo caso è banale ed allora ci permettiamo di saltarlo per un eccesso di pigrizia.

3. Considera la funzione ricorsiva strutturale $\texttt{semp} : \texttt{Expr} \to \texttt{Expr}$ seguente:

$$
\begin{array}{lcl}
\texttt{semp}\,(\texttt{num}\,n) & := & \texttt{num}\,n \\
\texttt{semp}\,(\texttt{add}\,e_1\,e_2) & := & \texttt{if}\,(\texttt{isZero}\,(\texttt{eval}\,e_1)) \\
 & & \quad (\texttt{semp}\,e_2) \\
 & & \quad (\texttt{if}\,(\texttt{isZero}\,(\texttt{eval}\,e_2)) \\
 & & \quad\quad (\texttt{semp}\,e_1) \\
 & & \quad\quad (\texttt{add}\,(\texttt{semp}\,e_1)\,(\texttt{semp}\,e_2)) \\
 & & \quad) \\
\texttt{semp}\,(\texttt{mult}\,e_1\,e_2) & := & \texttt{if}\,(\texttt{isZero}\,(\texttt{eval}\,e_1)) \\
 & & \quad (\texttt{num}\,\texttt{0}) \\
 & & \quad (\texttt{if}\,(\texttt{isZero}\,(\texttt{eval}\,e_2)) \\
 & & \quad\quad (\texttt{num}\,\texttt{0}) \\
 & & \quad\quad (\texttt{mult}\,(\texttt{semp}\,e_1)\,(\texttt{semp}\,e_2)) \\
 & & \quad).
\end{array}
$$

Sembra complicato, ma in realtà questa funzione fa semplicemente dei check per semplificare l'espressione togliendo ogni sotto-espressione "che fa $\texttt{0}$".

Per esempio, chiamando $e_1 :=: \texttt{mult}\,(\texttt{num}\,\texttt{6})\,(\texttt{num}\,\texttt{0}) : \texttt{Expr}$, $e_2 := \texttt{num}\,\texttt{3} : \texttt{Expr}$ e $e_3 := \texttt{add}\,(\texttt{num}\,\texttt{1})\,(\texttt{num}\,\texttt{0}) : \texttt{Expr}$, si ha:

$$
\begin{array}{lll}
= & \texttt{semp}\,(\texttt{add}\,(\texttt{mult}\,(\texttt{num}\,\texttt{6})\,(\texttt{num}\,\texttt{0}))\,(\texttt{mult}\,(\texttt{num}\,\texttt{3})\,(\texttt{add}\,(\texttt{num}\,\texttt{1})\,(\texttt{num}\,\texttt{0})))) & \\
= & \texttt{semp}\,(\texttt{add}\,e_1\,(\texttt{mult}\,e_2\,e_3)) & \textit{per def. di } e_1, e_2, e_3 \\
= & \texttt{semp}\,(\texttt{mult}\,e_2\,e_3) & \textit{perché } \texttt{eval}\,(e_1) = \texttt{0} \\
= & \texttt{if}\,(\texttt{isZero}\,(\texttt{eval}\,e_3))(\texttt{num}\,\texttt{0})(\texttt{mult}\,(\texttt{semp}\,e_2)\,(\texttt{semp}\,e_3)) & \textit{perché } \texttt{eval}\,(e_2) \neq \texttt{0} \\
= & \texttt{mult}\,(\texttt{semp}\,e_2)\,(\texttt{semp}\,e_3) & \textit{perché } \texttt{eval}\,(e_3) \neq \texttt{0} \\
= & \texttt{mult}\,(\texttt{num}\,\texttt{3})(\texttt{semp}\,(\texttt{add}\,(\texttt{num}\,\texttt{1})\,(\texttt{num}\,\texttt{0}))) & \\
= & \texttt{mult}\,(\texttt{num}\,\texttt{3})\,e_4 & \textit{perché } \texttt{eval}\,(\texttt{num}\,\texttt{1}) \neq \texttt{0} \\
= & \texttt{mult}\,(\texttt{num}\,\texttt{3})\,(\texttt{semp}\,(\texttt{num}\,\texttt{1})) & \textit{perché } \texttt{eval}\,(\texttt{num}\,\texttt{0}) = \texttt{0} \\
= & \texttt{mult}\,(\texttt{num}\,\texttt{3})\,(\texttt{num}\,\texttt{1}) &
\end{array}
$$

con $e_4 := \texttt{if}\,(\texttt{isZero}\,(\texttt{eval}\,(\texttt{num}\,\texttt{0})))\,(\texttt{semp}\,(\texttt{num}\,\texttt{1}))\,(\texttt{add}\,(\texttt{semp}\,(\texttt{num}\,\texttt{1}))\,(\texttt{semp}\,(\texttt{num}\,\texttt{0}))) : \texttt{Expr}$. Ovvero, abbiamo cancellato le parti che si valutavano a $\texttt{0}$.

Usando la seguente formula come assioma:

$$\forall e : \texttt{Expr}\,. \qquad (\texttt{eval}\,e = \texttt{0}) \vee (\texttt{eval}\,e \neq \texttt{0}),$$

dimostra il seguente:

Theorem 5

$$\forall e : \texttt{Expr}\,. \qquad (\texttt{eval}\,e \neq \texttt{0}) \to (\texttt{occZ}\,(\texttt{semp}\,e) = \texttt{false}).$$