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
  3. specificare le proprietà di sistemi temporizzati

1. La logica per definire linguaggi: variante generale

- In realtà l'abbiamo già fatto:

  $\{a^n b^n \mid n > 0\}$

  altro non è che un'abbreviazione della formula del prim'ordine:

  $x \in L \leftrightarrow \exists n\, (n \geq 1 \wedge x = a^n b^n)$

  Dove a sua volta l'operazione x^n è definita da:

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

  sulla base del (simbolo di) operazione elementare '.'

- Similmente, $L_1 = a^*b^*$ è definita da:
  $x \in L_1 \leftrightarrow x = \varepsilon \vee \exists y\,(x = a.y \wedge y \in L_1) \vee \exists y\,(x = y.b \wedge y \in L_1)$
- Posto $L_2 = b^*c^*$ e definito in maniera simile
- $L_3 = a^*b^*c^*$ ($= L_1.L_2$) è definito *anche* come:

$x \in L_3 \leftrightarrow$

$(x \in L_1) \vee (x \in L_2) \vee$

$\exists y\,((x = a.y \wedge (y \in L_2 \vee y \in L_3)) \vee (x = y.c \wedge (y \in L_3 \vee y \in L_1))$

- $L_4 = \{x \mid \#x_a = \#x_b\}$ con $\#x_a$ definita da:

$(x = \varepsilon \rightarrow \#x_a = 0) \wedge (x = a.y \rightarrow \#x_a = \#y_a + 1) \wedge$

$(x = b.y \rightarrow \#x_a = \#y_a)$

(con quantificazione <u>implicita</u> $\forall x\ \forall y$)

Logica monadica del prim'ordine: MFO

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

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

  - laddove a ∈ 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]

Alcune classiche abbreviazioni

- ovviamente:
  - $\varphi_1 \wedge \varphi_2 \quad \triangleq \quad \neg(\neg\varphi_1 \vee \neg\varphi_2)$
  - $\varphi_1 \Rightarrow \varphi_2 \quad \triangleq \quad \neg\varphi_1 \vee \varphi_2$
  - $\forall x\,(\varphi) \quad \triangleq \quad \neg\exists x\,(\neg\varphi)$
  - $x = y \quad \triangleq \quad \neg\,(x < y) \wedge \neg\,(y < x)$
  - $x \leq y \quad \triangleq \quad \neg\,(y < x)$
- ma possiamo anche definire:
  - la costante 0: $x = 0 \quad \triangleq \quad \forall y(\neg(y < x))$
  - la relazione successore:
    $succ(x, y) \quad \triangleq \quad x < y \wedge \neg\exists z(x < z \wedge z < y)$
  - 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)
- Formula che è vera su tutte e sole le parole il cui primo simbolo esiste ed è a:

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

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

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

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 $\exists z_1 \ldots z_{k-1}(y = z_{k-1}+1 \wedge \ldots \wedge z_1 = x+1)$
  - 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:
  $\exists x\,(\text{last}(x) \wedge a(x))$
- parole (di almeno 3 simboli) in cui il terzultimo simbolo è a:
  $\exists x\,(a(x) \wedge \exists y\,(y = x+2 \wedge \text{last}(y)))$
  - Oppure (abbreviate): $\exists x\,(a(x) \wedge \text{last}(x+2))$, $\exists y\,(a(y-2) \wedge \text{last}(y))$

Semantica

- Siano $w \in I^+$ e V_1 l'insieme delle variabili; un assegnamento è una funzione $v_1 : V_1 \rightarrow [0..|w|-1]$
  - $w, v_1 \models a(x)$ 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 esiste v'_1 con $v'_1(y) = v_1(y)$, $y \neq x$ tale che $w, v'_1 \models \varphi$
- Linguaggio di una formula chiusa φ:
  - $L(\varphi) = \{ w \in I^+ \mid w \models \varphi \}$

Proprietà di MFO

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

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}
- 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 $a(0) \wedge a(1) \wedge last(1)$ 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 "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 $\exists X(\varphi)$, 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 sui numeri naturali

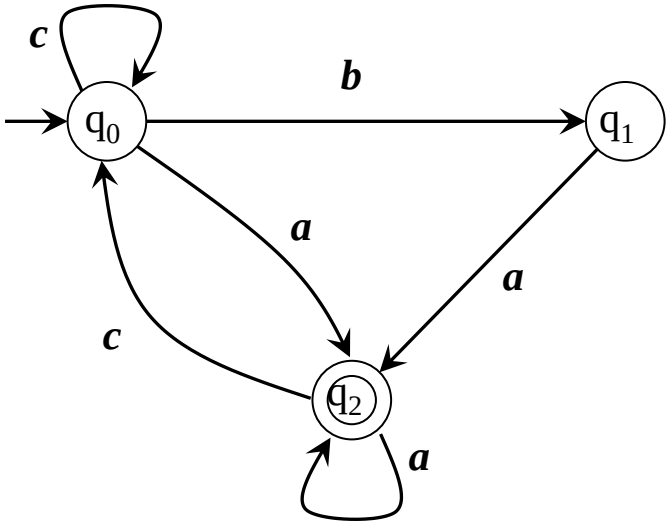
Semantica ed esempio

- 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 $v_1(x) \in v_2(X)$
  - $w, v_1, v_2 \models \exists X(\varphi)$ sse esiste v'_2, $v'_2(Y) = v_2(Y)$, $Y \neq X$ tale che $w, v_1, v'_2 \models \varphi$
- Possiamo dunque scrivere la formula che descrive il linguaggio L_p

