Induzione strutturale

Le tecniche di induzione possono essere usate anche per

- Liste
- Alberi binari
- Alberi binari etichettati

Liste

Consentono di memorizzare sequenze omogenee di dati di lunghezza variabile

La lista che contiene la sequenza $a_1, a_2, \ldots, a_k$ viene rappresentata come $[a_1, a_2, \ldots, a_k]$

- La lista vuota è $[\,]$

Liste e insiemi

- Sia liste che insiemi possono contenere elementi
- Simile notazione ma parentesi diverse
- Negli insiemi gli elementi ripetuti non contano, nelle liste sì

$$[0,0,1,2,3] \neq [0,1,2,3] \neq [2,0,1,3]$$
$$\{0,0,1,2,3\} = \{0,1,2,3\} = \{2,0,1,3\}$$

- Le liste sono sempre finite, gli insiemi possono essere infiniti ex. $\mathbb{R}$

Le liste induttivamente

Possiamo costruire qualunque lista partendo dalla lista vuota e con le operazioni a : che aggiungono un elemento in testa

- L'insieme L_A delle liste su A è il più piccolo insieme che soddisfa
  1. Clausola Base $[\,] \in L_A$
  2. Clausola Induttiva $\text{Se } a \in A$ e $lst \in L_a \text{ allora } a : lst \in L_A$
- Notazione: $(b : (e : (a : [\,]))) = b : e : a : [\,] = [b, e, a]$

1. $[\,] \in L_A \rightarrow$ Liste su A
2. $(a \in A \wedge lst \in L_A) \Rightarrow a : lst \in L_A$

Operazioni su Liste

- Lunghezza di una lista $len : L_A \rightarrow \mathbb{N}$
  1. $len([\,]) = 0$
  2. $len(a : lst) = len(lst) + 1$

Valutando $len([a, b, a])$

$$len([a,b,a]) = len(a : [b,a]) = len([b,a]) + 1 =$$
$$len(b : [a]) + 1 = len([a]) + 1 + 1 = len(a : [\,]) + 1 + 1 =$$
$$len([\,]) + 1 + 1 + 1 = 0 + 1 + 1 + 1 = 3$$

- Somma di una lista $sumList : L_{\mathbb{N}} \rightarrow \mathbb{N}$
  1. $sumList([\,]) = 0$
  2. $sumList(n : lst) = sumList(lst) + n$

Valutando $sumList([9, 5, 28])$

$$sumList([9,5,28]) = sumList(9 : [5,28]) = sumList([5,28]) + 9$$
$$sumList(5 : [28]) + 9 = sumList([28])5 + 9 = sumList(28 : [\,]) + 5 + 9$$
$$sumList([]) + 28 + 5 + 9 = 0 + 28 + 5 + 9 = 42$$

- Appartenenza di un elemento alla lista $L_A \times A \rightarrow Bool$
  1. $belList([\,], b) = f, \forall b \in A$
  2. $belList(a : lst, b) = t, \forall a, b \in A$ t.c. $a = b$
  3. $belList(a : lst, b) = belList(lst, b), \forall a, b \in A$ t.c. $a \neq b$

Valutiamo $belList([a, e, a], e)$

$$belList([a,e,a], e) = belList(a : [e,a], e) = belList([e,a], e)$$
$$belList(e : [a], e) = t$$

Valutiamo $belList([a, e, a], b)$

$$belList([a,e,a],b) = belList(a:[e,a],b) = belList([e,a],b)$$
$$belList(e:[a],b) = belList([a],b) = belList(a:[\,],b)$$
$$belList([\,],b) = f$$

- Concatenazione di due liste $app : L_A \times L_A \to L_A$
  1. $app([\,], lst_2) = lst_2$
  2. $app(a : lst_1, lst_2) = a : app(lst_1, lst_2)$

Valutiamo $app([d,f],[a,c,d])$

$$app([d,f],[a,c,d]) = app(d:[f],[a,c,d]) = d : app([f],[a,c,d])$$
$$d : app(f:[\,],[a,c,d]) = d : f : app([], [a,c,d]) = d : f : [a,c,d]$$
$$[d,f,a,c,d]$$

