Logica Matematica

Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

6 aprile 2022

La logica come formalismo descrittivo

Un ulteriore linguaggio di specifica

- Logica: un formalismo "universale" alternativo al linguaggio naturale
  - Vantaggi: non ambiguità, possibile dimostrare in modo automatico proprietà desiderate
- Applicata in contesti molto vari: da ingegneria informatica a ingegneria dei sistemi
- Esistono formalismi applicativi basati sulla logica:
  - Linguaggi di programmazione (Prolog, Datalog)
  - Linguaggi di specifica (Z è lo standard ISO/IEC 13568 per specifiche funzionali)

Usi esaminati in questo corso

- Specifica di linguaggi formali (logica monadica del primo e secondo ordine)
- Logica per la specifica di comportamento (I/O) di programmi

Logica del prim'ordine su parole finite

Sintassi ed interpretazione

- Consideriamo la logica del prim'ordine con predicati su una sola variabile. Consente di descrivere parole su un alfabeto $\mathbf{I}$
- Con $a \in \mathbf{I}$ una lettera predicativa per ogni simbolo di $\mathbf{I}$, una formula φ è :
  - $a(x)$: il predicato a applicato ad una variabile
  - $x < y$
  - $\neg\varphi$: negazione logica della formula
  - $\varphi \wedge \varphi$: congiunzione (and Booleano) di due formule
  - $\forall x\ (\varphi)$: quantificatore universale
- Interpretazione:
  - il dominio delle variabili è un sottoinsieme finito di $\mathbb{N}$ da pensare come posizioni
  - $<$ corrisponde alla relazione di minore tra le posizioni

Alcune abbreviazioni

Concetti ricorrenti

- Come sempre:
  - $\varphi_1 \vee \varphi_2 \stackrel{\Delta}{=} \neg(\neg\varphi_1 \wedge \neg\varphi_2)$: legge di De Morgan
  - $\varphi_1 \Rightarrow \varphi_2 \stackrel{\Delta}{=} \neg\varphi_1 \vee \varphi_2$: definizione di implicazione semplice
  - $\exists x(\varphi) \stackrel{\Delta}{=} \neg\forall x(\neg\varphi)$: proprietà dei quantificatori
  - $x = y \stackrel{\Delta}{=} \neg(x < y) \wedge \neg(x > y)$: definizione di uguaglianza
  - $x \leq y \stackrel{\Delta}{=} \neg(y < x)$
- In aggiunta:
  - La costante 0: $x = 0 \stackrel{\Delta}{=} \forall y\ (\neg(y < x))$
  - La funzione successore $S(x)$: $S(x) = y \stackrel{\Delta}{=} (x < y) \ \wedge \ \neg\exists z(x < z \wedge z < y)$
  - Le costanti $1, 2, 3, \ldots$ come $S(0), S(S(0)), S(S(S(0))), \ldots$

Interpretazione come parole su $\mathbf{I}$

Interpretazione di $a(x)$

- $a(x)$ è vero $\Leftrightarrow$ l'x-esimo simbolo di una parola $w \in \mathbf{I}^*$ è a
  - gli indici di w partono da 0

Esempi

- Formula vera su tutte e sole le parole non vuote che iniziano per a:
  $\exists x(x = 0 \wedge a(x))$
- Formula vera su tutte e sole le parole in cui ogni a è seguita da una b:
  $\forall x(\ a(x)\ \Rightarrow\ \exists y(y = S(x) \wedge b(y))\ )$
- Formula vera per la sola stringa vuota: $\forall x\ (a(x) \wedge \neg a(x))$
  - n.b. se problematico considerare ε, la formula non è mai vera

Altre abbreviazioni convenienti

Abbreviazioni per indici comodi

- $y = x + 1$ indica $y = S(x)$ (non esiste un'operatore di somma tra variabili)
- generalizzando, se $k \in \mathbb{N}, k > 1$ indichiamo con $y = x + k$
  $\exists z_1, z_2, \ldots, z_{k-1}(z_1 = x + 1, z_2 = z_1 + 1, \ldots, y = z_{k-1} + 1)$
- $y = x - 1$ indica $x = S(y)$, ovvero $x = y + 1$, così come $y = x - k$ indica $x = y + k$
- $last(x)$ indica $\neg\exists y(y > x)$