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

Da FSA a MSO

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



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

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

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:
- Ricerca in un array ordinato:
  $\{\forall i(1 \leq i < n \rightarrow a[i] \leq a[i+1])\}$
  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

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

  E' una specifica "adeguata"?
  (Pensiamo all'analogia "specifica = contratto")

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

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?
- Siamo sicuri che il problema dell'ordinamento venga sempre inteso alla stessa maniera, sia che si tratti di ordinare un array o una lista o un file?
- 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"

3. La logica per la specifica di proprietà di sistemi

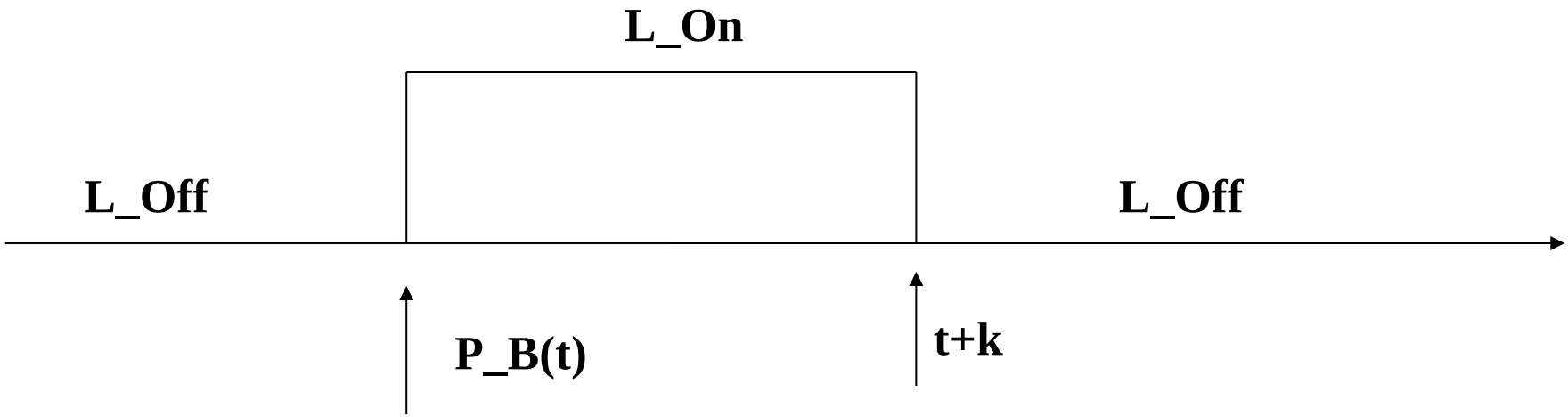
- "Se premo il pulsante si accende la luce entro Δ istanti":
  - P_B(t): Predicato che indica la pressione del pulsante all'istante t
  - L_On(t): Predicato che indica che all'istante t la luce è accesa

$$\forall t(P\_B(t) \rightarrow \exists u(t \leq u \leq t+\Delta \wedge L\_On(u)))$$

In realtà una specifica siffatta lascia molto a desiderare rispetto a quanto normalmente si chiede ad un pulsante di accensione della luce.

Scendiamo in qualche dettaglio

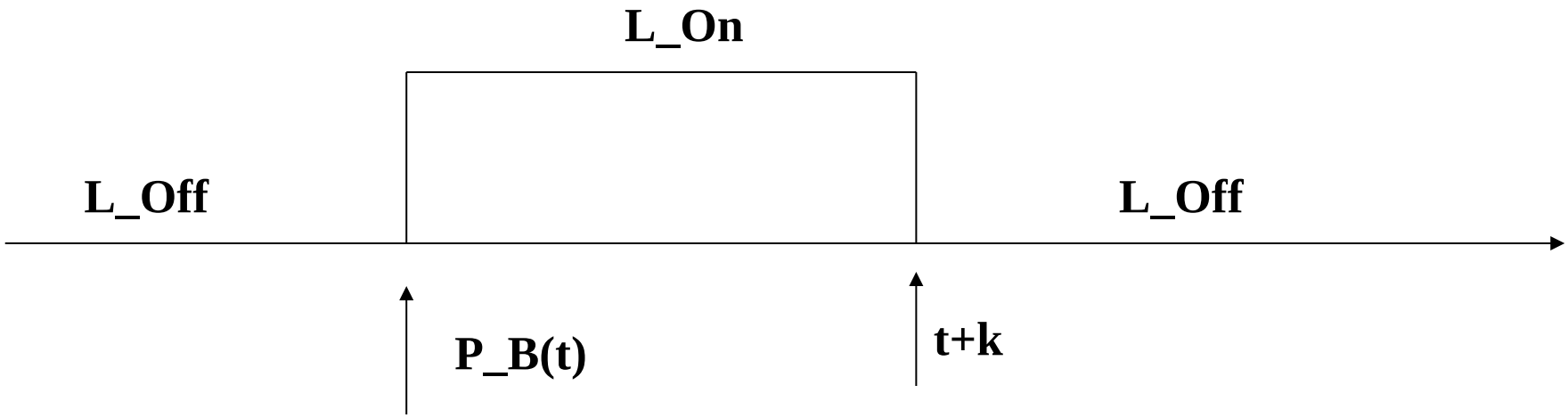
- Focalizziamo l'attenzione su un classico "pulsante a tempo" per la luce (se eccita di più la fantasia: un allarme e/o chiusura a tempo per cassaforti)



