# INDUZIONE STRUTTURALE

**Principio**

- **Base** → definire / dimostrare una proprietà sugli elementi base dell'insieme (non definiti in funzione di altri elementi)
- **Passo induttivo**
  - **Hp. induttiva**: assumere la proprietà sugli elementi dell'insieme
  - **Passo**: definizione / dimostrazione della proprietà sugli elementi composti
  - → la proprietà è assunta per tutti i componenti immediati

$X \subseteq \mathbb{N}$ Def:

**Base**: $0 \in X$

**Passo**: se $x \in X$ allora $x+3 \in X$

dim. $X = 3\mathbb{N}$

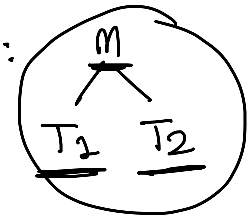
---

**Alberi binari** BT Nodi N

**Base**: $m \in N$ è un BT ($m \in BT$) .. $N \subseteq BT$

**Passo**: se $T_1$ e $T_2 \in BT$ allora $br(m, T_1, T_2) \in BT$

br:




**Foglie**: conta il numero di foglie di $T \in BT$

$$\begin{cases} foglie(m) = 1 \\ foglie(br(m, T_1, T_2)) = foglie(T_1) + foglie(T_2) \end{cases} \leftarrow$$

**Nodi**: conta il numero di nodi di $T \in BT$

$$\begin{cases} nodi(m) = 1 \\ nodi(br(m, T_1, T_2)) = nodi(T_1) + nodi(T_2) + 1 \end{cases}$$

**Branch**: conta il numero di diramazioni di $T \in BT$

$$\begin{cases} branch(n) = 0 \\ branch(br(n, T_1, T_2)) = branch(T_1) + branch(T_2) + 1 \end{cases}$$

$\forall T \in BT \quad . \quad foglie(T) = branch(T) + 1$

**Base** $\quad n \in N \quad foglie(n) = 1 + 0 = 1 + branch(n)$ ✓

$branch(n) = 0$

**Passo** $\quad br(n, T_1, T_2) \in BT \quad T_1, T_2 \in BT$

$\boxed{\text{Hp. ind}}$ → assume la proprietà per i componenti immediati

$foglie(T_1) = branch(T_1) + 1$

$foglie(T_2) = branch(T_2) + 1$

$$foglie(br(n, T_1, T_2)) \underset{\substack{\text{per def} \\ \text{foglie}}}{=} foglie(T_1) + foglie(T_2)$$

$$\underset{\text{Hp.ind}}{=} branch(T_1) + 1 + branch(T_2) + 1$$

$$= \underbrace{(branch(T_1) + branch(T_2) + 1)} + 1$$

$$= branch(br(n, T_1, T_2)) + 1 \quad ✓$$

---

$P \to aa \mid bb \mid bPb \mid aPa \qquad \Sigma = \{a, b\}^*$

def. insieme SP : $aa, bb \in SP$

se $x \in SP$ allora $axa \in SP$

$bxb \in SP$

Dimostriamo: $\forall x \in SP \quad \underbrace{\exists yz \in \{a,b\}^* \,.\, x = yz \ e \ y = reverse(z)}_{2}$

Base: $x = aa$, $\underbrace{y = a \quad z = a}$

$\Downarrow$

$x = yz = aa \qquad \underset{\overset{\shortparallel}{a}}{y} = reverse(z) = a$ ✓

$x = bb$, $\underbrace{y = b \quad z = b}$

$\Downarrow$

$x = yz = bb \qquad \underset{\overset{\shortparallel}{b}}{y} = reverse(z) = b$

Passo: $x \in SP$

$\hookrightarrow a\overset{x'}{\underline{x}}a$ Hp. ind. $\exists yz.\ \underline{x = yz}$ $y = reverse(z)$

dobbiamo dim. $\exists y'z'.\ x' = axa$
$reverse(z') = y'$

$y' = ay \quad z' = za \Rightarrow x' \underset{\text{def } x'}{=} axa \underset{\text{Hp. ind}}{=} ayza$

$= y'z'$

$reverse(z') = reverse(za) \overset{\text{def reverse}}{=} a\, reverse(z)$

$\underset{\text{Hp ind}}{=} ay \underset{\text{def } y'}{=} y'$

$\hookrightarrow bxb$ analogo

---

Grammatica espressioni aritmetiche

$E \rightarrow \underset{\overset{\cap}{\mathbb{N}}}{n} \mid E + E \mid E * E \quad \leadsto \quad Exp:$