- Inverso di una lista $rev : L_A \to L_A$
  1. $rev([\,]) = [\,]$
  2. $rev(a : lst) = app(rev(lst), a : [\,])$

Valutiamo $rev([b,e,a])$]

$$rev([b,e,a] = rev(b:e:a:[\,]) = app(rev(e:a:[\,]), b:[\,])$$
$$app(app(rev(a:[\,]), e:[\,]), b:[\,]) = app(app(app(rev([\,]), a:[\,]), e:[\,]), b:[\,])$$
$$app(app(app([\,], a:[\,]), e:[\,]), b:[\,]) = app(app(a:[\,], e:[\,]), b:[\,])$$
$$app(a : app([\,], e:[\,]), b:[\,]) = a : app(e:[\,], b:[\,] = a : e : app([\,], b:[\,])$$
$$a:e:b:[] = [a,e,b]$$

Principio di induzione su Liste

Sia P una proprietà su L_A, $P : L_A \to Bool$

- Se (Caso Base) $P([\,])$ è vera
- Passo induttivo
  se
  $P(lst)$ è vera allora anche $P(a : lst)$ lo è, per ogni $lst \in L_A, a \in A$, allora $P(lst)$ è vera per ogni $lst \in L_A$

$$\frac{P([\,]) \ \forall a \in A \,.\, \forall lst^{'} \in L_A \,.\, P(lst^{'}) \Rightarrow P(a : lst^{'})}{\forall lst \in L_A \,.\, P(lst)} \ \text{P.I. su } L_A$$

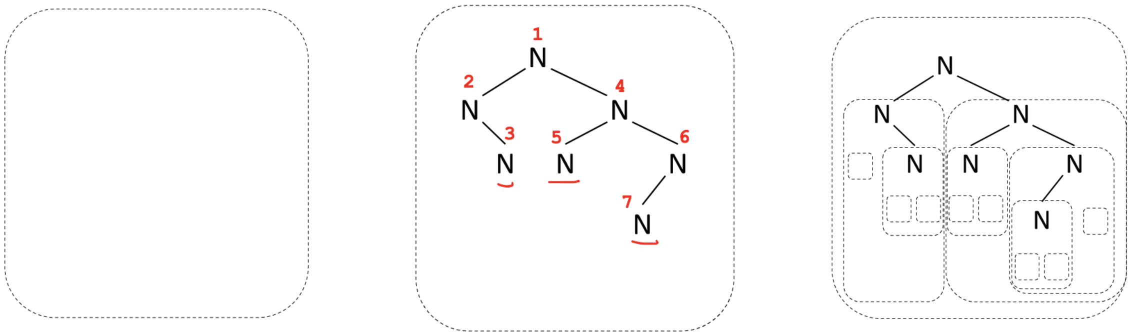
Alberi Binari

- Clausola Base $\lambda \in BT$, l'albero è vuoto
- Clausola Induttiva $\text{Se } t_1, t_2 \in BT \text{ allora } N(t_1, t_2) \in BT$

$N(t_1, t_2)$ è un nodo con due sottoalberi (sinistro e destro)

Foglie $\to$ nodi con entrambi i sottoalberi vuoti (λ)

Alberi radicati cardinali con $k = 2$



1. L'albero vuoto $\lambda \in BT$
2. L'albero radicato

$$N(N(\lambda, N(\lambda, \lambda)), N(N(\lambda, \lambda), N(N(\lambda, \lambda), \lambda)))$$

Funzioni strutturali su Alberi Binari

- Dimensione di un albero binario $size : BT \to \mathbb{N}$
  1. $size(\lambda) = 0$
  2. $size(N(t_1, t_2)) = size(t_1) + size(t_2) + 1$
- Altezza di un albero binario $height(\lambda) : BT \to \mathbb{N} \cup \{-1\}$
  1. $height(\lambda) = -1$
  2. $height(N(t_1, t_2)) = max(height(t_1), height(t_2)) + 1$

