# L'uso delle formule logiche come formalismo descrittivo

- Logica: formalismo "universale", molto vicino al linguaggio naturale
- Applicabile a contesti molto vari (non solo informatici, del resto il confine tra computer engineering e system engineering è molto labile)
- Sulla logica sono basati molti formalismi applicativi, dai linguaggi di programmazione a quelli di specifica

- Noi vedremo la logica per:
  1. descrivere linguaggi formali (MFO/MSO)
  2. specificare il comportamento (input/output) di programmi

# Frammento del I ordine della logica monadica: MFO

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

$$\varphi ::= a(x) \mid x < y \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x(\varphi)$$

  – dove $a \in I$, cioè introduciamo una lettera predicativa per ogni simbolo dell'alfabeto
- Interpretazione:
  – $<$ corrisponde alla solita relazione di minore
  – il dominio delle variabili è un sottoinsieme finito di numeri naturali $[0..n-1]$, che sono le *posizioni* dei caratteri nella parola

# Alcune classiche abbreviazioni

- $\varphi_1 \wedge \varphi_2 \quad := \neg(\neg\varphi_1 \vee \neg\varphi_2)$
- $\varphi_1 \rightarrow \varphi_2 \quad := \neg\varphi_1 \vee \varphi_2$
- $\forall x\, (\varphi) \quad := \neg\exists x\, (\neg\varphi)$
- $x = y \quad := \neg(x < y) \wedge \neg(y < x)$
- $x \leq y \quad := \neg(y < x)$
- la costante 0: $\quad x = 0 := \neg\exists y(y < x)$
- successore:
  $succ(x, y) := x < y \wedge \neg\exists z(x < z \wedge z < y)$
- le costanti 1, 2, 3, .... come successore di 0, 1, 2, ...

# Interpretazione come parola sull'alfabeto I

- data una parola $w \in I^*$, ed un simbolo $a \in I$:

  a(x) è vero sse l'x-esimo simbolo di w è a (il primo simbolo di w ha posizione 0)
- Formula che è soddisfatta da tutte e sole le parole il cui primo simbolo è *a*:

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

- Formula che è soddisfatta da tutte le parole in cui ogni *a* è seguita da una *b*

$$\forall x(a(x) \rightarrow \exists y(succ(x,y) \wedge b(y)))$$

# Altri esempi di abbreviazioni e formule

- Usiamo le seguenti abbreviazioni:
  - $y = x + 1$ per $succ(x,y)$
  - più in generale, se k è una costante > 1,
    $y = x + k$ per $\exists z_1 \ldots z_{k-1}(y = z_{k-1} + 1 \wedge z_{k-1} = z_{k-2} + 1 \ldots \wedge z_1 = x+1)$
  - $y = x - 1$ per $succ(y,x)$ (cioè $x = y + 1$)
  - $y = x - k$ per $x = y + k$
  - $last(x)$ per $\neg\exists y(x < y)$
- parole in cui l'ultimo simbolo è *a*: $\exists x(last(x) \wedge a(x))$
- parole (di almeno 3 simboli) in cui il terzultimo simbolo è *a*:
  $\exists x(a(x) \wedge \exists y(y = x+2 \wedge last(y)))$
  - Oppure, abbreviate: $\exists x(a(x) \wedge last(x+2))$, $\exists y(a(y-2) \wedge last(y))$

# Semantica

- Siano $w \in I^*$ e $V_1$ l'insieme delle variabili; un assegnamento è una funzione $v_1 : V_1 \rightarrow [0..|w|-1]$
  - $w, v_1 \models a(x)$ sse $w = uav$ e $|u| = v_1(x)$
  - $w, v_1 \models x < y$ sse $v_1(x) < v_1(y)$
  - $w, v_1 \models \neg\varphi$ sse non $w, v_1 \models \varphi$
  - $w, v_1 \models \varphi_1 \vee \varphi_2$ sse $w, v_1 \models \varphi_1$ oppure $w, v_1 \models \varphi_2$
  - $w, v_1 \models \exists x(\varphi)$ sse $|w| > 0$ ed esiste $i \in [0..|w|-1]$ tale che $w, v_1[i/x] \models \varphi$

  dove $v_1[i/x]$ è come $v_1$ ma assegna ad $x$ il valore $i$

- Linguaggio di una formula chiusa $\varphi$:

$$L(\varphi) = \{ w \in I^* \mid w \models \varphi \}$$

- Nota: la stringa vuota non può soddisfare q. esistenziali, quindi soddisfa solo quantificazioni universali

# Proprietà di MFO

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

# Proprietà di MFO (2)

- In MFO non si può esprimere il linguaggio $L_p$ fatto di tutte e sole le parole di lunghezza *pari* con I = {a}
(nota bene: alfabeto unario)

- 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

# Proprietà di MFO (3)

- I linguaggi definiti da MFO non sono chiusi rispetto alla * di Kleene:
- la formula MFO $a(0) \wedge a(1) \wedge last(1)$ definisce il linguaggio $L_{p2}$ fatto della sola parola {aa} di lunghezza 2.
- Abbiamo che $L_p = L_{p2}^*$
- MFO definisce i cosiddetti linguaggi star-free, cioè definibili tramite unione, intersezione, complemento e concatenazione di linguaggi finiti

