La logica nell’informatica

- La logica gioca un ruolo importante nell’informatica
  - *Logic plays a similar role in computer science to that played by calculus in the physical sciences and traditional engineering disciplines*

(M. Vardi, 2007)

Varietà di logica

- Esistono molti linguaggi logici
  - Con diversi livelli di astrazione rispetto al linguaggio naturale
- Esempi:
  - Logica Proposizionale
  - FOL
  - Description Logic
  - Logiche temporali

Applicazioni

- La logica è un formalismo "universale"
- Si applica a numerosi ambiti:
  - architetture (porte logiche)
  - ingegneria del software (specifica e verifica)
  - linguaggi di programmazione (semantica, programmazione logica)
  - database (calcolo relazionale, Datalog)
  - intelligenza artificiale (dimostrazioni automatiche di teoremi)
  - …

Applicazioni di base

- La logica proposizionale è usata nell'informatica per il progetto di circuiti
- FOL (più potente) è usata nella verifica dei programmi e nell'intelligenza artificiale
- Usi all'interno di questo corso:
  - definizione di linguaggi
  - specifica di proprietà di programmi

Linguaggi

- I linguaggi sono insiemi di stringhe su un alfabeto
- Esempio:
  - Il linguaggio di stringhe su {*a*, *b*} con lo stesso numero di *a* e di *b* e tutte le *a* prima è informalmente descritto dall'insieme $\{ a^n b^n \mid n \geq 0 \}$

In logica

- FOL può aiutare a descrivere un linguaggio
- Gli insiemi possono essere visti come abbreviazioni di formule FOL
- Problemi:
  - Che cosa dobbiamo descrivere?
  - Come definire le diverse parti?
  - Quali primitive possiamo assumere?

Esempio

- Come descrivere l'insieme $\{ a^n b^n \mid n \geq 0 \}$?
- Di fatto è una forma abbreviata di

$$\forall x\ (x \in L \leftrightarrow \exists n\ (n \geq 0 \wedge x = a^n . b^n))$$

  - Predicati: $\in L$, $\geq$, $=$
  - Funzioni: concatenazione, elevamento a potenza
- Cos'è x^n?

$$\forall n \forall x((n=0 \rightarrow x^n = \varepsilon) \wedge (n>0 \rightarrow x^n = x^{n-1} . x))$$

Osservazioni

- Occorrerebbe definire tutti i predicati e le funzioni non elementari (=, >, +, -, *, :, concatenazione…)
- Per questo motivo, nel seguito, per definire linguaggi faremo riferimento anche a logiche molto più ristrette, con una sintassi specifica già predisposta

$L_1 = a^* b^*$

- L_1 è il linguaggio delle stringhe su {a, b} con tutte le ‘a’ all’inizio
- Più precisamente, una stringa è in L_1 se
  - è la stringa vuota, oppure
  - è composta da un prefisso ‘a’ e da un suffisso y (che appartiene sempre a L_1)
  - è composta da un prefisso y (che appartiene sempre a L_1) e da un suffisso ‘b’
- Questo si può esprimere come

$$\forall x(x \in L_1 \leftrightarrow (x = \varepsilon) \vee \exists y\,(x = ay \wedge y \in L_1) \vee \exists y\,(x = yb \wedge y \in L_1))$$

$L_2 = a^*b^*c^*$ (1)

- L_2 è il linguaggio delle stringhe su {a, b, c} con tutte le ‘a’ all’inizio, poi tutte le ‘b’ e alla fine tutte le ‘c’
- L_2 può essere visto come $a^*b^* . b^*c^*$
  - a^*b^* è L_1
  - b^*c^* ha la stessa struttura di L_1 (chiamiamolo L_3)
- Una stringa appartiene a L_2 se
  - È in L_1 oppure
  - È in L_3 oppure
  - È composta da un prefisso ‘a’ e da un suffisso y (che appartiene a L_2 o a L_3) oppure
  - È composta da un prefisso y (che appartiene a L_1 o a L_2) e da un suffisso ‘c’

$L_2 = a^* b^* c^*$ (2)

- In FOL:

$$\forall x (x \in L_2 \leftrightarrow (x \in L_1) \vee (x \in L_3) \vee \exists y\, ((x = ay \wedge (y \in L_2 \vee y \in L_3)) \vee (x = yc \wedge (y \in L_2 \vee y \in L_1))))$$

x è in L_1 x è in L_3

Si può scomporre nel prefisso 'a' e nel suffisso y (che appartiene a L2 o a L3)

Si può scomporre nel prefisso y (che appartiene a L1 o a L2) e il suffisso 'c'

- … ci servono tutti?