Principio di induzione su Alberi Binari

Sia P una proprietà su BT, $P : BT \to Bool$

- Se (Caso Base) $P(\lambda)$ è vera
- Passo Induttivo
  se per ogni
  $t_1, t_2 \in BT$, se $P(t_1)$ e $P(t_2)$ sono vere, allora anche $P(N(t_1, t_2))$ lo è, allora $P(t)$ è vera per ogni $t \in BT$f

$$\frac{P(\lambda) \quad \forall t_1, t_2 \in BT \,.\, (P(t_1) \wedge P(t_2) \Rightarrow P(N(t_1, t_2)))}{\forall t \in BT \,.\, P(t)} \text{ P.I. su } BT$$

Dimostrazione per Induzione Strutturale che $height(t) \leq size(t)$

Definizione di $height(t)$

1. $height(\lambda) = -1$
2. $height(N(t_1, t_2)) = max(height(t_1), height(t_2)) + 1$

Definizione di $size(t)$

1. $size(\lambda) = 0$
2. $size(N(t_1, t_2)) = size(t_1) + size(t_2) + 1$

$$P\lambda) \Rightarrow height(\lambda) \leq size(\lambda) = -1 \leq 0$$
$$P(t_1) \wedge P(t_2) \Rightarrow P(N(t_1, t_2))$$
$$P(N(t_1, t_2)) = height(N(t_1, t_2)) \leq size(N(t_1, t_2))$$

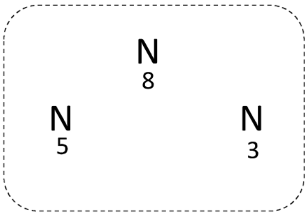
$$\begin{aligned} height(N(t_1, t_2)) = max(height(t_1), height(t_2)) + 1 \leq \\ max(size(t_1), size(t_2)) + 1 \leq \\ size(t_1) + size(t_2) + 1 = size(N(t_1, t_2)) \end{aligned}$$

Alberi Etichettati

Un albero etichettato è un albero che ha

- $\lambda \in BT_A$ l'albero vuoto
- se $t_1, t_2 \in BT_A$ allora $N(t_1, a, t_2) \in BT_A, \ \forall a \in A$

Esempio: $N(N(\lambda, 5, \lambda), 8, N(\lambda, 3, \lambda))$



Operazioni su alberi etichettati

- Appartenenza $belBT : BT_A \times A \rightarrow Bool$
  1. $belBT(\lambda, b) = f$
  2. $belBT(N(t_1, a, t_2), b) = t, \ \forall a, b \in A \ . \ a = b$
  3. $belBT(N(t_1, a, t_2),) = belBT(t_1, b) \vee belBT(t_2, b), \ \forall a, b \in A \ . \ a \not= b$

Valutiamo $belBT(N(N(\lambda, 5, \lambda), 8, N(\lambda, 3, \lambda)), 3)$

$$belBT(N(N(\lambda, 5, \lambda), 8, N(\lambda, 3, \lambda)), 3)$$
$$belBT(N(\lambda, 5, \lambda), 3) \vee belBT(N(\lambda, 3, \lambda)$$
$$belBT(\lambda, 3) \vee belBT(\lambda, 3) \vee t$$
$$f \vee f \vee t = t$$

- Scansione sequenziale dei nodi, secondo una certa strategia $visit : BT_A \rightarrow L_A$
  1. $visit(\lambda) = [\,]$
  2. $visit(N(t_1, a, t_2)) = app(visit(t_1), a : visit(t_2))$

$$
\begin{aligned}
&visit(N(N(\lambda,5,\lambda),8,N(\lambda,3,\lambda)))\\
&= app(visit(\lambda,5,\lambda),8 : visit(N(\lambda,3,\lambda))\\
&app(app(visit(\lambda),5 : visit(\lambda)),8 : app(visit(\lambda),3 : visit(\lambda)))\\
&app(app([\,],5 : [\,]),8 : app([\,],3 : [\,]))\\
&app(5 : [\,],8 : 3 : [\,])\\
&5 : 8 : 3 : [\,]\\
&[5,8,3]
\end{aligned}
$$

- Somma delle etichette di un albero etichettato $sumBT : BT_{\mathbb{N}} \to \mathbb{N}$
    1. $sumBT(\lambda) = 0$
    2. $sumBT(N(t_1,a,t_2)) = sumBT(t_1) + sumBT(t_2) + a$