# Logica monadica del secondo ordine (MSO)

- Per ottenere lo stesso potere espressivo degli FSA serve permettere di *quantificare su predicati monadici*
  - logica del I ordine: quantificazioni su **posizioni**
  - logica del II ordine: quantificazione su **insiemi di posizioni**, dove un insieme di posizioni è codificato da un **predicato monadico**: P(x) vuol dire x è una posizione per cui vale P(x)
- Ammettiamo quindi formule del tipo $\exists X(\varphi)$, dove X è una variabile il cui dominio è l'insieme dei predicati monadici
- per convenzione usiamo le lettere maiuscole per indicare variabili con dominio l'insieme dei predicati monadici, e lettere minuscole per indicare variabili sulle posizioni

# Semantica MSO

- 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 $v_1(x) \in v_2(X)$
  - $w, v_1, v_2 \models \exists X(\varphi)$ sse $|w| > 0$ ed esiste $S \subseteq [0..|w|-1]$ tale che $w, v_1, v_2[S/X] \models \varphi$

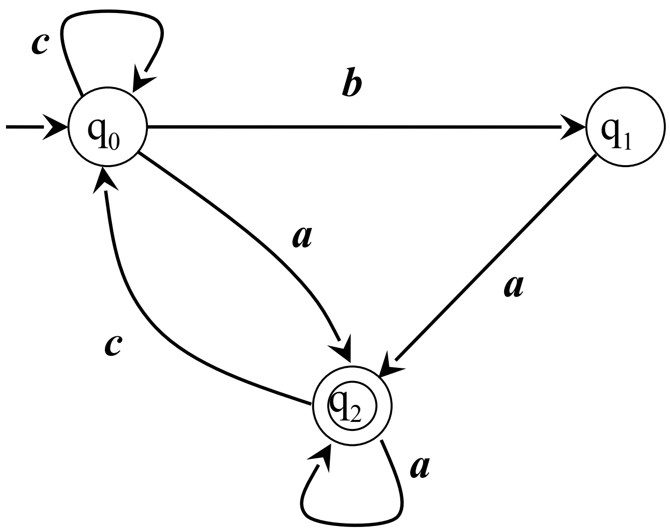
dove $v_2[S/X]$ è come $v_2$ ma assegna S ad X.

- Possiamo dunque scrivere la formula che descrive il linguaggio $L_p$

$$
\begin{aligned}
\exists P(\forall x(\neg P(0) \wedge \\
&(\neg P(x) \leftrightarrow P(x+1)) \wedge \\
&a(x) \wedge (\, last(x) \rightarrow P(x)\,) \quad ))
\end{aligned}
$$

# Da FSA a MSO

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




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

# Da MSO a FSA

- Data una formula MSO $\varphi$, è possibile costruire un FSA che accetta esattamente il linguaggio L definito da $\varphi$

  (Teorema di Büchi-Elgot-Trakhtenbrot)
  - la dimostrazione dell'esistenza è costruttiva (mostra come ottenere un FSA da una formula MSO) ed è disponibile sulla dispensa per gli interessati

- Quindi la classe dei linguaggi definibili da formule MSO coincide con i linguaggi regolari

# 2. La logica per definire proprietà dei programmi

- Specifica di un algoritmo di ricerca:
  La variabile logica *found* deve essere vera se e solo se esiste un elemento dell'array $a$, di $n$ elementi, uguale all'elemento cercato $x$:
  $found \leftrightarrow \exists i(1 \leq i \leq n \wedge a[i] = x)$
- Specifica di un algoritmo di inversione di un array:
  $\forall i(1 \leq i \leq n \rightarrow b[i] = a[n - i + 1])$

# Più in generale

- {Precondizione: *Pre*}
  Programma - o frammento di programma - *P*
  {Postcondizione: *Post*}

- Se vale *Pre* prima dell'esecuzione di *P* si vuole che *P* sia tale da far valere *Post* dopo la sua esecuzione

# Esempi

- *Ricerca in un array ordinato*:

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

  $P$

  $\{found \leftrightarrow \exists i(1 \leq i \leq n \wedge a[i] = x)\}$

- NB: ciò non significa affatto che P debba essere un algoritmo di ricerca binaria. Significa solo che chi lo realizza può sfruttare il fatto che *a* sia ordinato prima dell'esecuzione di P.
- Un normale algoritmo di ricerca sequenziale sarebbe corretto rispetto a questa specifica; al contrario un algoritmo di ricerca binaria non sarebbe corretto rispetto ad una specifica che avesse come precondizione semplicemente *True*.

*Ordinamento di un array di n elementi senza ripetizioni*:

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

*ORD*

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

è una specifica “adeguata”?

(Pensiamo all’analogia “specifica = contratto”) meglio:

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

- Se eliminiamo la prima riga della precondizione la specifica è ancora valida?
- In realtà anche un concetto ben noto come l'ordinamento è esposto a imprecisioni ed equivoci nell'uso informale del termine
- Pensiamo a requisiti del tipo "vogliamo automatizzare il rilascio di certificati, o la gestione dei cc bancari"