Note e considerazioni aggiuntive

- Quando l'ordine tra le lettere in un linguaggio è importante, la formula FOL definisce il linguaggio decomponendolo
  - Definizione "ricorsiva"
- Quando occorre contare le lettere, si può definire una funzione aggiuntiva

Esempio

- L_4={x∈{a,b}* | numero di ‘a’ uguale a numero di ‘b’}
  - #(x, a) è di arietà 2 e conta il numero di occorrenze del simbolo ‘a’ nella stringa x
  - Si può definire formalmente come:

$$\forall x \forall y((x = \varepsilon \rightarrow \#(x,a) = 0) \wedge$$
$$(x = a.y \rightarrow \#(x,a) = \#(y,a) + 1) \wedge (x = b.y \rightarrow \#(x,a) = \#(y,a))) \wedge$$
$$\forall x \forall y((x = \varepsilon \rightarrow \#(x,b) = 0) \wedge$$
$$(x = b.y \rightarrow \#(x,b) = \#(y,b) + 1) \wedge (x = a.y \rightarrow \#(x,b) = \#(y,b)))$$

    - La definizione dipende dall’alfabeto

- In FOL $\forall x(x \in L_4 \leftrightarrow \#(x,a) = \#(x,b))$

MFO: Logica monadica del prim'ordine

- Vediamo un frammento di logica del prim'ordine che ci permette di descrivere parole su un alfabeto I
- Sintassi

$$\phi := a(x) \mid x < y \mid \neg\phi \mid \phi \land \phi \mid \forall x(\phi)$$

  - laddove $a \in$ I, cioè introduciamo un predicato unario per ogni simbolo dell'alfabeto
- Interpretazione:
  - $<$ corrisponde alla solita relazione di minore
  - il dominio delle variabili è N

Alcune classiche abbreviazioni

- Ovviamente:
  - $\phi_1 \vee \phi_2 \quad \triangleq \quad \neg(\neg\phi_1 \wedge \neg\phi_2)$
  - $\phi_1 \Rightarrow \phi_2 \quad \triangleq \quad \neg\phi_1 \vee \phi_2$
  - $\exists x(\phi) \quad \triangleq \quad \neg\forall x(\neg\phi)$
  - $x = y \quad \triangleq \quad \neg (x < y) \wedge \neg (y < x)$
  - $x \leq y \quad \triangleq \quad \neg (y < x)$
- Ma possiamo anche definire:
  - la costante 0: $x = 0 \quad \triangleq \quad \forall y(\neg(y < x))$
  - il predicato per il successore succ(x,y):
    succ(x,y) $\triangleq \quad x < y \wedge \neg\exists z(x < z \wedge z < y)$
  - le costanti 1, 2, 3, ecc. come successore di 0, 1, 2, ecc.

Interpretazione come parola sull'alfabeto I

- Data una parola $w \in I^+$, ed un simbolo $a \in I$:
  - $a(x)$ è vero se, e solo se, l'x-esimo simbolo di w è a (il primo simbolo di w ha indice 0)
    - (w in realtà può anche essere vuota, ma le definizioni si complicano un po')
- Formula che è vera su tutte e sole le parole il cui primo simbolo esiste ed è a:

$$\exists x(x = 0 \land a(x))$$

- Formula che è vera su tutte le parole in cui ogni a è seguita da una b

$$\forall x(a(x) \Rightarrow \exists y\ (\mathrm{succ}(x,y) \land b(y)))$$

Altri esempi di abbreviazioni e formule

- Usiamo le seguenti abbreviazioni:
  - $y = x + 1$ per dire succ(x,y)
  - più in generale, se k è una costante > 1,
    $y = x + k$ per dire $\exists z_1 \ldots z_{k\text{-}1}(y = z_{k-1}+1 \land \ldots \land z_1 = x+1)$
  - $y = x - 1$ per dire succ(y,x) (cioè $x = y + 1$)
  - $y = x - k$ per dire $x = y + k$
  - last(x) per dire $\neg\exists y(y > x)$
- Parole (non vuote) in cui l'ultimo simbolo è a:
  $\exists x\ (\text{last}(x) \land a(x))$
- Parole (di almeno 3 simboli) in cui il terzultimo simbolo è a:
  $\exists x\ (a(x) \land \exists y\ (y = x+2 \land \text{last}(y)))$
  - Oppure (con un leggero abuso di notazione): $\exists x\ (a(x) \land \text{last}(x+2))$
    - oppure ancora: $\exists y\ (a(y\text{-}2) \land \text{last}(y))$

Semantica

- Siano $w \in I^+$ e V_1 l'insieme delle variabili; un *assegnamento* è una funzione $v_1 : V_1 \rightarrow [0..|w|-1]$ tale che
  - $w, v_1 \models a(x)$ sse $w = uav$ e $|u| = v_1(x)$
  - $w, v_1 \models x < y$ sse $v_1(x) < v_1(y)$
  - $w, v_1 \models \neg\phi$ sse non $w, v_1 \models \phi$
  - $w, v_1 \models \phi_1 \wedge \phi_2$ sse $w, v_1 \models \phi_1$ e $w, v_1 \models \phi_2$
  - $w, v_1 \models \forall x(\phi)$ sse $w, v'_1 \models \phi$ per ogni v'_1 con $v'_1(y) = v_1(y)$, $y \neq x$
- Linguaggio di una formula ϕ:
  - $L(\phi) = \{ w \in I^+ \mid \exists v_1 : w, v_1 \models \phi \}$

Note sulla stringa vuota

- Per semplicità, abbiamo definito la semantica (e i linguaggi definiti da una formula MFO) su parole non vuote:
  - $w \in I^+$
- È possibile includere anche le parole vuote, con qualche precauzione aggiuntiva (la stringa vuota non può soddisfare le quantificazioni esistenziali):
  - $w \in I^*$

Proprietà di MFO (1/2)

- I linguaggi esprimibili mediante MFO sono chiusi rispetto a unione, intersezione, complemento
  - basta fare "or", "and", "not" di formule

Proprietà di MFO (2/2)

- In MFO non si può esprimere il linguaggio L_p fatto di tutte e sole le parole di lunghezza pari con I = {*a*}
- MFO è strettamente meno potente degli FSA
  - data una formula MFO si può sempre costruire un FSA equivalente
    - non vediamo la costruzione
  - L_p può facilmente essere riconosciuto mediante un FSA
- I linguaggi definiti da MFO non sono chiusi rispetto alla * di Kleene:
  - la formula MFO $a(0) \land a(1) \land \text{last}(1)$ definisce il linguaggio L_{aa} fatto della sola parola {*aa*} di lunghezza 2.
  - Abbiamo che $L_p = L_{aa}^*$
  - MFO definisce i cosiddetti linguaggi *star-free*, cioè definibili tramite unione, intersezione, complemento e concatenazione di linguaggi finiti

MSO: Logica monadica del secondo ordine

- Per ottenere lo stesso potere espressivo degli FSA "basta" permettere di quantificare sui predicati monadici
  - quindi: logica del **secondo** ordine
  - (in pratica vuol dire poter quantificare anche su **insiemi** di posizioni)

- Ammettiamo formule del tipo $\exists X(\phi)$, dove X è una variabile il cui dominio è l'insieme dei predicati monadici
  - per convenzione usiamo le lettere maiuscole per indicare variabili aventi come dominio l'insieme dei predicati monadici, e lettere minuscole per indicare variabili sui numeri naturali