$$\forall t(P\_B(t) \rightarrow \forall u(t \leq u < t + k \rightarrow L\_On(u)) \wedge \forall v(t+k \leq v \rightarrow L\_Off(v))$$

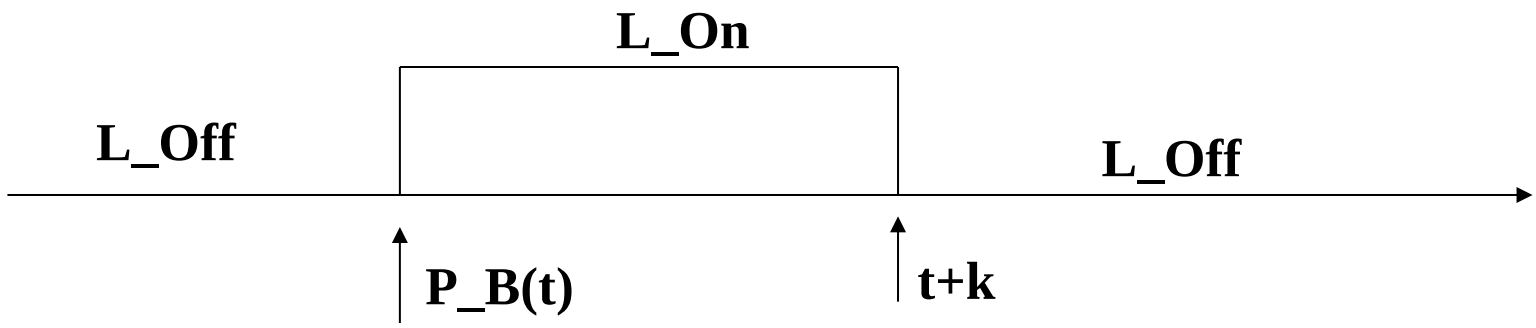
- **In realtà ci sono ancora molte cose che non vanno, un po' di caccia all'errore ...**

Proviamo questa:



$$\forall t(P\_B(t) \wedge L\_Off(t) \rightarrow \forall u(t \leq u < t + k \rightarrow L\_On(u)) \wedge L\_Off(t+k)) \wedge \forall v,w(L\_Off(v) \wedge \forall x(v \leq x \leq w \rightarrow \neg P\_B(x)) \rightarrow L\_Off(w))$$

Ancora un tentativo:



$\forall t(L\_On(t) \leftrightarrow \neg L\_Off(t)) \wedge$ *(discutibile)*

$\forall t(P\_B(t) \rightarrow \exists d(\forall u(t\text{-}d < u < t \vee t < u < t+d) \rightarrow \neg P\_B(u))) \wedge$

$\forall t(P\_B(t) \wedge \exists d(\forall u(t\text{-}d < u < t \rightarrow L\_Off(u))) \rightarrow$

$\forall u(t \leq u < t + k \rightarrow L\_On(u)) \wedge L\_Off(t+k)) \wedge$

$\forall v,w(L\_Off(v) \wedge \forall x(v \leq x \leq w \rightarrow \neg P\_B(x)) \rightarrow L\_Off(w))$

Variazioni sul tema:

- Pulsante di spegnimento
- Mantenimento luce accesa mediante pressione durante l'accensione
- Apertura e chiusura tende/paratoie/finestrini auto, …
  - Mantenimento pressione pulsanti
  - Interruzione o non interruzione movimento in corso
  - …..

- **Generalità e sistematicità dell'approccio**

- **Verso *metodi e linguaggi di specifica***

Dalla specifica alla prova: un cenno

- Dopo aver *specificato* i requisiti di un algoritmo (e.g., di ordinamento) e dopo aver *costruito* un tale algoritmo, occorre *verificare* la correttezza del medesimo:
  se ho a disposizione un modello matematico (e.g., *un'assiomatizzazione*) dell'implementazione costruita, in linea di principio posso ottenere la prova di correttezza come una *dimostrazione di teorema*.