Principio di induzione su Alberi Etichettati

Sia P una proprietà su BT_A, $P : BT_A \to Bool$

- Se (Caso Base) $P(\lambda)$ è vera
- Passo Induttivo
  se per ogni
  $t_1, t_2 \in BT_A$ e per ogni $a \in A$, se $P(t_1)$ e $P(t_2)$ sono vere allora $P(N(t_1,a,t_2))$ lo è, allora $P(t)$ è vera per ogni $t \in BT_A$

$$
\frac{P(\lambda) \quad \forall a \in A\,.\, \forall t_1,t_2 \in BT_A\,.\,(P(t_1) \land P(t_2) \Rightarrow P(N(t_1,a,t_2))}{\forall t \in BT_A\,.\,P(t)}
$$

Principio di Induzione Strutturale

Introduciamo la notazione di **segnatura** (rappresenta insiemi e operazioni) e la definizione induttiva di **termine** (rappresenta elementi dell'insieme).

Presentiamo il principio di Induzione Strutturale, parametrico rispetto alla segnatura.

Per ottenere segnature otteniamo i principi di induzione visti in precedenza

Segnature

Una segnatura è una famiglia di insiemi indicizzata da $\mathbb{N}$

$$
\mathcal{F} = \{\mathcal{F}_n\}_{n \in \mathbb{N}}
$$

Per $n \in \mathbb{N}$, $\mathcal{F}_n$ è l'insieme dei **Simboli di Arietà n,** o con n argomenti

I simboli $\mathcal{F}_0$ di arietà 0 sono le **costanti**

Spesso i simboli in $\mathcal{F}_1$ sono detti unari e quelli in $\mathcal{F}_2$ sono detti binari

Termini per una Segnatura

Data una segnatura $\mathcal{F}$, l'insieme $\mathcal{F}Term$ degli $\mathcal{F}$-termini è il più piccolo insieme che soddisfa

[Clausola Base] Per ogni costante $c \in \mathcal{F}_0,\ c \in \mathcal{F}Term$

[Clausola Induttiva] Per ogni $n \geq 1$ e per ogni simbolo $f \in \mathcal{F}_n$, se $t_1, t_2, \ldots, t_n \in \mathcal{F}Term$, allora $f(t_1, t_2, \ldots, t_n) \in \mathcal{F}Term$

Al variare della segnatura, i termini permettono di rappresentare numerose strutture dati definite induttivamente

Esempio, Consideriamo la segnatura $\mathcal{F}$ definita come:

$$\mathcal{F}_0 = \{a, b\} \quad \mathcal{F}_1 = \{f\} \quad \mathcal{F}_2 = \{g\} \quad \mathcal{F}_n = \emptyset, \ \forall n \geq 3$$

I termini validi possono essere ad esempio:

- $f(a)$
- $g(a, b)$
- $f(g(a, b))$
- $g(f(a), f(b))$

I termini non validi invece sono ad esempio:

- g perché vuole 2 argomenti
- $f(a, b)$ perché vuole solo un argomento
- $g(a, b, a)$ perché ci sono troppi argomenti

La segnatura $\mathcal{BT}$: alberi binari come termini

$$\mathcal{BT}_0 = \{\lambda\} \quad \mathcal{BT}_1 = \emptyset \quad \mathcal{BT}_2 = \{N\} \quad \mathcal{BT}_n = \emptyset, \ \forall n \geq 3$$

Applicando la definizione di termini

- Clausola Base: Per ogni costante $c \in \mathcal{F}_0, \ c \in \mathcal{F}Term$
- Clausola Induttiva: Per ogni $n \geq 1$ e per ogni simbolo $f \in \mathcal{F}_n$, se $t_1, t_2, \ldots, t_n \in \mathcal{F}Term$, allora $f(t_1, t_2, \ldots, t_n) \in \mathcal{F}Term$

otteniamo la definizione di alberi binari, $N \in \mathcal{BT}_2$

- Clausola Base: $\lambda \in BT$
- Clausola Induttiva: $\text{se } t_1, t_2 \in BT \text{ allora } N(t_1, t_2) \in BT$

La segnatura $\mathcal{L}^\mathcal{A}$: liste come termini

$$\mathcal{L^A}_0 = \{[\,]\} \quad \mathcal{L^A}_1 = \{a : \mid a \in A\} \quad \mathcal{L^A}_n = \emptyset, \ \forall n \geq 2$$