Esempi

- Parole non vuote terminanti per a: $\exists x(last(x) \wedge a(x))$
- Parole con almeno 3 simboli di cui il terzultimo è a
  $\exists x(a(x) \wedge \exists y(y = x + 2 \wedge last(y)))$ Abbreviando: $[\exists x(a(x) \wedge last(x + 2))]$

Semantica formale

Semantica dei componenti di una formula

- Dati $w \in \mathbf{I}^+$ e $\mathbf{V}_1$ insieme delle variabili, un assegnamento è una funzione $v_1 : \mathbf{V}_1 \to \{0, 1, \ldots, |w| - 1\}$
  - $w, v_1 \vDash a(x)$ se e solo se $w = uav$ e $|u| = v_1(x)$
  - $w, v_1 \vDash x < y$ se e solo se $v_1(x) < v_1(y)$
  - $w, v_1 \vDash \neg\varphi$ se e solo se $w, v_1 \nvDash \varphi$
  - $w, v_1 \vDash \varphi_1 \wedge \varphi_2$ se e solo se $w, v_1 \vDash \varphi_1$ e $w, v_1 \vDash \varphi_2$
  - $w, v_1 \vDash \forall x(\varphi)$ sse $w, v_1' \vDash \varphi$ per ogni v_1' con $v_1'(y) = v_1(y)$ con y diversa da x

Linguaggio definito da una formula

- $L(\varphi) = \{w \in \mathbf{I}^+ \mid \exists v : w, v \vDash \varphi\}$, con φ formula chiusa

Proprietà della MFO

Chiusura rispetto ad operazioni

- I linguaggi esprimibili con MFO sono chiusi per unione, intersezione, complemento
  - Basta combinare le formule con $\wedge$, $\vee$, $\neg$
- In MFO non posso esprimere $L = \{a^{2n}, n \in \mathbb{N}\}$ su $\mathbf{I} = \{a\}$
- MFO è strettamente *meno* potente degli FSA
  - Da una formula in MFO φ posso sempre costruire un FSA che riconosce $L(\varphi)$

Proprietà della MFO

Chiusura rispetto alla $*$ di Kleene

- I linguaggi definiti da una formula MFO non sono chiusi rispetto alla $*$ di Kleene
  - la formula $a(0) \wedge a(1) \wedge last(1)$ definisce $L = \{aa\}$
  - la $*$-chiusura di L è il linguaggio delle stringhe di a pari
- MFO è in grado di definire i linguaggi *star-free*: sono i linguaggi ottenuti per unione, intersezione, concatenazione e complemento di linguaggi finiti
- Come definire tutti i REG?

Logica Monadica del Secondo Ordine (MSO)

Quantificare insiemi di posizioni

- Per avere lo stesso potere espressivo degli FSA basta "solo" permettere di quantificare sui predicati monadici
  - In pratica, quantificare su *insiemi* di posizioni
  - Quantificazione su predicati del prim'ordine $\rightarrow$ logica del secondo ordine
- Ammettiamo formule come $\exists X\ (\varphi)$ con X appartenente all' insieme dei predicati monadici (insiemi di posizioni)
- Convenzione: usamo maiuscole e minuscole
  - Maiuscole per indicare variabili con dominio l'insieme dei predicati monadici
  - Minuscole per indicare variabili $\in \mathbb{N}$

Semantica

Assegnamento delle variabili

- L'assegnamento di variabili del 2º ordine (insieme $\mathbf{V}_2$) è una funzione $v_2 : \mathbf{V}_2 \to \wp(\{0, 1, \ldots, |w| - 1\})$
  - $w, v_1, v_2 \vDash X(x)$ se e solo se $v_1(x) \in v_2(X)$
  - $w, v_1, v_2 \vDash \forall X(\varphi)$ se e solo se $w, v_1' \vDash \varphi$ per ogni v_2' con $v_2'(Y) = v_2(Y)$, con Y diversa da X