Base $n \in Exp \quad (n \in \mathbb{N})$

Passo se $e_1, e_2 \in Exp$ allora

$e_1 + e_2 \in Exp$

$e_1 * e_2 \in Exp$

op: conta il numero di operatori in una espressione

$$op(n) = 0$$
$$op(e_1 + e_2) = op(e_1 * e_2) = op(e_1) + op(e_2) + 1$$

val: conta il numero di valori in una espressione

$$val(n) = 1$$
$$val(e_1 + e_2) = val(e_2 * e_2) = val(e_1) + val(e_2)$$

Dimostriamo $\underline{\forall e \in Exp \quad val(e) = op(e) + 1}$

**Base** $n \in Exp$. $\quad val(n) = 1 = 1 + 0 = 1 + op(n)$ ✓

**Passo** $e_1, e_2 \in Exp \qquad e_1 + e_2 \qquad$ (analogo $e_1 * e_2$)

dobbiamo dim. $val(e_1 + e_2) = op(e_1 + e_2) + 1$

Hp. ind $\begin{cases} val(e_1) = op(e_1) + 1 \\ val(e_2) = op(e_2) + 1 \end{cases}$

$$val(e_1 + e_2) \underset{\text{def val}}{=} val(e_1) + val(e_2) \underset{\text{Hp. ind}}{=} op(e_1) + 1 + op(e_2) + 1$$

$$= \underbrace{(op(e_1) + op(e_2) + 1)}_{\text{def op}} + 1$$

$$= op(e_1 + e_2) + 1$$

## COMPOSIZIONALITÀ

Il significato di ogni costrutto deve essere funzione del significato delle componenti immediate.

---

## SISTEMI DI TRANSIZIONE

→ Matematicamente precisi

→ concisi

→ Permetto di dare semantica in modo induttivo sulla struttura della sintassi

→ Questo avviene mediante un insieme di regole

→ Metodo di specifica generale → permette astrazione

**Def** $\langle \Gamma, \to \rangle$ è un sistema di transizione dove

$\Gamma$ insieme di configurazioni / stati $\gamma \in \Gamma$
(descrizione formale dello stato della macchina)

→ relazione binaria di transizione

$\to \subseteq \Gamma \times \Gamma$

<u>Notazioni</u> $(\gamma_1, \gamma_2) \in \to$ viene denotato $\gamma_1 \to \gamma_2$

$\gamma_0 \to^* \gamma_n \Leftrightarrow \exists\, \gamma_1 \gamma_2 \ldots \gamma_{n-1}$ t.c.
$\gamma_0 \to \gamma_1 \to \gamma_2 \to \ldots \to \gamma_{n-1} \to \gamma_n$

<u>Sistema di transizione terminale</u>: $\langle \Gamma, T, \to \rangle$ dove

$\langle \Gamma, \to \rangle$ è un sistema di transizione

$T \subseteq \Gamma$ insieme di configurazioni terminali

t.c. $\forall \gamma \in T.\ \nexists\, \gamma' \in \Gamma$
$\gamma \to \gamma'$

Grammatica $G = \langle V, T, P, S \rangle \rightarrow \langle \Gamma, \xrightarrow{T} \rangle$ Sistema di transizione

$\Gamma = P$

$T = A \rightarrow \alpha$
$\Uparrow$
$T^*$

$$\frac{A \rightarrow \alpha}{\beta A \gamma \rightarrow \beta \alpha \gamma} \equiv (A \rightarrow \alpha) \longrightarrow (\beta A \gamma \rightarrow \beta \alpha \gamma)$$

$S \longrightarrow \alpha \in T^*$

---

Non determinismo $\gamma \rightarrow \gamma' \rightsquigarrow \gamma'''$, $\gamma \rightarrow \gamma'' \rightsquigarrow \gamma'''$ $\qquad e_1 + e_2$

---

## CATEGORIE SINTATTICHE

↳ manipolare dati (espressioni)

↳ creazione/manipolazione di legami (dichiarazioni)

↳ trasformare stati (comandi)

Ambiente
Insieme dei legami
id ↔ oggetti

Memoria
permette di descrivere gli effetti di una trasformazione

ESPRESSIONI → denotano valori
→ vengono valutate per restituire un valore
→ sono equivalenti se denotano lo stesso valore
(sono uguali se sono sintatticamente uguali)

$6 * 1$

$2 * 3$

$4 + 2$

## DICHIARAZIONI

→ Denotano richieste di modifica degli ambienti

→ vengono elaborate per attuare le trasformazioni dell'ambiente

→ dichiarazioni equivalenti se vengono elaborate nello stesso ambiente

## COMANDI

→ Denotano richieste di modifica della memoria

→ vengono eseguiti per ottenere la trasformazione della memoria corrispondente

→ programmi equivalenti generano ∀ imput le stesse trasformazioni della memoria.