Applicando la definizione di termine, gli elementi di $\mathcal{L^A}Term$ sono proprio le liste su A

Ad esempio per $A = \{a, b, c\}$

$$\mathcal{L^A}_0 = \{[\,]\} \quad \mathcal{L^A}_1 = \{a :, b :, c :\} \quad \mathcal{L^A}_2 = \emptyset \ \ldots$$

La segnatura $\mathcal{N}$: i naturali come termini

$$\mathcal{N}_0 = Z \quad \mathcal{N}_1 = S \quad \mathcal{N}_n = \emptyset, \ \forall n \geq 2$$

Dalla definizione di termini otteniamo:

- $Z \in \mathcal{N}Term$

- Se $t \in \mathcal{N}Term$, allora $S(t) \in \mathcal{N}Term$

Questi termini sono in Biiezione con i numeri naturali

Per definire una funzione su $\mathcal{F}Term$ occorre:

- definire il valore della funzione per i simboli di arietà 0
- definire il valore della funzione per ogni simbolo di arietà $n \geq 1$, eventualmente utilizzando il valore della funzione calcolato su ognuno degli n-argomenti

Esempio: $val : \mathcal{N}Term \to \mathbb{N}$

1. $val(Z) = 0$
2. $val(S(x)) = val(x) + 1$

Definizione induttiva di somma su naturali:

1. $add(x, Z) = x$
2. $add(x, S(y)) = S(add(x, y))$

Il principio di induzione strutturale

Sia $\mathcal{F}$ una segnatura e P una proprietà su $\mathcal{F}Term$,

- Se (Caso Base) $P(c)$ è vera per ogni simbolo $c \in \mathcal{F}_0$
- E (Passo Induttivo)
  se per ogni
  $n \geq 1$, per ogni $f \in \mathcal{F}_n$ e per tutti i termini $t_1, \dots, t_n \in \mathcal{F}Term$ vale che se $P(t_1), \dots, P(t_n)$ sono vere allora anche $P(f(t_1, \dots, t_n))$ lo è, allora $P(t)$ è vera per ogni $t \in \mathcal{F}Term$

$$\frac{\forall c \in \mathcal{F}_0 \,.\, P(c) \quad \forall n \geq 1 \,.\, \forall f \in \mathcal{F}_n \,.\, \forall t_1, \dots, t_n \in \mathcal{F}Term \,.\, P(t_1) \wedge \dots \wedge P(t_n) \Rightarrow P(f(t_1, \dots, t_n))}{\forall t \in \mathcal{F}Term \,.\, P(t)}$$

Ricorsione

Una ricorsione è una cosa definita in termini di se stessa, una ricorsione che funziona è detta ben fondata.

Un esempio di ricorsione ben fondata è l'induzione

- Una definizione di una funzione rec è ricorsiva se in almeno un caso richiede di valutare la stessa funzione, su uno o più argomenti
- Tutte le definizioni per induzione sono ricorsive

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n-1)! & \text{altrimenti} \end{cases}$$

- Le definizioni per induzione che abbiamo visto sono ben date, cioè definiscono univocamente una funzione (cioè una relazione totale e univalente)

Una definizione ricorsiva ben data

$$due(n) = \begin{cases} 2 & \text{se } n > 100 \\ due(n+1) & \text{altrimenti} \end{cases} \qquad due : \mathbb{N} \to \mathbb{N}$$

Una definizione ricorsiva non ben data

$$g(n) = \begin{cases} 0 & \text{se } n = 0 \\ g(n+1) & \text{altrimenti} \end{cases} \quad g : \mathbb{N} \to \mathbb{N}$$

- Se usiamo la definizione per valutare $g(n)$ per $n > 0$, non finisce mai

  $g(n)$ è quindi una funzione parziale, indefinita per $n > 0$

Relazione associata a una definizione ricorsiva

$R_{rec} \subseteq D \times A$, $R_{rec} = \{(n, m)\}$ partetendo da $rec(n)$ e applicando iterativamente (e correttamente) le clausole della definizione, la valutazione termina con il risultato di m