Esempio

- Possiamo descrivere il linguaggio $L = \{a^{2n}, n \in \mathbb{N} \setminus \{0\}\}$

$$\begin{aligned} \exists P(\forall x(\quad & a(x) \wedge \\ & \neg P(0) \wedge \\ & \forall y(y = x + 1 \Rightarrow (\neg P(x) \Leftrightarrow P(y))) \wedge \\ & (last(x) \Rightarrow P(x)) \qquad\qquad)) \end{aligned}$$

Da MSO a FSA

Completare l'equivalenza

- Data una φ MSO, si può sempre costruire un FSA che accetta esattamente $L(\varphi)$ (teorema di Büchi-Elgot-Trakhtenbrot)
  - La dimostrazione dell'esistenza è costruttiva: mostra come costruire l'FSA a partire da una formula MSO (non la vediamo per semplicità)
- La classe dei linguaggi definibili via MSO coincide con REG

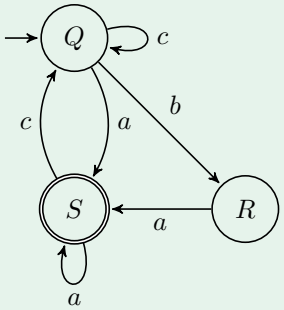
Da MSO a FSA

Un approccio costruttivo

- Per ogni $q \in \mathbf{Q}$ dell'FSA, definiamo una variabile $\mathbf{X}_q$ che rappresenta l'insieme di posizioni in una stringa accettata dove l'automa si trova in q
- L'automa non è in due stati contemporaneamente: per ogni coppia $\mathbf{X}_i, \mathbf{X}_j$, abbiamo $\neg\exists y(y \in \mathbf{X}_i \wedge y \in \mathbf{X}_j)$
- L'FSA parte da q_0 equivale a dire che $\forall x(x = 0 \Rightarrow x \in X_{q_0})$ delle posizioni dei caratteri letti dall' FSA partendo da q
- Ogni transizione $\delta(q_i, a) = q_j$ diventa:
  $\forall x, y(y = x + 1 \Rightarrow (x \in \mathbf{X}_i \wedge a(x) \wedge y \in \mathbf{X}_j))$
- L' accettazione via $\delta(q_i, a) \in \mathbf{F}$ diventa: $\forall x(last(x) \Rightarrow \bigvee_{\delta(q_i,a)\in\mathbf{F}}(x \in \mathbf{X}_i \wedge a(x))$

Da FSA a MSO

Un esempio pratico



$$
\begin{aligned}
&\exists Q, R, S(\\
&\quad \neg\exists z(\neg(Q(z) \wedge R(z)) \wedge \\
&\qquad \neg(Q(z) \wedge S(z)) \wedge \\
&\qquad \neg(R(z) \wedge S(z))) \quad \wedge \\
&\quad \forall x\,(x = 0 \Rightarrow x \in Q) \quad \wedge \\
&\quad \forall x, y(y = x + 1 \Rightarrow \\
&\qquad (Q(x) \wedge c(x) \wedge Q(y)) \vee \\
&\qquad (Q(x) \wedge b(x) \wedge R(y)) \vee \\
&\qquad (Q(x) \wedge a(x) \wedge S(y)) \vee \\
&\qquad (R(x) \wedge a(x) \wedge S(y)) \vee \\
&\qquad (S(x) \wedge a(x) \wedge S(y)) \vee \\
&\qquad (S(x) \wedge c(x) \wedge Q(y))) \quad \wedge \\
&\quad \forall x\,(last(x) \Rightarrow \; Q(x) \wedge a(x) \vee R(x) \wedge a(x) \vee \\
&\qquad\qquad S(x) \wedge a(x)))
\end{aligned}
$$