Semantica ed esempio

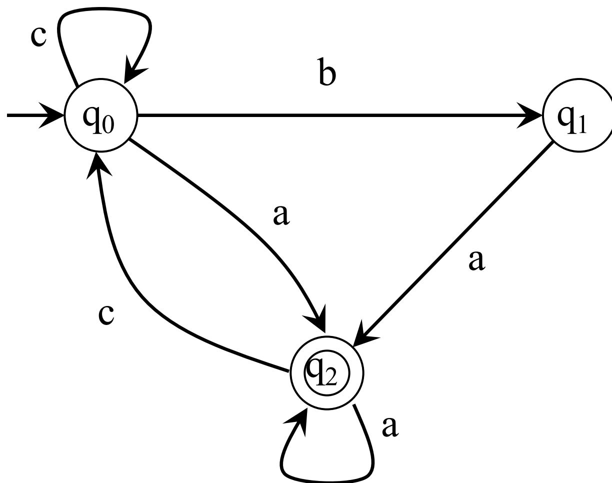
- L'assegnamento delle variabili del II ordine (insieme V_2) è una funzione $v_2 : V_2 \rightarrow \wp([0..|w|-1])$
  - $w, v_1, v_2 \models X(x)$ sse $v_1(x) \in v_2(X)$
  - $w, v_1, v_2 \models \exists X(\phi)$ sse $w, v_1, v'_2 \models \phi$ per qualche v'_2 con $v'_2(Y) = v_2(Y)$, $Y \neq X$
- Possiamo dunque scrivere la formula che descrive il linguaggio L_p

$$\exists P(\forall x(\neg P(0) \wedge (\neg P(x) \Leftrightarrow P(x+1)) \wedge a(x) \wedge (last(x) \Rightarrow P(x))))$$

- NB: l'indice dell'ultimo carattere di una stringa di lunghezza pari è dispari! P è un insieme di posizioni dispari

Da FSA a MSO

- In generale, grazie alle quantificazioni del second'ordine è possibile trovare, per ogni FSA, una formula MSO equivalente
- Esempio