Una funzione ricorsiva è ben data se la corrispondente relazione è univalente e totale

- $R_{due} = \{(n, 2) \mid n \in \mathbb{N}\}$ ben formata
- $R_g = \{(0, 0)\}$ non è totale

Abbiamo assunto che $\{D_1, D_2, \dots, D_k\}$ sia una partizione del dominio, per garantire che per ogni elemento del domino ci sia esattamente una clausola da applicare

Relazione di precedenza indotta da una definizione ricorsiva

La relazione di precedenza indotta da una definizione ricorsiva di $rec : D \to A$, è la relazione $\prec_{rec} \subseteq D \times D$ definita come

$$x \prec_{rec} y \text{ se } rec(y) \text{ è definita direttamente in termini di } rec(x)$$

Quindi se $x \prec_{rec} y$, allora per valutare $rec(y)$ devo valutare $rec(x)$

- Ma allora se c'è una catena discendente infinita

  $$d_1 \succ_{rec} d_2 \succ_{rec} d_3 \succ_{rec} \dots$$

  La valutazione di $rec(d_1)$ non termina

Relazioni ben fondate

Una relazione $\sqsubset \subseteq A \times A$ è ben fondata se non esiste una catena infinita decrescente di elementi di A

$$a_1 \sqsupset a_2 \sqsupset a_3 \sqsupset \dots$$

Proposizione

Una definizione ricorsiva $rec : D \to A$ è ben data se la relazione di precedenza indotta $\prec_{rec} \subseteq D \times D$ è ben fondata (e $\{D_1, D_2, \dots, D_k\}$ è partizione del dominio)

- $due(n)$ $\quad \prec_{due} \subseteq \mathbb{N} \times \mathbb{N}$ definita come $n + 1 \prec_{due} n$ se $n \leq 100$
- $g(n)$ $\quad \prec_g \subseteq \mathbb{N} \times \mathbb{N}$, definita come $n + 1 \prec_g n$ se $n > 0$

Fibonacci

$$fib(n) = \begin{cases} 1 & \text{se } n = 1 \text{ o } n = 2 \\ fib(n-1) + fib(n-2) & \text{altrimenti} \end{cases}$$

La relazione di precedenza indotta non è $succ$, ma la definizione è comunque ben data.

Fibonacci è ben data, formalmente

1. Se $\sqsubset$ è una relazione ben fondata e $\sqsubset_1 \subseteq \sqsubset$, allora anche $\sqsubset_1$ è ben fondata
2. Una relazione $\sqsubset$ è ben fondata se e solo se lo è la sua chiusura transitiva $\sqsubset^+$

Poiché $succ$ è ben fondata e la chiusura transitiva di $succ$ è $<$, qualunque funzione ricorsiva definita sui naturali usando il suo valore su argomenti strettamente minori è ben data

Altri tipi di ricorsione

Queste erano funzioni ricorsive dirette, ci sono altri tipi di ricorsioni, con caratteristiche proprie:

- Ricorsione Annidata
  Funzione 91 di MacCarthy, è univalente, ma non è immediato vedere che sia totale

$$f(n) = \begin{cases} n - 10 & \text{se } n > 100 \\ f(f(n+11)) & \text{se } n \leq 100 \end{cases}$$

- Ricorsione Mutua
  Può essere eliminata ottenendo ricorsione diretta, stesse problematiche per la terminazione

$$ping(n) = \begin{cases} 0 & \text{se } n = 0 \\ pong(n-1) & \text{altrimenti} \end{cases} \quad pong(n) = \begin{cases} 0 & \text{se } n = 0 \\ ping(n-1) & \text{altrimenti} \end{cases}$$

- Ricorsione Procedurale
  Abbiamo visto la ricorsione per funzioni nel senso matematico del termine
  Gli stessi concetti si applicano a funzioni di linguaggi di programmazione e a procedure

```
void stampa_array(int a[], int i, int n) {
    if (i < n) {
        printf("%d", a[i]);
        stampa_array(a, i+1, n);
    }
}

int main(void) {
    int arr[5] = {1,2,3,4,5};
    stampa_array(arr, 0, 5);
    return 0;
}
```

- Ricorsione in Coda
- Ricorsione Inefficiente