Logica per definire proprietà dei programmi

Un formalismo per definire gli effetti

- Specifica di un algoritmo di ricerca: la variabile found $\in \{0, 1\}$ deve valere 1 se e solo se esiste un elemento dell' array a di n elementi uguale all' elemento x cercato
  - found $\Leftrightarrow \exists i(a[i] = x \wedge 0 \leq i \leq n - 1)$
- Specifica di un algoritmo di inversione out-of-place di un array a in un array b
  - $\forall i, (0 \leq i \leq n - 1 \Rightarrow b[i] = a[n - 1 - i])$

Più in generale

Pre- e Post-condizioni

- Precondizioni: insieme di condizioni che devono essere vere prima dell'esecuzione di un programma P affinchè sia vero un insieme di fatti (*post-condizioni*) dopo l' esecuzione
- Esempio: ricerca di un elemento x in un array ordinato a

  Pre $\{\forall i, (0 \leq i \leq n-2 \Rightarrow a[i] \leq a[i+1])\}$

  - Esecuzione del programma `P`

  Post $\{\text{found} \Leftrightarrow \exists i(a[i] = x \wedge 0 \leq i \leq n-1)\}$
- Controesempio: un algoritmo di ricerca binaria non garantirebbe post se pre fosse semplicemente $\{True\}$
- N.B. le pre e post precedenti non implicano che `P` sia un algoritmo di ricerca binaria: una ricerca lineare funziona ugualmente

Un ulteriore esempio

Ordinamento di array di n elementi senza ripetizioni

Pre $\{\neg\exists i, j\ (\ 0{\leq}i{\leq}n{-}1\ \wedge\ 0{\leq}j{\leq}n{-}1\ \wedge\ a[i]{=}a[j]\ \wedge\ i{\neq}j)\}$

- Esecuzione di `ORD`

Post $\{\forall i, (\ 0{\leq}i{\leq}n{-}2 \Rightarrow a[i] \leq a[i+1])\}$

"Buone" specifiche

- É una specifica "adeguata"?
- La specifica agisce come un "contratto" con chi deve sviluppare `ORD`, così come con chi usa il programma sviluppato

Ordinamento di array di n elementi senza ripetizioni

Una specifica più accurata

Pre $\{ \neg\exists i,j\ (\ 0{\leq}i{\leq}n{-}1\ \wedge\ 0{\leq}j{\leq}n{-}1\ \wedge\ a[i]{=}a[j]\ \wedge\ i{\neq}j)\ \wedge\ \forall i(\ 0{\leq}i{\leq}n{-}1 \Rightarrow a[i] = b[i])\ \}$

- Esecuzione di `ORD`

Post $\{\ \forall i,(\ 0{\leq}i{\leq}n{-}2 \Rightarrow a[i]{\leq}a[i+1]) \wedge$
$\forall i(\ 0{\leq}i{\leq}n{-}1 \Rightarrow \exists j(\ 0{\leq}j{\leq}n{-}1\ \wedge a[i] = b[j])) \wedge$
$\forall j(\ 0{\leq}j{\leq}n{-}1 \Rightarrow \exists i(\ 0{\leq}j{\leq}n{-}1\ \wedge a[i] = b[j]))\ \}$

"Buone" specifiche

- Se eliminiamo la prima porzione della Pre, la specifica è ancora valida?
- La specifica data è un "buon" modello anche per l' ordinamento di una lista o un file?

Verso metodi e linguaggi di specifica

Verifiche di correttezza di implementazioni

1. Specificare i requisiti di un algoritmo in un'opportuna logica
2. Implementare l'algoritmo in un opportuno linguaggio
3. Ottenere la correttezza dell'implementazione come dimostrazione (automatizzata) di un teorema

Logica come descrizione di "dati"

- É possibile scegliere un'opportuna logica per descrivere un insieme di concetti
  - e.g., RDF per pagine web, logiche descrittive per dati biomedici
- Se la logica è opportuna (= è possibile calcolare la verità di un dato teorema) possiamo automatizzare la validazione di nostre deduzioni su vaste quantità di dati