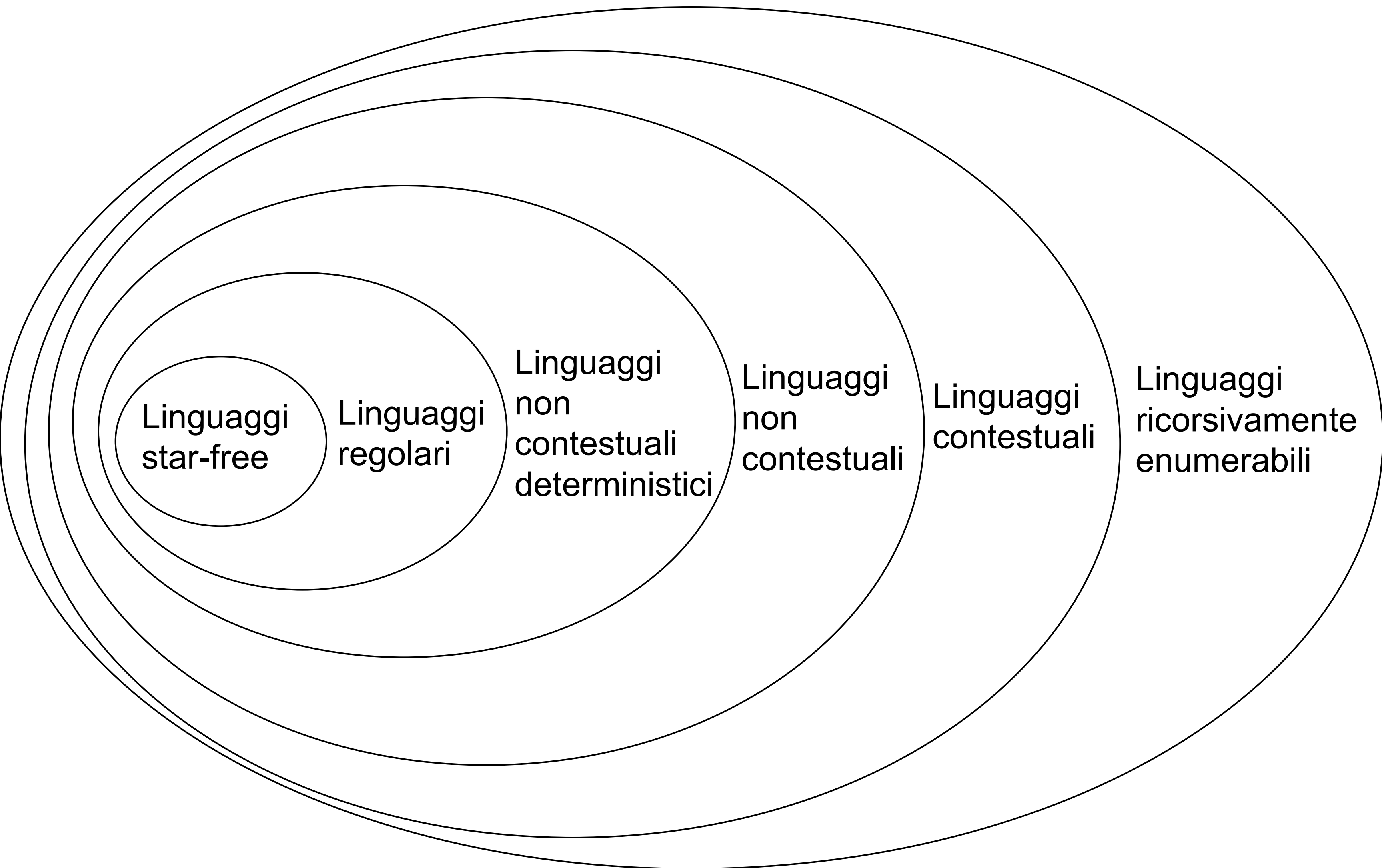


$$
\begin{array}{l}
\exists Q_0,Q_1,Q_2 \;(\\
\quad \forall z(\quad \neg(Q_0(z) \wedge Q_1(z)) \wedge \neg(Q_0(z) \wedge Q_2(z)) \wedge \\
\quad \qquad \neg(Q_1(z) \wedge Q_2(z))) \wedge \\
\quad Q_0(0) \wedge \\
\quad \forall x \;(\quad (\neg last(x) \Rightarrow (\\
\qquad\qquad Q_0(x) \wedge c(x) \wedge Q_0(x+1) \vee \\
\qquad\qquad Q_0(x) \wedge b(x) \wedge Q_1(x+1) \vee \\
\qquad\qquad Q_0(x) \wedge a(x) \wedge Q_2(x+1) \vee \\
\qquad\qquad Q_1(x) \wedge a(x) \wedge Q_2(x+1) \vee \\
\qquad\qquad Q_2(x) \wedge c(x) \wedge Q_0(x+1) \vee \\
\qquad\qquad Q_2(x) \wedge a(x) \wedge Q_2(x+1)) \;\; \wedge \\
\qquad (last(x) \Rightarrow \quad Q_0(x) \wedge a(x) \vee \\
\qquad\qquad Q_2(x) \wedge a(x) \vee \\
\qquad\qquad Q_1(x) \wedge a(x))))
\end{array}
$$

Da MSO a FSA

- Data una formula MSO ϕ, è possibile costruire un FSA che accetta esattamente il linguaggio L definito da ϕ (teorema di Büchi-Elgot-Trakhtenbrot)
  - la dimostrazione dell'esistenza è costruttiva (mostra come ottenere un FSA da una formula MSO), ma non la vediamo per semplicità
- Quindi la classe dei linguaggi definibili da formule MSO coincide con i linguaggi regolari

Il bersaglio



Precondizioni e postcondizioni

- Quando si programma una funzione è importante definire precisamente che **cosa** fa, senza necessariamente descrivere **come** lo fa
- Questo è lo scopo di precondizioni e postcondizioni
  - La **precondizione** indica cosa deve valere prima che la funzione sia invocata
  - La **postcondizione** indica cosa deve valere dopo che la funziona ha finito la propria esecuzione

Struttura generale (notazione di Hoare)

{Precondizione: *Pre*}
Programma: *P*
{Postcondizione: *Post*}

- La precondizione è verificata prima dell'esecuzione di *P*, mentre la postcondizione è verificata dopo
- *P* deve essere tale per cui, se *Pre* vale prima dell'esecuzione, allora *Post* vale dopo l'esecuzione

Come definirle?

- Le precondizioni e postcondizioni possono essere definite in modi diversi
  - Linguaggio naturale
  - Linguaggi per le asserzioni
  - Linguaggi ad-hoc

$\rightarrow$ FOL può essere usata per questo scopo

Algoritmo di ricerca (1)

- Sia P un programma che implementa la ricerca di un elemento x in un array ordinato di n elementi
  - Precondizione: l’array è ordinato
  - Postcondizione: la variabile logica `found` (un *flag*) dev’essere vera se e solo se l’elemento x esiste nell’array a
- Nota: P non implementa necessariamente un algoritmo di ricerca binaria
  - Ma la precondizione è necessaria in quel caso

Algoritmo di ricerca (2)

- La precondizione può essere formalizzata così:

$$\{\forall i(1 \leq i \leq n-1 \rightarrow a[i] \leq a[i+1])\}$$

- La postcondizione è $\{found \leftrightarrow \exists i(1 \leq i \leq n \wedge a[i] = x)\}$
- Quindi la struttura complessiva è

$$\{\forall i(1 \leq i \leq n-1 \rightarrow a[i] \leq a[i+1])\}$$
$$P$$
$$\{found \leftrightarrow \exists i(1 \leq i \leq n \wedge a[i] = x)\}$$

- Notare che gli elementi di un array sono indicati, come di consueto, con parentesi quadre

Ordinamento (1)

- Sia ORD un programma che ordina un array a di n elementi senza ripetizioni
  - Precondizione: l'array non contiene ripetizioni
  - Postcondizione: l'array ottenuto è ordinato (se un elemento x precede un elemento y nell'array, allora x<y)

Formalmente:

$$\{\neg\exists i, j(1 \le i \le n \land 1 \le j \le n \land i \ne j \ \land a[i] = a[j])\}$$

$$ORD$$

$$\{\forall i(1 \le i \le n-1 \rightarrow a[i] \le a[i+1])\}$$

Ordinamento (2)

- Questa specifica è adeguata?
- Consideriamo l'esempio seguente:
  - a prima dell'esecuzione di ORD è [7 6 2 4 22]
  - a dopo l'esecuzione di ORD è [2 6 10 10 22]

  Soddisfa la postcondizione!
- La postcondizione deve asserire che tutti e soli gli elementi nell'array da ordinare sono contenuti nell'array ordinato

Ordinamento (3)

- Usiamo un array b (non usato in ORD) per riferirci all'array a prima dell'esecuzione
  - Occorre aggiungere alla precondizione che b è esattamente come a
- La soluzione diventa

$$\{\neg\exists i,j(1\le i\le n\wedge 1\le j\le n\wedge i\ne j\wedge a[i]=a[j])\quad\wedge$$
$$\forall i(1\le i\le n\rightarrow a[i]=b[i])\}$$
$$ORD$$
$$\{\forall i(1\le i<n\rightarrow a[i]\le a[i+1])\quad\wedge$$
$$\forall i(1\le i\le n\rightarrow\exists j((1\le j\le n)\wedge(a[i]=b[j])))\quad\wedge$$
$$\forall j(1\le j\le n\rightarrow\exists i((1\le i\le n)\wedge(a[i]=b[j])))\}$$

Note

- Una specifica deve essere considerata come un “contratto”
  - Deve contenere tutte le informazioni
  - Senza assunzioni a priori
- Quando qualche condizione è eliminata dalla precondizione, la specifica diventa insoddisfacente

Dalla specifica alla prova: cenni

- Dopo aver *specificato* i requisiti di un algoritmo (o di un sistema), occorre *verificare* la correttezza del medesimo
- Se ho a disposizione un modello matematico (ad esempio un’assiomatizzazione) dell’implementazione costruita, in linea di principio potrei ottenere la prova di correttezza come una *dimostrazione di teorema*